cadence®

Encounter® Test: Reference: Test Pattern Formats

Product Version 12.1.101
February 2013

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Contents

List of Figures

Preface

Typographic and Syntax Conventions

The Encounter Test library set uses the following typographic and syntax conventions.

- Text that you type, such as commands, filenames, and dialog values, appears in Courier type.

  Example: Type `build_model -h` to display help for the command.

- Variables appear in Courier italic type.

  Example: Use `TB_SPACE_SCRIPT=`*`input_filename`* to specify the name of the script that determines where Encounter Test binary files are stored.

- Optional arguments are enclosed in brackets.

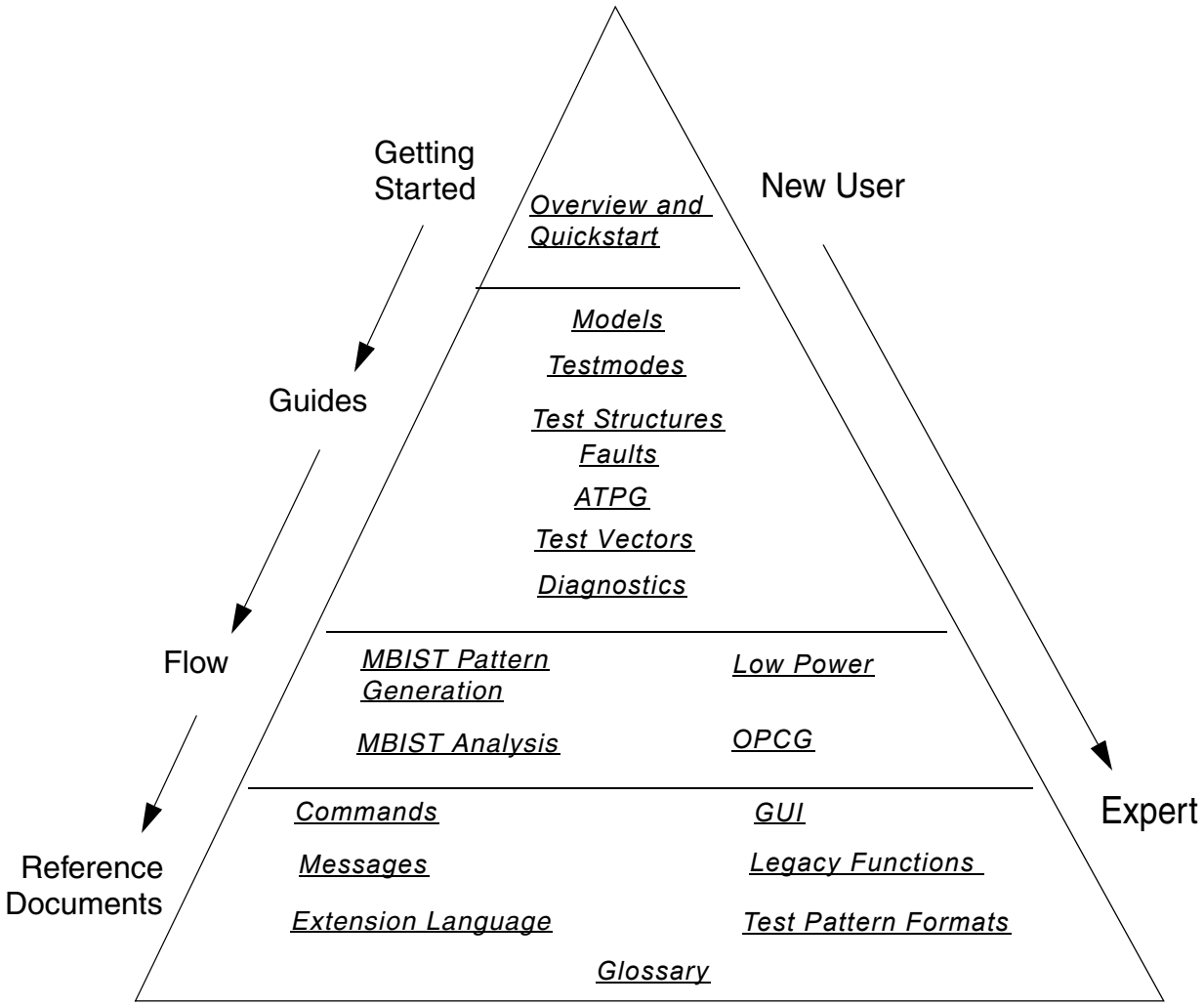
  Example: `[simulation=gp|hsscan]`

- User interface elements, such as field names, button names, menus, menu commands, and items in clickable list boxes, appear in Helvetica italic type.

  Example: Select *File - Delete - Model* and fill in the information about the model.

Encounter Test Documentation Roadmap

The following figure depicts a recommended flow for traversing the documentation structure.



Getting Help for Encounter Test and Diagnostics

Use the following methods to obtain help information:

1. From the *<installation_dir>*`/tools/bin` directory, type `cdnshelp` at the command prompt.

2. To view a book, double-click the desired product book collection and double-click the desired book title in the lower pane to open the book.

Click the *Help* or *?* buttons on Encounter Test forms to navigate to help for the form and its related topics.

Refer to the following in the *Encounter Test: Reference: GUI* for additional details:

- "Help Pull-down" describes the *Help* selections for the Encounter Test main window.
- "View Schematic Help Pull-down" describes the Help selections for the Encounter Test View Schematic window.

Contacting Customer Service

There are several ways to get help for your Cadence product.

- Cadence Online Customer Support

  Cadence online customer support offers answers to your most common technical questions. It lets you search more than 40,000 FAQs, notifications, software updates, and technical solutions documents that give step-by-step instructions on how to solve known problems. It also gives you product-specific e-mail notifications, software updates, service request tracking, up-to-date release information, full site search capabilities, software update ordering, and much more.

  Go to http://www.cadence.com/support/pages/default.aspx for more information on Cadence Online Customer Support.

- Cadence Customer Response Center (CRC)

  A qualified Applications Engineer is ready to answer all of your technical questions on the use of this product through the Cadence Customer Response Center (CRC). Contact the CRC through Cadence Online Support. Go to http://support.cadence.com and click the *Contact Customer Support* link to view contact information for your region.

- IBM Field Design Center Customers

  Contact IBM EDA Customer Services at 1-802-769-6753, FAX 1-802-769-7226. From outside the United States call 001-1-802-769-6753, FAX 001-1-802-769-7226. The e-mail address is edahelp@us.ibm.com.

Encounter Test And Diagnostics Licenses

Refer to "Encounter Test and Diagnostics Product License Configuration" in *Encounter Test: Release: What's New* for details on product license structure and requirements.

Using Encounter Test Contrib Scripts

The files and Perl scripts shipped in the `<ET installation path>/ etc/tb/ contrib` directory of the Encounter Test product installation are not considered as "licensed materials". These files are provided AS IS and there is no express, implied, or statutory obligation of support or maintenance of such files by Cadence. These scripts should be considered as samples that you can customize to create functions to meet your specific requirements.

What We Changed for This Edition

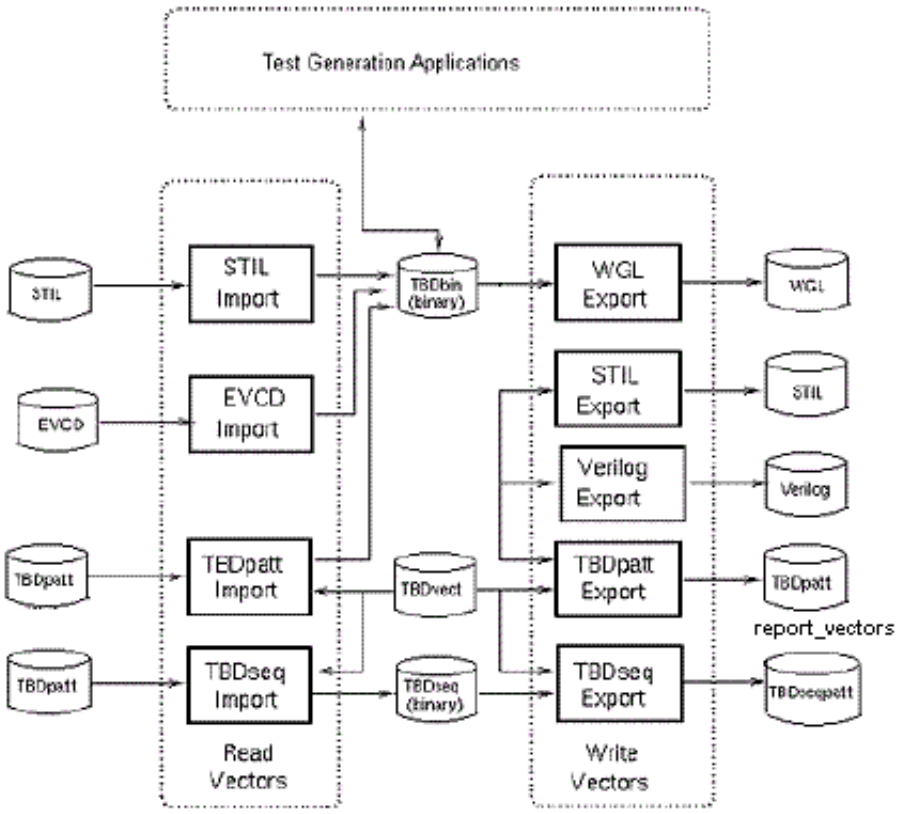
There are no significant modifications specific to this version of the manual.

1

Test Vector Overview

Figure 1-1 shows the interaction of Encounter Test applications and resulting vectors.

Figure 1-1 Test Data Interface Overview



An Overview to Test Vector Formats

The `write_vectors` command can create output in WGL, STIL, and Verilog for the generated test vectors. The following sections cover each of the output languages:

- WGL Pattern Data Format on page 144
- STIL Pattern Data Format on page 159
- Verilog Pattern Data Format on page 173

Note: Encounter Test can read STIL and EVCD formats to import test patterns. EVCD, a part of the IEEE Standard 1364-2001, is the recommended format for importing test patterns. Refer to read_vectors in the *Encounter Test: Reference: Commands* for more information.

Encounter Test can also create a proprietary version of the test pattern data known as TBDpatt. Refer to TBDpatt and TBDseqPatt Format on page 15 for more information. Encounter Test creates the TBDpatt format by using the report_vectors command. Refer to report_vectors in the *Encounter Test: Reference: Commands* for more information.

Use the TBDpatt format to assign unique event sequences for custom control logic within a design, such as OPCG (On Product Clock Generator). To assign unique test sequences, build a test mode using the following command:

`build_testmode seqdef=`*TBDpatt filename*

Then run ATPG to create test patterns and introduce a special pattern application sequence:

`create_tests sequencefile=`*TBDseqpatt filename*

Refer to build_testmode in the *Encounter Test: Reference: Commands* for more information.

Tip

The `create_tests` command will be removed in the next major release. Run `create_scanchain_tests` and then run `create_logic_tests` instead of `create_tests`.

To generate test patterns, use the following command:

`write_vectors language=`*<wgl | stil | verilog>*

Refer to write_vectors in the *Encounter Test: Reference: Commands* for more information.

The parameters of the write_vectors command are independent of the target test language. For a complete list of the supported parameters, run the write_vectors -H command.

Note: Select *ATPG - Write Vectors* to generate test patterns using the graphical user interface.

2

Test Vector Formats

This chapter describes the following test pattern formats:

TBDpatt and TBDseqPatt Format

This section defines the TBDpatt and TBDseqPatt test pattern formats. These formats are proprietary to Cadence Encounter Test. The TBDpatt files are produced as output and consumed as input. Encounter Test also produces industry standard WGL, STIL, and Verilog test pattern formats. Refer to An Overview to Test Vector Formats on page 12 for more information.

TBDpatt files contain test patterns (experiments) in ASCII form. TBDseqPatt files are a specialized form of TBDpatt files containing sequence definitions in ASCII form and a subset of the statements used in TBDpatt files. The structure of both these files is identical, but they contain different types of data. TBDpatt files contain actual test pattern data and the test sequences and timing information associated with that data. TBDseqPatt files contain only sequence definitions and associated application objects, such as timing data, with no vector data. Therefore, the contents of TBDseqPatt files cannot be directly applied for fault simulation or written out in STIL, WGL, or Verilog. We commonly refer to the language of both the TBDseqPatt and TBDpatt files as the TBDpatt language.

The main purpose of TBDseqPatt files is to supply Encounter Test with any special sequences that must be applied. Some of the commonly-used special sequences are mentioned below:

- Initialize the design
  - This is referred to as the Mode Initialization Sequence or `modeinit`

- An example of this is turning off the IEEE 1149.1 boundary scan logic to access the parallel scan chains in the design or programming the on-product clocking logic and synchronizing the PLLs.

- Define a special `test` application sequence
  - This can be used to force test patterns created in Encounter Test have a user-defined form and to ensure the correct operation of the hardware in an on-product clocking environment.
- Special sequences to enter into or out of scan shift modes of operation
  - The Scan Preconditioning Sequence (`scanprecond`) is used to set up the design state to initialize a scan operation.
  - The Scan Exit Sequence (`scanexit`) is used to set up the design for test operation when the scan operation is complete.
- Define any special sequences that must be applied during the actual scan shift operation
  - The Scan Sequence (`scansequence`) actually applies the scan load and measure values and performs the shift operation.

To import a special initialization or scan sequence, provide a TBDpatt file as input when building a test mode:

```
build_test_mode testmode=OPCG_mode seqpath=<directory path> seqdef=<A TBDpatt
input file with "modeinit">
```

To import a file with special sequences to apply the test patterns, provide a TBDseqPatt file on the pattern generation command line, as follows:

```
create_logic_tests testmode=OPCG_mode sequencefile=<TBDseqPatt file>
testsequence=<A sequence defined in the TBDseqPatt file>
```

Or read in the definitions in a separate step using `read_sequence_definition`, as follows:

```
read_sequence_definition testmode=OPCG_mode importfile=<TBDseqPatt file>
create_logic_tests testmode=OPCG_mode testsequence=<A sequence defined in the
TBDseqPatt file>
```

To produce a TBDpatt file from an existing Encounter Test experiment, use the report_vectors command. To view the automatically-generated TBDseqPatt sequences for MBIST and OPMISR+ test modes, use the report_sequences command. An example for an OPMISR+ compression logic test mode is given below. The output for an OPMISR+ test mode will also contain some of the above-mentioned sequences.

```
report_sequences testmode=OPMISRPLUS outputfile=TBDseqPatt.OPMISRPLUS format=node
```

The test pattern data itself is organized in a hierarchy of objects. The higher level objects group the lower level objects in a meaningful way. At the lowest level of the hierarchy are events which represent the actual test pattern data. In the TBDpatt and TBDseqPatt files, each of these objects is identified by a numbering system which reflects its position in the test data hierarchy.

Each object may contain special application objects and keyed data in addition to the test data hierarchy. Refer to "Application Object" on page 110 for detailed information about the application objects. Refer to "Keyed Data" on page 137 for detailed information about keyed data.

Detailed information about each object is included in the following sections.

Following is the structural hierarchy of a `TBDseqPatt` file. The file is shown as the top level of the hierarchy. It consists of a header, vector correspondence lists (as comments) and sequence definitions.

```
TBDseqPatt file header
      Include "fileName";
      //Vector Correspondence List
      Sequence Definition
      Pattern
      Event
```

The header indicates that this file is in the TBDpatt format and describes how the input is represented within the file.

The vector correspondence data contains information on how to map an event's data, which is in vector form, to individual pins or registers.

A sequence definition contains patterns, which contain events. These define the specific actions to be taken on the tester or in the simulation environment.

Following is the structural hierarchy of a `TBDpatt` file. The file is shown as the top level of the hierarchy. It consists of vector correspondence lists (as comments) and experiment blocks. If audits indicate that the test patterns contained in these Vectors may be suspect, an audit summary is written to the pattern file following the vector correspondence. An experiment block is a collection of test sections, each of which is a collection of tester loops, and so on.

```
TBDpatt file header
// Vector Correspondence List
   Experiment Block
     Test_Section
        Tester_Loop
            Test_Procedure
               Test_Sequence
                   Pattern
                        Event
```

For additional information about test data structural hierarchy, refer to "Encounter Test Vector Data" in the *Encounter Test: Guide 6: Test Vectors*.

`TBDpatt` data can exist in two forms:

- with stimulus and response data in vector form
- with stimulus and response data in node list form

In the vector form, all primary input, primary output, scannable latch, and weight values are listed as strings of logic values. The pin to which the value applies is determined by the position of the value in the vector. An example of a PI stimulus vector is:

```
Stim_PI (): 0110111;
```

By looking at the example, it is possible to determine that this design has seven primary inputs. This is the case since all vectors must fully specify the entity type that is being stimulated. The correspondence of each vector value to a particular PI pin is specified in the vector correspondence. This can be written in a separate file using write_vector_correspondence or included when writing the TBDpatt or TBDseqPatt files. The vector correspondence is used to determine this relationship for both import and export of TBDpatt. It is determined automatically by the system and cannot be directly modified by the user. All user-created test sequences in vector form must conform to the order specified in the vector correspondence.

The following is an example of vector correspondence data:

```
#Vector_Correspondence
#
# Note:
#   If the original position of any scancell is changed within the Scan_Load or
#     Scan_Unload vectors below, it is possible that some of the information
#     included in the controllable and observable scan chain commentary will no
```

```
#     longer be true.  The potentially incorrect fields would include,
#     First_Stim_bit, Last_Stim_Bit, First_Meas_Bit and Last_Meas_Bit.
#
# Legend:
#
#  tf     => the test function for the corresponding primary input/output pin
#            AC    => A shift clock
#            AS    => A shift and system clock
#            BC    => B shift clock
#            BDY   => boundary (test pin -- data)
#            BI    => bi-directional inhibit
#            BS    => B shift and system clock
#            CHI   => channel input
#            CHO   => channel output
#            CI    => clock isolation
#            CMI   => channel mask input
#            CME   => channel mask enable
#            CML   => channel mask load clock
#            CMLE  => channel mask load enable
#            CMI_A => channel mask load A clock
#            CMI_B => channel mask load B clock
#            CTL   => boundary (test pin -- control)
#            EC    => shift clock for edge-sensitive flip-flops
#            ES    => clock for both shift and system function of edge-sensitive
#                  flip-flops.
#            LH    => linehold
#            ME    => MISR enable
#            MO    => MISR observe
#            MMI   => MISR mask input
#            MME   => MISR mask enable
#            MML   => MISR mask load clock
#            MMLE  => MISR mask load enable
#            MMI_A => MISR mask load A clock
#            MMI_B => MISR mask load B clock
#            MRE   => MISR reset Enable
#            MRD   => MISR read
#            MRST  => MISR reset
#            NIC   => no interconnect
#            OI    => output inhibit
#            PC    => P clock
#            PGE   => PRPG load enable
#            PLD   => PRPG load
#            PR    => PRPG restore
#            PS    => P and system clock
#            PV    => PRPG save
#            SC    => system clock
#            SE    => scan enable
#            SI    => scan-in
#            SIG   => scan in gate
#            SO    => scan-out
#            SOF   => scan out fill
#            SOG   => scan out gate
#            TC    => test constraint
#            TCK   => test clock
#            TI    => test inhibit
#            TMS   => test mode select
#            TRST  => test reset
#            WS    => weight select
```

```
#
#  index  => Encounter Test Manufacturing model pin index
#
#
# PI:
#     (PI 1 = "Pin.f.l.fulladder.nl.A",   #  index = 0
#      PI 2 = "Pin.f.l.fulladder.nl.B",   #  index = 1
#      PI 3 = "Pin.f.l.fulladder.nl.CLK",   #  index = 2  tf = -ES
#      PI 4 = "Pin.f.l.fulladder.nl.Test",   #  index = 3  tf = +SE
#      PI 5 = "Pin.f.l.fulladder.nl.carryin",   #  index = 4
#      PI 6 = "Pin.f.l.fulladder.nl.scanin")   #  index = 5  tf =  SI
#
# PO:
#     (PO 1 = "Pin.f.l.fulladder.nl.SnC",   #  index = 6
#      PO 2 = "Pin.f.l.fulladder.nl.carryout",   #  index = 7  tf =  SO
#      PO 3 = "Pin.f.l.fulladder.nl.sum")   #  index = 8
#
# Scan_Chain Definition
#   Legend:
#     Load_Node      => Pin where logic values are placed for transfer into a
#                       scancell via the load operation (e.g. a scan-in primary
#                       input).
#     Unload_Node    => Pin where scancell logic values appear as the result of an
#                       unload operation (e.g. a scan-out primary output).
#    index        => Encounter Test Manufacturing model pin index for load/unload
nodes
#                       index for stim/measure scancells.
#     Load_Sect      => The id of the load section.
#     Unload_Sect    => The id of the unload section.
#     Bit_Length     => The number of bit positions in controllable/observable
#                       scan chain.
#     Number_Of_RSLs => The number of Representative Stim Latches/Flops in the
#                       controllable scan chain.
#    Number_Of_SSLs => The number of Skewed Stim Latches/Flops in the controllable
#                       scan chain.
#     Number_Of_RMLs => The number of Representative Measure Latches/Flops in the
#                       observable scan chain.
#     First_Stim_Bit => The bit position in the Scan_Load vector where RSL
#                       values for this scan chain begin.
#     Last_Stim_Bit  => The bit position in the Scan_Load vector where RSL
#                       values for this scan chain stop.
#     First_Meas_Bit => The bit position in the Scan_Unload vector where RML
#                       values for this scan chain begin.
#     Last_Meas_Bit  => The bit position in the Scan_Unload vector where RML
#                       values for this scan chain stop.
#
# controllable scan chain 1:
#    Load_Node = "Pin.f.l.fulladder.nl.scanin"   index = 5
#    Load_sect = 3
#    Bit_Length = 5
#    Number_Of_RSLs = 5#    Number_Of_SSLs = 0
```

```
#    First_Stim_Bit = 1  Last_Stim_Bit = 5
#
# observable scan chain 1:
#    Unload_Node = "Pin.f.l.fulladder.nl.carryout"   index = 7
#    Unload_sect = 3
#    Bit_Length = 5
#    Number_Of_RMLs = 5
#    First_Meas_Bit = 1  Last_Meas_Bit = 5
#
#  RSL    => Representative Stim Latch/Flop
#  SSL    => Skewed Stim Latch/Flop
#  RML    => Representative Measure Latch/Flop
#  CR     => The id of the controllable scan chain which includes this scancell.
#            This id can be correlated to the scan chain definition
#            information listed above.
#  OR     => The id of the observable scan chain which includes this scancell.
#            This id can be correlated to the scan chain definition
#            information listed above.
#  pos    => Position in the scan chain which this scancell occupies.
#            For a controllable scan chain, the first scancell which receives a
#            value from the load node is in position 1 (i.e., scancell
#            closest to the load node).  For a observable scan chain, the
#            scancell whose value reaches the unload node first is in
#            position 1 (i.e., scancell closest to the unload node).
#  index =>  Encounter Test Manufacturing hierModel index for the RSL,
#            RML or SSL (scancell) block.
# invert => "Yes" means there is inversion in the scan chain
#            between this scancell and the scan chain I/O pin.  For
#            a stim scancell, the inversion is with respect to the scan
#            chain input pin;  for a measure scancell, the inversion
#            is with respect to the scan chain output pin.
#            "No" means there is no inversion between the scan chain
#            I/O pin and the scancell.
#
# Scan_Load:
#     (RSL 1 = "Block.f.l.fulladder.nl.chain1.ScanReg.slave",   #  CR = 1  pos = 1
index = 13  invert = no
#      RSL 2 = "Block.f.l.fulladder.nl.chain2.ScanReg.slave",   #  CR = 1  pos = 2
index = 19  invert = no
#      RSL 3 = "Block.f.l.fulladder.nl.chain3.ScanReg.slave",   #  CR = 1  pos = 3
index = 25  invert = no
#      RSL 4 = "Block.f.l.fulladder.nl.chain4.ScanReg.slave",   #  CR = 1  pos = 4
index = 31  invert = no
#      RSL 5 = "Block.f.l.fulladder.nl.chain5.ScanReg.slave")   #  CR = 1  pos = 5
index = 37  invert = no
#
# Scan_Unload:
#     (RML 1 = "Block.f.l.fulladder.nl.chain5.ScanReg.slave",   #  OR = 1  pos = 1
index = 37  invert = no
#      RML 2 = "Block.f.l.fulladder.nl.chain4.ScanReg.slave",   #  OR = 1  pos = 2
index = 31  invert = no
```

```
#     RML 3 = "Block.f.l.fulladder.nl.chain3.ScanReg.slave",   #  OR = 1  pos = 3
index = 25  invert = no
#     RML 4 = "Block.f.l.fulladder.nl.chain2.ScanReg.slave",   #  OR = 1  pos = 4
index = 19  invert = no
#     RML 5 = "Block.f.l.fulladder.nl.chain1.ScanReg.slave")   #  OR = 1  pos = 5
index = 13  invert = no
#      ;
# End of vector correspondence
```

This vector correspondence data describes a 1-bit full adder design with a 5-bit controllable and observable scan chain running through it. The scan chain loads through a primary input named scan_in and is observed through the primary output named carry_out. The above-mentioned data shows the order in which all of the primary inputs, primary outputs, and scan chains will be presented when test pattern data is written out in vector (as opposed to node) form in the TBDpatt or TBDseqPatt files. For example, to stim the scan enable pin named Test to a 1, the vector data would be as follows:

```
Event 1.1.1 Stim_PI(): ...1..;
```

Similarly, to measure a value of 0 from only the register named chain4.ScanReg.slave, the vector data would be as follows:

```
Event 1.4.1 Scan_Unload (default_value = X): .0...;
```

In the node list form, PIs, POs and scannable latches are listed as a node=logic value pair, where node specifies the pin or flop being referenced. For example,

```
Stim_PI ():
     "Pin.f.l.PGMUX.n1.DATA00"=0
     "Pin.f.l.PGMUX.n1.DATA01"=0;
```

The above-mentioned example shows the pin names specified in full form. You can also use the short form of the name, for example:

```
Stim_PI ():
     "DATA00"=0
     "DATA01"=0;
```

In the short form of the name, the top-level block name (PGMUX) and type of entity (Pin) are implicit.

This form of output generally consumes more space than the vector form, but in some situations is easier to analyze visually. `Pulse`, `Stim_Clock`, `Pulse_PPI`, and `Stim_PPI_Clock` events are always written in this form, regardless of the form specified to be the TBDpatt format header.

For both vector and node list formats, all referenced nodes are given in either name or index form. The name form specifies the node as a hierarchical pin or block name and must be contained within quotation marks. The index form specifies the node as an Encounter Test model index.

A `TBDpatt` file in name form can often be imported and reused for an edited version of that same design, as long as the primary input and output pin names have not changed and the hierarchical latch names have not changed. Such processing is not recommended if the index form is used since the model indices will almost assuredly be different between the original design and a modified version of that same design. The vector formatted `TBDpatt` files may be reusable if the number of primary I/O pins, latches, and their vector ordering does not change, but some changes to test mode pin assignments may cause warnings or errors when simulating these vectors. For example, if an unassigned PI is changed to a +TI pin assignment, your TBDpatt pattern data may have contained an assignment to 0 on this pin, causing an error when simulated in the new mode.

The description of `TBDpatt` which follows contains some references to test mode names. To understand this, it is important to know that, while Vector data is stored in a test mode-qualified file, there are some situations in which, during the initialization of a test mode, it is necessary to switch temporarily to another test mode. As an example, in a test for an LBIST mode, initialization of fixed-value latches and linear feedback shift registers (LFSRs) can be accomplished by switching to a test mode in which these latches are scannable, using that test mode to initialize them, and then switching to the target (LBIST) mode.

The format and syntax of the `TBDpatt` file will be enhanced over time to include new features. This means that future `TBDpatt` files generated by Encounter Test may contain new attributes, objects, or event types. It is intended that these changes in syntax be backward compatible; where possible, the new features will be made optional so that older `TBDpatt` files will continue to be usable by Encounter Test.

The comment characters allowed in `TBDpatt` and `TBDseqPatt` files are:

line comments:

- *//*
- --
- *#*

block comments:

- */* block of comments* /* - *" /*"* to start and *"*/"* to end block comments.

TBDpatt_Format

A `TBDpatt` file begins with a `TBDpatt_Format` header statement. Two attributes on this statement identify the formatting conventions used within the `TBDpatt` file:

- `mode` - the value may be either node or vector. This attribute is optional. TBDseqPatt files are expected to be in node format. The default value is vector, however, node format may legally be interspersed in the vector form.
  - `mode = node` specifies that PIs, POs, and flops for various events, such as Stim_PI and Scan_Load are named individually and are assigned values, as shown below:

    ```
    Event 1 Stim_PI(): "scan_enable" = 0;
    Event 2 Pulse(): "capture_clk" = +;
    ```

  - `mode = vector` specifies that a string of values is given without names, and each of these values is mapped to a PI, PO, or flop according to the vector correspondence for all events except Pulse, Stim_Clock, and the corresponding PPI clock events. For example:

    ```
    Event 1 Stim_PI(): ......1.....;
    Event 2 Pulse(): "capture_clk" = +;
    ```

- `model_entity_form` - the value may be either name, hier_index, or flat_index. This attribute is mandatory. Following is an example of the `TBDpatt_Format` statement.

  ```
  TBDpatt_Format (mode=vector, model_entity_form=name);
  ```

  - `model_entity_form = name` specifies to expect either short or full entity names to specify all relevant event actions and value assignments. For example:

    ```
    Event 1 Stim_PI(): "scan_enable" = 0;
    Event 2 Pulse(): "capture_clk" = +;
    ```

  - `model_entity_form = hier_index` specifies the use of hierarchical model indices (refer to Hierarchical and Flattened Model Characteristics in the *Encounter Test: Guide 1: Models* for more information) to specify entities involved in a given event or value assignment. For example:

    ```
    Event 1 Stim_PI(): 124 = 0;
    Event 2 Pulse(): 12 = +;
    ```

  - `model_entity_form = flat_index` specifies the use of flat model indices (refer to Hierarchical and Flattened Model Characteristics in the *Encounter Test: Guide 1: Models* for more information) to specify the entities involved in a given event or value assignment. For example:

    ```
    Event 1 Stim_PI(): 123 = 0;
    Event 2 Pulse(): 11 = +;
    ```

  Following is an example of the `TBDpatt_Format` statement.

  ```
  TBDpatt_Format (mode=node, model_entity_form=name);
  ```

  This header specifies that PIs, POs, and flops for various events, such as `Stim_PI` and `Scan_Load`, are named individually and are assigned values, as opposed to the vector

representation, in which a string of values is given, assigned to PIs, POs, or flops according to the vector correspondence.

Experiment

An experiment contains the test data generated from one or more test generation application runs. Each experiment that is committed to the master test data will have its own test data grouped separately from the test data of other experiments that may also have been committed. This may be useful in tracking down the source of certain test data. The experiment is the highest level division of a pattern's odometer number, which fully specifies the context of all test patterns and events. When TBDpatt is used as input to read_vectors, experiment statements cannot be used to separate the data, although they are a required element of the data structure. The read_vectors command renumbers and collapses all odometer numbers down to a simple, sequentially numbered form, removing any logical (lacking physical significance) hierarchical divisions.

Following is an example of an `Experiment` statement.

```
[ Experiment exper1 1;
        .
        .
        .
] Experiment exper1 1;
```

where:

`exper1` is the experiment name that was specified when the source test data (Vectors) was created and 1 is the first experiment.

An Experiment contains one or more test sections. There is no order dependency for Experiments, that is, they can be applied in any desired order. In addition to test sections, an Experiment may contain Keyed Data and Define_Sequence application objects.

The experiment object has these defined attributes:

- `TDM`

  This attribute is specified for an experiment that was created by Test Data Migration, and is intended only as a programming interface and is not supported as part of read_vectors; that is, an experiment with this attribute cannot be imported.

- `manipulated`

  When this attribute is set, it indicates the patterns contained within the experiment have already been manipulated and should not be further manipulated.

Test_Section

A `Test_Section` contains tests of a particular type, such as logic, shift register, and IDDQ.

A `Test_Section` is characterized by the following attributes. Note that each of these attributes applies to the entire test section and cannot change within a test section.

- `fast_forward`

  The presence of this attribute for a `logic_WRP` type test section indicates that the test section contains tests which use a pseudo-random pattern generator feature that allows the skipping of some tests. When used for a `logic_LBIST` type test section, this attribute is synonymous with the `fast_forward_sequences` attribute, and its presence indicates that the test data was produced by an early version of Encounter Test which did not support `fast_forward_pins`. When this attribute is present, the test data includes signatures for just the effective tests and an effective cycle mask that tells which tests to skip.

- `fast_forward_pins`

  This attribute is used for a `logic_LBIST` type test section to indicate that the tests use a pseudo-random pattern generator feature that allows the skipping of some tests. Two implementations of the fast_forward feature are supported. This attribute indicates that the skipping mechanism is controlled by pins labeled with the `PRPG_SAVE` (PV) attribute and the `PRPG_RESTORE` (PR) attribute. The other implementation would be indicated by the `fast_forward_sequences` attribute on the test section. When this attribute is present, the test data includes signatures for just the effective tests and an effective cycle mask that tells which tests to skip.

- `fast_forward_sequences`

  This attribute is used for a `logic_LBIST` type test section to indicate that the tests use a pseudo-random pattern generator feature that allows the skipping of some tests. Two implementations of the fast_forward feature are supported. This attribute indicates that the skipping mechanism is controlled by user-defined sequences (of type `prpgsave` and `prpgrestore`). The other implementation would be indicated by the `fast_forward_pins` attribute on the test section. When this attribute is present, the test data includes signatures for just the effective tests and an effective cycle mask that tells which tests to skip.

- `tester_termination = v`

  Indicates how the tester should terminate the product output pins for this test section.

  v may be 0 (tester must supply termination to logic 0), 1 (tester must supply termination to logic 1), and none (tester does not supply any termination). This attribute is optional. The default is none.

Differential pins that are inverted using the CORRELATE attribute receive the opposite termination value. Pins with a TTERM attribute always keep the TTERM value regardless of the termination value, even if tester termination is none.

- `termination_domination = keyword`

  Indicates whether tester-supplied termination shall be applied to pins which already have product termination, and if so, which will dominate.

  A keyword of tester indicates that tester-supplied termination is applied to all three-state output pins regardless of any product-supplied value for a pin. A *keyword* value of product indicates that tester-supplied termination is applied only to pins without product-supplied termination.

  This attribute is optional. If termination_domination is not specified, it defaults to the TDR value if tester_termination is specified as 0 or 1, or to tester if tester_termination is defaulted or specified to none. If dominance is specified when termination is defaulted or specified to none, dominance is ignored.

- `test_section_type = keyword`

  This attribute is required. Types of test sections are:

```
logic
logic_WRP
logic_LBIST
flush
scan
channel_scan
driver_receiver
macro
IDDq
IEEE_1149.1_integrity
ICT_stuck_driver
ICT_stuck_driver_diagnostic
ICT_shorted_nets_log(n+2)
ICT_shorted_nets_2*logn
ICT_shorted_nets_n+1
IOWRAP_stuck_driver
IOWRAP_shorted_nets_log(n+2)
IOWRAP_shorted_nets_2*logn
IOWRAP_shorted_nets_n+1
parametric
path
ecid
```

- `test_type = keyword`

  The keyword should be one of the following:

  - `opcbist`:

    This type is used for test data applied by an on-product controller. The designation of "`opcbist`" is used by test compilers to recognize that some aspects of the test are programmed into the product; for example, the number of test iterations is stored in a BIST controller register. With OPCG, the timing of the tests is under control of the product, so a Test_Section having a test_type of opcbist will normally contain patterns that have the static format. Check with the manufacturer or test expert if you are unsure whether to use this keyword.

  - `dynamic`:

    If the `test_type` is not `opcbist`, and the Test_Section contains any patterns in the dynamic format, the `Test_Section's test_type` is dynamic. Dynamic format patterns are patterns with a distinct launch and capture time frame, designed to create and capture transitions. Any patterns created with true-time test will be in dynamic format.

  - `static`:

    This is the `test_type` if the `Test_Section` is not `opcbist` and contains only static patterns.

    This attribute is optional; the default is static.

- `pin_timing`

  The presence of this attribute indicates the test section uses customized pin timings.

- `tester_PRPGs`

  The presence of this attribute indicates that the test section contains tests which use pseudo-random pattern generators connected to the design's primary inputs.

- `product_PRPGs`

  The presence of this attribute indicates that the test section contains tests which use pseudo-random pattern generators that are contained on the design itself, for example OPMISR block.

- `tester_signatures`

  The presence of this attribute indicates that the test section contains tests which specify latch and primary output responses in terms of compressed signatures collected by signature registers connected to the design's primary outputs.

- `product_signatures`

  The presence of this attribute indicates that the test section contains tests which specify latch responses in terms of compressed signatures collected by signature registers contained on the design itself, for example, OPMISR signature registers.

- `simulated`

  The presence of this attribute indicates that the test data of this test section was simulated. The absence of this attribute indicates that the test data of this test section has not been simulated, or was exported then re-imported after simulation, so it may have been subject to manual edits. Read Vectors resets this attribute and therefore, it cannot be set manually.

Following is an example of a Test_Section statement.

```
[ Test_Section 1.2 (tester_termination = 0, termination_domination = tester,
test_section_type = logic, test_type = dynamic);
    .
    .
    .
]Test_Section 1.2;
```

The 1.2 following `Test_Section` indicates that this is the second test section in the first experiment.

The `Test_Sections` within an experiment can be applied in any order. A `Test_Section` is a collection of `Tester_Loop`s. A `Test_Section` may also contain Keyed Data.

Tester_Loop

The term `Tester_Loop` derives from its original usefulness in allowing a diagnostic program to apply the `Tester_Loop` data repetitively for analysis. At the beginning of a `Tester_Loop`, the design is assumed to be in an unknown internal state (all X). Within Encounter Test, `Tester_Loop`s often contain many more tests than a diagnostic procedure would want for looping, and the `Tester_Loop` denotes only that the test contained therein can be used independently of tests in any other `Tester_Loop`s. Although it is guaranteed that `Tester_Loop`s can be applied independently of each other, often it is possible to apply even smaller sections of test data independently for diagnostic procedures. The `Tester_Loop` statement has an attribute (procedures_have_memory) which indicates whether or not the `Test_Procedures` that it contains may be applied independently of each other. Furthermore, the `Test_Procedure` statement has a similar attribute which indicates whether or not the `Test_Sequences` are independent of each other.

Upon entry to a `Tester_Loop` object in the test data, it may be important to ensure that the design has been placed into the appropriate test mode of operation. To this end, each `Tester_Loop` begins with a special init `Test_Procedure` whose purpose is to bring the design from the unknown state into the target test mode stability state. An `init Test_Procedure` is always the first Test_Procedure inside a `Tester_Loop` and contains a single init type of `Test_Sequence`. This init `Test_Sequence` may either have been provided by user input during test mode definition or it may have been generated automatically by Encounter Test.

A `Tester_Loop` has the following attributes:

- `procedures_have_memory`

  The presence of this attribute means that some test procedures within the `tester_loop` may use state information resulting from the preceding test procedure. The omission of this attribute means that each `Test_Procedure` assumes the design is in the state that was reached by the init `Test_Procedure` (the first Test_Procedure within the `Tester_Loop`), (i.e., the procedures are logically independent of each other).

- `ram_init`

  This attribute indicates ram initialization is required by simulation to obtain and apply array initialization values.

- `accumulated_signatures`

  The presence of this attribute indicates that signature events detected within the tester_loop are accumulated from the start of the tester_loop. This is similar in concept to the `procedures_have_memory` attribute in that the order of execution of the test procedures are executed is critical to achieving the correct expected value response from the design.

Following is an example of a `Tester_Loop` statement.

```
[ Tester_Loop 1.2.1 ();
     .
     .
     .
]Tester_Loop 1.2.1;
```

The 1.2.1 following the `Tester_Loop` indicates that this is the first `Tester_Loop` in test section 1.2.

`Tester_Loops` are numbered relatively within a `Test_Section` for identification purposes.

A `Tester_Loop` is a collection of `Test_Procedures`. A `Tester_Loop` may also contain keyed data, `Macro_Tester_Loop` application objects, and `TestSeqType` objects.

Test_Procedure

A `Test_Procedure` is a collection of one or more test sequences. Unless otherwise stated by `Test_Procedure` attributes, all the constituent test sequences have the same clocking structure (see the non-uniform_sequences attribute below). Test procedures that are automatically created by Encounter Test will contain a maximum of 32 test sequences. This number of sequences is optimal for Encounter Test's high speed scan-based simulator (refer to Test Simulation Concepts in *Encounter Test: Guide 6: Test Vectors* for more information), which is capable of simultaneously simulating 32 uniform test sequences in a single pass.

A Test_Procedure has the following attributes:

- `non-uniform_sequences`

  The presence of this attribute means that the sequences in this test procedure do not have the same form. Such differences can include the addition or removal of one or more events, or changes to the clocks that are pulsed or stimulated. Each non-uniform sequence will have to be simulated independently, therefore, simulation times will increase when this attribute is present.

- `sequences_have_memory`

  The presence of this attribute means that some `Test_Sequences` within the `Test_Procedure` use state information resulting from the preceding `Test_Sequence`. The omission of this attribute means that each `Test_Sequence` assumes the design is in the state that was reached by the init `Test_Procedure` (the first `Test_Procedure` within the `Tester_Loop`), (i.e., the sequences are logically independent of each other).

- `slow_to_turn_off`

  The presence of this attribute indicates the tests are specifically generated to detect clock slow-to-turn-off faults. These dynamic faults cause the trailing edge of a clock pulse to arrive at a memory element later in time than it is supposed to. These tests require special treatment in the Encounter Test simulator. This attribute is automatically added by Encounter Test and should not be added or removed manually unless you require this special behavior from the simulator.

- `type`

  Two fundamental types of `Test_Procedure` are supported:

- `normal`

  Most Test_Procedures are of this type, and have the function and content already described.

  Note: If the *type* attribute is not specified in an imported `TBDpatt` file, the default type is *normal.*

- `init`

  This type of test procedure contains only the initialization sequence for the test mode. Such a test procedure should be the first appearing in the tester loop, and is followed by normal test procedures.

  Note: While importing a set of vectors using read_vectors, if a mode initialization test procedure and sequence are present, it should be identical to the mode initialization which has been defined for the given test mode. If the modes do not match, Encounter Test ignores the initialization procedure and replaces it with the one which has been defined for the test mode.

The Test_Procedure has attributes that inform manufacturing of test coverage attained at given points in an experiment. The test coverage attributes apply only to the order listed in the `TBDpatt` file. If the test data is applied in any other order, these numbers are not accurate.

The following test coverage attributes are used:

- `static_faults`

  This attribute gives the number of static faults uniquely detected in a `Test_Procedure` in a logic `Test_Section`.

- `iddq_faults`

  This attribute gives the number of IDDq faults uniquely detected in a `Test_Procedure` in an IDDq `Test_Section`.

- `dynamic_faults`

  This attribute gives the number of dynamic (transition) faults uniquely detected with a `Test_Procedure`.

- `driverReceiver_faults`

  This attribute gives the number of driver/receiver faults uniquely detected with a `Test_Procedure`.

- `percent_static_faults`

This attribute gives the percentage of static faults (# of static faults detected in this mode / total # of static faults) detected at this point in the experiment.

- `percent_iddq_faults`

  This attribute gives the percentage of IDDq faults (# of IDDq faults detected in this mode / total # of IDDq faults) detected by IDDq tests at this point in the experiment.

- `percent_dynamic_faults`

  This attribute gives the percentage of dynamic faults (# of dynamic faults detected in this mode / total # of dynamic faults) detected at this point in the experiment.

- `percent_driverReceiver_faults`

  This attribute gives the percentage of driver & receiver faults (# of driver & receiver faults detected in this mode / total # of driver & receiver faults) detected at this point in the experiment.

Following is an example of a `Test_Procedure` statement.

```
[ Test_Procedure 1.2.1.3 (sequences_have_memory,slow_to_turn_off);
     .
     .
     .
]Test_Procedure 1.2.1.3;
```

The 1.2.1.3 following Test_Procedure indicates that this is the third test procedure in `Tester_Loop` 1.2.1.

A Test_Procedure is a collection of one or more test sequences. A Test_Procedure may contain keyed data and the following Application Objects.

- `Macro_Test_Procedure`
- `Ignore_Measures`

The following is an example of Tester_Loop and Test_Procedure to specify that the test mode initialization sequence (modeinit) loads the required values into internal FF's (`procedures_have_memory` and `sequences_have_memory`) and into RAMs (`ram_init`) within the design. If you are using a complicated initialization sequence as the base logic state for very specialized and focused pattern generation, warn Encounter Test that there are FF's and RAMS that have been loaded with required logic values.

```
TBDpatt_Format (mode=node, model_entity_form=name);
 [Experiment MBIST;
    [Test_Section (tester_termination=none, test_section_type=logic,
test_type=static) ;
  [Tester_Loop(procedures_have_memory,ram_init ) ;
  [Test_Procedure(sequences_have_memory);
  [Test_Sequence ( ) ;
  [Pattern(pattern_type=static);
```

```
Event Pulse ( ):
...
```

Test_Sequence

A Test_Sequence is a sequence of test patterns geared toward detecting a specific set of faults (defects). Although a single Test_Procedure can contain any number of test sequences, it is recommended to limit this number to 32 sequences and ensure that they are uniform. This is the optimum configuration for simulating with the high speed scan-based simulator of Encounter Test. A Test_Sequence has the following attributes:

- `type` = *keyword*

  The type attribute is optional; the default value is normal. The type attribute identifies the test sequence as one of the following:

  - `normal`

    This is the usual case, especially for stored-pattern tests.

  - `init`

    This test sequence type contains the initialization patterns for the test mode. This test sequence will appear as the only one within an init test procedure. For user-supplied vectors, this is expected to be identical to the initialization procedure that has already been created for the given test mode.

  - `setup`

    This `test_sequence` type contains only the initial events for the containing test procedure. Such a test sequence will appear only as the first test sequence within a test procedure. It is used for weighted random pattern testing to specify the signal weights and to initialize the product scan chains with their initial random values.

    In general, the setup sequence is used when there is OPC (on-product clock or control) logic that requires initialization before running a series of normal `Test_Sequences`. Except for its use in weighted random pattern and BIST testing, the setup sequence is useful only for initializing OPC latches.

  - `loop`

    This is a type of test sequence that would be considered as normal except that it is applied repetitively. The number of repetitions is specified via the `repeat` attribute. This sequence type is used in situations such as random-pattern testing, where the hardware creating the test patterns and analyzing the output results can be controlled by repetitive events within the test sequence to create a non-repetitive result.

- `miscompare`

  Identifies that one or more patterns in this test sequence contain an `Expect` or `Measure` event whose values did not agree with those predicted during simulation. Note that Measure events are used for comparisons when the simulation option Compare Measures is specified for the General Purpose Simulator. Read Vectors removes this attribute and, therefore, you cannot set it directly.

- `repeat = n`

  Supplies the number of times this test sequence is to be repeated. This attribute is valid only on loop type test sequences. If `type=loop` is specified, then `repeat= n` must also be specified, where `n` is a positive integer.

- `collapsible`

  This attribute directs the test generator to simplify the test sequence by collapsing the sequence into a single pattern or a small set of patterns. The collapsed version of the sequence is defined by patterns having the type collapsed appearing at the beginning of the sequence. The original patterns are kept, but not being of type collapsed, they can be ignored by subsequent applications wishing to process the collapsed form of the sequence.

  Collapsing “removes” patterns that contain only pseudo primary input events, combines successive patterns containing `Wait_Osc` events into a single `Wait_Osc` event, and replaces `Channel_Scan` events with the equivalent `Wait_Osc` event. The result of collapsing is a single `Wait_Osc` event for each running oscillator. As a result, no extra dead tester cycles will be set aside for these events.

  If an application or test compiler is processing the collapsed form of sequences, this attribute tells it to ignore non-collapsed patterns in this sequence. If the sequence is not marked `collapsible`, then it should not contain any collapsed patterns, and all patterns should be processed normally.

- `extra_release_capture`

  This attribute directs the test compiler to recognize and use the `Apply_Release_Capture` event, which will appear in a non-loop pattern near the end of the sequence. If this attribute does not appear on the `Test_Sequence`, then any `Apply_Release_Capture` event would be used only by diagnostics routines, and ignored when running the full test to collect the final signature.

Audit Attributes

These attributes are set by Encounter Test. While reading vectors, all audit information is reset as described below. Audit information is kept for any Force events contained in the

sequence and for any pseudo PI events contained in the sequence as well as additional audit information.

This audit information is expressed in `TBDpatt` format by the following attribute keywords:

- `Forces_unverified,`
- `Forces_verified, Forces_bad`

  One of these will be present if Force events are found within this sequence definition

  Note: `Forces_verified` indicates that the simulation has verified that the force events in this sequence are redundant because the forced nets were already at the values indicated in the Force event and remained there throughout the processing of the hold parameter, without requiring any special processing.

- `PPI_unverified,`
- `PPI_verified,`
- `PPI_bad`

  One of these will be present if Pseudo-PI events are found within this sequence definition or if this is not a modeinit sequence definition and there are any Pseudo-PIs defined for this test mode with stability values (TIs, TCs, or clocks). Note that the stability pins (and stability pseudo PIs) are always assumed to be at their stability states after the mode initialization sequence has been applied.

- `Failed_verification`

  This flag is set if Verify On Product Clock Sequences was run on this sequence definition and some check (unique to the sequence verifier) failed.

- `Invalid_oscillator`

  This flag is set if any Start_Osc, Stop_Osc, or Wait_Osc event is found on a pin that is neither a clock nor oTI.

Following is an example of a Test_Sequence.

```
[ Test_Sequence 1.2.1.1.1 (type=init);
     .
     .
     .
]Test_Sequence 1.2.1.1.1;
```

The 1.2.1.1.1 following Test_Sequence indicates that this is the first test sequence in test procedure 1.2.1.1.

A Test_Sequence contains an ordered collection set of Patterns. A Test_Sequence may also contain keyed data and the following Application Objects:

- `Macro_Test_Sequence`
- `Timing_ID`
- `SeqDef`

If this test sequence has been timed, the `SeqDef` and `Timing_ID` refer to the timing data for this test sequence's dynamic pattern. For a `Test_Sequence` to have an associated set of timings, it must contain a dynamic pattern. Only one dynamic pattern is allowed within a `Test_Sequence`.

Pattern

A pattern is an ordered set of events. A typical pattern is intended to contain all the events to be applied within one tester cycle. In the absence of timing data, multiple events within the same pattern must be applied to the design in the order found in the test data stream. In the case of a timed delay test (pattern_type=dynamic), if the containing sequence has a timing_ID, then the timings from the referred-to timing data will be used, which may change their relative order of application. Pattern has these attributes:

- `miscompare`

  Identifies whether or not this pattern contains an `Expect` or `Measure` event whose values did not agree with those predicted during simulation. Note that Measure events are used for comparisons when the simulation option Compare Measures is specified for the General Purpose Simulator.

- `pattern_type =` *`keyword`*

  Indicates the type of pattern, and is one of the following:

  - `static` - a typical pattern used in the application of static tests.
  - `dynamic` - a pattern that contains a separate launch and capture time frame used to excite and detect transition defects. Dynamic patterns use custom timing if a Timing_ID exists. Only one dynamic pattern is allowed within a `Test_Sequence`.
  - `begin_loop` - a pattern that denotes the beginning of a loop. The only event that should appear within this pattern is `Repeat`. This is one of two ways to specify loops within TBD. The other is by use of the loop type of test sequence.

    An example of a begin_loop:

    ```
    [Pattern 5 (pattern_type = begin_loop);
       Event 5.1 Repeat (): 3;
    ```

```
   ]Pattern 5;
[Pattern 6;
   Event 6.1 Pulse:
   "MASTERCLK(0)"=+;
   ]Pattern 6;
[ Pattern 7 (pattern_type = end_loop);
   ] Pattern 7;
```

- `end_loop` - A pattern that denotes the end of a loop. This pattern should not contain any events.
- `collapsed` - A Test_Sequence that is *collapsible,* will contain one or more `collapsed` patterns. A test data compiler can process either the collapsed form of the sequence or the uncollapsed form. In processing the collapsed form, only patterns that have the `collapsed` attribute are processed, up to the first `non-loop` pattern. In processing the uncollapsed form of the sequence, the collapsed patterns are skipped, and all other patterns are processed.

  The collapsing process currently supported by Encounter Test consists of counting the number of oscillator pulses applied through one iteration of the test sequence loop, including the scan operation, and representing this number in a `Wait_Osc` event in a single collapsed pattern.
- `non_loop` - A *loop* type Test_Sequence may contain patterns at the end that are not part of the loop. The pattern containing BIST or WRPT signatures is in this category. The first pattern within a loop test sequence that is actually outside the range of the loop must be type `non_loop`.

The `pattern_type` attribute is optional; the default is static.

Following is an example of a `Pattern` statement.

```
[ Pattern 1.2.1.3.1.5 (pattern_type = dynamic);
   .
   .
   .
]Pattern 1.2.1.3.1.5;
```

The 1.2.1.3.1.5 following Pattern indicates that this is the fifth pattern in test sequence 1.2.1.3.1.

Event

An event is a container for stimulus and response test data and any other data for which ordering is important.

Events are contained in pattern blocks, and appear in the same order as they occurred in the simulation.

An event may have attributes, depending on its type.

- `timed_type`

  Several events share the `timed_type` attribute. These are tags that are added to events to describe the intentions of a particular event with respect to launching, propagating, or capturing a transition fault. They are not required for delay test sequence events but can be useful for informational purposes. For non-transition test patterns, `timed_type` should be unspecified or `none`, which is the default value. This attribute has the following values:

  - `none`

    The `timed_type` attribute is unspecified or this particular event does assume the time at which it occurs. This may occur on timed delay test dynamic patterns, but does not necessarily indicate that the event is independent of timing. It only indicates that this extra information was not provided.

  - `release`

    This event triggers a transition for a delay test.

  - `propagate`

    This event occurs during a timed test between the release and capture events.

  - `capture`

    This event ends the timed portion of the test by capturing the result. This may be a clock pulse that strobes the data input to some latch or flip-flop, or a primary output measure (where the strobing occurs in the tester).

The `Signature` events share the following attributes:

- `iteration`

  Specifies a number (as `iteration=`*`n`*) which tells how many repetitions of the containing test sequence are needed to obtain the signature. This allows premature termination of the test sequence for the purpose of obtaining intermediate signatures for diagnosing failures. When *`n`* equals the `Repeat` count on the test sequence, then this is the signature at the end of the test sequence when no test sequence cycles were skipped.

- `fast_forward`

  Specifies that the signature is the signature at the end of the current test sequence when the ineffective test sequence cycles were skipped. The effective test sequence cycles are specified by the `Effective_Cycle_Mask` event. Intermediate signatures are not

supplied for fast forward, and therefore this attribute is not accepted in combination with the `iteration=n` attribute.

- `final`

  Specifies that this is the end of the test; no more test sequences or test procedures follow within the containing tester loop. If this signature is ignored, the test results will be lost.

Either `iteration` or `fast_forward` must be specified; `final` is specified only where required.

The following table lists the events that Encounter Test provides:

Event Type	Applied To	Event Name
Physical simulation events	All architectures	Pulse Stim_Clock Stim_PI Stim_PI_Plus_Random
	WRP/LBIST	PI_Weight Pulse_Tester_PRPG_Clocks Pulse_Tester_SISR_Clocks
Non-physical (simulation-only) events		Force Internal_Response Pulse_PPI Release Stim_PPI Stim_PPI_Clock
Measure operations		Measure_Current Measure_PO

Placeholder Events (for custom test vector definitions)		Dummy_Skewed_Scan_Unload Dummy_Scan_Load Dummy_Scan_Unload Dummy_Skewed_Scan_Load Put_Stim_PI
Custom Scan Protocol Definition (may only be used within Define_Sequence constructs)	All	Apply Set_Scan_Data
	Full scan	Measure_Scan_Data
	OPMISR/OPMISR+	Set_CME_Data Set_CMI_Data Measure_MISR_Data
	XOR compression	Set_CME_Data Set_CMI_Data Measure_Scan_Data
	On-Product Clocking Control	Load_OPCG_Controls Pulse_PPI Set_OLI_Data Start_Osc Stop_Osc Wait_Osc
Diagnostic aids (LSSD)	OPMISR/OPMISR+	Diagnostic_Skewed_Scan_Unload Diagnostic_Scan_Unload
	GSD	Diagnostic_Scan_Unload

	XOR compression	Diagnostic_Skewed_Scan_Unload Diagnostic_Scan_Unload
Diagnostic aids (WRP/LBIST)		Latch_Values
Masking operations	WRP/LBIST	Effective_Cycle_Mask Tester_SISR_Mask
	OPMISR/OPMISR+	Effective_Cycle_Mask Fix_MISR Load_Channel_Masks Use_Channel_Masks
	XOR compression	Effective_Cycle_Mask Load_Channel_Masks Use_Channel_Masks
Test Data Migration / Core test operations		Internal_Response Internal_Scan_Load
Scannable Flop Shift/Load/Unload operations (WRP/LBIST)		Channel_Scan Connect_Tester_PRPG Latch_Weight Product_PRPG_Signature Tester_PRPG_Seed Tester_PRPG_Signature Tester_SISR_Seed

Scannable Flop Shift/Load/Unload operations (WRP/ LBIST)	Full/partial scan	Skewed_Scan_Unload Skewed_Unload_SR Compact_Scan_Load Compact_Skewed_Scan_Load Dummy_Skewed_Scan_Unload Dummy_Scan_Unload Dummy_Scan_Load Dummy_Skewed_Scan_Load Load_SR Skewed_Load_SR Scan_Load Scan_Unload Skewed_Scan_Load Unload_SR
	OPMISR/OPMISR+	Channel_Scan Compact_Scan_Load Compact_Skewed_Scan_Load Diagnostic_Skewed_Scan_Unload Diagnostic_Scan_Unload Dummy_Scan_Load Dummy_Skewed_Scan_Load Product_MISR_Signature Scan_Load

	XOR compression	Skewed_Compressed_Output_Stream Compact_Scan_Load Compressed_Input_Stream Skewed_Compressed_Input_Stream Compressed_Output_Stream Dummy_Skewed_Scan_Unload Dummy_Scan_Unload Dummy_Scan_Load Dummy_Skewed_Scan_Load Diagnostic_Scan_Unload Load_SR Skewed_Load_SR Scan_Load Skewed_Scan_Load
Scannable Flop Shift/Load/Unload operations (GSD)	Full/partial scan	Compact_Scan_Load Dummy_Scan_Load Dummy_Scan_Unload Load_SR Scan_Load Scan_Unload Unload_SR

OPMISR/OPMISR+	Channel_Scan
	Compact_Scan_Load
	Diagnostic_Scan_Unload
	Dummy_Scan_Load
	Product_MISR_Signature
	Scan_Load
XOR compression	Compact_Scan_Load
	Compressed_Input_Stream
	Compressed_Output_Stream
	Diagnostic_Scan_Unload
	Dummy_Scan_Load
	Dummy_Scan_Unload
	Load_SR
	Scan_Load

A description of each event is covered in later sections.

Apply

This event invokes a `Define_Sequence` by specifying the name of the sequence. It is similar in concept to a subroutine in a structured programming language. This event has no attributes.

Example:

```
Event 6.1.1 Apply (): Scan_Preconditioning_Sequence;
```

Apply events are legal for all `Define_Sequence` types and the following additional contexts:

- scanop sequences
- scansection sequences
- Channel_Mask_Load_Sequence
- MISR_Mask_Load_Sequence

Note: While Apply events are allowed in all `Define_Sequence` types, any pattern that contains one or more `Apply` events cannot contain other event types.

The Define_Sequence types that are being applied must either have already been imported or existing previously in the TBDseqPatt being imported.

All `Apply` events that are not part of the scan operation are removed and the patterns of the sequence being applied are inserted. The Define_Sequence name and the date/time of its last update is saved on the first inserted pattern. All other inserted patterns store the `Define_Sequence` type of the expanded `Apply`.

The following table shows the use of the `Apply` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST		X	
	OPMISR/OPMISR+		X	
	XOR Compression		X	
	Full/Partial Scan		X	
GSD	WRP/LBIST		X	
	OPMISR/OPMISR+		X	
	XOR Compression		X	
	Full/Partial Scan		X	

Apply_Release_Capture

This event causes the release-capture portion of the test sequence to be applied. The release-capture portion of the test sequence is defined as all patterns and events from the beginning of the sequence up to but not including the Channel_Scan event (or the first non_loop pattern, whichever comes first). The release-capture sequence is automatically provided to the processing application through Encounter Test API. The release-capture sequence is provided in either uncollapsed form or (if the sequence has the `collapsible` attribute) the collapsed form.

If the containing Test_Sequence does not have the `extra_release_capture` attribute, the `Apply_Release_Capture` event may be skipped at the discretion of the processing application.

Example:

```
Event 1.1.1.2.2.6.1 Apply_Release_Capture ();
```

Note: Release-capture is not a typical user operation and its use requires expert knowledge of Encounter Test and its API interface.

The following table shows the use of the `Apply_Release_Capture` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			X
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			X
GSD	WRP/LBIST			X
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			X

Begin_Test_Mode

This event denotes entry into the specified test mode and implies the establishment of the stability state for the specified test mode. When exporting the data out of the Encounter Test environment, this event should be interpreted as an invocation of the modeinit sequence for the specified test mode. Subsequent to this event, until the end of the current test sequence all scan operations (`Scan_Load`, `Scan_Unload`) apply to the specified test mode, and if using the vector format, the vector correspondence lists for this test mode are used to interpret the data. This event has no attributes.

Example:

```
Event 1.1.1.1.1.25.1 Begin_Test_Mode (): lssd_static;
```

Restriction: This event is allowed only within a modeinit sequence definition or an `init` type of test sequence.

Also note that IBM's TDS/6000 system differs from Encounter Test processing in that it does not simulate the Parent Mode Initialization sequence. This should not be a concern if you

have a normal mode initialization, however for complex resets of the chip, this should be added to the child's Mode Init immediately following the `Begin_Test_Mode` event to ensure TDS/6000 software places the patterns in the file.

The following table shows the use of the `Begin_Test_Mode` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X		type=init
	OPMISR/OPMISR+	X		type=init
	XOR Compression	X		type=init
	Full/Partial Scan	X		type=init
GSD	WRP/LBIST	X		type=init
	OPMISR/OPMISR+	X		type=init
	XOR Compression	X		type=init
	Full/Partial Scan	X		type=init

Channel_Scan

This event is used in connection with Weighted Random Pattern (WRP), LBIST, and OPMISR-based testing to specify when the scan operation is to take place. Refer to Appendix A, “Scan Operation,” for a diagram.

This event may be qualified by the following attribute keywords; any combination (or none) of these may be present:

- `block_signature_register`

  No tester SISRs are to be clocked during the scan; normally (in the absence of this attribute), the tester SISR connected to each scan chain output is clocked at each cycle of the scan operation. If there are any on-board MISRs (indicated by the presence of a product_signatures attribute on the test section), then there must be one or more MISR Enable (ME) primary inputs that are used to block the functioning of these MISRs. This blocking is accomplished by inverting the value on the ME inputs when the scan preconditioning sequence is applied. The `block_signature_register` attribute is specified when the channel scan is used to initialize the scan chains. This prevents unknown (unpredictable) signals from being shifted into the SISRs or MISRs.

- `fast_forward`

  This is a signal that the channel scan is run in the *fast forward* mode; that is, it includes operations that save the PRPG states into PRPG save registers, restore the PRPG states from PRPG save registers, and for tester PRPGs, a PRPG clock pulse. If the test section has either the `fast_forward` attribute or `fast_forward_sequences` attribute, the restore operation occurs first (preceding the actual channel scan). If the test section has the `fast_forward_pins` attribute, then the restore operation occurs on the last cycle of the channel scan. The tester PRPG manipulations (restoring, incrementing, saving) always occur first. The PRPG save operation for on-product PRPGs (LBIST) is positioned within the channel scan operation such that the state of the PRPG after n+1 scan cycles is saved (where *n* is the number of tests to be skipped). None of these operations appear explicitly in `TBDpatt` or Vectors, but are specified implicitly by this attribute.

- `fast_forward_save`

  This is a signal that the channel scan is run in the *fast forward* mode (see the `fast_forward channel_scan` attribute), but excluding the restore operation at the beginning of the channel scan. If the test section has the `fast_forward_pins` attribute, then the restore operation occurs in its place on the last channel scan cycle for both `fast_forward` and `fast_forward_save` channel scans.

- `skewed_load`

  Causes an extra A shift clock pulse to be applied at the end of the scan operation, resulting in the two latches of each SRL being set to independent values.

- `skewed_unload`

  Causes an extra B shift clock pulse to be applied at the beginning of the scan operation. The effect is to unload the B_SHIFT_CLOCK measure latches (L1s in an LSSD design). When this attribute is absent, the representative measure latches (RMLs) are observed (these are L2s in an LSSD design).

  Example:

  ```
  Event 1.1.1.1.1.28.1 Channel_Scan (block_signature_register, skewed_load,
  skewed_unload);
  ```

- `padding_cycles=`*n*

  Specifies whether the channel_scan must have *n* cycles of padding with all zeros applied to the scan-in pins before applying the scan-in data for the next test. By default, `padding_cycles=0` so there are no padding cycles unless required.

- `overlap=yes|no`

Specifies whether overlapping with the next scan-in is allowed. Refer to "Overlapped/ Non-Overlapped Scans" on page 169 for additional information.

The following table shows the use of the `Channel_Scan` event

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			X
	OPMISR/OPMISR+			X
	XOR Compression			
	Full/Partial Scan			
GSD	WRP/LBIST			X
	OPMISR/OPMISR+			X
	XOR Compression			
	Full/Partial Scan			

Compact_Scan_Load

This event is a compact form of the `Scan_Load` event which is used to scan in latch values. This is the form that is generated by Automatic Test Pattern Generation since it can significantly reduce the size of the Vectors. Use of the `Compact_Scan_Load` for manual patterns may also reduce the size of the Vectors created during Read Vectors as well as the Vectors resulting from simulation. The values specified are the same as for the `Scan_Load` event.

For information on the Scan_Load event, refer to Scan_Load on page 83.

The default_value attribute for Compact_Scan_Load has 2 more values than it does for Scan_Load: `default_value=0|1|X|scan_0|scan_1|random|repeat`. If `default_value=random`, you also need to specify the `seed=`*`value`* attribute; where the value is the seed for the randomly generated fill values.

If you are writing manual patterns, it is recommended that you:

1. Use `Compact_Scan_Load` rather than `Scan_Load` whenever any of the latch values are not explicitly specified. In the rare instance that every latch value is explicitly specified `Scan_Load` is actually more compact than `Compact_Scan_Load`.

2. *Do not* specify the `default_value` or seed. The vector will be filled with the correct values based on your specification of `latchfill` and `latchfillstatic`/ `latchfilldynamic` during Test Simulation (TGResim).

3. The latch values may be specified using:

   - expanded vector format - latch values are specified as a string of values, one for every RSL in the order specified in the vector correspondence. Unspecified values are represented with a “.”.

     For example:

     ```
     Event 1.1.1.2.6.1.1 Compact_Scan_Load():
     0.....;
     ```

   - node list format - latch values are specified with `rslname=`*`value`*; where `rslname` is the name of the RSL latch primitive block, or a pin/net that can be traced back to the RSL's latch primitive block without any ambiguity. Unspecified latches are not included in the list. For example:

     ```
     Event 1.1.1.2.6.1.1 Compact_Scan_Load():
     latch1_block_name=0;
     ```

   - unexpanded vector format - latch values are specified as string(s) of values. Each string represents one word, 32 latch values, in flatmodel index order. The words are numbered starting with 0. This format is how the `Compact_Scan_Load` events are actually stored in the Vectors and is the most compact form. It is difficult to decipher and is intended for use by Encounter Test development. However, it may be useful if you are editing the patterns from a large Vectors file and you don't need to change any latch values. For example:

     ```
     Event 1.1.1.2.6.1.1 Compact_Scan_Load():
     0=..0...;
     ```

Notes

- The same latch is set to 0 in all three examples.
- The `TBDpatt_format mode=`*`value`* setting doesn't matter.

The following table shows the use of the `Compact_Scan_Load` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			

	OPMISR/OPMISR+	X
	XOR Compression	X
	Full/Partial Scan	X
GSD	WRP/LBIST	
	OPMISR/OPMISR+	X
	XOR Compression	X
	Full/Partial Scan	X

Compact_Skewed_Scan_Load

This event is analogous to the `Compact_Scan_Load` event with the exception that it is the compact version of the `Skewed_Scan_Load` event. See Compact_Scan_Load on page 50 and Skewed_Scan_Load on page 96.

The following table shows the use of the `Compact_Skewed_Scan_Load` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			X
GSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Compressed_Input_Stream

This event lists values for all scan-in (`SI`) pins for every scan cycle. The values specified are the same as for the `Scan_Load` event, but should be all 0 or 1 (no X values) by the time this event is being sent to a tester; however, unlike the `Scan_Load` event, there is no need to pre-pad short scan chains with zeros since any padding is already built into the supplied values.

For information on the `Scan_Load` event, refer to Scan_Load on page 83.

The following table shows the use of the `Compressed_Input_Stream` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			X
	Full/Partial Scan			
GSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			X
	Full/Partial Scan			

Compressed_Output_Stream

This event lists values for each Scan_Out (`SO`) or Misr_Observe (`MO`) pin for every scan cycle. Values include 0/1/X. The attribute `overlap=no` indicates overlapping with the next scan-in is disallowed and the values provided in the event were computed with the assumption that the scan-in (SI) pins will bet set to zeros (0) for all scan cycles. The `imported` attribute indicates the data was imported from a foreign pattern source (IEEE 1450 STIL).

Refer to "Overlapped/Non-Overlapped Scans" on page 169 for additional information.

The following table shows the use of the `Compressed_Output_Stream` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			X
	Full/Partial Scan			

GSD	WRP/LBIST		
	OPMISR/OPMISR+		
	XOR Compression		X
	Full/Partial Scan		

Connect_Tester_PRPG

This event is used in weighted random pattern (WRP) testing to specify that the indicated primary inputs are to receive pseudo-random values. The associated tester PRPGs are assumed to have been initialized by some previous `Tester_PRPG_Seed` event. WRP testing assumes that all non-clock primary inputs that are not being held to a constant value are to receive pseudo-random values. Therefore, the `Connect_Tester_PRPG` event is used only when the PRPG(s) has been temporarily disconnected to apply some deterministic value to the primary input(s) via the `Stim_PI` event. This event has the `timed_type` attribute.

Example:

```
    Event 1.1.1.1.1.36.1 Connect_Tester_PRPG ():
"Pin.f.l.lsrfh0b.nl.a1"
"Pin.f.l.lsrfh0b.nl.a2";
```

The following table shows the use of the `Connect_Tester_PRPG` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X		
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			
GSD	WRP/LBIST	X		
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Diagnostic_Skewed_Scan_Unload

This event, along with a corresponding `Diagnostic_Scan_Unload`, is used to store the values for all B_SHIFT_CLOCK measure latches (BMLs) and B_SHIFT_CLOCK cell latches (BCLs) after the initial clock pulse. The format of this event is similar to the `Skewed_Scan_Unload` event, however a `Diagnostics_Observe` sequence must be applied before expect data can be scanned out to be observed on the tester.

Refer to the following for related information:

- "Skewed_Scan_Unload" on page 92
- "Diagnostic_Scan_Unload" on page 55
- "Define_Sequence" on page 111

The following table shows the use of the `Diagnostic_Skewed_Scan_Unload` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			
GSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Diagnostic_Scan_Unload

This event, along with a corresponding Diagnostic_Skewed_Scan_Unload, is used to store the values for all representative measurable latches (RMLs) and representative cell latches (RCLs) after the initial clock pulse. The format of this event is similar to the Scan_Unload event, however a Diagnostics_Observe sequence must be applied before expect data can be scanned out to be observed on the tester.

Example:

```
Event 1.1.1.2.1.4.1 Diagnostics_Scan_Unload ():
```

Refer to the following for related information:

- "Diagnostic_Skewed_Scan_Unload" on page 55
- "Scan_Unload" on page 71
- "Define_Sequence" on page 111

The following table shows the use of the `Diagnostics_Scan_Unload` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			
GSD	WRP/LBIST			
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			

Dummy_Skewed_Scan_Unload

This event is used as a place holder by Encounter Test. When importing sequence definitions of type test during Test Mode creation you must use `Dummy_Skewed_Scan_Unload` events instead of real `Skewed_Scan_Unload` events because Encounter Test has not yet defined the scan chains at the time the sequence definitions are being read in during Test Mode definition processing. This event can also be used within manual patterns as a space saving device. Actual measured values can not be determined until the input patterns are simulated. During simulation Encounter Test will convert all `Dummy_Skewed_Scan_Unload` events to real `Skewed_Scan_Unload` events.

Example:

```
Event 1.1.1.1.1.3.1 Dummy_Skewed_Scan_Unload ();
```

Refer to "Skewed_Scan_Unload" on page 92 for more information.

The following table shows the use of the `Dummy_Skewed_Scan_Unload` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression	X		X
	Full/Partial Scan	X		X
GSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Dummy_Scan_Unload

This event is used as a place holder by Encounter Test. When importing sequence definitions of type test during Test Mode creation you must use `Dummy_Scan_Unload` events instead of real `Scan_Unload` events because Encounter Test has not yet defined the scan chains at the time the sequence definitions are being read in during Test Mode definition processing. This event can also be used within manual patterns as a space saving device. Actual measured values can not be determined until the input patterns are simulated. During simulation Encounter Test will convert all `Dummy_Scan_Unload` events to real `Scan_Unload` events.

Example:

```
Event 1.1.1.1.1.3.1 Dummy_Scan_Unload ();
```

Refer to "Scan_Unload" on page 71 for more information.

```
Event 1.1.1.1.1.3.1 Dummy_Scan_Unload ();
```

The following table shows the use of the `Dummy_Scan_Unload` event:

	Mode initialization	Custom scan protocol	Test pattern sequences

LSSD	WRP/LBIST		
	OPMISR/OPMISR+		
	XOR Compression	X	X
	Full/Partial Scan	X	X
GSD	WRP/LBIST		
	OPMISR/OPMISR+		
	XOR Compression	X	X
	Full/Partial Scan	X	X

Dummy_Scan_Load

This event is used as a place holder by Encounter Test. When importing sequence definitions of type test during Test Mode creation you must use `Dummy_Scan_Load` events instead of real `Scan_Load` events because Encounter Test has not yet defined the scan chains at the time the sequence definitions are being read in during Test Mode definition processing. When the sequence definitions are actually used by Encounter Test, the `Dummy_Scan_Load` events will be converted to real `Scan_Load` events.

This event has no attributes.

Example:

```
Event 1.1.1.1.1.3.1 Dummy_Scan_Load ();
```

Refer to "Scan_Load" on page 83 for more information.

```
Event 1.1.1.1.1.3.1 Dummy_Scan_Unload ();
```

The following table shows the use of the `Dummy_Scan_Load` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+	X		X
	XOR Compression	X		X
	Full/Partial Scan	X		X

GSD	WRP/LBIST		
	OPMISR/OPMISR+	X	X
	XOR Compression	X	X
	Full/Partial Scan	X	X

Dummy_Skewed_Scan_Load

This event is used as a place holder by Encounter Test. When importing sequence definitions of type test during Test Mode creation you must use `Dummy_Skewed_Scan_Load` events instead of real `Skewed_Scan_Load` events because Encounter Test has not yet defined the scan chains at the time the sequence definitions are being read in during TestMode definition processing. When the sequence definitions are used by Encounter Test the `Dummy_Skewed_Scan_Load` events will be converted to real `Skewed_Scan_Load` events.

This event has no attributes.

Example:

```
Event 1.1.1.1.1.3.1 Dummy_Skewed_Scan_Load ();
```

Refer to "Skewed_Scan_Load" on page 96 for more information.

```
Event 1.1.1.1.1.3.1 Dummy_Scan_Unload ();
```

The following table shows the use of the `Dummy_Skewed_Scan_Load` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+	X		X
	XOR Compression	X		X
	Full/Partial Scan	X		X
GSD	WRP/LBIST			
	OPMISR/OPMISR+	X		X
	XOR Compression	X		X
	Full/Partial Scan	X		X

Effective_Cycle_Mask

This event contains a bit string that tells which iterations of the containing test sequence should be applied in fast forward mode. If the i-th bit in this string is 1, then the i-th iteration of the test sequence should be applied; if the i-th bit is 0, then the channel scan event and all Pulse Tester SISR Clocks events are skipped during the i-th iteration of the test sequence. The mask is written as a hexadecimal string. This event has no attributes.

Example:

```
Event 1.1.1.1.1.34.2 Effective_Cycle_Mask ():
FEF3BFF ECA3F7E3 6241A0B4 980500C0 24200090 00040020 3010C600 00044002
0200810 00601000 20001000 00104000 00200A40 00102100 00004010 00020052
08020000 00406000 00000000 00000100 ;
```

The following table shows the use of the `Effective_Cycle_Mask` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			X
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			
GSD	WRP/LBIST			X
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			

Expect

This event specifies expected values for a set of nodes. The nodes may be internal nets or primary inputs or primary outputs. This event has the `timed_type` attribute. The expect event may contain data for resimulation or diagnostics. The diagnostics describe possible defects in terms of a failure node and a specific pin or net.

Example:

```
Event 1.1.1.1.1.7.1 Expect ():
```

```
"Pin.f.l.lsrfh0b.nl.a5"=1
"Pin.f.l.lsrfh0b.nl.a3"=1
"Pin.f.l.lsrfh0b.nl.a9"=1;
    Detected_fault:
SDT on Driver Pin "00"
SNT on Net "srf10hm.s30"
SNT on Net "srf10hn.s30";
```

The following table shows the use of the `Expect` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X		X
	OPMISR/OPMISR+	X		X
	XOR Compression	X		X
	Full/Partial Scan	X		X
GSD	WRP/LBIST	X		X
	OPMISR/OPMISR+	X		X
	XOR Compression	X		X
	Full/Partial Scan	X		X

Fix_MISR

This event is used to modify a prior test's `Product_MISR_Signature` event. It specifies a value to be XORed with a prior test's MISR values to take into account how the overlapping of the scan operation for this test will influence the prior test's MISR signature. This is intended to allow `Test_Sequences` to be easily reordered. The format of this object is similar to the `Product_MISR_Signature` event in that it contains values for each (observable) MISR.

Example:

```
Event 1.1.1.3.1.2.1 Fix_MISR:
030A0200;
```

The following table shows the use of the `Fix_MISR` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+	X		X
	XOR Compression			
	Full/Partial Scan			
GSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Force

This event is a declaration about the state of an internal node (often this will be a latch). It allows a user to force a value on the internal net directly. There are many instances where this may be useful. Some examples are:

- Logic has been pruned from the design model so the simulator does not know the complete behavior.
- A long or complicated initializing sequence is required to produce the state, but simulation of this sequence has been bypassed to save execution time.
- The design is not initializable. A frequency divider is one example of a case where it is valid to make an assumption about the initial state even though the initial state is unpredictable.
- The power-on state is predictable.

The Force event specifies a list of nodes and a value for each.

The Force event should be used with caution, and its proper use requires familiarity not only with the design and its function, but also with the way latches are modeled for Encounter Test. In particular, edge-triggered flip-flops are modeled with a pair of latches in tandem, and one of these latches always has its clock “on.” If the Force is applied to the latch whose clock is “on” at that point, then the value may be lost before the clock is again pulsed.

This event has one attribute, `hold`. If `hold` is specified, the value will persist on the specified node continually until a `Release` event is specified for this node, or until the design is reset, as it might be at a test sequence. If `hold` is not specified, then the forced value persists in simulation until the design is stabilized. Then the simulator immediately processes a "virtual" release event for the node. See "Release" on page 82 to find out what the simulator does when processing a release event.

Example:

```
    Event 1.1.1.1.1.4.1 Force ():
"Block.f.l.lsrfh0b.nl.opcg1.srfh10m.021"=0
"Block.f.l.lsrfh0b.nl.opcg1.srfh10n.021"=0;
```

The following table shows the use of the `Force` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X
GSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X

Internal_Response

This event specifies the waveform for a single net within the containing pattern. This event is typically an output of simulation and is used as input to the waveform display tool. The event identifies a net and a series of time = value pairs that describe the behavior of the net during the pattern. A time zero value is always provided and is followed by a time = value pair for each change on the net. The time units are in picoseconds. This event has no attributes.

Example:

```
    Event 1.1.1.1.1.36.2 Internal_Response ():
"Pin.f.l.lsrfh0b.nl.srf10hd.aa000aa10";
Time 0 = 0
```

```
Time 1 = x
Time 2 = 1;
```

The internal response values are described in the following diagram. The five horizontal lines represent single logic values. H is a weak 1, L is a weak 0. The ten vertical lines are different types of unknown values: An A is 1 or H; an N is Z, L or 0; etc.

```
1----+---------------+---------------+-----------+-
     A               |               |           |
H----+---+-----------M---+-----------+----+------+-
         B           |   |           P    |      |
Z--------+--+--------+---W---+------+----+------X-
            C            |   |      |    Q      |
L-----------+--+---------+---N------+----+------+-
                D            |           |      |
0-------------+-------------+-----------+------+--
```

The following table shows the use of the `Internal_Response` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			X
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			X
GSD	WRP/LBIST			X
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			X

Internal_Scan_Load

This event lists the values that will be loaded into all flop scan cells as a result of preceding events. This may be used with unknown compression mechanism to denote what values end up in the scan cells as a result of applying all of the preceding events. This event is used by

simulators and will not be included in any tests converted to another pattern format (WGL, STIL).
i

Notes

1. If using the node/name format of `TBDpatt`, scan cells may be identified by either referencing the primitive block or output pin name directly or by referencing the containing cell output pin name if there is no fan-in along the path from the primitive pin to the cell pin.
2. If you are not explicitly specifying values for all scan cells, the compact version of this event should be used to achieve a reduction in the size of the Vectors file that results from Read Vectors. Refer to "Compact_Scan_Load" on page 50 for additional information.

This event has the following attributes:

- `default_value=0 |1 |X |scan_0 |scan_1`

  This specifies what value should be scanned into a scan cell whose value is not uniquely specified. A default value of X means that cell will be set arbitrarily by the tester or TDS software. A default value of 0 (or 1) means that the unspecified cells will be set to 0 (or 1). The value required on the scan data primary input will depend upon the number of inversions between it and each defaulted flop. A default value of scan_0 (or scan_1) means that 0 (or 1) will be placed on the scan data primary input when the defaulted cells are to be loaded. The value resulting in each defaulted cell will depend upon the number of inversions between it and the scan data primary input. If this attribute is not used, the default value is X.

- `manipulatecopy`

  Indicates the event is a copy of an original that was manipulated for control pipeline. Refer to "Pipelined Control Signals" in the *Encounter Test: Reference: Legacy Functions*.

Refer to Appendix A, "Scan Operation" for a diagram.

Example:

```
Event 1.1.1.1.1.1.1 Internal_Scan_Load (default_value=0):
"internal_scan_flop.Q" = 1;
```

The following table shows the use of the `Internal_Scan_Load` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X		X
	OPMISR/OPMISR+	X		X
	XOR Compression	X		X
	Full/Partial Scan	X		X
GSD	WRP/LBIST	X		X
	OPMISR/OPMISR+	X		X
	XOR Compression	X		X
	Full/Partial Scan	X		X

Latch_Values

This event is used to record the values of all representative measure latches when signature analysis is being used. The latch values are useful for debug and diagnosis. This event type has the attributes `iteration`, `skewed_unload`, `fast_forward`, and `final`.

Example:

```
    Event 1.1.1.1.1.36.2 Latch_Values (iteration=256):
FEFB3BFF ECA3F7E3 6241A0B4 980500C0;
```

This event is always printed in a hex vector format. This is similar to the vector format of a Scan_Load event, except that the bit vector contains only 1's and 0's and is converted to hexadecimal.

The following table shows the use of the `Latch_Values` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			X
	OPMISR/OPMISR+			

	XOR Compression	
	Full/Partial Scan	
GSD	WRP/LBIST	X
	OPMISR/OPMISR+	
	XOR Compression	
	Full/Partial Scan	

Latch_Weight

This event is used in weighted random pattern testing to specify the bias to be applied in the selection of random values to be scanned into the latches. In concept, a weight is the probability that the value will be a 1. Each weight is specified in terms of equivalent AND/OR inputs --the number of unbiased random signals that are ANDed together or ORed together to produce the weighted random value. Each set of correlated latches can have a different weight. This implies the use of a high-speed buffer to dynamically alter the weights during scan operations at the tester. This event has no attributes.

```
Vector format example:
    Event 1.1.1.1.1.16.1 Latch_Weight ():
O4O5-.;

Node format example:
    Event 1.1.1.1.1.16.1 Latch_Weight ():
"Block.f.l.lsrfh0b.nl.srf10hm.021"=O4
"Block.f.l.lsrfh0b.nl.srf10hn.021"=O5;
```

The following table shows the use of the `Latch_Weight` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X		X
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			
GSD	WRP/LBIST			
	OPMISR/OPMISR+			

	XOR Compression
	Full/Partial Scan

Load_Channel_Masks

This event specifies the complete list of channel mask bit registers and the values to which they should be loaded. When present in a Test_Sequence, this event appears at the beginning of the pattern that contains the `Channel_Scan` event. This event also includes a `scan_cycle` number on which the masks are to be loaded.

Example:

```
Event 2.1.1.2.1.4.2 Load_Channel_Masks (cycle=0):
Mask Reg 1 = 1000 # masks just chain 1
Mask Reg 2 = 0100; # masks just chain 2
```

The following table shows the use of the `Load_Channel_Masks` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+	X		X
	XOR Compression	X		X
	Full/Partial Scan			
GSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Load_OPCG_Controls

The `Load_OPCG_Controls` event is specified either in mode initialization or setup sequences. This event, when specified in the mode initialization sequence, specifies the values to load into the PLL programming registers.

When used in the mode initialization sequence, place this event between the `Begin_Test_Mode` event and the `Scan_Load` event. These values are then converted into values for the `Scan_Load` event to be loaded by the parent testmode scan load operation.

When used in the setup sequence, this event specifies values to be copied into the `Scan_Load` event for each subsequent test sequence for all the scan loadable OPCG registers. If there are OPCG registers that are loaded by the OPCG scan operation then the `Load_OPCG_Controls` event specifies that the OPCG scan operation is to be invoked to load those registers. Refer to OPCG Controls and Identifying Test Function Pins (Advanced) in the *Encounter Test: Guide 2: Testmodes* for more information.

Example:

```
Event 1.1.1.1.1.2.1   Load_OPCG_Controls ():
"MULT" = 00100
"RANGEA" = 1101
"RANGEB" = 010
"TUNE" = 1000111000 ;
```

The following table shows the use of the `Load_OPCG_Controls` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X		type=init
	OPMISR/OPMISR+	X		type=init
	XOR Compression	X		type=init
	Full/Partial Scan	X		type=init
GSD	WRP/LBIST	X		type=init
	OPMISR/OPMISR+	X		type=init
	XOR Compression	X		type=init
	Full/Partial Scan	X		type=init

Load_SR

This event scans in latch values in a scan design environment, the same as the `Scan_Load` event. It differs from the `Scan_Load` event in that the values to be scanned in are those to be applied at the input of a controllable register rather than on internal latches. It has two attributes:

- `stim_register`

  This is the ID of the associated controllable register. It correlates to the *Controllable Register Application Object* which describes this controllable register.

- `number_shifts`

  The number of shifts necessary to scan in the given values. The loading of the controllable register starts with the first (left-most) logic value in the stim vector and continues for the number of shifts stated until all required values have been loaded.

The values specified in the load vector are those to be loaded into representative stim latches. Inversions between the representative stim latch and the scan-in are factored into the load vector value.

Note: All `Load_SR` events within a pattern take place in parallel.

Example:

```
    Event 1.1.1.1.1.4.1 Load_SR (stim_register = 4, number_shifts = 9);
101010010;
```

The following table shows the use of the `Load_SR` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression	X		X
	Full/Partial Scan	X		X
GSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression	X		X
	Full/Partial Scan	X		X

Measure_Current

This event is used in IDDq testing to specify when the power supply current should be measured. There is no data associated with this event; the test system must get the

identification of the power supply pin(s) and the acceptable current limit from sources outside of Encounter Test. This event has no attributes.

Example:

```
Event 1.1.1.1.1.5.1 Measure_Current ();
```

The following table shows the use of the `Measure_Current` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			X
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			X
GSD	WRP/LBIST			X
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			X

Scan_Unload

This event scans out latch values in a scan design environment. A Define_Sequence with the attribute of scanop is used to perform the unload operation. The values specified are those in the latches; the value may appear inverted on the scan data primary output due to the intervening logic between the latch and the primary output. It specifies values for the latches

in the Measure Latch Correspondence List (representative measure latches). Refer to Appendix A, “Scan Operation” for a diagram.

Notes

- If using the node/name format of `TBDpatt`, RML latches may be identified by either referencing the latch primitive block or output pin name directly or by referencing the containing cell output pin name if there is no fan-in along the path from the latch primitive pin to the cell pin.
- If a Vectors file is created using Version 4.1 or later, and expansion of Scan_Load occurs, the primary inputs are no longer restored to their pre-scan states, instead are left in their scan-corrupted states.

This event has one attribute: `default_value =0 |1 |X |scan_0 |scan_1`. This specifies what value to look for from a latch whose value is not uniquely specified. A default value of X means that the values of the unspecified latches will be ignored by the tester or TDS software. A default value of 0 (or 1) means that the unspecified latches will be measured for 0 (or 1). The value appearing on the scan data primary output will depend upon the number of inversions between it and each defaulted latch. A default value of scan_0 (or scan_1) means that 0 (or 1) is expected on the scan data primary output when the defaulted latches are to be measured. The value scanned out of each defaulted latch will depend upon the number of inversions between it and the scan data primary output. If this attribute is not used, the default value is X.

Vector format example:

```
    Event 1.1.1.1.1.3.3 Scan_Unload ():
10;
Node format example:
    Event 1.1.1.1.1.3.3 Scan_Unload ():
"Block.f.l.lsrfh0b.nl.srf10hm.021"=0
"Block.f.l.lsrfh0b.nl.srf10hn.021"=1;
```

The following table shows the use of the `Scan_Unload` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			

	XOR Compression	
	Full/Partial Scan	X
GSD	WRP/LBIST	
	OPMISR/OPMISR+	
	XOR Compression	
	Full/Partial Scan	

Measure_MISR_Data

This event is found in the misr_observe scan sequence. When it is used within a scan sequence, it directs when values on `MISR_OBSERVE` test function pins are to be strobed into the tester for comparison. These pins will usually be the same as the scan I/O pins. This event has no attributes.

Example:

```
Event 7.1.2 Measure_MISR_Data ():
"Pin.f.l.top.nl.SCAN_PIN[0]";
```

For additional information, refer to "On-Product MISR Controls" in the *Encounter Test: Guide 2: Testmodes* and Define_Sequence types.

The following table shows the use of the `Measure_MISR_Data` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			X
	XOR Compression			
	Full/Partial Scan			
GSD	WRP/LBIST			
	OPMISR/OPMISR+			X
	XOR Compression			
	Full/Partial Scan			

Measure_PO

This event contains expected PO values. This event type causes the design primary output pins to be observed (and simulators to produce the expected values for each output pin). By definition, design primary output pins are to be ignored except when explicitly specified by this event type. This event has the `timed_type` attribute.

Vector format example:

```
    Event 1.1.1.1.1.3.1 Measure_PO ():
111;
```

Node format example:

```
    Event 1.1.1.1.1.3.1 Measure_PO ():
"Pin.f.l.lsrfh0b.nl.00"=1
"Pin.f.l.lsrfh0b.nl.01"=1
"Pin.f.l.lsrfh0b.nl.02"=1;
```

The following table shows the use of the `Measure_PO` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			X
GSD	WRP/LBIST			
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			X

Measure_Scan_Data

This event is used inside a Define_Sequence with type scansequence for scanning or unloading latches. By its placement within the sequence definition it tells when to observe the primary output(s) on which the latch values appear, and it specifies which primary outputs to look at. This event has no attributes.

Example:

```
    Event 1.1.1.1.1.9.1 Measure_Scan_Data ():
"Pin.f.l.lsrfh0b.nl.02";
```

The following table shows the use of the `Measure_Scan_Data` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression		X	
	Full/Partial Scan		X	
GSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression		X	
	Full/Partial Scan		X	

PI_Weight

This event is used in weighted random pattern test to specify the bias to be applied in the selection of random values to be put on the primary inputs. In concept, a weight is the probability that the value will be a 1. Each weight is specified in terms of equivalent AND/OR inputs --the number of unbiased random signals that are ANDed together or ORed together to produce the weighted random value. This event has no attributes.

Vector format example:

```
    Event 1.1.1.1.1.14.1 PI_Weight ():
-.-.A2A3-.-.-.-.O2O4;
Node format example:
    Event 1.1.1.1.1.14.1 PI_Weight ():
"Pin.f.l.lsrfh0b.nl.a2"=A2
"Pin.f.l.lsrfh0b.nl.a3"=A3
"Pin.f.l.lsrfh0b.nl.a8"=O2
"Pin.f.l.lsrfh0b.nl.a9"=O4;
```

The following table shows the use of the `PI_Weight` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X		
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			
GSD	WRP/LBIST	X		
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Product_PRPG_Signature

This event is used to record intermediate on-product PRPG states to allow application of a subset of the pseudo-random patterns for diagnostic analysis. This event type has the attributes `iteration`, `fast_forward`, and `final`.

Example:

```
    Event 1.1.1.3.2.8.2 Product_PRPG_Signature (iteration=256):
```

The following table shows the use of the `Product_PRPG_Signature` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X		
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			
GSD	WRP/LBIST	X		
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Product_MISR_Signature

This event is used to record signatures against which to compare the contents of on-board MISRs. The comparison is done either at the end of the test, or for diagnosis, at some intermediate point. This event type has the attributes `iteration`, `fast_forward`, and `final`.

Example:

```
Event 1.1.1.3.2.8.4 Product_MISR_Signature (iteration=256):
```

Data values for the MISR bits are printed in hexadecimal notation in the order shown by the vector correspondence information. By default, this order is bit 1 of MISR 1, bit 2 of MISR 1,...the last bit of MISR 1, bit 1 of MISR 2, etc.

The following table shows the use of the `Product_MISR_SIGNATURE` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+	X		

	XOR Compression	
	Full/Partial Scan	
GSD	WRP/LBIST	
	OPMISR/OPMISR+	X
	XOR Compression	
	Full/Partial Scan	

Pulse

This event identifies a list of clocks to be pulsed and the direction to be pulsed. This event has the timed_type attribute.

Note: Beware of putting multiple clocks in one Pulse event when writing manual patterns or sequence definitions. Under certain conditions, this could cause a mismatch between Encounter Test's results and the hardware. When you are applying multiple clock pulses, you should put each one in a separate Pulse event. This will ensure that they are simulated by Encounter Test in the same order that they are applied at the tester.

Example:

```
Event 1.1.1.2.1.5.1 Pulse (timed_type = release):
```

The following table shows the use of the `Pulse` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X
GSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X

A pulse event applies both the clock rise and fall edges in a single event, thus returning the clock to its off-state at the end of the event. This event does not allow any other event to occur between the clock edges.

Pulse_PPI

This event identifies a list of pseudo primary inputs to be pulsed and the direction to be pulsed. Pseudo primary inputs are always referred to by name; the `TBDpatt_Format` statement has no effect on the format of this event. This event has the timed_type attribute.

Example:

```
    Event 1.1.1.1.1.3.4 Pulse_PPI ():
"OPCout"=+;
```

The following table shows the use of the `Pulse_PPI` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X
GSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X

Pulse_Tester_PRPG_Clocks

This event specifies that new random values are to be applied at the primary inputs by means of pulsing a clock inside the tester that controls the PRPGs that supply these values to the design primary inputs. This event is used in the application of weighted random pattern tests. All tester PRPGs are pulsed, including those connected to scan data primary inputs and any that are temporarily disconnected from a pin. This event has the timed_type attribute.

Example:

```
Event 2.1.1 Pulse_Tester_PRPG_Clocks ();
```

The following table shows the use of the `Pulse_Tester_PRPG_Clocks` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			
GSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Pulse_Tester_SISR_Clocks

This event is used in weighted random pattern testing to specify when the design primary outputs are to be observed by means of clocking the signature registers that are connected to the primary outputs. All tester SISRs are pulsed, including those connected to scan data primary outputs and any that are temporarily disconnected from a pin. This event has the timed_type attribute.

Example:

```
Event 2.1.1 Pulse_Tester_SISR_Clocks ();
```

The following table shows the use of the `Pulse_Tester_SISR_Clocks` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+			
	XOR Compression			

	Full/Partial Scan			
GSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Put_Stim_PI

This event provides a list of primary inputs and the values to which they should be set. It is used in a test sequence definition to tell when the automatically generated primary input vector is to be applied. The test pattern generator will replace the `Put_Stim_PI` event by its own primary input stimulus event (`Stim_PI` or `Stim_PI_Plus_Random`), but any primary input values specified in the `Put_Stim_PI` event will override the corresponding values specified by the automatic test pattern generator.

This event is used only in sequence definitions that are identified with the `testsequence` keyword on one of the `create_*_tests` commands. `Put_Stim_PI` events are not allowed in test sequences identified with the `tgtemplate` keyword.

This event has no attributes.

Example:

```
Event 2.2.1 Put_Stim_PI ():
```

The following table shows the use of the `Put_Stim_PI` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			X
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			X
GSD	WRP/LBIST			X
	OPMISR/OPMISR+			X
	XOR Compression			X

	Full/Partial Scan	X

Release

This event discontinues the effect of a previous `Force` event on each node specified by the `Release` event. This allows each specified node to assume the value consistent with its current input values existing at the time the `Release` event is processed. If one of the specified nodes changes as the result of simulating the `Release` event, this means that the `Force` was still doing something "unnatural" to the design, and this causes the simulator to produce an informational message to that effect (see message "TFS-884" in the *Encounter Test: Reference: Messages)*.

Example:

```
    Event 1.1.1.1.1.8.1 Release ():
"Pin.f.l.opgckt.nl.opcg1.master.DOUT";
```

This event has no attributes.

The following table shows the use of the `Release` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X
GSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X

Repeat

This event should appear only inside a pattern with the `begin_loop` attribute. It specifies the number of loop iterations to be applied. This event has no attributes.

Example:

```
Event Repeat (): 30;
```

Pattern loop repeats in excess of 4,294,967,295 will be broken into multiple pattern loops. This value can be controlled with the `read_vectors` keyword `maxpatternloops`.

The following table shows the use of the `Repeat` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X
GSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X

Scan_Load

This event scans in flop scan cell values in a scan design environment. A `Define_Sequence` with the attribute of `scanop` is used to perform the load operation. The values specified are those to be loaded into the scan cells; the value applied to the scan data primary input may have to be inverted, due to the intervening logic between the primary input and the cell. It specifies values for the cells in the Scan Load Correspondence List and for all other scan cells whose states are implied by loading the representative RSLs. Any controllable cell not so specified is to be loaded to a random value picked by the simulator. If a cell is specified that

is not an RSL, then the value is moved to the corresponding RSL. An error is indicated if there is no corresponding RSL for the cell.

Notes

1. If using the node/name format of `TBDpatt`, RSLs may be identified by either referencing the primitive block or output pin name directly or by referencing the containing cell output pin name if there is no fan-in along the path from the primitive pin to the cell pin.
2. If you are not explicitly specifying values for all RSLs, the compact version of this event should be used to achieve a reduction in the size of the Vectors file that results from Read Vectors. Refer to "Compact_Scan_Load" on page 50 for additional information.

This event has the following attribute:

- `default_value=0 |1 |X |scan_0 |scan_1`

  This specifies what value should be scanned into a scan cell whose value is not uniquely specified. A default value of X means that cell will be set arbitrarily by the tester or TDS software. A default value of 0 (or 1) means that the unspecified cells will be set to 0 (or 1). The value required on the scan data primary input will depend upon the number of inversions between it and each defaulted flop. A default value of scan_0 (or scan_1) means that 0 (or 1) will be placed on the scan data primary input when the defaulted cells are to be loaded. The value resulting in each defaulted cell will depend upon the number of inversions between it and the scan data primary input. If this attribute is not used, the default value is X.

Refer to Appendix A, "Scan Operation" for a diagram.

Example:

```
Event 1.1.1.1.1.1.1 Scan_Load (default_value=0):
```

The following table shows the use of the `Scan_Load` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			X
	XOR Compression			X

	Full/Partial Scan	X
GSD	WRP/LBIST	
	OPMISR/OPMISR+	X
	XOR Compression	X
	Full/Partial Scan	X

Set_CME_Data

This event is placed into the scan_cycle to indicate when the Channel_Mask_Enable (CME) pins should be applied (just before the scan-in pins).

Example:

```
Event 3.1.1 Set_CME_Data ():
"Pin.f.l.a2901scan.nl.q31_in"
"Pin.f.l.a2901scan.nl.ram0_in" ;
```

The following table shows the use of the `Set_CME_Data` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			
GSD	WRP/LBIST			
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			

Set_CMI_Data

This event is placed into the channelmaskcycle sequence to indicate when the Channel_Mask_Input (CMI) pins should be applied (just before the pulse of he Channel_Mask_Load (CML) clock.

Example:

```
Event 9.1.1 Set_CMI_Data ():
"Pin.f.l.a2901scan.nl.scanin"
"Pin.f.l.a2901scan.nl.ram31in" ;
Event 9.1.3 Pulse ():
```

The following table shows the use of the `Set_CMI_Data` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			
GSD	WRP/LBIST			
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			

Set_OLI_Data

This event indicates the scan cycle location where the loading of OPCG register should be applied to the `OLI` pins.

Example:

```
Event 9.1.1 Set_OLI_Data ():
"Pin.f.l.a2901scan.nl.scanin"
"Pin.f.l.a2901scan.nl.ram31in" ;
Event 9.1.3 Pulse ():
```

The following table shows the use of the `Set_OLI_Data` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X
GSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X

Set_Scan_Data

This event is used inside a `Define_Sequence` with type scansequence for scanning or loading latches. This event lists PIs that are scan data inputs that are supposed to be stimulated at that point in the scan sequence. By its placement within the sequence definition it tells when to place on the PI the values that are to be loaded into the latches. This event has no attributes.

Example:

```
Event Set_Scan Data (): "SI1" "SI2";
```

The following table shows the use of the `Set_Scan_Data` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST		X	
	OPMISR/OPMISR+		X	
	XOR Compression		X	
	Full/Partial Scan		X	

GSD	WRP/LBIST	X
	OPMISR/OPMISR+	X
	XOR Compression	X
	Full/Partial Scan	X

Start_Osc

This event starts an oscillating waveform on the specified pin(s). All pins in the `Start_Osc` event must be in the same correlated set. The corresponding pin should have the ±`OSC` attribute This oscillating waveform is applied concurrently with subsequent events, until the waveform is stopped, either by an expiration of the cycle count specified in this event, or by a `Stop_Osc` event. In addition, a `Stim_Clock` or `Pulse` event on this pin will stop the oscillator. This event is intended to be used to run On-Product Clock Generation (OPCG) logic. The OPCG logic typically ignores all input oscillators until a triggering event occurs (normally the assertion of a `GO` test function pin), at which point the OPCG logic fires off some number of internal clock pulses based on the input oscillators (often fed through on-product Phase-Locked Loops - PLLs) and custom OPCG programming. The OPCG logic's finite-state machine (FSM) then shuts down and waits for another triggering event.

This event has the following optional attributes:

- `pulses_per_cycle = n`

  This attribute is specified if the oscillator signal is supplied by the tester, with a fixed number of oscillator pulses per tester cycle. Specify the number of oscillator pulses per tester cycle. attribute defines the oscillator to be synchronous with the tester. If this attribute is not specified, it is assumed that the oscillator will be asynchronous to the tester.

  Specify a positive integer.

- `cycles = n`

  This attribute is specified if the oscillator signal is to have a specific number of cycles applied to it and then stop. Specify the total number of oscillator pulses to be applied to this pin. If this attribute is not specified, it is assumed that the oscillator will continue indefinitely until explicitly stopped.

  Specify a positive integer.

- `up n.nn ts`

  This attribute is specified to denote the time to be spent in the positive portion of the oscillator cycle. It is paired with the `down` attribute to complete the specification of the

period for the oscillator cycle. For example, to specify a 500 MHz oscillator with 50% duty cycle, specify (up 1.0 ns, down 1.0 ns). This attribute can be specified for oscillators that are synchronous with the tester to specify the approximate period and duty cycle (and imply the frequency) for the input oscillator, but tester programming may not achieve the exact values specified for the up and down attributes. This attribute is more important for asynchronous oscillators to specify to test engineering what frequency and duty cycle is needed for the input oscillator; for asynchronous oscillators, if this attribute is not specified, it is assumed that the oscillator will be set to an appropriate frequency by a test engineer.

Specify a positive number and a time scale (ps, ns, us, or ms to indicate the time is specified in pico-, nano-, micro-, or milli- seconds).

- `down` *`n.nn`* `ts`

  This attribute is specified to denote the time to be spent in the negative portion of the oscillator cycle. It is paired with the up attribute to complete the specification of the period for the oscillator cycle. For example, to specify a 333 MHz oscillator with 33% duty cycle, specify (up 1.0 ns, down 2.0 ns). This attribute can be specified for oscillators that are synchronous with the tester to specify the approximate period and duty cycle (and imply the frequency) for the input oscillator, but tester programming may not achieve the exact values specified for the up and down attributes. This attribute is more important for asynchronous oscillators to specify to test engineering what frequency and duty cycle is needed for the input oscillator; for asynchronous oscillators, if this attribute is not specified, it is assumed that the oscillator will be set to an appropriate frequency by a test engineer.

  Specify a positive number and a time scale (ps, ns, us or ms to indicate the time is specified in pico-, nano-, micro-, or milli- seconds).

Example:

```
Event Start_Osc (pulses_per_cycle=3,up 1.5 ns, down 1.5 ns) : "B"=+;
```

In the example, the tester is instructed to pulse pin `B` 3 times per tester cycle evenly distributed within the tester cycle to create an oscillating input. By specifying a period of 3ns, it is expected that the resulting oscillator signal will run at approximately 333MHz. To do this, the tester cycle will need run as close to 111MHz as is possible (about 9ns cycle time). While the oscillator is pulsing pin `B`, other events following this one may be applied to the product. Normally, Encounter Test simulators set this pin to X, and it will remain there until some other event specifies a value to place on the pin.

The value plus (`+`) in the preceding example is the starting value of the oscillator, applied when the `Start_Osc` event is processed. Pulses will continuously occur on the oscillator, but will normally take affect via OPCG logic only in response to a triggering event, typically caused by the raising of a GO signal. Once triggered, the OPCG logic should run until it

consumes sufficient input oscillator cycles to produce the internal clocks as programmed. This is indicated by use of `Wait_Osc` events within Test_Sequences that specify to wait for some number of cycles to occur before applying any other stimulus to the device.

The following table shows the use of the `Start_Osc` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X	X	
	OPMISR/OPMISR+	X	X	
	XOR Compression	X	X	
	Full/Partial Scan	X	X	
GSD	WRP/LBIST	X	X	
	OPMISR/OPMISR+	X	X	
	XOR Compression	X	X	
	Full/Partial Scan	X	X	

Skewed_Compressed_Input_Stream

This event causes a “skewed load” by adding an extra A shift clock pulse to be applied at the end of the scan operation. Because this shifts one more bit of data into the scan chains, this event contains one more bit of data for each scan-in (`SI`) pin. A consequence of using this event is that, for LSSD designs, the L1 master latch and its companion L2 slave latch can be set to different/independent values.

The listed values are the values described for the Compressed_Input_Stream event.

The following table shows the use of the `Skewed_Compressed_Input_Stream` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			

	XOR Compression	X
	Full/Partial Scan	
GSD	WRP/LBIST	
	OPMISR/OPMISR+	
	XOR Compression	X
	Full/Partial Scan	

Skewed_Compressed_Output_Stream

This event causes a “skewed unload” by starting the scan-out by applying all of the LSSD B shift clocks prior to the full count of scan shift cycles, with the result that the master L1 latches are observed instead of the slave L2 latches. This event lists the values described for the Compressed_Output_Stream event.

The event attribute `overlap=no` indicates overlapping with the next scan-in is disallowed and the values provided in the event were computed with the assumption that the scan-in (SI) pins will bet set to zeros (0) for all scan cycles. The `imported` attribute indicates the data was imported from a foreign pattern source (IEEE 1450 STIL).

The following table shows the use of the `Skewed_Compressed_Output_Stream` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			X
	Full/Partial Scan			
GSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Skewed_Internal_Scan_Load

This event lists the values that will be loaded into all flop scan cells as a result of preceding events, similar to the Internal_Scan_Load event with the addition that this event also stores the skewed scan latch (SSL) values.

Skewed_Scan_Unload

This event scans out latch values in a LSSD scan design environment. It differs from the `Scan_Unload` event in that the `Skewed_Scan_Unload` event includes the application of the skewunload sequence. This observes the values of the latches preceding the latches controlled by the B clocks in the scan chains. Values are reported for representative measure latches (RML) and the values reported for the RMLs are the values after the initial B clock pulse. Refer to Appendix A, “Scan Operation” for a diagram.

Note: If using the node/name format of `TBDpatt`, RML latches may be identified by either referencing the latch primitive block or output pin name directly or by referencing the containing cell output pin name if there is no fan-in along the path from the latch primitive pin to the cell pin.

This event has one attribute: *`default_value`* = `0 |1|X|scan_0 |scan_1`. This specifies what value to look for from a latch whose value is not uniquely specified. A default value of X means that values from the unspecified latches will be ignored by the tester or TDS software. A default value of 0 (or 1) means that the unspecified latches will be measured for 0 (or 1). The value appearing on the scan data primary output will depend upon the number of inversions between it and each defaulted latch. A default value of scan_0 (or scan_1) means that 0 (or 1) is expected on the scan data primary output when the defaulted latches are to be measured. The value scanned out of each defaulted latch will depend upon the number of inversions between it and the scan data primary output. If this attribute is not used, the default value is X.

Vector format example:

```
Event 1.1.1.1.1.4.1 Skewed_Scan_Unload ():
11;
Node format example:
    Event 1.1.1.1.1.4.1 Skewed_Scan_Unload ():
"Block.f.l.lsrfh0b.nl.srf10hm.021"=1
"Block.f.l.lsrfh0b.nl.srf10hn.021"=1;
```

The following table shows the use of the `Skewed_Scan_Unload` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			X
GSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Skewed_Unload_SR

This event scans out latch values in a scan design environment, the same as the `Skewed_Scan_Unload` event. It differs from the `Skewed_Scan_Unload` event in that the vector of values given are those to be observed at the output of a observable scan chain rather than on internal latches. It has two attributes:

- `measure_register`

  This is the ID of the associated observable scan chain. It correlates to the *Measure Register Application Object* which describes this observable scan chain.

- `number_shifts`

  The number of shifts necessary, after the initial B_SHIFT_CLOCK pulse, to scan out the given values. The unloading of the measure register starts with the first (left-most) logic value in the measure vector and continues for the number of shifts stated until all values have been scanned out.

The `Skewed_Unload_SR` event differs from the `Unload_SR` event in that it includes the application of the skewunload sequence. This observes the values of the latches preceding the latches controlled by the B clocks in the observable scan chains. Values in the unload vector are those scanned out of representative measure latches (RMLs), and the values reported for the RMLs are the values after the initial B clock pulse. Scan path inversions

between the latch being scanned out and the observable scan chain output are factored into the scan-out values given.

Note: All `Skewed_Unload_SR` events within a pattern take place in parallel.

Example:

```
   Event 1.1.1.1.1.4.1 Skewed_Unload_SR (measure_register = 3, number_shifts = 8):
110100010;
```

The following table shows the use of the `Skewed_Unload_SR` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			X
GSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Skewed_Load_SR

This event scans in latch values in a scan design environment, the same as the `Skewed_Scan_Load` event. It differs from the `Skewed_Scan_Load` event in that the values to be scanned in are given by reference to a controllable register ID and a vector of values to be applied to the input of that controllable register, rather than as a set of latch values. It has two attributes:

- `stim_register`

  This is the ID of the associated controllable register. It correlates to the *Controllable Register Application Object* which describes this controllable register.

- `number_shifts`

The number of shifts necessary to scan in the given values. The loading of the controllable register starts with the first (left-most) logic value in the stim vector and continues for the number of shifts stated, followed by the pulsing of all identified A scan clocks, until all required values have been loaded.

The values specified in the load vector are those to be loaded into representative stim latches. Inversions between the representative stim latch and the scan-in are factored into the load vector values.

Note: All `Skewed_Load_SR` events within a pattern take place in parallel.

Example:

```
    Event 1.1.1.1.1.4.1 Skewed_Load_SR
    (stim_register = 6, number_shifts = 7);
1011010;
```

The following table shows the use of the `Skewed_Load_SR` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression	X		
	Full/Partial Scan	X		
GSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Stim_Clock

This event type provides a list of clock primary inputs and the values to which they should be set. This event has the `timed_type` attribute.

Example:

```
    Event 1.1.1.4.1.3.1 Stim_Clock ():
```

The following table shows the use of the `Stim_Clock` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X
GSD	WRP/LBIST			
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X

This event can produce a single edge of a clock, therefore leaving the clock in the on-state at the end of the event or pattern.

The `stim_clock` event allows other events to occur between the rising and falling edge of the clock, as shown below:

```
Stim_clock:clk=1
Stim_PI (other pins)
Measure_PO:
Stim_clock: clk=0
```

Skewed_Scan_Load

This event scans in latch values in a scan design environment. It differs from the `Scan_Load` event only in that the scan operation for `Skewed_Scan_Load` includes an extra pulse on all identified “A” scan clocks, as implied by its name. The effect is to reach latch states that cannot be reached by the normal scan sequence. An extra value gets loaded into the first

latch in a scan chain if this latch is controlled by an “A” clock. The specification of this extra value gives this event a slightly different appearance from the `Scan_Load` event.

Notes

1. If using the node/name format of `TBDpatt`, RSL latches may be identified by either referencing the latch primitive block or output pin name directly or by referencing the containing cell output pin name if there is no fan-in along the path from the latch primitive pin to the cell pin.
2. If you are not explicitly specifying values for all RSLs, the compact version of this event should be used to achieve a reduction in the size of the Vectors file that results from Read Vectors. Refer to “Compact_Skewed_Scan_Load” on page 52 for additional information.
3. This event can be compacted to achieve a reduction in the size of the Vectors file. Refer to “Compact_Skewed_Scan_Load” on page 52 for additional information.

This event has the following attribute:

- `default_value=0 |1 |X |scan_0 |scan_1`

  This specifies what value should be scanned into a latch whose value is not uniquely specified. A default value of X means that latch will be set arbitrarily by the tester or TDS software. A default value of 0 (or 1) means that the unspecified latches will be set to 0 (or 1). The value required on the scan data primary input will depend upon the number of inversions between it and each defaulted latch. A default value of scan_0 (or scan_1) means that 0 (or 1) will be placed on the scan data primary input when the defaulted latches are to be loaded. The value resulting in each defaulted latch will depend upon the number of inversions between it and the scan primary input. If this attribute is not used, the default value is X.

Refer to Appendix A, “Scan Operation” for a diagram.

Example:

```
Event 1.1.1.2.1.1.1 Skewed_Scan_Load ():
```

The following table shows the use of the `Skewed_scan_Load` event:

	Mode initialization	Custom scan protocol	Test pattern sequences

LSSD	WRP/LBIST	
	OPMISR/OPMISR+	X
	XOR Compression	X
	Full/Partial Scan	X
GSD	WRP/LBIST	
	OPMISR/OPMISR+	
	XOR Compression	
	Full/Partial Scan	

Stim_PI

This event type provides a list of primary inputs and the values to which they should be set. The simulator will not randomize unspecified PIs. When this event is shown in the vector format, periods (.) are used as place holders for those primary inputs that are not to be affected by this event. This event has the `timed_type` attribute.

Example:

```
Event 5.2.1 Stim_PI ():
```

The following table shows the use of the `Stim_PI` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X
GSD	WRP/LBIST			
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X

Stim_PI_Plus_Random

This event type is to be used only as input to a simulator. This event type provides a list of primary inputs and the values to which they should be set. It implies that any primary input which can properly be stimulated (with the exceptions noted below) will be set to a simulator supplied random value. When this event is shown in the vector format, periods (.) are used as place holders for those primary inputs which are to be randomized and for those that are not to be affected by this event. This event has the `timed_type` attribute.

Notes

The following classes of primary inputs are not randomized, even if their vector positions are not specified (occupied by periods):

- Test Inhibit (TI)

  These pins are never allowed to be taken out of their specified stability state. They are not touched by the `Stim_PI_Plus_Random` event.

- Clocks

  These pins are not touched by the `Stim_PI_Plus_Random` event. They should be at their stability values, as it is not recommended to change any other primary inputs while a clock is at its defined on state; however, the Encounter Test simulators do not verify this.

- Three-state primary inputs

  These would be bi-directional pins that are driven from the design by three-state drivers. They are set to the high-impedance value if left unspecified in the `Stim_PI_Plus_Random` event.

- Lineheld (LH) primary inputs

  These are PIs that are specified in the linehold file.

Example:

```
Event 1.1.1.2.1.1.1 Stim_PI_Plus_Random ():
```

The following table shows the use of the `Stim_PI_Plus_Random` event:

	Mode initialization	Custom scan protocol	Test pattern sequences

LSSD	WRP/LBIST	
	OPMISR/OPMISR+	X
	XOR Compression	X
	Full/Partial Scan	X
GSD	WRP/LBIST	
	OPMISR/OPMISR+	X
	XOR Compression	X
	Full/Partial Scan	X

Stim_PPI

This event specifies a list of pseudo primary inputs to be set to specific values. Treatment of this event by Encounter Test applications is identical to the Stim_PI event, except that this event is for pseudo PIs only. TDS (manufacturing) applications (other than diagnostic simulators) should ignore this event. The values are stored as a vector, and the event is printed in node list format.

Pseudo primary inputs are always referred to by name; the `TBDpatt_Format` statement has no effect on the format of this event. This event hs the `timed_type` attribute.

Example:

```
Event 1.1.1 Stim_PPI (timed_type=release):
```

The following table shows the use of the `Stim_PPI` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X
GSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X

	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X

Stim_PPI_Clock

This event specifies a list of pseudo primary inputs to be set to specific values. Treatment of this event by Encounter Test applications is identical to the `Stim_Clock` event, except that this event is for pseudo PIs only.

Pseudo primary inputs are always referred to by name; the `TBDpatt_Format` statement has no effect on the format of this event. This event has the `timed_type` attribute.

Example:

```
Event 1.27.2 Stim_PPI_Clock:
```

The following table shows the use of the `Stim_PPI_Clock` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X
GSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X

Stop_Osc

This event provides a list of primary inputs to be quiesced from a previous `Start_Osc` event. The quiescent value for each pin is the opposite of the value specified on that pin in its previous `Start_Osc` event.

All pins in the `Stop_Osc` event must be in the same correlated set.

This event has no attributes.

Example:

```
    Event Stop_Osc ():
"Pin.f.l.opcgckt.nl.B"=0;
```

The following table shows the use of the `Stop_Osc` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X	X	
	OPMISR/OPMISR+	X	X	
	XOR Compression	X	X	
	Full/Partial Scan	X	X	
GSD	WRP/LBIST	X	X	
	OPMISR/OPMISR+	X	X	
	XOR Compression	X	X	
	Full/Partial Scan	X	X	

Tester_PRPG_Seed

This event specifies the initial values to be loaded into the tester's PRPGs that are connected to the design primary inputs. In the vector format there is one seed for each primary input in the vector correspondence list. In the node list format, each seed value is prefixed by the name of the primary input. The seed values are specified as hexadecimal strings. This event has no attributes.

Example:

```
    Event 1.1.1.2.1.1.1     Tester_PRPG_Seed ():
```

The following table shows the use of the `Tester_PRPG_Seed` event:

	Mode initialization	Custom scan protocol	Test pattern sequences

LSSD	WRP/LBIST	X
	OPMISR/OPMISR+	
	XOR Compression	
	Full/Partial Scan	
GSD	WRP/LBIST	X
	OPMISR/OPMISR+	
	XOR Compression	
	Full/Partial Scan	

Tester_PRPG_Signature

This event is used to specify intermediate primary input PRPG states to allow application of a subset of the pseudo-random patterns for diagnostic analysis. This event type has the attributes `iteration`, `fast_forward`, and `final`.

Example:

```
Event 1.1.1.3.2.8.1     Tester_PRPG_Signature (iteration=256):
```

The following table shows the use of the `Tester_PRPG_Signature` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			X
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			
GSD	WRP/LBIST			X
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Tester_SISR_Mask

This event tells which primary outputs have active signature registers connected for the current `Test_Procedure`. This allows a SISR that is assigned to a pin to be disconnected during the `Test_Procedure`. This flexibility is required on bidirectional pins to support testers that do not have true bidirectional pin support. For one `Test_Procedure`, the pin may be monitored by a SISR and receive no stimulus from the tester; for another test procedure, the pin may receive stimulus (either a line-hold, stims, or from a PRPG) from the tester, but have its SISR disconnected.

The mask is defined as a bit string, one bit per primary output in the primary output correlation list; a 1 indicates that the corresponding primary output has a SISR that is connected for the test procedure, while a 0 indicates that either there is no SISR assigned to the primary output or the SISR is disconnected. When a SISR is not connected, Encounter Test assumes that it does not get clocked during the test procedure.

This event appears only in a *setup* type `Test_Sequence`. It has no attributes.

Example:

```
Event 1.1.1.3.1.1.4     Tester_SISR_Mask ():
```

The following table shows the use of the `Tester_SISR_Mask` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			X
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			
GSD	WRP/LBIST			X
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Tester_SISR_Seed

This event specifies the initial values to be loaded into the tester's SISRs that are connected to the design primary outputs. In the vector format there is one seed for each primary output in the vector correspondence list. In the node list format, each seed value is prefixed by the name of the primary output. The seed values are specified as hexadecimal strings.

This event has the `fast_forward` attribute. The `fast_forward` attribute allows the SISR to be seeded differently for normal or `fast_forward` test application. This makes it possible to have the SISR seeds included at the beginning of each `Test_Procedure` and still have the option of either (a) comparing signatures and reseeding between successive `Test_Procedures` or (b) continuing to the next `Test_Procedure` without comparing signatures and reseeding.

Example:

```
Event 1.1.1.3.1.1.2     Tester_SISR_Seed ():
```

The following table shows the use of the `Tester_SISR_Seed` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			X
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			
GSD	WRP/LBIST			X
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Tester_SISR_Signature

This event is used to specify the primary output signatures against which to compare the contents of the tester SISRs (single-input signature registers). The comparison is done either at the end of the test, or for diagnosis, at some intermediate point. This event type has the attributes `iteration`, `fast_forward`, and `final`.

Example:

```
Event 1.1.1.2.2.7.3 Tester_SISR_Signature (iteration=256):
```

The following table shows the use of the `Tester_SISR_Signature` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			X
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			
GSD	WRP/LBIST			X
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			

Unload_SR

This event scans out latch values in a scan design environment, the same as the `Scan_Unload` event. It differs from the `Scan_Unload` event in that the scanned-out values are given by reference to a observable scan chain ID and a vector of values to be observed at the output of that observable scan chain, rather than as a set of latch values. It has two attributes:

- `measure_register`

  This is the ID of the associated observable scan chain. It correlates to the *Measure Register Application Object* which describes this observable scan chain.

- `number_shifts`

  The number of shifts necessary to scan out the given values. The unloading of the observable scan chain starts with the first (left-most) logic value in the measure vector and continues for the number of shifts stated until all values have been scanned out.

Values are those scanned out of representative measure latches (RMLs). Scan path inversions between the latch being scanned out and the observable scan chain output are factored into the scan-out values given.

Note: All `Unload_SR` events within a pattern take place in parallel.

Example:

```
    Event 1.1.1.1.1.4.1 Unload_SR (measure_register = 3, number_shifts = 8)
110100010;
```

The following table shows the use of the `Unload_SR` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			X
GSD	WRP/LBIST			
	OPMISR/OPMISR+			
	XOR Compression			
	Full/Partial Scan			X

Use_Channel_Masks

The `Use_Channel_Masks` event specifies what masking is to occur for each and every scan cycle in which response data could be unloaded into the MISR. There should be as many entries as there are scan cycles to completely load and unload the scan chains for each test. Each entry is an encoded value for the state of the channel_mask_enable (`CME`) signals. The value `0` signifies to perform no masking on that scan cycle and so the `CME` pins should be set to their defined stability values. For easier reading, zero (`0`) values are printed as periods (.) and simply indicate that no masking is being done of those scan cycles. A value of `1` selects to use mask register 1 for that scan cycle if there is a mask register defined and to mask all channels if there is no mask register. A value of two indicates to use mask register `2` on that scan cycles or to use the mask-all channels capability if there is no mask register 2. Similarly on up to a value of up to `15`, that indicates to use mask register `15` or the mask-all channels capability if mask register `15` does not exist.

Example:

```
Mask_Reg 1 = 10100010
Mask_Reg 2 = 11111111;
```

```
        Event 1.1.1.2.10.4.2   Use_Channel_Masks ():
1.1.11......1.......;
```

The following table shows the use of the `Use_Channel_Masks` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST			
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			
GSD	WRP/LBIST			
	OPMISR/OPMISR+			X
	XOR Compression			X
	Full/Partial Scan			

Wait_Osc

This event synchronizes other events with the operation of a free-running oscillator which has been initiated by a `Start_Osc` event. The oscillator pin is identified by the `Wait_Osc` event. When a `Wait_Osc` event is encountered, all further action halts (if necessary) until the specified number of oscillator cycles have elapsed. The oscillator cycle count is specified as an integer, representing the number of cycles since the previous `Wait_Osc` (or `Start_Osc`) event.

All pins in the `Wait_Osc` event must be in the same correlated set. A cycle count of 0 means that any preceding events were applied independent of the oscillator.

When two or more `Wait_Osc` events appear consecutively, they are assumed to be concurrent; the “wait time” is the maximum time specified by any one of the `Wait_Osc` events in the group, where the time is calculated by multiplying the number of cycles specified by the period of that oscillator. Encounter Test will indicate an error condition if the nostability attribute is set in some `Wait_Osc` event for which the number of cycles is not sufficient to account for the maximum time calculated for all `Wait_Osc` events in that consecutive group.

This event has the following attributes:

Cycles

Expressed as `Cycles=` *n* where *n* is the number of oscillator cycles since the previous `Wait_Osc` or `Start_Osc` event.

This attribute is required; there is no default.

nostability

The presence of this attribute means that the design is not expected to be stable at this time. A `Wait_Osc` event with this attribute would be placed at a point in the sequence where the Encounter Test sequence verifier needs to know the relative time in terms of oscillator pulses, but should not be checking for stability. If this attribute is not present, the sequence verifier will issue an error message if the design is not stable.

off

The presence of this attribute means that subsequent events are to be applied independent of the oscillator signal. The default, with this attribute omitted, means that subsequent events are synchronized with the oscillator signal, and implies that another `Wait_Osc` event will follow some intervening events, possibly in another pattern.

Example:

```
[ Pattern;
  Event Scan_Load (): 1011100010010101;
] Pattern;
[ Pattern;
  Event Stim_PI ():.0..101001;
  Event Measure_PO ():;
] Pattern;
[ Pattern;
  Event Wait_Osc (Cycles=0): "Pin.f.l.ckt.nl.osc";
  Event Stim_PI ():.1........;   # Apply "go" signal to apply a
  Event Pulse_PPI (): C1=+;        system clock cycle from OPCG logic.
  Event Pulse_PPI (): C2=+;
  Event Wait_Osc (Cycles=4,Off): "Pin.f.l.ckt.nl.osc";
] Pattern;
[ Pattern;
  Event Scan_Unload ():;
] Pattern;
```

The following table shows the use of the `Wait_Osc` event:

		Mode initialization	Custom scan protocol	Test pattern sequences
LSSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X
GSD	WRP/LBIST	X	X	X
	OPMISR/OPMISR+	X	X	X
	XOR Compression	X	X	X
	Full/Partial Scan	X	X	X

Application Object

An application object can be added to certain TBD objects. An application object has the purpose of providing a means of associating Encounter Test data with the test pattern data object that would be awkward to handle as simple attributes of the object. Application objects have unique names as described below. The application object is placed immediately after the begin statement of the TBD object to which it is attached, and may be interspersed with Keyed Data. Application objects can not be attached to events.

Begin_Test_Mode Application Objects

There are two types of application objects that are used to indicate the current test mode of the design.

- [`Going_To_Mode`: *`Test Modename`*];

  This object is on the first pattern of a series of patterns that initialize the named test mode.

- [`In_Test_Mode`: *`Test Modename`*];

  This object is on the last pattern of a series of patterns that initialize the named test mode.

These objects are used when writing a `sigobs` sequence, whenever the readout occurs in a different mode from the LBIST mode. They may also appear in exported `TBDpatt` if the expand modeinit sequence was specified.

Sequence Definition Application Objects

There are six types of Sequence Definition application objects:

- `Define_Sequence` - a sequence definition containing a sequence of patterns. See "Define_Sequence" on page 111.
- `SeqDef` - a referral to a Define_Sequence. See "SeqDef" on page 123.
- `SetupSeq` - a referral to another sequence definition that provides design initialization (beyond that of the mode initialization sequence) specific to the test sequence (definition) to which this object is attached. See "SetupSeq" on page 123.
- `EndupSeq` - a referral to another sequence definition that quiesces the BIST controller at the end of running BIST. See "EndupSeq" on page 123.
- `TestSeqType` - a descriptive key used by Manufacturing to catalogue tests in their database. See "TestSeqType" on page 124.
- `Lineholds` - required lineholds to make this sequence work. See "Lineholds" on page 124.

These objects are described in more detail below.

Define_Sequence

A sequence definition is a set of patterns which, when applied in the given sequence, accomplishes some purpose. Sequence definitions are stored in the `TBDseq` file.

LBIST test sequence definitions, when they are used, are copied from the `TBDseq` file into Vectors, and thus, when the Vectors file is printed, they appear in `TBDpatt`. This protects against the possibility that the user may import a new or changed version of the test sequence after it is used; information from the sequence definition will be used for delay test. All the sequence definitions used in an experiment will appear as application objects immediately following the `[Experiment` statement.

A `Define_Sequence` has the following attributes:

- `collapsible`

Refer to "Test_Sequence" on page 34 for an explanation of this attribute. The `Test_Sequence` inherits the `collapsible` attribute from the `Define_Sequence` from which the `Test_Sequence` is derived.

- `extra_release_capture`

  Refer to "Test_Sequence" on page 34 for an explanation of this attribute. The `Test_Sequence` inherits the `extra_release_capture` attribute from the `Define_Sequence` from which the `Test_Sequence` is derived.

- `osc_running`

  This attribute is used by Encounter Test applications to tell whether there are running oscillators at the end of the sequence definition. This attribute is set automatically under the following conditions:

  - There is a `Start_Osc` event for a pin in the sequence and no `Stop_Osc` event for the same pin in the sequence following the last `Start_Osc` for that pin.
  - All patterns in the sequence contain `Wait_Osc` events. In this case, Encounter Test assumes there is an oscillator running even though the sequence does not contain a `Start_Osc` event.

  Read Vectors automatically sets this attribute, ignoring its setting in the input data.

- `repeat`

  Specify the number of times this test sequence is to be repeated. This attribute is valid only on `test` and `scansequence` type sequences. On a `test` sequence definition, this attribute is used only for LBIST. If the sequence is used to drive stored pattern test generation or WRPT, the `repeat` attribute is ignored. WRPT actually replaces it with its own repeat count.

  This attribute is optional. If not specified, the number of repetitions for each application is ascertained by other means. For a test sequence, the number of repetitions is specified by a user parameter. For a scan sequence, the number of repetitions depends on the length of the scan chains.

- `sequences_have_memory`

  This attribute has meaning only for the "test" type of sequence definition. When the sequence definition is used to drive test generation, any `Test_Procedure` containing `Test_Sequences` derived from this sequence definition will inherit this attribute.

  The presence of this attribute means that this test sequence uses (observes) the starting state of some memory element and that this memory element gets updated during the course of the sequence so that successive `Test_Sequences` derived from this

defined sequence will have results that are dependent upon activity that occurred in a previous iteration of the test.

For a stored-pattern test, you may be able to omit this attribute even though the above-stated condition is true. If this attribute is not specified, then Encounter Test will set all memory elements (except TC and fixed value latches) to X (unknown values) at the beginning of each iteration of the sequence, thereby removing this dependency upon previous iterations. However, there may be cases where setting all the non-fixed value latches to X destroys the effectiveness of the test. If that is the case, then you should specify this attribute. However, this attribute should not be used if the only memory elements to be preserved are LFSRs (for WRPT and LBIST) and OPC latches. Note that OPC latches are defined as those that are observable only through paths that pass through cut points.

- `dummy`

  This attribute is used on `misrreset` sequences if these conditions exist:

  - On-Product MISR test mode
  - The MISR reset clock is also the pipeline clock

  The presence of this attribute indicates that `misrreset` sequences are not applied during a scan event, but instead are explicitly applied within a manipulated test sequence.

- `type`

  The sequence type attributes and their meanings are:

  - `controlpipelinefill`

    This sequence type is added during pattern manipulation for patterns inserted after a scan event that cause the explicit scanning in of the last number of scan bits.

  - `include`

    - This sequence type includes an optional file that the user can specify.

  - `modeinit`

    A sequence that brings the design from an unknown state to the stability state for the test mode. If the test mode has fixed-value latches, the modeinit sequence loads them. For more information, refer to "Requirements of Initialization Sequences" in the *Encounter Test: Guide 2: Testmodes*.

  - `scanop`

The overall scan operation that loads and unloads the scannable latches. This sequence consists of a series of Apply events, which identifies the order in which the scan sections are to be applied. See Appendix A, "Scan Operation" for a diagram.

- `scanentry`

  An optional sequence component of the scan operation that exists primarily in support of 1149.1 scan protocols. It is used for those cases in which 1149.1 TAP scan stored pattern test generation is performed in some state of the TAP controller other than Run-Test/Idle.

  The `scanentry` sequence exists to allow each scan section to start at the same TAP controller state, or at least in compatible states. This would make it possible to support dropping scan sections from the scanop without affecting the viability of the scan sections retained. This section dropping capability is not presently supported, but the `scanentry` sequence is nonetheless defined and supported to forestall a migration problem should the Encounter Test support posture change.

- `scansection`

  A sequence that loads and unloads some subset of the scannable latches. Encounter Test automatically produces only a single scansection sequence. A scansection sequence as created by Encounter Test consists of a series of Apply events that identify the scanprecond, skewunload, scansequence, and skewload sequences that, when applied in succession, accomplish the scan operation. You can also create your own scansection, refer to "Custom Scan Protocols" in the *Encounter Test: Guide 2: Testmodes*.

- `scanexit`

  An optional sequence component of the scan operation that exists primarily in support of 1149.1 scan protocols and only in cases where there is more than one scan section. It exists to return to the state in which test generation is being performed, after all scan sections have been processed.

- `scanprecond`

  A sequence that brings the design from its stability state to the state where this section's scansequence can be applied. This is usually one or two patterns that apply the values to scan gates, output inhibits, and the like.

- `skewunload`

  A sequence that can be appended in front of the scan sequence to perform a skewed unload. For LSSD designs, this consists of a single pattern that pulses all the B shift clocks.

- `scansequence`

The clocking sequence used for loading and unloading all the latches that are scannable in the scan section. Within the sequence are dummy stim and measure events (`Set_Scan_Data` and `Measure_Scan_Data`) that tell when and on which pins to insert or observe the stim latch or measure latch values respectively.

- `misrobserve`

  This sequence establishes the MISR Observe state by setting the `MISR_READ` pins to their value. A `Measure_MISR_Data` event is then included to observe the MISR contents on the `MISR_OBSERVE` pins. Another Stim_PI is then applied if needed to reset the values on the `MISR_READ` pins back to the scan state.

  For additional information, refer to:

  - "On-Product MISR Controls" in the *Encounter Test: Guide 1: Models*.
  - "Measure_MISR_Data" on page 73
  - "Stim_PI" on page 98

  The `misrobserve` sequence exists in the `scanop` after the scansequence and before the skewload and scansectionexit sequences. Refer to Appendix A, "Scan Operation" for details on the scanop structure.

- `misrreset`

  The `misrreset` sequence immediately follows the `MISR_Observe_Sequence`. It establishes the MISR Reset state by setting the `MISR_RESET_ENABLE` pins to their values, then pulses the MISR Reset clock and pulses all the B clocks. Another `Stim_PI` is then applied if needed to reset the values on the `MISR_RESET_ENABLE` pins back to the scan state.

  For additional information, refer to:

  - "On-Product MISR Controls" in the *Encounter Test: Guide 2: Testmodes*.
  - "MISR Reset" design state in the *Encounter Test: Guide 2: Testmodes*.
  - "Stim_PI" on page 98

  The `misrreset` sequence exists in the `scanop` after the scansequence and before the skewload and scansectionexit sequences. Refer to Appendix A, "Scan Operation" for details on the scanop structure.

- `diagobserve`

  This sequence is used to switch the design from the On-Product MISR test mode to the diagnostics test mode where the channel latch data can be scanned out. This

sequence may be empty if there is no difference in the stability state between the On-Product MISR test mode and the diagnostics test mode.

- `diagreturn`

  This sequence is used to switch the design from the diagnostics test mode to the On-Product MISR test mode after a diagnostic measure event. If the `Diagnostics_Observe_Sequence` is empty, this sequence will be empty also.

- `nonscanflush`

  The nonscanflush sequence is used by Encounter Test in two situations. In both cases, the `nonscanflush` sequence is automatically inserted into the test sequence immediately after the scannable latches have been loaded.

  - Generation of pseudo-random tests. The nonscanflush sequence re-initializes the non-scannable memory elements after each scan load, allowing them to have new pseudo-random states for each test, along with the scannable memory elements.
  - Generation of scan-based or weighted random pattern tests with the `nonscanlatch=flush` option.

- `scanlastbit`

  An optional component sequence of the scan operation that exists primarily in support of 1149.1 scan protocols. It scans the last bit of the scan section for which the preceding `scansequence` scanned the first N-1 bits. The reason for its existence and the reason the `scansequence` stops one bit short is to accommodate those cases in which the scanop returns the TAP controller to some state other than Shift-DR. This cannot happen without at some point entering the Exit1-DR state, and going from the Shift-DR state to the Exit1-DR state causes one final shift to occur, hence the name `scanlastbit`.

- `skewload`

  A sequence that can be appended to the end of the scan sequence to perform a skewed load. For LSSD designs, this consists of a single pattern that pulses all the A shift clocks.

- `scansectionexit`

  The `scansectionexit` sequence exists to allow each scan section to end at the same TAP controller state, or at least in compatible states. This would make it possible to support dropping scan sections from the scanop without affecting the viability of the scan sections retained. This section dropping capability is not presently supported, but the `scansectionexit` sequence is nonetheless

defined and supported to forestall a migration problem should the Encounter Test support posture change.

- `loadsuffix`

  A sequence that is optionally applied immediately after the scanop sequence. When applied, the loadsuffix sequence is copied into the test sequence immediately after the scan load event. The loadsuffix sequence is used to move values from the scan shift registers into any stable scan bits attached to them. This mechanism is used, for example, to deal with the UPDATE stage of the 1149.1 TAP controller operation. In LSSD terms, the loadsuffix sequence is used to load L3 latches, which should normally be fed from L1 latches. When scan chains are reported, these are included in the listing correlated with a specific bit position within a controllable scan chain.

- `prpgsave`

  A sequence that copies the state of all on-board PRPGs into their respective shadow registers. This sequence is used when applying tests using Fast Forward.

- `prpgrestore`

  A sequence that copies the state of all on-board PRPG shadow registers into the corresponding PRPGs. This sequence is used when applying tests using Fast Forward.

- `test`

  A sequence that specifies the order of operations (patterns and events) for a `Test_Sequence`. This type of sequence definition may be thought of as a template to be used for some `Test_Sequences`.

- `setup`

  This sequence definition type becomes a setup type `Test_Sequence` when it is used in a Vectors Experiment. Refer to `setup` on page 34.

- `endup`

  Endup type sequences are to be applied at the end of a BIST operation and before reading out the MISR contents. An endup sequence may be necessary for shutting down a phase-locked loop, or moving a BIST controller state machine into some desired end state. It is not required to have an endup sequence if the design has no such special requirements.

- `sigobs`

  A signature observation sequence is used for reading out the results of a BIST test. A typical signature observation sequence consists of switching the design into a

scan mode, and then a `Scan_Unload` event to scan out the MISR (and, for diagnostics, the channel latch) states.

- `channelmaskprecon`

  A channel mask preconditioning sequence defines how to precondition the design to allow loading the mask bits. This sequence is applied on top of the scan state and enables loading of the channel mask register from the `Channel_Mask_Input (CMI)` pins.

- `channelmaskcycle`

  This sequence iterates one cycle of the shift-loading of the channel mask registers.

- `channelmaskexit`

  This sequence returns to the scan state so that scan cycles can begin.

- `channelmaskload`

  This sequence applies the prior three sequences in order.

- `scanfill`

  This sequence contains the number (1 or more) of scan cycles to be explicitly simulated at the end of a scan load. The sequence is copied into all tests created during ATPG immediately following all `scan_load` events and all scan data is backed up in the scan chains n bit positions to account for these extra shift cycles. This update of the ATPG created tests allows simulators to correctly compute values that are loaded into non-scan latches/flops as a result of the clocks applied during the last n cycles of the scan load operation. These additional scan cycles are added to the normal scan depth based on the length of the longest scan chain. This sequence is automatically generated with *n* explicit scan cycles.

  The difference between scanfill and nonscanflush is that the latter is a custom sequence of patterns that are applied after the scan chain has been loaded. The nonscanflush patterns are used to load latches that are not on a scan chain. The scan chain is frozen during this sequence and values from bits within the scan chain are loaded into non-scan latches.

  Keep the following points in consideration while working with scanfill:

  - Introduce a scanfill sequence during: `build_testmode seqdef=xxx seqpath=yyy`

    An example is shown below:

    ```
    [ Define_Sequence xxx   (scanfill) ;
    [ Pattern  (pattern_type=static);
    Event  Stim_PI ():
    ```

```
...any stim PI events ;
] Pattern  ;
[ Pattern  (pattern_type=static);
Event  Set_Scan_Data ()
...List the Scan input pins  ;
Event Pulse () :
...list the clock or clocks to pulse;
] Pattern  ;
];
## Set_Scan_Data () tells Encounter Test when to apply the scan input
values.
```

- Define the number of scan shifts within the scanfill sequence using the SCAN statement with the input modedef file.
- After ATPG:

  If you write out the TDBPatt, you will see that the `Stim_Latch` event placed before the custom scanfill sequence will have the scan chain values offset by the appropriate number of scan shifts. The bits closest to the scan out are "do not care" and should be filled with the values set for these and all the other don't care bits in the vector (these bits are included in the bit bucket anyway after the scanfill is executed).

  The `Stim_Latch` event is followed by the custom user-defined scanfill sequence. The scan data in pins are either explicit ATPG requested values or do not care and thus the latch fill settings determine what gets applied.
- Only limited error checking can be done with a scanfill sequence. Most errors are detected when test patterns fail simulation or on the tester
- The scanfill sequence must first apply any PI stimulus needed to return to the scan state as the normal `scan_load` sequences set the design into the test generation (TG) state if there are any Test Constraint (TC) test function pins defined. This sequence must then also return the design back to the TG state.

- `opcgload`

  This sequence is used to load OPCG registers and applies the following sequences:

  - `opcgprecon`

    This sequence puts the design into the scan state for the OPCG registers based on the assumption that the design starts in the TG state. Note that this loading of OPCG registers is done via a setup sequence that is applied prior to any associated test_sequence and as such the same entry condition as for any test_sequence is assumed, that is, the TG state.
  - `opcgcycle`

This sequence is repeated and shifts the OPCG scan chains 1 bit for each cycle.

- `opcgexit`

  This sequence returns to the TG state of the design after the OPCG registers are loaded.

- `premanipulate_copy`

  It is necessary to save the initial (unmanipulated) latch values for designs with scan control pipelines and `Scan_Load` events rather then `Compact_Scan_Load` events. During manipulation of the test vectors an `Internal_Scan_Load` event containing the initial latch values is created and stored within the manipulated test vectors. The pattern containing this `Internal_Scan_Load` event is marked with a sequence attribute of `premanipulate_copy` making it readily identifiable during simulation.

- Audit attributes

These attributes are set by Encounter Test. Upon Import, all audit information is reset as described below. Audit information is kept for any Force events contained in the sequence and for any pseudo PI events contained in the sequence as well as additional audit information as described below. The audits are carried as binary flags in Vectors:

a. Sequence contains Force events

This flag is set if any Force events are to be found within this sequence definition.

b. Sequence contains pseudo_PI events

This flag is set if any `Stim_PPI`, `Pulse_PPI`, or `Stim_PPI_Clock` events are to be found within this sequence definition, or if this is not a modeinit sequence definition and there are any pseudo PIs defined for this test mode with stability values (TIs, TCs, or clocks). Note that stability pins (and stability pseudo PIs) are always assumed to be at their stability states after the mode initialization sequence has been applied.

c. Verification has been run on this sequence

This flag is set if Verify Sequences was run on this sequence definition. Read Vectors resets this flag.

d. Sequence failed verification

This flag is set if Verify Sequences was run on this sequence definition and some check (unique to the sequence verifier) failed. Read Vectors resets this flag.

e. Force events verified

This flag is set if all Force events were found to be redundant--the forced nets were already at the “forced-to” state and would have remained there throughout the duration of the hold parameter even if they had not been forced. Read Vectors resets this flag.

f. `pseudo_PI` events verified

This flag is set if verification has been run and all pseudo PI events were found to be correct. Read Vectors resets this flag.

g. Sequence contains an oscillator event on a non-clock pin.

This flag is set if any `Start Osc`, `Stop Osc`, or `Wait Osc` is found on a pin that is neither a clock nor oTI.

This audit information is expressed in `TBDpatt` format by the following attribute keywords:

- `Forces_unverified`

  This attribute keyword will appear if flag 1=1, flag 3=0, and flag 5=0.

- `Forces_verified`

  This attribute keyword will appear if flag 5=1.

- `Forces_bad`

  This attribute keyword will appear if flag 1=1, flag 3=1, and flag 5=0.

- `PPI_unverified`

  This attribute keyword will appear if flag 2=1, flag 3=0, and flag 6=0.

- `PPI_verified`

  This attribute keyword will appear if flag 6=1.

- `PPI_bad`

  This attribute keyword will appear if flag 2=1, flag 3=1, and flag 6=0.

- `Failed_verification`

  This attribute keyword will appear if flag 4=1.

- `Invalid_oscillator`

  This attribute keyword will appear if flag 7=1.

- `manipulation_required`

This audit indicates patterns must be manipulated before they can be committed. This audit is set during any application that creates test patterns if the test mode statistics indicate that a non-zero pipeline depth has been specified. Refer to "ASSIGN" in the *Encounter Test: Guide 2: Testmodes* for related information.

- `not_manipulated`

  This audit indicates the patterns have not yet been manipulated and must be manipulated before they can be committed. This audit is set by an application that sets the `manipulation_required` audit. The audit is reset only by using `insert_pipeline_vector_sequence`. Refer to "insert_vector_pipeline_sequence" in the *Encounter Test: Reference: Commands*.

There is no *audit* keyword.

- `user_defined`

  This attribute is present if the associated sequence was defined by the user rather than system generated.

  The writer of a sequence definition may choose to invoke another sequence by means of the Apply event. Sequences invoked by Apply in a manually written sequence do not necessarily have a specified sequence type.

  `Define_Sequence` is a collection of patterns. Only one dynamic pattern is allowed within a `Define_Sequence`. A sequence definition may contain keyed data and `Timing_Data`, `Lineholds`, `SetupSeq`, `EndupSeq`, and `TestSeqType` application objects.

  Following is an example showing the placement of a `Define_Sequence` statement in an Experiment:

```
[ Experiment myExperiment 2 (type = loop, repeat = 256);
   [ Define_Sequence sequence_abc 19951002204813 1.1 (test);
            .
            .
            .
] Define_Sequence sequence_abc 1.1;
[ Test_Section 2.1;
            .
            .
            .
```

  where:

  - `sequence_abc` is the name of the sequence definition
  - `19951002204813` is the date-time stamp of when the sequence definition was created

SeqDef

This includes the name of the sequence definition used by this test sequence for timing information or for LBIST and the date and time that the sequence definition was created. The sequence definition being referred to must exist as an application object on the experiment which this test sequence is in.

A SeqDef object can appear only in a `Test_Sequence`.

SetupSeq

This specifies the name of another sequence definition to be used as the setup sequence in the test procedure when the original sequence definition (the one that contains this `SetupSeq` object) is used as a test sequence. Any initialization requirements needed to run the original sequence definition as a test sequence must be satisfied by either the test mode initialization sequence, the test sequence itself, or the setup sequence (or a combination thereof). If there is no need for a setup sequence, then this object should be omitted.

For example, suppose the sequence named `XYZ_Setup` is required to initialize OPC logic before running a series of tests using the test sequence named `OPCG_Test_Seq_XYZ`. They would be coded as follows for import into the `TBDseq` file. All usages of this pair of sequences refer only to `OPCG_Test_Seq_XYZ`. `XYZ_Setup`, the setup sequence, will be automatically used as needed.

```
[ Define_Sequence OPCG_Test_Seq_XYZ (test);
   [ SetupSeq=XYZ_Setup ];
   [ Pattern ();

    .
    .
    .
] Define_Sequence OPCG_Test_Seq_XYZ;
[ Define_Sequence XYZ_Setup (setup);
   [ Pattern ();

    .
    .
    .
] Define_Sequence XYZ_Setup;
```

EndupSeq

Specify the name of a sequence definition that runs at the end of the BIST process to shut down, or “quiesce” the BIST controller. If there is no special quiescing process for your BIST controller, then this object is omitted.

Example syntax:

```
[ EndupSeq=lbiststop ];
```

TestSeqType

This is a character string that describes the test, used by Manufacturing as a sort of catalogue index in the data base where the tests for a design are stored. The exact character string to use depends upon certain parameters of the test sequence as defined by your manufacturer. Some manufacturers may not require that this object be specified at all.

Example syntax:

```
[ TestSeqType=C2WCC1 ];
```

Lineholds

This is a list of lineholds that are required for the sequence to work properly.

Note: It may also include lineholds that are not required--Encounter Test does not check for the specification of unneeded lineholds.

The purpose of this is to ensure that if lineholds are required to get the sequence to pass the Sequence Verifier properly, then the same lineholds will be used with this test sequence in the test generation process. This is the only vehicle by which lineholds can be specified to the Sequence Verifier.

The syntax of each linehold is `"<object name>"=<value>` where:

- `<object name>` is either a pin name, a net name, a block name, or a PPI name
- `<value>` is `0`, `1`, `X`, `Z`, `W`, `L`, or `H`

Example syntax:

```
[ Lineholds () :
   "Netname_A"=1
   "Netname_B"=0 ;
] Lineholds;
```

Sort_keys

This application object is associated with a Test_Sequence and stores the number of faults detected by the `Test_Sequence`. The syntax of `sort_keys`:

```
[ sort_keys (sone = n ) key= value {key= value... };]
```

Key names represent test objectives and the values represent the number of each test objective detected by the `Test_Sequence` to which the object is attached. `key` can be one of:

- ❑ `null`
- ❑ `static`
- ❑ `dynamic`
- ❑ `iddq`
- ❑ `driver_receiver`
- ❑ `stuck_driver`
- ❑ `dynamic_stuck_driver`
- ❑ `shorted_net`
- ❑ `user` – provides a means for users to sort patterns using their own criteria.

`value` is an integer greater than or equal to zero (0)

- *sone* specifies whether fault detection counts are based on sofe (sone=1), dictionary mode (sone=255), or a value between 1 and 255. SONE stands for *S* top *O* n *N* th Error.

Ignore_Measures

`Ignore_Measures` causes the specified primary outputs or scannable latch states to be set to X in all `Measure_PO` or `Scan_Unload` events within the associated `Test_Procedure`. It also prevents faults from being marked tested at the specified primary outputs and latches. While `Ignore_Measures` causes the specified primary outputs and latches to be masked for observation, it does not affect the states on their associated nets during simulation. `Ignore_Measures` is used for dynamic tests when the path(s) feeding the specified latch or PO are longer than the maximum path length, or there are incomplete or incorrect delays in the logic that feeds it. `Ignore_Measures` may also be used in static tests when there is any reason not to trust the simulated values at a given latch or primary output.

`Ignore_Measures` can appear in a `Define_Sequence`, a `Test_Procedure`, a `Test_Sequence`, or `Timing_Data`.

When importing test sequence definitions, include the object on the `Timing_Data` object, if `Timing_Data` is specified, or on the `Define_Sequence`, if no `Timing_Data` exists. `Test_Procedures` automatically generated by Encounter Test from this `Define_Sequence` will include the `Ignore_Measures` object copied from the `Define_Sequence` or `Timing_Data`. When importing Encounter Test pattern data

containing `Test_Procedure` objects, place the `Ignore_Measures` object directly on the `Test_Procedure`. If also included on an associated `Define_Sequence` or `Timing_Data` object, the `Ignore_Measures` objects should be consistent, although Encounter Test will not make any check for consistency.

There are two types of `Ignore_Measures`:

- `Ignore_Latches`

  This keyword identifies a list of latches that are not to be observed.

- `Ignore_POs`

  This keyword identifies a list of primary outputs that are not to be observed.

These application objects have one attribute: `default_value = 0|1`. This value allows the specification of just the latches or POs which are to be measured, thus reducing the effort required when the majority of latches or POs are to be ignored. The default is 0.

The following is an example of the `Ignore_Measure` construct in TBD:

```
[ Test_Procedure 1.1.1.2 (static_faults = 35919, percent_static_faults =
50.325752);
   [Ignore_Measures;
      Ignore_Latches (default_value=0):
         Block.f.l.cd.nl.cd_bufl_a.sden2_ph2_lt.zph2_lt.scan_lt.sq1.master=1;
      Ignore_POs:
         Pin.f.l.cd.nl.cd_primary_output_pin_5=1;
    ]Ignore_Measures;
.....
```

Notes

1. The `Ignore_Measure` object is valid only on a `Test_Procedure`, `Test_Sequence`, `Define_Sequences`, and as part of the `TimingData`.
2. A “1” means to ignore measures on the latch/PO.
3. `Ignore_Measure` is also supported in vector format.

Application Objects for Test Data Migration

This section describes objects used or created by Test Data Migration applications.

Create Core Tests Application Objects

The following application objects are created or used by Create Core Tests.

- `Macro_Tester_Loop`

  This is used to specify the name of a macro test algorithm to be used for a macro group. It is a reset point for each macro group (that is, a complete test algorithm is started at a tester loop and is not finished until the end of the tester loop). Using a different tester loop for each test algorithm-macro group combination forces them to be independent so that diagnostic data can be collected for each individual group.

  A `Macro_Tester_Loop` statement can appear only in a `Tester_Loop`. Following is an example showing the placement of the `Macro_Tester_Loop` statement in a `Tester_Loop`:

```
[Tester_Loop 1.1.1 ();
  [Macro_Tester_Loop(macro_algorithm=alg1,mic_name=mic1);
  macros_in_group=(65, 140, 296);
  ]Macro_Tester_Loop;
```

- `Macro_Test_Procedure`

  This designates that the containing `Test_Procedure` defines a macro operation, and specifies the name of the operation.

  A `Macro_Test_Procedure` statement can appear only in a `Test_Procedure`. Following is an example showing the placement of the `Macro_Test_Procedure` statement in a `Test_Procedure`:

```
[Test_Procedure 1.1.1.1 ();
  [Macro_Test_Procedure (macro_operation=read);
  ]Macro_Test_Procedure;
```

  Each operation can be further divided into sub-groups in the contained test sequences.

- `Macro_Test_Sequence`

  This is used to define macro sub-groups for an operation. The sequence contains all the patterns necessary to perform the operation for this sub-group.

  A `Macro_Test_Sequence` statement can appear only in a `Test_Sequence`. Following is an example showing the placement of the `Macro_Test_Sequence` statement in a Test_Sequence:

```
[Test_Sequence 1.1.1.1.1 ();
   [ Macro_Test_Sequence;
   macros_in_subgroup=(65, 140);
   ] Macro_Test_Sequence;
```

Application Objects for TDM Vectors

When *Convert Vectors for Core Tests* is invoked, a special kind of Vectors is created, called *TDM* Vectors, which contains no references to internal structure elements of the entity for which test data is being migrated. It is necessary to so restrict the migrateable Vectors

because the Encounter Test model for the higher level structure to which test data is being migrated may not contain the full logic expansion for the lower level entity whose test data is being migrated. This is commonly the case, for instance, when test data is being migrated from a core to an ASIC+core package.

Note: `TDM` Vectors cannot be read into Encounter Test except as an input to Create Core Tests when test data migration is being performed.

There are four application objects present in `TDM` Vectors to make it structure-independent. These application objects are attached to the Vectors handle.

- PI Vector

  The *PI Vector Application Object* gives the correspondence between PI node identifiers and PI pin names. It contains the following for each primary input pin:

  - Primary input pin node ID
  - Primary input pin name
  - Test function pin attributes

  Note: A bi-directional I/O is present in both the *PI Vector Application Object* and the *PO Vector Application Object.*

- PO Vector

  The *PO Vector Application Object* gives the correspondence between PO node identifiers and PO pin names. It contains the following for each primary output pin:

  - Primary output pin node ID
  - Primary output pin name
  - Test function pin attributes

  Note: A bi-directional I/O is present in both the *PI Vector Application Object* and the *PO Vector Application Object.*

- Controllable Register

  A *Controllable Register Application Object* is present for each controllable register defined for the test mode. It contains the following information:

  - Controllable register ID (an integer)
  - Scan-in pin node ID
  - Number of bits

- Name of load section preconditioning sequence
- Name of load sequence

- Observable Scan Chain

  A *Observable Scan ChainApplication Object* is present for each observable scan chain defined for the test mode. It contains the following information:

  - Observable scan chain ID (an integer)
  - Scan-out pin node ID
  - Number of bits
  - Name of unload section preconditioning sequence
  - Name of unload sequence

Pattern_Source Application Object

This object is attached to the experiment level of TBDbin hierarchy and serves to identify the original source of the patterns. The `Pattern_Source` object contains the following information:

- Source ID - the source record ID used to relate this object to a specific `Pattern_Map` object (integer).
- Source Type - the type of source file (integer): STIL (0) or TBDbin (1) or EVCD (2).
- Source File Name - the file name, excluding the path, of the STIL or TBDbin file used as the source of the patterns (string).
- Source File Date - the date of the source file in *`MM/DD/YY`* format (string).
- Source File Time - the time of the source file in *`HH:MM:SS`* format (string).
- Source File Version - the version of the source file (string).

The following is an example of the `Pattern_Souce` object:

```
[ Pattern_Source ( Source_ID = 0,  Type = "TBDbin",  File =
"TBDbin.lssd_static.gsc",  Date = "02/05/04",  Time = "56:21:15",  Version = "0" )
 ] Pattern_Source;
```

Pattern_Map Application Object

This object is attached to each pattern of the TBDbin hierarchy that contains a measure event. and serves to identify the original source of the patterns. The `Pattern_Map` object contains the following information:

- Operation ID - the operation ID that relates this object to a specific `Pin_Map` object (integer).
- Source ID - the source record ID used to relate this object to a to a Pattern_source object (integer).
- Reference - a location reference into the source file (as identified by the Source ID field) which caused the measure contained within the pattern to be created. For STIL source files, this is the STIL statement number. For TBDbin source files, this is the TBD *odometer* value. The type of source file is identified in the referenced Source ID object.

The following is an example of the `Pattern_Map` object:

```
[ Pattern_Map ( Operation_ID = 1,  Source_ID = 0,  Reference ="1.1.1.2.4.1" )
      ] Pattern_Map;
```

Pin_Map Application Object

This object is attached to the experiment level of TBDbin hierarchy and serves to identify the original source of the patterns. The `Pin_Map` object contains the following information:

- Operation ID - the operation ID that relates this object to a specific `Pattern_Map` object (integer).
- Operation Name - the name of the operation (string)
- Instance - the cell instance identifier; the identifier is the hierIndex of the cell.
- SoC Node - the SoC node value; repeated for each cell output pin.
- Pin Name - the pin name associated with the node; repeated for each cell output pin.
- Scan Offset - the Scan_Offset value for the pin; repeated for each cell output pin.
- Scan Length - the length of the scan chain; repeated for each cell output pin.
- Invert - the Invert value for the pin; repeated for each cell output pin.

The following is an example of the `Pin_Map` object:

```
 [ Pin_Map( Operation_ID = 1,  Operation_Name = "IDDQTEST", Instance = 30715, ):
SoC_Node = 22698,  Pin_Name = "DONEOUT", Scan_Offset = 0, Scan_Length = 0,  Invert = no;
SoC_Node = 22649,  Pin_Name = "INITBOUT", Scan_Offset = 0, Scan_Length = 0,  Invert = no;
 ] Pin_Map
```

Target_Faults

This object lists the faults targeted by ATPG when the `Test_Sequence` was created. It lists the faults and the observation point (scan cell or primary output) intended to detect the fault. The syntax is as follows:

```
Fault = fault_index MeasurePoint =  name
```

Example:

```
[ Test_Sequence 2.1.1.2.2 ();
[ Target_Faults;
Fault = 22538 MReg = 2 Bit = 21
Fault = 57122 MReg = 0 PO = 11
Fault = 19425 MReg = 1 Bit = 13
] Target_Faults;
```

Delay Test Application Objects

The following application objects are created and used by Encounter True Time delay test.

Timing_Data

`Timing_Data` is attached to a `Define_Sequence` application object. `Timing_Data` contains the pin timings plus other application objects which were used in the derivation of the pin timings.

The timings represent the points in time that the events in the `Define_Sequence's` dynamic pattern change state during functional simulation or on a tester. You can have multiple `Timing_Data` attributes if each timing is derived under different circumstances (process variations, lineholds, `observe_points`, and delay data).

The following is a list of the Timing_Data attributes and their definitions:

- `timing_data_type`

  This indicates whether the timing data was derived manually or automatically. When `TBDpatt` or `TBDseqpatt` is imported and `Timing_Data` exists, `timing_data_type` is set to manual. This attribute is required.

- `number_of_cycles`

  This indicates how many tester cycles are contained within the pin timings.

- `early`

Specify the best, nominal and worst case for the early mode. See "Delay Timing Concepts" in the *Encounter Test: Guide 5: ATPG* for more information.

- `late`

  Specify the best, nominal and worst case for the late mode. See "Delay Timing Concepts" in the *Encounter Test: Guide 5: ATPG* for more information.

- `delay_file_date_time_stamp`

  If the timings are automatically generated, this defines the instance of the delayModel that was used. This can be used to see if the timings match the delayModel.

- `delay_file_name`

  This specifies the instance name of the delay file used to generate the timings.

- `VDD`

  This specifies the voltage of the primary power supply used when developing the delays used to create the timings.

- `VTT`

  This specifies the voltage of a secondary power supply used when developing the delays used to create the timings.

- `temp`

  This specifies the temperature used when developing the delays used to create the timings. This is the temperature used to test the product.

- `delay_file_audit_string`

  This specifies the audit string from the import of the `delayModel`. This includes the date and time stamp that were used to create the delays.

- `maximum_path_length`

  This specifies the maximum path length that was used when deriving the tests. Any paths over this size were ignored. The primary outputs of the paths that have been ignored are listed in the Ignore_Measures event.

- `minimum_path_length`

  This specifies the minimum path length that was used when deriving the tests. Any paths under this size were ignored. If all paths are smaller than this, then all paths will be considered and this minimum will be ignored.

- `cycles_to_repeat`

This describes whether the dynamic pattern was timed expecting that it would be repeated without the design being allowed to settle between repetitions.

Pin_Timing

This object is attached to the `Timing_Data` object and defines the tester cycles and the times within each tester cycle when specific events are to occur. The events are identified by their relative positions within the dynamic pattern which constitutes the timed portion of the test. The `Pin_Timing` object consists of a list of unique timings. There are two types of unique timing entries:

- tester cycle

  These define the time that each tester cycle starts. The first always starts at time 0 and all other times are specified relative to this point. This entry type includes the following fields:

  - `tester_cycle` - a keyword to indicate that this is a tester cycle entry
  - The time of this tester cycle
  - The cycle number

    For example, `tester_cycle 0.000000 ps cycle 1`.

- individual pin timings

  These define the time that a specific action is to occur on each pin. The action is defined by identifying the event and its type. The pin is identified by name. The timing information includes both the time relative to the beginning of the first cycle and the number of the tester cycle. This entry type includes the following fields:

  - A keyword to identify the TimingType of either:
    - `stim_PIs` (see "Stim_PI" on page 98)
    - `stim_clocks` (see "Stim_Clock" on page 95) *`leading_edge_of_pulse`*
    - `trailing_edge_of_pulse`
    - `PO_strobe`
    - `stim_PPIs` (see "Stim_PPI" on page 100)
    - `stim_PPI_clocks` (see "Stim_PPI_Clock" on page 101)
    - `leading_edge_of_PPI_pulse` (see "Pulse_PPI" on page 79)
    - `trailing_edge_of_PPI_pulse`

- A keyword to identify the NodeType of:
  - `PI` - a pin identified by name, hierarchical index, or flat index depending on the `TBDpatt_Format` `model_entity_form`.
  - `PO` - a pin identified by name, hierarchical index, or flat index depending on the `TBDpatt_Format` `model_entity_form`.
  - `PPI` - a pseudo PI name
  - `cut_point` - a net identified by name, hierarchical index, or flat index depending on the `TBDpatt_Format` `model_entity_form`.

    Timing on cut points can be expressed either on the individual cut point or on the pseudo PI. If pin timings are given for both a cut point and its pseudo PI, the cut point timing takes precedence. Thus, if a pseudo PI represents 20 cut points and 19 of them have the same timing, this can easily be expressed with one pin timing or the pseudo PI and another pin timing for the unique cut point.
- The direction of the signal:
  - `Rising` - 0 to 1 or Z to 1
  - `Falling` - 1 to 0 or Z to 0
  - *RorF* - either direction
  - *to_Z* - 0 to Z or 1 to Z
- The time from the first timed event (e.g., 0 ps).
- The tester cycle that this `Pin_Timing` resides within.
- The number of the event within the dynamic pattern that corresponds with this `Pin_Timing`.
- The node id. Interpretation of this depends on the `NodeType` field.

PI or PO

A pin identified by name, hierarchical index, or a flat index depending on the `TBDpatt_Format` `model_entity_form`. See "TBDpatt_Format" on page 23

PPI

A pseudo PI name.

cut_point

A net identified by name, hierarchical index, or flat index depending on the `TBDpatt_Format` model_entity_form. See "TBDpatt_Format" on page 23

Timing on cut pints can be expressed either on the individual cut point or on the pseudo PI. If pin timings are given for both a cut point and its pseudo PI, the cut point timing takes precedence. Thus, if a pseudo PI represents 20 cut points and 19 of them have the same timing, this can easily be expressed with one pin timing for the pseudo PI and another pin timing for the unique cut point.

Figure 2-1 shows a timing diagram used to create the example `Pin_Timing` object of Figure 2-2.

Figure 2-1 Timings of a Dynamic Pattern

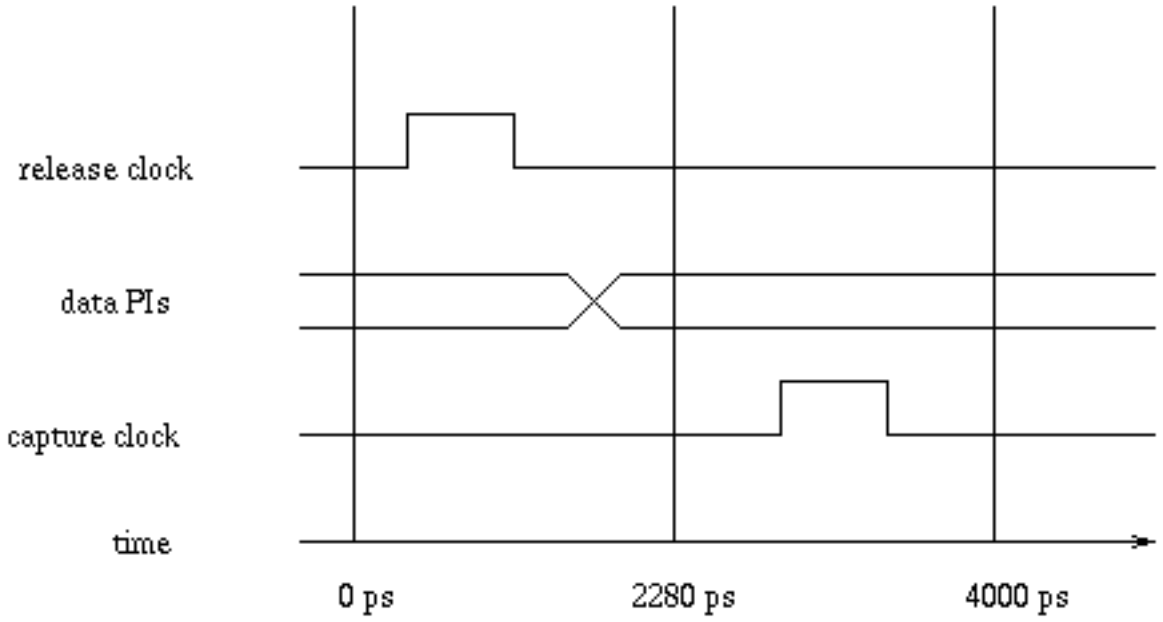


Figure 2-2 A Sample Timing for Dynamic Patterns

```
  [ Pin_Timing:
   tester_cycle 0.000000 ps cycle 1;
   leading_edge_of_pulse Rising 200.000000 ps cycle 1 event 1 "release_clock" IN0";
   trailing_edge_of_pulse Falling 600.000000 ps cycle 1 event 1 "release_clock"IN0";
   stim_PIs RorF 950.000000 ps cycle 1 event 2 "Pin.f.l.omniTest.nl.PIN0";
   stim_PIs RorF 950.000000 ps cycle 1 event 2 "Pin.f.l.omniTest.nl.PIN1";
   stim_PIs RorF 950.000000 ps cycle 1 event 2 "Pin.f.l.omniTest.nl.PIN2";
   stim_PIs RorF 950.000000 ps cycle 1 event 2 "Pin.f.l.omniTest.nl.input1";
   tester_cycle 2280.000000 ps cycle 2;
   leading_edge_of_pulse Rising 2400.000000 ps cycle 2 event 3 "capture_clock" IN0";
   trailing_edge_of_pulse Falling 2800.000000 ps cycle 2 event 3 "capture_clockIN0";
   tester_cycle 4000.000000 ps cycle 3;
  ] Pin_Timing;
 ] Timing_Data 1.1.1;
 [ Pattern 1.1.1 (pattern_type = static);
   Event 1.1.1.1 Stim_PI_Plus_Random ():
 ] Pattern 1.1.1;
 [ Pattern 1.1.2 (pattern_type = dynamic);
   Event 1.1.2.1 Pulse (timed_type=release):
"release_clock"= +;
   Event 1.1.2.2 Stim_PI (timed_type=release);
   ;       # (The sequence definition is a template, so individual stim
           # values do not necessarily appear here.)
   Event 1.1.2.3 Pulse (timed_type = capture):
"capture_clock"= +;
 ] Pattern 1.1.2;
 [ Pattern 1.1.3 (pattern_type = static);
   Event 1.1.3.1  Scan_Unload ():
 ] Pattern 1.1.3;
```

Timing_Lineholds

This object is attached to the `Timing_Data` object and is one of the application objects used in the derivation of the pin timings. Timing derivation is controlled by user entered or automatically generated static lineholds. The `Timing_Lineholds` are values that are guaranteed not to change during the release portion of the sequence and so can be used to eliminate logic that can not affect the tests. Only static lineholds are allowed to be `Timing_Lineholds`. The set of `Timing_Lineholds` can contain both automatically and manually generated lineholds.

Observe_Points

This object is attached to the `Timing_Data` object and is one of the application objects used in the derivation of the pin timings. This is a list of hierpins. When `Observe_Points` is present, only those POs and memory elements were used in the generation of timings.

Setup_Patterns

This object is attached to the `Timing_Data` object and is one of the application objects used in the derivation of the pin timings. These patterns are used as an auxiliary way to setup the state of the design before timings are calculated.

The `Setup_Patterns` are patterns that represent how the design is initialized by the sequences that are portions of the test sequence being timed.

Timing_ID

This object is attached to a test sequence and contains the index of the timing data used by this test sequence for timing information. This index refers to the relative order of the timing data within the `Define_Sequence`. For example, the following `Test_Sequence` refers to the second set of `Timing_Data` within the `Define_Sequence` named "`Atest`" and dated "19950330104313".

```
[ Test_Sequence 2 (type = loop, repeat = 256);
  [ SeqDef=(Atest,"19950330104313") ] SeqDef;
  [ Timing_ID = 2 ];
  [ Pattern 1 (pattern_type = static);
```

Keyed Data

Keyed_Data provides a means for passing information through the test pattern data file that Encounter Test applications may or may not recognize. Keyed data can be added to any TBD object in the hierarchy from experiment to event and to a `define_sequence`. The keyed data is placed immediately after the begin statement of the block to which it is attached.

Keyed data consists of character string pairs. In each pair, the two character strings are separated by an = sign. Each character string should be enclosed in double quotes (`"`). The string to the left of the equal sign is referred to as the key and the character string to the right is the data. Both the key and the data can include any characters, including blanks, except the newline (carriage return) character. For example, keyed data is coded as follows:

```
[Test_Sequence ();
 [Keyed_Data;
  "Keynumber1"="This keyed data is attached to the test sequence"
 ]Keyed_Data;
 [Pattern (pattern_type=static);
 [Keyed_Data;
  "Patkey" = "This is keyed data attached to a pattern"
 ]Keyed_Data;
 Event 1 Pulse():
 "SystemClock"=+;
 ]Pattern;
]Test_Sequence;
```

Multiple comments can be embedded in Keyed_Data that can be subsequently be viewed in the TBD output when using General Purpose or High Speed Scan Based Simulation. The keyword `logMsg` produces these comments when used per the following example:

```
[Keyed_Data;
 "logMsg" = "this comment will appear in the log"
 "logMsg" = " These patterns were written by John Smith on March 5 2002"
] Keyed_Data;
```

For General Purpose Simulation, `logMsg` may be used on TBD entities down to, and including `Test_Procedure`. A `logMsg` comment on a `Test_Sequence` or a Pattern is ignored. Specifying `msglevel` lower than *2* inhibits the printing of log messages.

For High Speed Scan Based Simulation, `logMsg` may be used down to, and including the Pattern entity. Specifying `quiet=yes` inhibits the printing of log messages.

Note: Keyed_Data objects for `TG=IGNORE`, `TG=IGNORE_FIRST`, and `TG=IGNORE_LAST` are only recognized when found on the pattern level of the hierarchy of user sequences of type test. Refer to TG=Keyed Data in the *Encounter Test: Guide 5: ATPG* for more information.

Simulation Options

Encounter Test simulators stores the specified simulation options as `Keyed_Data` in each Test Section. Use either of the following methods to produce a report:

- Via GUI, click *Report - Vector Simulation Options* to display the **Report Vector Simulations** window. Refer to "Report Vector Simulation Options" in the *Encounter Test: Reference: GUI*.
- Via commands, use `report_vector_simulation_options`. Refer to "report_vector_simulation_options" in the Encounter Test: Reference: Commands.

An example of stored simulation options `Keyed_Data` in a `TBDpatt` file follows:

```
[ Experiment ttclogic 1 ();
 [ Test_Section 1.1 (tester_termination = 0,  termination_domination = product,
         test_section_type = logic, test_type = static,
         simulated);
   [ Keyed_Data;
    "SimOptions" = "simulate=hsscan(12Jan04) choppers=safe contentionmessage=all
    contentionremove=yes contentionreport=soft infinitex=default
    keepers=safe latchfill2=repeat latchfill=random latchfilldynamic=100.00
    latchfillstatic=100.00 ... xmask=no zmemory=no"
   ] Keyed_Data;
```

Summary Information

A `TBDpatt` file (written by Encounter Test) contains a summary including the number of occurrences of each level of the hierarchy.

```
# experiments = n
# test sections = n
# tester loops = n
# test procedures = n
# test sequences = n
# patterns = n
# events = n
```

Adjusting Default Event Timing

By default Encounter Test creates specific timing templates for stuck-at fault test patterns. This section describes the default event order and provides you with information on how to adjust the timing:

test_cycle timeplate - defines the order of events and the timing for functional capture. The normal order of events is

1. Apply input values
2. Pulse the functional capture clock
3. Strobe any expected output values

   Use the following write_vectors parameters to adjust the default timing:

   ```
   testperiod=<int>  ## Adjust the tester cycle 80ns = default
   testpioffset=<int>  ## Adjust when PI's are set 0ns =default
   testbidioffset=<int> ## Adjust when bidi's are set  0ns=default
   teststrobeoffset=<int>  Adjust when to strobe  72ns=default
   testpioffsetlist=<pin name]=<int>[,<pin name>=<int>]  ## The other entries
   adjust the timing of all pins in a specific group. This allows you to control
   the timing of a specific pin. Many times this is used to ensure that the clocks
   are fired in a specific sequence.
   ```

scan_cycle timeplate - Defines the timing and order of events for the scan chain shift operation. The normal order of events is:

1. Strobe outputs
2. Set inputs
3. Pulse the scan shift clocks

   You can adjust the timing of inputs, clocks and strobe times with write_vectors parameters.

   ```
   scanperiod=<int>    ## Adjust the tester cycle 80ns = default
   scanpioffset=<int>  ## Adjust when PI's are set. 16ns =default
   scanbidioffset=<int> ## Adjust when bidi's are set. 16ns = default
   scanstrobeoffset=<int>  ## Adjust when to strobe. 0ns = default
   ```

```
scanpioffsetlist=<pin name]=<int>[,<pin name>=<int>] ## The other entries
adjust the timing of all pins in a specific group. This allows you to control
the timing of a specific pin.
```

Defining Equation-based Timings

Specify `write_vectors equationbasedtimings=yes` to enable Encounter Test define the timing information in variables for WGL, STIL, and Verilog patterns. You can then use these defined variables in the file to change the timing information during pattern translation into the ATE format.

When defining equation-based timing, in addition to variables to support limited timeplates and dynamic sequences, Encounter Test also defines a scaling variable, `Scaling_percent`, which allows you to scale all timing variables. The default value for this variable is `100` or `100%`, which means that no timing variables will be scaled.

Note: This scaling variable does not adjust any timings specified to zero.

Encounter Test works with the following restrictions while defining equation-based timings:

- Equation-based timings is not implemented for a variable with the value zero. This is because a variable is not supported in the first position of the timeplate, so it must be set to `0ns`.
- Encounter Test does not support two events at the same time, which includes a `0ns` followed by a variable set to `0`. The `write_vectors` command will continue to produce equation based timings if a `0` is encountered but the PI or PO set to `0` will not be equation based and therefore cannot be modified through a variable.
- The number of characters allowed on the right side of any equation is limited to 147 including blanks. Therefore, if *`<pinName>`* exceeds 45 characters, `write_vectors` will replace the value with the PI or PO entry number instead of the pin name.

Refer to the following for sample equation-based settings and corresponding timeplates for WGL, STIL, and Verilog formats:

- "Equation-based Timing for WGL Patterns" on page 149
- "Equation-based Timing for STIL Patterns" on page 165
- "Equation-based Timing for Verilog Patterns" on page 188

OPCG Test Pattern Application Sequence

TBDseqpatt data is used by designs that require a specialized pattern application sequence for:

- Entering or exiting scan shift mode that is normally done by toggling the scan enable signal
- Issuing a double clock pulse for launch and capture for delay test patterns
- Designs in general that have on-chip clock generation logic, that is, where hardware tester cannot issue a clock pulse to an input pin.

The following example illustrates a special sequence to get OPCG logic to issue the true-time delay test launch and capture clocks.

```
create_logic_delay_tests testseq=<TBDSeqpatt file name>
testsequence=opcg_sys_capture testmode=FULLSCAN_TIMED
```

The following is a sample TBDseqpatt sequence:

Note: In the example:

- `mode=node` defines the reference to node and pin names
- The `Define_Sequence` name is referenced through the `testsequence` parameter. Several different capture clock test sequences may be defined within `TBDSeqPatt`.

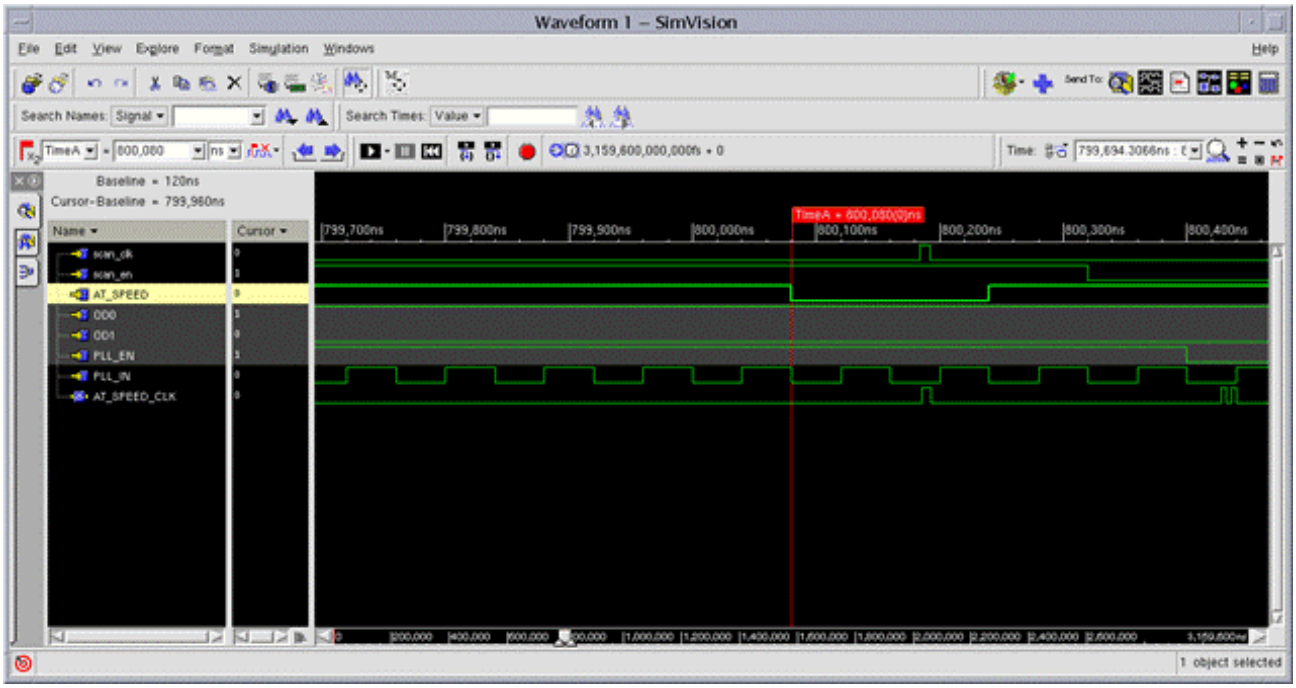
```
TBDpatt_Format(mode=node,model_entity_form=name);
[ Define_Sequence opcg_sys_capture  (test);
# This is the scan load event.
# In this example there are no scan preconditioning sequences
#   required to get the device into scan shift mode. The
#   control pins defined in the pin assignment file are sufficient.
#   If one were required you would see it here.
# Within the input pin assignment file, you will find scan enable, scan data in and
# out pins, and other pins that control the device
  [ Pattern (pattern_type = static );
    Event Scan_Load():;
  ] Pattern;
# In this next section, we wait for the scan enable signal to settle.
# TG=IGNORE is very important. It tells Encounter Test to NOT
# attempt to align/insert the ATPG pattern here.
  [ Pattern (pattern_type = static );
    [Keyed_Data;  TG=IGNORE ] Keyed_Data ;
    Event Wait_Osc (cycles=10): "PLL_IN";
] Pattern;

# Now that we've done the scan load and have set up our input pattern
# we now go through the sequence of events that will get the OPCG logic
# to spit out a launch and capture clock. PLL_EN is the signal that
# controls this operation. When we set it to logic 1 (remember that
# this signal was assigned "+GO" the OPCG logic will issue our clocks.
# Stim_PI_Plus_Random tells Encounter Test to apply an explicit
# value to PLL_EN and to apply the ATPG pattern specific values
# to all of the other pins. Do not use TG=IGNORE here.
  [ Pattern (pattern_type = static );
    Event Stim_PI_Plus_Random(): "PLL_EN"=1;
  ] Pattern;
# The Wait_Osc command synchronizes other events with the
# tester supplied input reference clock.
```

```
# Stim_PI starts the state machine by lowering the PLL_EN signal.
  [ Pattern (pattern_type = static );
  [Keyed_Data;  TG=IGNORE ] Keyed_Data ;
  Event Wait_Osc (cycles=0): "PLL_IN";
    Event Stim_PI(): "PLL_EN"=0;
  ] Pattern;
# This tells Encounter Test that, after you have exercised the
# pattern sequence above then the OPCG logic will issue a
# launch/release and capture pulse at the PPI (cutpoint)
# NOTE: This is a "trust me" situation. There is no way
#   Encounter Test (or any other ATPG tool) can really
#   verify that a double clock pulse was issued OR what
#   the timing of these pulses is.
  [ Pattern  (pattern_type = dynamic) ;
    Event Pulse_PPI(timed_type=release):  "PLL_CLK"=+ ;
    Event Pulse_PPI(timed_type=capture):  "PLL_CLK"=+ ;
  ] Pattern;
# This Wait_Osc event defines how many PLL_IN
# clocks to issue before continuing on to
# the next sequence of events.
# For our design these 10 cycles allow the state machine
# within the OPCG logic to reset.
  [ Pattern (pattern_type = static );
    [Keyed_Data;  TG=IGNORE ] Keyed_Data ;
    Event Wait_Osc (cycles=10,off): "PLL_IN";
    Event Stim_PI(): "PLL_EN"=1;
  ] Pattern;
# This is a standard scan unload event.
# No special scan event sequences are required for our design.
  [ Pattern (pattern_type = static );
    Event Scan_Unload():;
] Pattern;
] Define_Sequence sys_capture  ;
```

The following figure represents a scan shift when `scan_en=1` and `AT_SPEED=0`. Note the following:

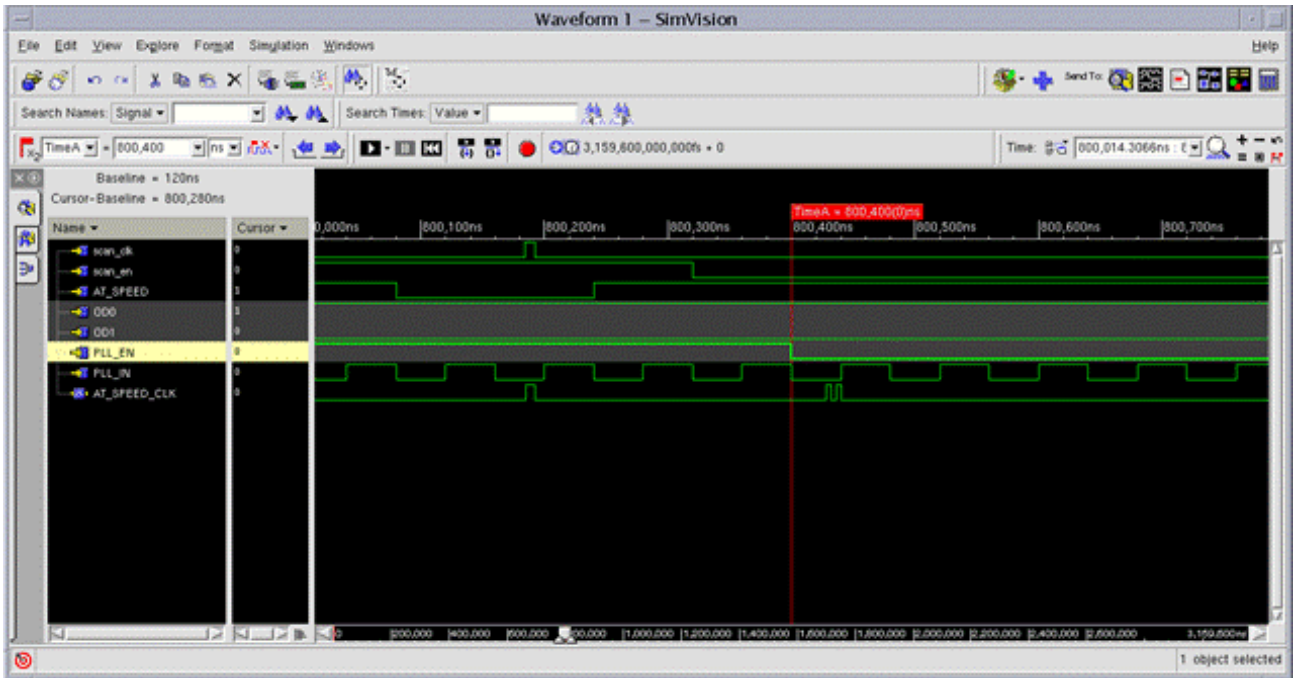
Figure 2-3 OPCG Scan Shift



Note: This is a broadside load Verilog simulation and, therefore, there is only one scan shift clock (scan_clk) to load the data. No additional application sequence is required to enter or exit scan shift mode. All necessary data for this was provided in the pin assignment file.

The following figure shows a special sequence to have OPCG issue at-speed delay test clocks. For the sample design, the OPCG logic will issue the launch and capture pulse when PLL_EN goes to 0. Note that `AT_SPEED=1` and this selects the OPCG output as the clock source.

Figure 2-4 Sequence with OPCG Issuing At-speed Delay Test Clocks



WGL Pattern Data Format

An Overview to WGL Pattern Data Format

Encounter Test can export test vectors in TSSI** Waveform Generation Language (WGL). The following discussion assumes a basic knowledge of WGL. For more information on the WGL language and specifications, contact TSSI (Test Systems Strategies Inc. - www.tessi.com).

Encounter Test creates one or more WGL “vector” files and optionally, a WGL “signals” file for each testmode. A testmode typically has one vector file containing scan string confidence patterns (refer to create_scanchain_tests in the *Encounter Test: Reference: Commands* for more information) and one or more vector files to test the functional logic.

Vector files contain WGL which represents the actual test data and any WGL constructs particular to that test data. The optional signals file contains WGL constructs that are “common” to multiple vector files. If the signals file is not created, then these constructs are contained in each vector file.

The test data is divided into multiple vector files in order to reduce the overall processing time of a single, large WGL file. You can choose one of these methods for creating the files using either the graphical interface or command line:

- Create WGL data file from uncommitted tests for a specific testmode. Any test patterns that you create are placed into an experiment. Specify the name of the experiment while executing the create_tests command:

  ```
  write_vectors language=wgl testmode=FULLSCAN INexperiment=scan_chain_test
  <options>
  ```

- After test generation, commit your test patterns using the commit_tests command and then create the WGL data file. When you commit a set of tests, the experiment name is eliminated and the patterns become part of the master pattern set. To write these patterns, you only need to reference the test mode.

  ```
  write_vectors  language=wgl testmode=FULLSCAN <options>
  ```

where in each case:

testmode is the test mode name

inexperiment is the name of the experimental pattern set from your create_tests run

Commentary is included in each vector file which correlates its contents to the source vectors with regard to experiment, test section, tester loops, test procedures, test sequences, and patterns.

The following test section types are supported:

Flush	Scan	Channel_Scan
Logic	Macro	Logic_LBIST
Driver and Receiver	IOWRAP Stuck Driver	ICT Stuck Driver
IEEE 1149.1 Integrity	IOWRAP Shorted Nets	ICT Shorted Nets

A WGL file has one of the following naming conventions:

- WGL.*<testmode name>*.signals
- WGL.*<testmode name>*.*<pattern description>*

By default, the WGL data is stored in the following directory:

<your_workdir>/testresults/wgl

A complete set of example WGL files is shown in Appendix C, “WGL Pattern Data Examples”.

Basic File Structure and Content of WGL Signals File

A WGL signals file starts with comment data. Comments begin with ## and end at a carriage return. The comment information contains useful information such as clock values, scan chain affiliation, and scan control but is not processed by WGL translation software.

Comment data includes the following:

- Encounter Test version used to create the file
- Date and time of file creation
- Part entity, variation, and iteration names (project is not specified)
- Applicable testmode name
- Source Experiment and Test Section identification
- Selected input parameters and their values
- Applicable tester termination information

The next section of the file contains legal WGL syntax defining the signal list for the design. This list includes all the ports on the design. The order of the signals in the list is important because their relative order is implied within the WGL vectors. The following is an example of the WGL syntax:

```
signal
   "CME0" : input; ## information about this input
   "DATA[0]" :bidir; ## information on this bidi
   "A[31]"  : output;  ## information on this output
   "SYS_CLK" : input; ## information about this input
end
```

The next section is the scancell section that lists the FF in each scan chain. The following is an example of the scancell section:

```
scancell
   "TOP.Level1.Level2.REGA.DFF";
   "TOP.Level1.Level2.REGB.DFF";
   "TOP.Level1.Level2b.REGA.DFF";
end
```

Following the scancell section is the scanchain section defining all the scan chains within the design. Each chain has a unique name and is referenced by the test vectors by its name.

Each scan chain is defined from Scan Input pin to Scan Output pin and lists every FF in the chain from Scan In to Scan Out. The FF instance names are the names defined in the above-mentioned scancell section. The following is an example of the scanchain section:

```
scanchain
   "MREG_1_FULLSCAN"  [
   "SCAN_IN_1",
   "TOP.Level1.Level2.REGA.DFF",
   "TOP.Level1.Level2.REGB.DFF",
   "TOP.Level1.Level2b.REGA.DFF,
   SCAN_OUT_1"
      ];
   "MREG_2_FULLSCAN"  [
   "SCAN_IN_2",
   "TOP.Level1.Level2.REGC.DFF",
   "TOP.Level1.Level2.REGD.DFF",
   !, "TOP.Level1.Level2b.REGE.DFF,
   SCAN_OUT_2"
      ];
end
```

Note: The `!` represents an inversion on the scan chain between REGD and REGE.

Basic File Structure and Contents of WGL Vector Files

A WGL vector file is divided into five sections: header, timing definitions, structure definition,, scanstate, and actual test vectors. The following section discusses each section in detail.

Header

The header section starts with comment data providing an overview of the WGL test list. The comment data contains the following:

- Encounter Test version used to create the file
- Date and time of file creation
- Part entity, variation, and iteration names (project is not specified)
- Applicable testmode name
- Source Experiment and Test Section identification
- Selected input parameters and their values

- Applicable tester termination information

Next in the header section is the waveform statement that points to the signals file. The following is a sample waveform statement:

```
waveform "WGL.lssd.logic.ex2.ts1"
        include "WGL.lssd.signals";
```

The first line of the statement contains the name of the vectors file and the `include` statement imbeds the common signals file defined in the previous section. If you do not specify the `include` statement, then these constructs are placed in-line in each vector file. (`write_vectors signalsfile=no`)

Note: You need only one signals file for each testmode. Every vector file created for a specific testmode will reference this signals file.

Timing Definitions

The next section is the timing definitions section that defines one or more WGL timeplates depending on the requirements of the test data. The number of timing timeplates depends on the number of event sequences required to apply the test patterns. These timeplates define the time when, within a tester cycle, a value should be applied to a specific pin. If the pin is an output, the timeplate defines when the pin should be observed within the tester cycle. All times are relative to the starting of a tester cycle.

Refer to Adjusting Default Event Timing on page 139 for information on default event timing.

The following examples show the default timing and event order that Encounter Test will assign to test modes. Refer to Default Timings for Clocks in *Encounter Test: Guide 6: Test Vectors* for more information.

```
timeplate "scan_cycle" period 80 ns
   "CME0"    := input [ 0ns:P, 16.00ns:S ];
   "DATA[0]" := input [ 0ns:P, 16.00ns:S ];
   "SYS_CLK" := input [ 0ns:P, 24.00ns:S, 32.00ns:U ];
   "A[31]"   := output[ 0.0ns:Q'edge ];
   "DATA[0]" := output[ 0.0ns:Q'edge ];
end
timeplate "test_cycle_FULLSCAN" period 80 ns
   "CME0"    := input [ 0.0ns:S ];
   "DATA[0]" := input [ 0.0ns:S ];
   "SYS_CLK" := input [ 0ns:P, 8.00ns:S, 16.00ns:U ];
   "A[31]"   := output[ 0.0ns:X, 72.00:Q'edge ];
   "DATA[0]" := output[ 0.0ns:X, 72.00:Q'edge ];
end
```

In the above example:

- The entry within the double quotes ("") is the pin name.

- The symbol `:=` is followed by the pin direction, `input` or `output`. If the pin is a BIDI then it will appear twice
- The symbol (such as `D`, `N`, `U`) that follows the time specification occurs at that time.

The different symbols (also known as state characters) and their definitions are as follows:

- `D` - Force logic low
- `U` - Force logic high
- `N` - Force logic unknown
- `Z` - Force logic high impedance
- `S` - Force logic substituted from pattern
- `C` - Force complement of substituted shape
- `P` - Force logic using previous format shape
- `L` - Compare logic low
- `H` - Compare logic high

The timeplates are used to modulate the signals by providing signal timings and shapes. These are referenced by vector and scan constructs within the pattern block. Both the scan_cycle and test_cycle timeplates define an `8ns` wide pulse for the input clock pin SYS_CLK. However, the scan_cycle strobes the `A` and `DATA` pins at time `0` within the tester cycle while the test_cycle strobes these pins at time 72ns within the tester cycle.

Equation-based Timing for WGL Patterns

If you specify `write_vectors equationbasedtimings=yes,` Encounter Test defines the timing information in variables, which you can use to modify the timing information in the WGL file.

A sample equation-based timing settings for WGL is given below:

```
##*************************************************************************##
##                          EQUATION BASED TIMINGS                         ##
##*************************************************************************##

    equationsheet specifications
       exprset default
          scaling_percent := 100 ;
          scaling_factor  := scaling_percent / 100 ;
          testpioffset     := 0.000000ns * scaling_factor ;
          testbidioffset   := 0.000000ns * scaling_factor ;
          teststrobeoffset := 72.000000ns * scaling_factor ;
          testperiod       := 80.000000ns * scaling_factor ;
```

```
         testpulsewidth   := 8.000000ns * scaling_factor ;
         test_A := 8.000000ns * scaling_factor ;
         test_B := 24.000000ns * scaling_factor ;
         test_C := 8.000000ns * scaling_factor ;
         scan_length := 16 ;
      end
   end

   equationsheet equations
      exprset clocksoff
         testpioffset_end := testpioffset + testpulsewidth ;
         scanpioffset_end := scanpioffset + scanpulsewidth ;
         testperiod_flush       := testperiod * scan_length ;
         teststrobeoffset_flush := teststrobeoffset * scan_length ;
         test_A_end := test_A + testpulsewidth ;
         test_B_end := test_B + testpulsewidth ;
         test_C_end := test_C + testpulsewidth ;
      end
   end

   equationdefaults
      specifications:default, equations:clocksoff;
   end
```

The timing definitions in the timeplate with sample equation-based timings will be as follows:

```
##************************************************************************##
##                            TIMING DEFINITIONS                          ##
##************************************************************************##

   timeplate "test_cycle_stimclks" period testperiod_flush
     "A" := input  [ 0ns:P, test_A:S ];
     "B" := input  [ 0ns:P, test_B:S ];
     "C" := input  [ 0ns:P, test_C:S ];
     "CS" := input  [ 0ns:S ];
     "DI1" := input  [ 0ns:S ];
     "DI2" := input  [ 0ns:S ];
     "DI3" := input  [ 0ns:S ];
.
.
.
     "DO4" := output [ 0ns:X, teststrobeoffset_flush:Q'edge ];
     "SO1" := output [ 0ns:X, teststrobeoffset_flush:Q'edge ];
     "SO1_BIDI" := output [ 0ns:X, teststrobeoffset_flush:Q'edge ];
     "SO2" := output [ 0ns:X, teststrobeoffset_flush:Q'edge ];
     "SO2_BIDI" := output [ 0ns:X, teststrobeoffset_flush:Q'edge ];
   end
```

scanstate Construct

Encounter Test defines a `scanstate` construct for each scan event in the vectors. Each scanstate pattern starts with a label, which is SS followed by the ATPG pattern number and ends with a semi-colon. The label is referenced in the WGL `scan vector` statement. The `scanstate` section also contains the scanchain followed by the set of logic values.

```
scanstate
```

```
"SS.2.1.1.2.1.1.1" :=
   "MREG_1_FULLSCAN" (010010011)
   "MREG_2_FULLSCAN" (0111000100000001);
   "SS.2.1.1.2.1.4.1" :=
   "MREG_1_FULLSCAN" (010010011)
   "MREG_2_FULLSCAN" (0111000100000001);
   "SS.2.1.1.2.2.1.1" :=
   "MREG_1_FULLSCAN" (110100110)
   "MREG_2_FULLSCAN" (0110001000000010);
...
"SS.2.1.1.5.10.5.1" :=
   "MREG_1_FULLSCAN" (110001011)
   "MREG_2_FULLSCAN" (1111110001001110);
end
```

Each `SS` scanstate entry will reference one or more of the scan chains that exist within the design. Every scan chain that is referenced is defined within the WGL `scanchain` construct. Some points to consider:

- The logic values will be for both input stimulus and expected output values. The WGL Vector will reference the scan chains within each `SS` scanstate entry through an `input` or `output` keyword.
- A logic value will be assigned for every FF in the scan chain that is referenced. There will be neither any implied logic values to complete the scan chain reference nor any fill logic values for the shorter scan chains to match the length of the longest scan chain.
- The logic value is the value to be applied at the Scan input pin or measured at the scan output pin. Translation software does not need to account for inversion internal to any scan chain.
- Any specific `SS` scanstate entry may reference a subset of the total number of scan chains. Different modes of operation such as loading the MASK scanchain for compression logic need to reference only the subset of scan chains that are actually being exercised.

Test Vectors

Encounter Test creates a pattern block construct containing vector and scan constructs that represent the actual test data events applied to the design.

The first entry is the `pattern` statement that defines the pins and the pin order that are referenced within the vector and scan pattern statements.

```
write_vectors  usesignalsgroup=NO  (default = yes)
```

The statement specifies that Encounter Test should not list all the design pins individually in the pattern statement but instead use the predefined WGL constructs ALLINPUT,

ALLOUTPUT, and ALLBIDIR. The implied pin order is the order in which these pins appear in the signals definition.

The pattern statement is followed by vector and scan statements, which are the actual test patterns. Each statement first refers to a timing timeplate. The first set of logic values are for the design pins. The order is defined in the pattern statement. Encounter Test places the logic values for inputs, bidirectionals as inputs, outputs, and bidirectionals as outputs on separate lines.

The scan statement includes the input and output values for the scan chains using a scanstate reference.

The Test Vectors section ends with comment data within the patterns. The most important information in the comment data is the EVENT. It is recommended that users reference the EVENT pattern number for diagnostic purposes to identify the failing pattern.

ATE vendors and other companies that process WGL format are required to capture the pattern number on the EVENT and report it as part of the CHIP-PAD-PATTERN format imported into Encounter Test for failure analysis.

Note: The EVENT data is not part of the legal WGL syntax and is not required for the normal processing of the WGL test patterns.

The following is a sample test vectors section.

As you analyze this example please note the following:

- `pattern MAIN` defines the inputs followed by bidirectionals as inputs, followed by the outputs, and then bidirectional pins as outputs.
- In the vector and scan statements, the stimulus and observe values for the pins follow the same order with each pin group on its own line.
- In the vector and scan statements, there is a reference to `test_cycle` or `scan_cycle`. These apply the event order and timing to each pin.
- The scan statement includes input and/or output statements that reference a scan chain that has already been defined and apply input or observe values to that scan chain through the scanstate entries that were defined.

```
##*******************************************************##
##                        TEST VECTORS                   ##
##*******************************************************##
pattern  MAIN ("CME0", "SYS_CLK",  <other inputs>,
   "DATA[0]":I, <other bidirectional pins as input>,
   "A[31],  <other output pins>,
   "DATA[0]":O, <other bidirectional pins as output> )
vector ( +, "test_cycle" ) := [
```

```
   000XXXXX111X0XX01
   ZZ
   XXXXXX
   --];
## Inserted the Scan Sequence:  Scan_Sequence
scan   ( +, "scan_cycle" ) := [
   110XXXXX111X0--01
   ZZ
   XXXXXX
--],
   input  [ "MREG_1_FULLSCAN" :  "SS.2.1.1.2.1.1.1" ] ,
   input  [ "MREG_2_FULLSCAN" :  "SS.2.1.1.2.1.1.1" ] ;
## Processing the Static: EVENT 2.1.1.2.1.2.1:  Stim_PI:
## Processing the Static: EVENT 2.1.1.2.1.3.1:  Stim_PI:
## Processing the Static: EVENT 2.1.1.2.1.4.1:  Measure_PO:
vector ( +, "test_cycle" ) := [
   000101110000011110
   11
   000000
   --];
##*******************************************************##
##  TEST SEQUENCE.............2   TYPE..........normal  ##
##*******************************************************##
## Processing the Static: EVENT 2.1.1.2.2.1.1:  Scan_Load:
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
vector ( +, "test_cycle" ) := [
   000101111110011101
   ZZ
   XXXXXX
   --];
vector ( +, "test_cycle" ) := [
   000101111110011101
   ZZ
   XXXXXX
   --];
## Inserted the Scan Sequence:  Scan_Sequence
scan   ( +, "scan_cycle" ) := [
   1101011111100--01
   ZZ
   XXXXXX
   --],
   input  [ "MREG_1_FULLSCAN" :  "SS.2.1.1.2.2.1.1" ] ,
   input  [ "MREG_2_FULLSCAN" :  "SS.2.1.1.2.2.1.1" ] ;
## Processing the Static: EVENT 2.1.1.2.2.2.1:  Stim_PI:
## Processing the Static: EVENT 2.1.1.2.2.3.1:  Stim_PI:
## Processing the Static: EVENT 2.1.1.2.2.4.1:  Measure_PO:
vector ( +, "test_cycle" ) := [
   00001100100011010
   --
   011100
   00];
##*******************************************************##
##  TEST SEQUENCE.............14  TYPE........normal   ##
##*******************************************************##
## Processing the Static: EVENT 2.1.1.8.14.1.1:  Scan_Load:  ( Overlap is in Effect)
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
vector ( +, "test_cycle" ) := [
   000010011111101101
```

```
   ZZ
   XXXXXX
   --];
vector ( +, "test_cycle" ) := [
   0000100111110101
   ZZ
   XXXXXX
   --];
## Inserted the Scan Sequence:  Skewed_Unload_Sequence
vector ( +, "test_cycle" ) := [
   0100100111110101
   ZZ
   XXXXXX
   --];
## Inserted the Scan Sequence:  Scan_Sequence
scan   ( +, "scan_cycle" ) := [
   1100100111110--01
   --
   XXXXXX
   --],
   input  [ "MREG_1_FULLSCAN" :  "SS.2.1.1.8.14.1.1" ] ,
   input  [ "MREG_2_FULLSCAN" :  "SS.2.1.1.8.14.1.1" ] ,
   output [ "MREG_1_FULLSCAN" :  "SS.2.1.1.8.13.4.1" ] ,
   output [ "MREG_2_FULLSCAN" :  "SS.2.1.1.8.13.4.1" ] ;
## Processing the Static: EVENT 2.1.1.8.14.2.1:  Stim_PI:
## Processing the Static: EVENT 2.1.1.8.14.2.2:  Measure_PO:
vector ( +, "test_cycle" ) := [
   0000110000001101 0
   --
   000000
   00];
## Processing the Static: EVENT 2.1.1.8.14.3.1:  Pulse:
vector ( +, "test_cycle" ) := [
   0010110000001101 0
   ZZ
   XXXXXX
   --];
...
## Inserted final non-scan Pattern
vector ( +, "test_cycle" ) := [
   0000110011100XX01
   ZZ
   XXXXXX
   --];
end end
```

Encounter Test performs data compression on PI and PO events wherever possible to minimize the number of WGL vectors generated from the source vectors. PI and PO events are collected into a single WGL vector regardless of pattern boundaries, unless the user option to respect pattern boundaries is used, as long as their event order is consistent with the defined WGL timeplate order. When an event which violates this order is encountered, such as a PI stim after a PO measure, the WGL vector is written and the compression begins for the next vector. Therefore, it is possible to compress several vectors into a single WGL vector. Scan events are not compressed.

Scan statements relate scan values to scan-in and scan-out signals through an indirect process. Each statement relates one or more scan chains defined by the scanchain construct

to cell values defined by the scanstate construct. The scanstate values for a chain are in the same order as the cells in the scanchain construct. The scanchain construct also identifies the scan-in and scan-out signals and any cell to cell inversion. Thus, WGL processors can map the scan chain cell values to their respective bits in the scanstate and can then map the bits to the respective scan-in and scan-out signals. Yon must take the scan chain inversions into account during this process.

WGL File Structure Variations

The following sections discuss variations on the basic WGL file structure described above.

Overlapped/Non-Overlapped Scans

The default is to overlap scan-outs with scan-ins. This overlap reduces the number of cycles required for scanning of test data into and out of the scan chains. However, overlap is not always possible. For instance, overlap cannot be done between tester loops since tester loops are intended to be independent. In addition, overlap cannot be done where there are intervening events which must take place between the scan-out and the subsequent scan-in. Therefore, for those cases where overlap is not possible a non-overlapped scan sequence is produced even though the user has requested scan overlap.

Optionally, the user may request a non-overlap scan style, whether or not overlap is possible. By not overlapping scans the number of tester cycles required for scan effectively doubles. This is not recommended because it increases the time to test the product and thus increases the cost of test. Use the following command to produce a non-overlapped scan:

```
write_vectors scanoverlap=NO
```

Macro Test Types

Support for Macro tests is very similar to Logic tests with the following exceptions:

- Macro tests typically contain loops to perform a repetitive application of clocks or to increment macro counters or address values, etc. Therefore, the pattern loop and end loop vectors events must be represented in the WGL.
- Macro tests may also contain pulses on non-clock PIs due to selecting a non-clock PI correspondence point for a macro clock.

The above macro test considerations are reflected in the test vector section by using the WGL `loop` and `end` statements to represent loop events and by changing pulse events on non-clock PIs into a sequence of two vectors - the first to turn the non-clock PI to its on state and

the second one to return it to its off state. If the non-clock PI is not originally off prior to the pulse, a third vector is added to establish this state.

```
   ...
## processed EVENT: 1.1.1.3.1.1.1
## processed EVENT: 1.1.1.3.1.1.2
   vector ( +, "task_cycle_lssd" ) := [
      00110001X0000XXX0
      ZZ
      XXXXXX
      00];
      vector ( +, "task_cycle_lssd" ) := [
      00110001X0000XXX1
      ZZ
      XXXXXX
      00];
   ...
```

The above example shows how a negative-active pulse on a non-clock PI is treated in the WGL using two consecutive vectors, the first is a stim to 0 and the second is a stim to 1.

Note: Encounter Test supplies logic values to design pins in the following order:

- Inputs
- Bidrectional as inputs
- Outputs
- Bidirectionals as outputs

LBIST Test Types

Support for LBIST tests involves consideration of the following LBIST test concepts:

- Scanning in the LBIST mode is somewhat different than normal scanning. The vectors will contain channel_scan events in lieu of the deterministic scan events. These channel_scan events also use attributes, as opposed to unique scan event types, to differentiate between normal and skewed scans. Therefore, unique LBIST "channel scan" sequences must be supported.
- The core of the LBIST test sequence involves cycling product LBIST clocks. These clocks cycle the on-product PRPGs and MISRs and move test values along the LBIST channels. Therefore, looping on the LBIST test sequence is required.
- LBIST test results are represented by "signatures." These are the values that remain in the PRPGs and MISRs after the LBIST test sequence has completed. Signatures are represented as hexadecimal values by signature events. These must be converted to parent mode values and compared to the product values by a scan-out after the final

sequence cycle has completed. Note, only the final signatures for each LBIST test sequence will be compared (see below).

These considerations are reflected in the test vector section by using WGL `loop` and `end` statements and by adding the necessary sequences to return to the parent mode and perform a scan-out.

The following sample code explains the use of LBIST test types:

```
##*****************************************************##
##  TEST SEQUENCE...........2  TYPE...............loop ##
##*****************************************************##


loop 256
## processed EVENT: 1.2.1.2.2.1.1
## processed EVENT: 1.2.1.2.2.2.1
## inserted  SCAN SEQUENCE: Scan_Preconditioning_Sequence
## inserted  Stability
   vector ( +, "task_cycle_lbist" ) := [
        00110000011000010
        ZZ
        XXXXXX
        --];
## inserted  SCAN SEQUENCE: Skewed_Unload_Sequence (Skewed Unload)
   vector ( +, "task_cycle_lbist" ) := [
        01010000011000010
        ZZ
        XXXXXX
        --];
loop 4
   vector ( +, "task_cycle_lbist" ) := [
        11010000011000010
        ZZ
        XXXXXX
        --];
end
end
   vector ( +, "task_cycle_lssd" ) := [
        00010000011000010
        ZZ
        XXXXXX
        --];
## processed EVENT: 1.2.1.2.2.3.2
## inserted  SCAN SEQUENCE: Scan_Preconditioning_Sequence
## inserted  SCAN SEQUENCE: Scan_Sequence (Scan)
   vector ( +, "task_cycle_lssd" ) := [
        00010000111000001
        ZZ
        XXXXXX
        --];
   scan  ( +, "scan_cycle" ) := [
        11010000111000001
        --
        XXXXXX
        --],
```

```
        output [ "MREG_1_FULLSCAN" : "SS.1.2.1.2.2.3.2" ],
        output [ "MREG_2_FULLSCAN" : "SS.1.2.1.2.2.3.2" ];
```

In the example above, the inner loop represents a `channel_scan` event that cycles the product clocks to increment the on-product PRPGs and MISRs and to move test values along the LBIST channels. The outer loop performs N cycles of the LBIST test sequence. After applying the N LBIST tester cycles, the following vector statements show how the parent scan state is entered and the scan statement with the output scanchain values show how a scan-out is performed to unload the MISR values.

Timed Test Types

Support for timed tests involves consideration of the following:

- Timing Data objects exist within the vectors file which specify explicit timings for test period, strobe offset, and primary signal I/Os.
- Each timing data object applies a "dynamic" pattern within one or more test sequences.
- One or more test cycles may be required for each dynamic pattern.

The above considerations are reflected in the WGL by defining a unique timeplate for each timing data object and then referencing the appropriate timeplate for each dynamic pattern found within the test data. The following is an example of a dynamic timeplate.

Note: The actual structure and syntax of the at-speed timing templates is similar to the templates for static test. The only difference is the timing accuracy and different event orders such as two clock pulses in the same tester cycle.

```
timeplate "TBautoLogicSeq8_7_0" period 12500.0000ps
       "A" := input[ 0ns:D, 1250.0000ps:S, 1750.0000ps:D ];
       "B" := input[ 0ns:D, 2000.0000ps:S, 2500.0000ps:D ];
       "C" := input[ 0ns:D, 250.0000ps:S, 750.000ps:D ];
       "CS" := input[ 0ns:P ];
       ...

       "DO3" := output[ 0ns:X, 10000.00ps:Q, 1250.00ps:X ];
       "DO4" := output[ 0ns:X, 10000.00ps:Q, 1250.00ps:X ];
       "SO1" := output[ 0ns:X, 10000.00ps:Q, 1250.00ps:X ];
       "B1" := output[ 0ns:X, 10000.00ps:Q, 1250.00ps:X ];
       "SO2" := output[ 0ns:X, 10000.00ps:Q, 1250.00ps:X ];
       "B2" := output[ 0ns:X, 10000.00ps:Q, 1250.00ps:X ];
end
       ...
```

The timeplate has a unique name that is constructed from its sequence name and the timing ID appended with a cycle number. Each timeplate includes signal waveform descriptions, which contain the calculated times from the timing data objects. When a test sequence

containing a dynamic pattern is encountered in vectors, the converter produces a vector statement that references the appropriate timeplate as shown below:

```
...
     vector ( +, "TBautoLogicSeq8_7_0" ) := [
          -1---------------
          --
          XXXXXX
          --];
          ...
```

If there are multiple cycles, multiple vector statements are produced, each referencing the respective timeplate. User-specified timings are used for untimed static patterns.

STIL Pattern Data Format

Encounter Test can export test vectors in Standard Test Interface Language (STIL), based on IEEE Standard 1450-1999 format. The following discussion assumes a basic knowledge scan based test patterns and the STIL language. For a formal description of the STIL language and its syntax please see the IEEE Standard 1450-1999 language specification.

Encounter Test creates one or more STIL “vector” files and optionally, a STIL “signals” file for each testmode. Vector files contain STIL which represents the actual test data and any STIL constructs particular to that test data. The optional signals file contains STIL constructs that are “common” to multiple vector files. If the signals file is not created, then these constructs are contained in each vector file.

The test data is divided into multiple vector files in order to reduce the overall processing time of a single, large STIL file. You can choose one of these methods for creating the files using either the graphical interface or command line:

- Create a STIL file from committed tests for each testsection found within each experiment in the source vectors file. Optionally, a STIL file for each tester loop within each test section within each experiment can be created. The file name is of the form:

  `STIL.`*`testmode.testsectiontype`*`.ex#.ts#[.tl#]`

- Create a STIL file from experimental tests for each testsection found within each experiment in the source vectors file. Optionally, a STIL file for each tester loop within each test section within each experiment can be created. The file name is of the form:

  `STIL.`*`testmode.experiment.testsectiontype`*`.ex#.ts#[.tl#]`

where in each case:

testmode is the test mode name

experiment is the name of the experiment

testsectiontype is the test section type (e.g. scan)

ex# is the experiment number

ts# is the test section number

tl# is the test loop number (optional)

Commentary is included in each vector file which correlates its contents to the source vectors with regard to experiment, test section, tester loops, test procedures, test sequences, and patterns.

The following test section types are supported:

- Flush
- Logic
- Driver and Receiver
- IEEE 1149.1 Integrity
- Scan
- Macro
- IOWRAP Stuck Driver
- IOWRAP Shorted Nets
- Channel_Scan
- Logic_LBIST
- ICT Stuck Driver
- ICT Shorted Nets

A complete set of example STIL files is shown in Appendix D, “STIL Pattern Data Examples”.

Basic STIL File Structure

Encounter Test limits the set of STIL constructs used in order to maximize tester compatibility. Only the following STIL constructs are used:

- Comments (text beginning with `//` and ending at a carriage return or text beginning with `/*` and ending in `*/`). Comments can be ignored when generating a tester program.
- The `include` statement, if the option to generate a common signals file is used during export.
- `Signals`, `SignalGroups`, `MacroDefs`, `Timing`, `PatternBurst`, `PatternExec`, and `Pattern` block constructs.
- `WaveformTable`, `Vector`, and `Shift` constructs within the `MacroDefs` block.
- `WaveformTable` and `Waveforms` constructs within the `Timing` block.
- `Macro` constructs within the `Pattern` block.

Conceptually, each vector file can be divided into five sections: a header section, a signals definition section, a macro definition section, a timing section, and the actual test vectors. The following discusses each section in more detail.

Header

A header section begins each STIL file, an example of which is shown below.

```
STIL 1.0
//************************************************************************//
//                            STIL VECTOR FILE                            //
//   Encounter(TM) Test Encounter Test 2.0.0 Sep 30, 2003 (aix43_64 TDA20)   //
//************************************************************************//
//                                                                        //
//  FILE CREATED.............July 31, 2001 at 12:58:06                   //
//                                                                        //
//  PROJECT NAME.............lbc                                         //
//                                                                        //
//  TESTMODE.................lssd                                        //
//                                                                        //
//  TDR......................dummy_tester_lssd                           //
//                                                                        //
//  TEST PERIOD..............80          TEST STROBE TYPE..........window  //
//  TEST PULSE WIDTH.........8           TEST TIME UNITS...........ns      //
//  TEST PI OFFSET...........0                                            //
//  TEST BIDI OFFSET.........0                                            //
//  TEST STROBE OFFSET.......72          X VALUE...................Z       //
//                                                                        //
//  SCAN PERIOD..............80          SCAN STROBE TYPE..........window  //
//  SCAN PULSE WIDTH.........8           SCAN TIME UNITS...........ns      //
//  SCAN PI OFFSET...........16                                           //
//  SCAN BIDI OFFSET.........16                                           //
//  SCAN STROBE OFFSET.......0           SCAN OVERLAP..............yes     //
//                                                                        //
//  EXPERIMENT...............2           DATA FORMAT...............binary  //
//                                                                        //
//  TEST SECTION.............1           TEST SECTION TYPE.........logic   //
//  TESTER TERMINATION.......0           TERMINATION DOMINATION....tester  //
//                                                                        //
//************************************************************************//
Include "STIL.lssd.signals";
```

The following information is included in the header section:

- The Encounter Test version used to create the file
- Date and time of file creation
- Part entity, variation, and iteration names (project is not specified)
- Applicable testmode name
- Source Experiment and Test Section identification
- Selected input parameters and their values

- Applicable tester termination information

The `include` statement imbeds the common `signals` file which contains the `Signals`, `SignalGroups`, and `MacroDefs` constructs common to all vector files. If the `include` statement is not present, then these constructs are placed in-line in each vector file.

Structure

This data is normally contained within the common “signals” file since it is applicable to all vector files. However, when the “signals” file is not generated, this data is contained within each vector file. See the examples below.

Encounter Test uses the `Signals` construct to define a STIL signals for each PI and PO of the design.

```
//*************************************************************************//
//                              DEFINE SIGNALS                             //
//*************************************************************************//

Signals {
   "A" In;
   "B" In;
   "C" In;
   "CS" In;

   ...

   "DO3" Out;
   "DO4" Out;
   "SO1" Out;
   "SO1_BIDI" InOut;
   "SO2" Out;
   "SO2_BIDI" InOut;
}  /* end Signals */
```

Encounter Test uses the `SignalGroups` construct to group I/O signals into meaningful collections which are then referenced, as needed, in the `MacroDefs` and `Pattern` blocks. Groups are defined for all PIs, POs, BIDIs, all clocks, all scan-inputs and scan-outputs, and all scan chains.

```
//*************************************************************************//
//                          DEFINE SIGNAL GROUPS                           //
//*************************************************************************//

SignalGroups {
   ALLPIs = '"A"+"B"+"C"+"CS"+"DI1"+"DI2"+"DI3"+"DI4"+"ENABLE1"+"ENABLE2"
            +"ME"+"PS"+"SEL"+"SI1"+"SI2"+"ST1"+"ST2"';
   ALLPOs = '"DO1"+"DO2"+"DO3"+"DO4"+"SO1"+"SO2"';

   ALLIOs = '"SO1_BIDI"+"SO2_BIDI"';

   ALLACs_lssd = '"A"';
```

```
   ALLBCs_lssd = '"B"';

   ALLSIs_lssd = '"SI1"+"SI2"';
   ALLSOs_lssd = '"SO1_BIDI"+"SO2_BIDI"';

   SI0001_lssd = '"SI1"' { ScanIn 9; }
   SI0002_lssd = '"SI2"' { ScanIn 16; }

   SO0001_lssd = '"SO1_BIDI"' { ScanOut 9; }
   SO0002_lssd = '"SO2_BIDI"' { ScanOut 16; }
}  /* end SignalGroups  */
```

The `ScanStructures` construct is used to list the scan chain definitions in a STIL signals file. Use the Write Vectors option to *Include scan chain definitions* (`includescanregs=yes` for the command line) to produce scan chain information.

```
//*************************************************************************//
//                            DEFINE SCAN CHAINS                           //
//*************************************************************************//

ScanStructures {
        ScanChain "Control_Observe_Reg_1_nobs" {
        ScanLength 24;
        ScanIn "BSI";
        ScanCells
        " rcvr1.slave"
        " rcvr5.slave"
        " rcvr6.slave"
        " rcvr7.slave"
             .
             .
             .
        " drvrcv4.slave3"
        " drv1.slave"
        " drv2.slave" ;
        ScanOut "BSO"; }

        ScanChain "Control_Observe_Reg_2_nobs" {
        ScanLength 9;
        ScanIn "SI1";
        ScanCells
        " fvsrl.slave"
        " prpg.srl1.slave"
        " prpg.srl2.slave"
        " prpg.srl3.slave"
        " prpg.srl4.slave"
        " misr.srl1.slave"
        " misr.srl2.slave"
        " misr.srl3.slave"
        " misr.srl4.slave" ;
        ScanOut "SO1"; }

        ScanChain "Control_Observe_Reg_3_nobs" {
        ScanLength 16;
        ScanIn "SI2";
        ScanCells
        " channel1.srl1.slave"
        " channel1.srl2.slave"
        " channel1.srl3.slave"
 " channel1.srl4.slave"
```

```
            .
            .
            .
        " channel3.srl4.slave"
        " channel4.srl1.slave"
        " channel4.srl2.slave"
        " channel4.srl3.slave"
        " channel4.srl4.slave" ;
        ScanOut "SO2"; }
}
```

The `MacroDefs` construct is used to defined several macros which are used to apply the actual parallel or scan cycle data. The `TEST` macro is used within the `Pattern` block whenever a parallel cycle is required. The `Vector` construct references the signal groups: `ALLPIs, ALLPOs`, and if applicable, `ALLIOs`.

Likewise, the `SCAN_lssd` macro is used within the `Pattern` block whenever a scan cycle is required. The `Shift` and `Vector` constructs reference the signal groups: `ALLSOs_lssd`, `ALLSIs_lssd`, and any applicable clock groups, e.g., `ALLACs_lssd` and `ALLBCs_lssd`.

```
//*************************************************************************//
//                              DEFINE MACROS                              //
//*************************************************************************//
MacroDefs {
   TEST { WaveformTable test_cycle;
     Vector {
         ALLPIs = %;
         ALLPOs = %;
         ALLIOs = %;
     }  /* end Vector  */
   }  /* end TEST  */

   SCAN_lssd { WaveformTable scan_cycle;
     Condition {
         ALLSIs_lssd = 00;
         ALLPOs = XXXXXX;
     }  /* end Condition */
     Shift { Vector {
         ALLSOs_lssd = #;
         ALLSIs_lssd = #;
         ALLACs_lssd = P;
         ALLBCs_lssd = P;
         }  /* end Vector  */
     }  /* end Shift  */
   }  /* end SCAN_lssd  */

}  /* end MacroDefs  */
```

Timing

Encounter Test defines one or more STIL `WaveformTables`, depending on the requirements of the test data. Examples are shown below.

```
Timing {
  WaveformTable test_cycle { Period '80ns';
     Waveforms {
         "A" { 01ZP { '0ns' P/P/P/D; '8ns' D/U/Z/U; '16ns' D/U/Z/D; } }
         "B" { 01ZP { '0ns' P/P/P/D; '24ns' D/U/Z/U; '32ns' D/U/Z/D; } }
         "C" { 01ZP { '0ns' P/P/P/D; '8ns' D/U/Z/U; '16ns' D/U/Z/D; } }
         "CS" { 01Z { '0ns' D/U/Z; } }

         ...

         "DO3" { LHTX { '0ns' X; '72ns' l/h/t/x; '80ns' X; } }
         "DO4" { LHTX { '0ns' X; '72ns' l/h/t/x; '80ns' X; } }
         "SO1" { LHTX { '0ns' X; '72ns' l/h/t/x; '80ns' X; } }
         "SO1_BIDI" { LHTX { '0ns' X; '72ns' l/h/t/x; '80ns' X; } }
         "SO2" { LHTX { '0ns' X; '72ns' l/h/t/x; '80ns' X; } }
         "SO2_BIDI" { LHTX { '0ns' X; '72ns' l/h/t/x; '80ns' X; } }
      }  /* end Waveforms  */
   }  /* end WaveformTable  */

   WaveformTable scan_cycle_lssd { Period '80ns';
      Waveforms {
         "A" { 01ZP { '0ns' P/P/P/D; '24ns' D/U/Z/U; '32ns' D/U/Z/D; } }
         "B" { 01ZP { '0ns' P/P/P/D; '40ns' D/U/Z/U; '48ns' D/U/Z/D; } }
         "C" { 01ZP { '0ns' P/P/P/D; '16ns' D/U/Z/U; '24ns' D/U/Z/D; } }
         "CS" { 01Z { '0ns' P; '16ns' D/U/Z; } }

         ...

         "DO3" { LHTX { '0ns' l/h/t/x; '8ns' X; } }
         "DO4" { LHTX { '0ns' l/h/t/x; '8ns' X; } }
         "SO1" { LHTX { '0ns' l/h/t/x; '8ns' X; } }
         "SO1_BIDI" { LHTX { '0ns' l/h/t/x; '8ns' X; } }
         "SO2" { LHTX { '0ns' l/h/t/x; '8ns' X; } }
         "SO2_BIDI" { LHTX { '0ns' l/h/t/x; '8ns' X; } }
      }  /* end Waveforms  */
   }  /* end WaveformTable  */
}  /* end Timing  */
```

The `WaveformTables` are used to "modulate" the signals, i.e., provide signal timings and shapes. They are referenced by the `TEST` and `SCAN_lssd` macros within the `MacroDefs` block. The `test_cycle WaveformTable` is used for all test (parallel) cycles. The `scan_cycle_lssd WaveformTable` is used for all scan cycles.

Refer to "Default Timings for Clocks" in the *Encounter Test: Guide 6: Test Vectors* for related information.

Equation-based Timing for STIL Patterns

If you specify `write_vectors equationbasedtimings=yes,` Encounter Test defines the timing information in variables, which you can use to modify the timing information in the STIL file.

```
//*************************************************************************//
//                          EQUATION BASED TIMINGS                         //
//*************************************************************************//
```

```
Spec equation_based_spec {
  Category EQUATION_BASED {
    "scaling_percent" = '100' ;
    "scaling_factor"  = '"scaling_percent"/100' ;
    "testpioffset"     = '0.000000ns*"scaling_factor"' ;
    "testbidioffset"   = '0.000000ns*"scaling_factor"' ;
    "teststrobeoffset" = '72.000000ns*"scaling_factor"' ;
    "testperiod"       = '80.000000ns*"scaling_factor"' ;
    "testpulsewidth"   = '8.000000ns*"scaling_factor"' ;
    "test_A" = '8.000000ns*"scaling_factor"' ;
    "test_B" = '24.000000ns*"scaling_factor"' ;
    "test_C" = '8.000000ns*"scaling_factor"' ;

    "scanstrobeoffset" = '0.000000ns*"scaling_factor"' ;
    "scanpioffset"     = '16.000000ns*"scaling_factor"' ;
    "scanbidioffset"   = '16.000000ns*"scaling_factor"' ;
    "scanperiod"       = '80.000000ns*"scaling_factor"' ;
    "scanpulsewidth"   = '8.000000ns*"scaling_factor"' ;
    "scan_A" = '24.000000ns*"scaling_factor"' ;
    "scan_B" = '40.000000ns*"scaling_factor"' ;
    "testpioffset_end" = '"testpioffset"+"testpulsewidth"' ;
    "scanpioffset_end" = '"scanpioffset"+"scanpulsewidth"' ;
    "test_A_end" = '"test_A"+"testpulsewidth"' ;
    "scan_A_end" = '"scan_A"+"scanpulsewidth"' ;
    "test_B_end" = '"test_B"+"testpulsewidth"' ;
    "scan_B_end" = '"scan_B"+"scanpulsewidth"' ;
    "test_C_end" = '"test_C"+"testpulsewidth"' ;
} }
```

The timing definitions in the timeplate with sample equation-based timings will be as follows:

```
//************************************************************************//
//                          TIMING DEFINITIONS                            //
//************************************************************************//

  Timing {
    WaveformTable "test_cycle" { Period '"testperiod"';
      Waveforms {
      "A" { 01ZP { '0ns' P/P/P/P; '"test_A"' D/U/Z/U; '"test_A_end"' D/U/Z/D; } }
      "B" { 01ZP { '0ns' P/P/P/P; '"test_B"' D/U/Z/U; '"test_B_end"' D/U/Z/D; } }
      "C" { 01ZP { '0ns' P/P/P/P; '"test_C"' D/U/Z/U; '"test_C_end"' D/U/Z/D; } }
      "CS" { 01Z  { '"testpioffset"' D/U/Z; } }
      "DI1" { 01Z  { '"testpioffset"' D/U/Z; } }
      .
      .
      .
      "DO4" { LHTX { '0ns' X/X/X/X; '"teststrobeoffset"' L/H/T/X; } }
      "SO1" { LHTX { '0ns' X/X/X/X; '"teststrobeoffset"' L/H/T/X; } }
      "SO1_BIDI" { LHTX { '0ns' X/X/X/X; '"teststrobeoffset"' L/H/T/X; } }
      "SO2" { LHTX { '0ns' X/X/X/X; '"teststrobeoffset"' L/H/T/X; } }
      "SO2_BIDI" { LHTX { '0ns' X/X/X/X; '"teststrobeoffset"' L/H/T/X; } }
  } }
```

Test Vectors

Encounter Test creates a *Pattern* block construct containing *Macro* constructs which invoke either the *TEST* or *SCAN_lssd* macros which, in turn, apply the test vector values to the product I/Os. See the example below.

```
//*************************************************************************//
//                              TEST VECTORS                               //
//*************************************************************************//

PatternBurst
   MAIN_BRST { PatList { MAIN_TEST; }
         Termination { ALLPOs TerminateHigh; ALLBOs TerminateHigh; } }

PatternExec
   MAIN_EXEC { PatternBurst MAIN_BRST; }

Pattern
   MAIN_TEST {

//*************************************************************************//
//  TESTER LOOP...............1        PROCEDURES HAVE MEMORY....no         //
//  TEST PROCEDURE............1        TYPE......................init       //
//  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no         //
//  TEST SEQUENCE.............1        TYPE......................init       //
/*************************************************************************//

// processed EVENT: 2.1.1.1.1.1.1
// processed EVENT: 2.1.1.1.1.2.1

//*************************************************************************//
//  TEST PROCEDURE............2           TYPE......................normal   //
//  SLOW TO TURN OFF..........false       SEQUENCES HAVE MEMORY.....no       //
//  STATIC FAULTS.............17          PERCENT STATIC FAULTS.....58.75000 //
//  TEST SEQUENCE.............1           TYPE......................normal   //
//*************************************************************************//

// processed EVENT: 2.1.1.2.1.1.1
// inserted  SCAN SEQUENCE: Scan_Preconditioning_Sequence (Scan Preconditioning)
   Macro TEST {
      ALLPIs = 000XXXXXXXXXXXXX;
      ALLPOs = XXXXXX;
      ALLIOs = ZZ; }
// inserted  SCAN SEQUENCE:; Scan_Sequence (Scan)
   Macro TEST {
      ALLPIs = 000XXXXX111X0XX01;
      ALLPOs = XXXXXX;
      ALLIOs = ZZ; }
   Macro SCAN_lssd {
      SI0001_lssd = 000110010;
      SI0002_lssd = 1010000111010100; }
// processed EVENT: 2.1.1.2.1.2.1
// processed EVENT: 2.1.1.2.1.3.1
// processed EVENT: 2.1.1.2.1.4.1

//*************************************************************************//
//  TEST SEQUENCE.............2           TYPE......................normal   //
//*************************************************************************//
```

```
// processed EVENT: 2.1.1.2.2.1.1
// inserted  SCAN SEQUENCE: Scan_Preconditioning_Sequence (Scan Preconditioning)
   Macro TEST {
      ALLPIs = 00010000100111110;
      ALLPOs = LLHLLL;
      ALLIOs = LL; }
// inserted  SCAN_lssd SEQUENCE: Scan_Sequence (Scan)
   Macro TEST {
      ALLPIs = 00010000111101101;
      ALLPOs = XXXXXX;
      ALLIOs = XX; }
   Macro SCAN_lssd {
      SI0001_lssd = 100100101;
      SI0002_lssd = 0011110010110111; }
// processed EVENT: 2.1.1.2.2.2.1
// processed EVENT: 2.1.1.2.2.3.1
// processed EVENT: 2.1.1.2.2.4.1
   Macro TEST {
      ALLPIs = 00010110000010110;
      ALLIOs = 11;
      ALLPOs = LLLLLL; }

     ...

//*************************************************************************//
//  TEST SEQUENCE............10          TYPE.......................normal   //
//*************************************************************************//

// inserted  SCAN SEQUENCE: Scan_Preconditioning_Sequence (Scan Preconditioning)
   Macro TEST {
      ALLPIs = P0010111001010110;
      ALLPOs = XXXXXX; }
      ALLIOs = ZZ;
// inserted  SCAN SEQUENCE: Skewed_Unload_Sequence (Skewed Unload)
// inserted  SCAN SEQUENCE: Scan_Sequence (Scan)
// processed EVENT: 2.1.1.5.10.1.1
   Macro TEST {
      ALLPIs = 0P010111111000101;
      ALLPOs = XXXXXX; }
      ALLIOs = ZZ;
   Macro SCAN_lssd {
      SO0001_lssd = HLLHHLHHH;
      SO0002_lssd = LHLLHLHHLHHLHLHH;
      SI0001_lssd = 100011011;
      SI0002_lssd = 1110001000100000; }
// processed EVENT: 2.1.1.5.10.2.1
// processed EVENT: 2.1.1.5.10.2.2
   Macro TEST {
      ALLPIs = 00010110001010010;
      ALLPOs = LLLLLL; }
      ALLIOs = ZZ;
// processed EVENT: 2.1.1.5.10.3.1
// processed EVENT: 2.1.1.5.10.3.2
   Macro TEST {
      ALLPIs = 00010110001010010;
      ALLPOs = LLLLLL; }
      ALLIOs = ZZ;
// processed EVENT: 2.1.1.5.10.4.1
// processed EVENT: 2.1.1.5.10.5.1
// inserted  SCAN SEQUENCE: Scan_Preconditioning_Sequence (Scan Preconditioning)
```

```
   Macro TEST {
      ALLPIs = 00010000100111110;
      ALLPOs = LLHLLL;
      ALLIOs = LL; }
// inserted  SCAN SEQUENCE: Skewed_Unload_Sequence (Skewed Unload)
// inserted  SCAN SEQUENCE: Scan_Sequence (Scan)
   Macro TEST {
      ALLPIs = 0P001011111000001;
      ALLPOs = XXXXXX;
      ALLIOs = XX; }
   Macro SCAN_lssd {
      SO0001_lssd = HHLLLLLLL;
      SO0002_lssd = HHHHLLHLHHLLHHLH;

}  /* end MAIN_TEST  */
```

In the above example, the `PatternExec` construct does a backward reference, by name, to the `PatternBurst` which in turn does a forward (one of the few allowed in STIL) to the `Pattern` construct, by name. The `PatternBurst` also defines the termination characteristics of the `ALLPOs` and `ALLIOs` signal groups. The termination characteristics are determined by the termination specified in the vectors Test Section.

Within the `Pattern` construct, `TEST` macro invocations are used to apply and measure values to PIs and POs while `SCAN_lssd` macro invocations are used to apply and measure scan values. Values are passed to the macros via the assignments made within the invocation.

Encounter Test performs data compression on PI and PO events where possible in order to minimize the number of STIL vectors generated from the source vectors. PI and PO events are collected into a single STIL vector regardless of pattern boundaries, unless the user option to respect pattern boundaries is used, as long as their event order is consistent with the defined STIL WaveformTable order. Once an event is encountered which violates this order, e.g., a PI stim after a PO measure, the STIL vector is written and compression begins for the next vector. Thus, it is possible that several vectors are compressed into a single STIL vector. Scan events are not compressed.

STIL File Variations

The following sections discuss variations on the basic STIL file structure described above.

Overlapped/Non-Overlapped Scans

The default is to overlap scan-outs with scan-ins. This overlap reduces the number of cycles required for scanning of test data into and out of the scan chains. However, overlap is not always possible. For instance, overlap cannot be done between tester loops since tester loops are intended to be independent. In addition, overlap cannot be done where there are

intervening events which must take place between the scan-out and the subsequent scan-in. Therefore, for those cases where overlap is not possible a non-overlapped scan sequence is produced even though the user has requested scan overlap.

Optionally, the user may request a non-overlap scan style, whether or not overlap is possible. By not overlapping scans the number of tester cycles required for scan effectively doubles. This is not usually desired as it increases the time to test the product and thus increases the cost of test. However, non-overlap may be required in some cases.

The STIL example below shows a subset of the test vector section which implements a non-overlapped scan.

```
      ...

// processed EVENT: 2.1.1.5.2.5.1
// inserted  SCAN SEQUENCE: Scan_Preconditioning_Sequence (Scan Preconditioning)
// inserted  SCAN SEQUENCE: Skewed_Unload_Sequence (Skewed Unload)
// inserted  SCAN SEQUENCE: Scan_Sequence (Scan)
   Macro TEST {
      ALLPIs = 0P001001111001001;
      ALLPOs = XXXXXX; }
      ALLIOs = ZZ;
   Macro SCAN_lssd {
      SO0001_lssd = LHLLHHLHL;
      SO0002_lssd = LLHHLHHLHHHLHHHH; }

//*************************************************************************//
//  TEST SEQUENCE.............3           TYPE.......................normal   //
//*************************************************************************//

// processed EVENT: 2.1.1.5.3.1.1
// inserted  SCAN SEQUENCE: Scan_Preconditioning_Sequence (Scan Preconditioning)
// inserted  SCAN SEQUENCE: Scan_Sequence (Scan)
   Macro SCAN_lssd {
      SI0001_lssd = 000101101;
      SI0002_lssd = 1111111100000110; }

      ...
```

Macro Test Types

Support for Macro tests is very similar to Logic tests with the following exceptions:

- Macro tests typically contain loops to perform a repetitive application of clocks or to increment macro counters or address values. Therefore, the pattern loop and end loop events must be represented in the STIL.
- Macro tests may also contain pulses on non-clock PIs due to selecting a non-clock PI correspondence point for a macro clock.

The above macro test considerations are reflected in the test vector section by using the STIL `Loop` construct to represent loop events and by changing pulse events on non-clock PIs into

a sequence of two vectors: the first to turn the non-clock PI to its “on” state, the second to return it to its “off” state. If the non-clock PI is not originally off prior to the pulse, a third vector is added to establish this state.

```
     ...

// processed EVENT: 1.1.1.3.1.1.1
// processed EVENT: 1.1.1.3.1.1.2
   Macro TEST {
     ALLPIs = 00110001X0000XX0X;
     ALLPOs = XXXXXX;
     ALLIOs = 00; }
   Macro TEST {
     ALLPIs = 00110001X0000XX1X;
     ALLPOs = XXXXXX;
     ALLIOs = 00; }

     ...
```

The above example shows how a negative-active pulse on a non-clock PI is treated in the STIL using two consecutive vectors, the first a stim to 0 and the second a stim to 1.

LBIST Test Types

Support for LBIST tests involves consideration of the following LBIST test concepts:

- Scanning in the LBIST mode is somewhat different than normal scanning. The vectors will contain `channel_scan` events in lieu of the deterministic scan events. These `channel_scan` events also use attributes, as opposed to unique scan event types, to differentiate between normal and skewed scans. Therefore, unique LBIST “channel scan” sequences must be supported.
- The core of the LBIST test sequence involves cycling product LBIST clocks. These clocks cycle the on-product PRPGs and MISRs and move test values along the LBIST channels. Therefore, looping on the LBIST test sequence is required.
- LBIST test results are represented by “signatures.” These are the values that remain in the PRPGs and MISRs after the LBIST test sequence has completed. Signatures are represented as hexadecimal values by signature events. These must be converted to parent mode values and compared to the product values by a scan-out after the final sequence cycle has completed. Note, only the final signatures for each LBIST test sequence will be compared (see below).

These considerations are reflected in the test vector section by using STIL the `Loop` construct and by adding the necessary sequences to return to the parent mode and perform a scan-out.

```
//*************************************************************************//
//  TEST SEQUENCE.............2          TYPE.......................loop     //
//*************************************************************************//
```

```
Loop 256 {   /* Test Sequence Loop */
// processed EVENT: 1.2.1.2.2.1.1
// processed EVENT: 1.2.1.2.2.2.1
// inserted  SCAN SEQUENCE: Scan_Preconditioning_Sequence (Scan Preconditioning)
   Macro TEST {
      ALLPIs = 00P10000011000010;
      ALLIOs = ZZ;
      ALLPOs = XXXXXX; }
// inserted  SCAN SEQUENCE: Skewed_Unload_Sequence (Skewed Unload)
   Macro TEST {
      ALLPIs = 0P010000011000010;
      ALLIOs = ZZ;
      ALLPOs = XXXXXX; }
Loop 4 {   /* Channel Scan Loop  */
   Macro TEST {
      ALLPIs = PP010000011000010;
      ALLIOs = ZZ;
      ALLPOs = XXXXXX; }
} /* end Channel Scan Loop */
} /* end Test Sequence Loop */
   Macro TEST {
      ALLPIs = 00010000011000010;
      ALLIOs = ZZ;
      ALLPOs = XXXXXX; }
// processed EVENT: 1.2.1.2.2.3.2
// inserted  SCAN SEQUENCE: Scan_Preconditioning_Sequence (Scan Preconditioning)
// inserted  SCAN SEQUENCE: Scan_Sequence (Scan)
   Macro TEST {
      ALLPIs = 00010000111000001;
      ALLIOs = ZZ;
      ALLPOs = XXXXXX; }
   Macro SCAN_lssd {
      SO0001_lssd = LHLLXXXXX;
      SO0002_lssd = XXXXXXXXXXXXXXXX; }
```

In the example above, the inner `Loop` represents a channel_scan event which cycles the product clocks to increment the on-product PRPGs and MISRs and to move test values along the LBIST channels whereas the outer `Loop` performs "`N`" cycles of the LBIST test sequence. This example also shows how the parent scan state is entered and a scan-out is performed to unload the MISR values.

Timed Test Types

Support for timed tests involves consideration of the following:

- "Timing Data" objects exist within the vectors file which specify explicit timings for test period, strobe offset, and primary signal I/Os.
- Each timing data object applies a "dynamic" pattern within one or more test sequences.
- One or more test cycles may be required for each dynamic pattern.

The above considerations are reflected in the STIL by defining a unique `WaveformTable` for each timing data object and then referencing the appropriate `WaveformTable` for each dynamic pattern found within the test data. The following is an example of a dynamic `WaveformTable`.

```
WaveformTable TBautoLogicSeq8_7_0_cycle { Period '12500.0000ps';
   Waveforms {
      "A" { 01ZP { '0ns' P/P/P/D; '1250.0000ps' D/U/Z/U; '1750.0000ps' D/U/Z/D; } }
      "B" { 01ZP { '0ns' P/P/P/D; '2000.0000ps' D/U/Z/U; '2500.0000ps' D/U/Z/D; } }
      "C" { 01ZP { '0ns' P/P/P/D; '250.0000ps' D/U/Z/U; '750.0000ps' D/U/Z/D; } }
      "CS" { 01Z { '0ns' P; } }

      ...

      "DO3" { LHTX { '0ns' X; '1000.0000ps' l/h/t/x; '1250.0000ps' X; } }
      "DO4" { LHTX { '0ns' X; '1000.0000ps' l/h/t/x; '1250.0000ps' X; } }
      "SO1" { LHTX { '0ns' X; '1000.0000ps' l/h/t/x; '1250.0000ps' X; } }
      "SO1_BIDI" { LHTX { '0ns' X; '1000.0000ps' l/h/t/x; '1250.0000ps' X; } }
      "SO2" { LHTX { '0ns' X; '1000.0000ps' l/h/t/x; '1250.0000ps' X; } }
      "SO2_BIDI" { LHTX { '0ns' X; '1000.0000ps' l/h/t/x; '1250.0000ps' X; } }
     }  /* end Waveforms  */
   }  /* end WaveformTable  */

      ...
```

The `WaveformTable` is given a unique name which is constructed from its sequence name and timing ID appended with a cycle number. Within each `WaveformTable` are the signal waveform descriptions which contain the calculated times from the timing data objects. When a test sequence containing a dynamic pattern is encountered in the vectors, the converter produces a macro invocation which references the appropriate `WaveformTable` as shown below.

```
      ...

   Macro TBautoLogicSeq8_7_0 {
      ALLPIs = 0P01000001000010;
      ALLPOs = XXXXXX;
      ALLIOs = XX; }

      ...
```

Multiple vector statements are produced if there are multiple cycles required, each referencing the appropriate `WaveformTable`. For untimed “static” patterns, the user specified timings are used as usual.

Verilog Pattern Data Format

Encounter Test can export test vectors in Cadence Design Systems, Inc. Verilog format. The following discussion assumes a basic knowledge of Verilog. For more information, refer to Cadence Design Verilog documentation.

Encounter Test creates a common “task definition” file and one or more Verilog “vector” files for each exported set of tests. The task definition file contains the Verilog task definitions which describe the application of parallel and scan vectors. This file is “included” in each vector file and eliminates the redundant definition of these tasks. The vector files contain Verilog language statements that represent the test vectors in a format and which can be used as input to a Verilog simulator or to applications which convert Verilog to tester programs.

The test data is divided into multiple vector files in order to reduce the overall processing time of a single, large Verilog file. You can choose one of the following methods for creating the files using either the graphical interface or command line:

- Create Verilog from committed tests for each testsection found within each experiment in the source vectors file. Optionally, you can create Verilog for each tester loop within each test section within each experiment. The file name is of the form:

  `VER.`*`testmode.testsectiontype.ex#.ts#[.tl#]`*

- Create Verilog from uncommitted tests for each testsection found within each experiment in the source vectors file. Optionally, you can create Verilog for each tester loop within each test section within each experiment. The file name is of the form:

  `VER.`*`testmode.experiment.testsectiontype.ex#.ts#[.tl#]`*

where in each case:

`testmode` is the test mode name

`experiment` is the name of the experiment

`testsectiontype` is the test section type (e.g. scan)

`ex#` is the experiment number

`ts#` is the test section number

`tl#` is the test loop number (optional)

Commentary is included in each file which correlates its contents to the source vectors with regard to experiment, test section, tester loops, test procedures, test sequences and patterns.

The following Vectors test section types are supported:

- Flush
- Logic
- Scan
- Macro
- Channel_Scan
- Logic_LBIST

- Driver and Receiver
- IOWRAP Stuck Driver
- ICT Stuck Driver
- IEEE 1149.1 Integrity
- IOWRAP Shorted Nets
- ICT Shorted Nets

A complete set of example Verilog files is shown in "Verilog Pattern Data Examples" on page 285.

Verilog Basic File Structure

Conceptually, the Verilog can be divided into four sections: a header section, a section which connects the vectors to the structure, a set of predefined tasks for simulating a parallel test cycle, a scan cycle, etc. and the actual test vectors. The following discusses each section in more detail. Note that the task definition and vector files are described as a single logical entity although they are produced as separate files.

Header

A header section begins each Verilog file, an example of which is shown below.

```
/*************************************************************************//
//                          VERILOG TASKDEF FILE                           //
//  Encounter(R) Test and Diagnostics 3.0.    Oct 07, 2005 (linux24 ET30)   //
//*************************************************************************//
//                                                                         //
//  FILE CREATED.............October 07, 2005 at 16:13:04                 //
//                                                                         //
//  PROJECT NAME.............lbc                                          //
//                                                                         //
//  TESTMODE.................lssd                                         //
//                                                                         //
//  TDR......................dummy_tester_lssd                            //
//                                                                         //
//  TEST PERIOD..............80       TEST TIME UNITS...........ns        //
//  TEST PULSE WIDTH.........8                                            //
//  TEST STROBE OFFSET.......72       TEST STROBE TYPE..........edge      //
//  TEST BIDI OFFSET.........0                                            //
//  TEST PI OFFSET...........0        X VALUE...................X         //
//                                                                         //
//  TEST PI OFFSET for pin "A" (PI # 1) is ......................8         //
//  TEST PI OFFSET for pin "B" (PI # 2) is ......................24        //
//  TEST PI OFFSET for pin "C" (PI # 3) is ......................8         //
//                                                                         //
//  SCAN FORMAT..............serial   SCAN OVERLAP.............yes       //
//  SCAN PERIOD..............80       SCAN TIME UNITS...........ns        //
//  SCAN PULSE WIDTH.........8                                            //
//  SCAN STROBE OFFSET.......0        SCAN STROBE TYPE..........edge      //
//  SCAN BIDI OFFSET.........16                                           //
//  SCAN PI OFFSET...........16       X VALUE...................X         //
//                                                                         //
//  SCAN PI OFFSET for pin "A" (PI # 1) is ......................24        //
//  SCAN PI OFFSET for pin "B" (PI # 2) is ......................40        //
//                                                                         //
```

```
//*************************************************************************//
```

The following information is included in the header section:

- Encounter Test version used to create the file
- Date and time of file creation
- Project identification data
- Applicable testmode name
- Source Experiment and Test Section identification
- Specification of selected command line parameters
- Applicable tester termination information

This section also supplies the Verilog "`timescale`" and "`module`" constructs. The `timescale` construct defines the units of time to be applied to timing statements used in the task definitions. The `module` construct is required at the beginning of each Verilog file. It is terminated by an "`endmodule`" construct at the end of the file. The module name is a catenation of the entity name, testmode name, experiment number, test section number, and, if applicable, tester loop number.

Structure Connection

This section defines the various Verilog variables used to manipulate or sample the primary I/O ports during simulation. An example is shown below.

```
//*************************************************************************//
//                 DEFINE VARIABLES FOR ALL PRIMARY I/O PORTS              //
//*************************************************************************//
   reg    [1:19] stim_PIs;

   reg    [1:19] stim_CIs;

   reg    [1:08] resp_POs,
                 comp_POs;

   reg    [1:19] buss_PIs;

   tri0   [1:08] buss_POs;
```

The `buss_PIs` variable contains a bit position for each PI and bidirectional PI net. Pure PIs are mapped to positions in `buss_PIs` by the structure instantiation such that when a `buss_PIs` bit changes value it is propagated to the corresponding PI of the structure. BIDI PIs are mapped to their corresponding POs by use of an assign statement (see below).

Similarly, the `buss_POs` variable contains a bit position for each PO net. POs, including BIDI POs, are mapped to `buss_POs` positions by the structure instantiation such that when simulation changes a PO value it is reflected in the corresponding bit position in `buss_POs`.

The `stim_PIs` variable contains a bit position for each PI and bidirectional PI net and is used to hold the values assigned to these nets by the test vectors. When the parallel cycle task is invoked it transfers the values from `stim_PIs` to `buss_PIs` which in turn propagates the values to the structure PIs and bidirectional POs for simulation. The `stim_CIs` variable is used to manipulate clock pulses on clock PIs in a similar manner.

The `resp_POs` variable contains a bit position for each PO net, including bidirectional POs. The `resp_POs` variable is used to hold the response values expected by the test vectors for these nets. When the parallel cycle task is invoked it compares the values in `buss_POs`, as calculated by the simulation, to those in `resp_POs`. When a miscompare is detected, a flag is set in `comp_POs` for the corresponding PO net.

This section also defines similar variables for input stim, output compare, and shift masks for all scan chains.

```
//*************************************************************************//
//                    DEFINE VARIABLES FOR ALL SHIFT CHAINS                //
//*************************************************************************//

   reg    [1:32] stim_SLs;

   reg    [1:02] stim_SSs;

   reg    [1:32] resp_MLs;
```

The `stim_SLs` and `resp_MLs` variables hold the values to be applied to and measured at all flops or latches. The `stim_SSs` variable holds values to be applied to all skewed stim latch values, if any.

The test vector section assigns stim and expect values from the vector scan events to the `stim_SLs` and `resp_MLs` variables and then invokes the necessary scan task(s) which perform the actual scan operation. Since scan chains will vary in length, shorter chains are required to be “padded” with X's so that chains can be scanned in parallel. This padding is accomplished by use of selectively assigning bit values to the `stim_SLs` and `resp_MLs` variables in the test vector section. Since unassigned bits in these variables are never changed, shorter chains are effectively padded with X values.

Connection to the structure is accomplished as shown below. This statement “instantiates” the structural module and assigns Verilog variables to each of its ports (PIs and POs).

```
//*************************************************************************//
//        INSTANTIATE THE STRUCTURE AND CONNECT TO VERILOG VARIABLES       //
//*************************************************************************//

  lbistchip
```

```
    lbistchip_inst (
       .A             (buss_PIs[01]),
       .B             (buss_PIs[02]),
       .C             (buss_PIs[03]),
       .CS            (buss_PIs[04]),
       .DI1           (buss_PIs[05]),
       .DI2           (buss_PIs[06]),
       .DI3           (buss_PIs[07]),
       .DI4           (buss_PIs[08]),
       .ENABLE1       (buss_PIs[09]),
       .ENABLE2       (buss_PIs[10]),
       .ME            (buss_PIs[11]),
       .PS            (buss_PIs[12]),
       .SEL           (buss_PIs[13]),
       .SI1           (buss_PIs[14]),
       .SI2           (buss_PIs[15]),
       .ST1           (buss_PIs[18]),
       .ST2           (buss_PIs[19]),

       .DO1           (buss_POs[01]),
       .DO2           (buss_POs[02]),
       .DO3           (buss_POs[03]),
       .DO4           (buss_POs[04]),
       .SO1           (buss_POs[05]),
       .SO1_BIDI      (buss_POs[06]),
       .SO2           (buss_POs[07]),
       .SO2_BIDI      (buss_POs[08]));
```

Connection is done by name so the order of the ports in the instantiation is not dependent on the order of ports in the structural definition. Pure input ports are connected to the `buss_PIs` variable. Output and bidirectional ports are connect to the `buss_POs` variable. The top cell name is used as the name of the module. The top cell name appended with `_inst` is used as the name of the module instance.

Finally, it may become necessary to make several other connections to the structure. This is done for bidirectional PIs, which are considered to be POs by Verilog, and, in the case of a parallel scan format, to internal latches. An example of bidirectional pin connections is shown below.

```
//*************************************************************************//
//                        MAKE SOME OTHER CONNECTIONS                      //
//*************************************************************************//

assign
    buss_POs[06] = buss_PIs[16],
    buss_POs[08] = buss_PIs[17];
```

These assignments place the values put into the `buss_PIs` variable bidi net positions by the various tasks onto bidirectional PO nets which have been connected to the `buss_POs` variables by the instantiation of the structure, as shown above.

Tasks

Encounter Test defines various tasks depending on the type of test patterns created. For example, Encounter Test can create tasks to apply the static FULLSCAN patterns or to apply patterns via the XOR or OPMISR compression logic. This section covers some of these tasks that will help you understand any Verilog pattern set created by Encounter Test.

Note: Refer to the Verilog mainsim file created by Encounter Test to know more about all the tasks.

sim_setup

This is the first task invoked by Encounter Test and it initializes the `global_term` variable to `Z`. Next, it assigns all the design port names to the `name_PO` variable, which is used for both debug messages and as part of any miscompare messages.

Lastly, it searches for the following NCSim "+" arguments that may have been applied:

- `+DEBUG` - This turns on the runtime debug messages.
- `+HEARTBEAT` - This turns on the heartbeat messages that provide the current pattern number being processed and assures that the simulation is still active.
- `+START_RANGE=<odometer>` and `+END_RANGE=<odometer>` - These allow you to simulate only a subsection of the entire pattern set within the vector file.
- `+FAILSET` - When set, this argument dumps the simulation miscompare messages to a separate file.

Refer to NC-Sim Considerations in *Encounter Test: Guide 6: Test Vectors* for more information.

sim_vector_file

This task processes the content of the vector files, one at a time, and then returns the control back to the initial block. If another file needs to be processed in between, the initial block processes it and then returns the control back to this task.

The task initializes the `stim`, `part`, and `resp` variables, which are defined by Encounter Test to contain the pattern stimulus, capture the part response, and hold the expected responses.

The task then reads each line of the vector file. Each line is processed by a CASE STATEMENT that extracts the first entry on each line to determine the appropriate course of action.

This task includes opcode definitions, which have reference to the following:

- Skewed latches - Skewed latches exist only within the LSSD domain. Encounter Test can add an extra "A" or "B" clock to the scan shift sequence. This "skews" the normal scan shift to where the L1 latch may not have the same logic value as the L2 latch for a specific position on the scan chain.
- Core tests - Core logic patterns are patterns that were created for a module that is now instantiated into the design. These patterns are mapped/migrated from a core or internal module to the top-level design.

Table 2-1 on page 180 lists the valid opcode definitions used by this task:

Table 2-1 Opcode Definitions

Function	Explanation
`000:`	This line stops simulation. When inserted in the middle of a file, it will stop NCSim at that point. *Tip* From the Ncsim prompt, issue reset to reset simulation to time 0 and run xx ps to run up to the desired point within simulation. This helps you avoid having to edit the pattern file.
`100 <comment>:`	This is a comment line
`101 <PI>:`	This line starts oscillator pins. These are pins assigned the `OSC` flag within the pin assignment file. The `PI` entry points to the OSC pin.

102 *<PI>:*	This line stops the oscillation on OSC pins. The *PI* entry points to the OSC pin.
200 *<stim_PIs>:*	This line contains the input stimulus for the PIs. *stim_PIs* is the variable that is set.
201 *<stim_CIs>:*	This line contains input clock stimulus. *stim_CIs* is the variable that is set.
202 *<resp_POs>:*	This line contains the expected PO responses. *resp_POs* is the variable that is set.
203 *<global_term>:*	This line contains the global termination value that is applied to all design bidirectional pins. *global_term* is the variable that is set.
204 *<stim_SSs>:*	This line contains the scan input stimulus values for *stim_SSs*. These are the skewed scan latch input stimulus values.
205 *<stim_CSs>:*	This line contains the scan input stimulus values for *stim_CSs*. These are the skewed scan latch stimulus values from a CORE logic test pattern set.

300 *<testModeNumber> <stim_SLs>:* **Note:** If `write_vectors scanvariables=byregister`, the opcode definition is as follows: 300 *<testModeNumber> <regNumber><stim_SLs>:*	This line contains the scan input stimulus values. *stim_SLs* is the variable that is set. *testModeNumber* defines the testmode being used. Different test modes result in different scan chain configurations. The second entry on this line is a test mode. The most likely reason for multiple test modes is the presence of `FULLSCAN` and `OPMISRPLUS` or `XOR` compression logic within the design. *regNumber* is an integer with `1` as the minimum value and the number of stim or measure registers as the maximum value.
301 *<testModeNumber> <resp_MLs>:* **Note:** If `write_vectors scanvariables=byregister`, the opcode definition is as follows: 301 *<testModeNumber> <regNumber><resp_MLs>:*	This line contains the expected scan chain values from the design. *resp_MLs* is the variable that is set. This also includes different test modes. *regNumber* is an integer with `1` as the minimum value and the number of stim or measure registers as the maximum value.
302 *<testModeNumber> <stim_CEs>:* **Note:** If `write_vectors scanvariables=byregister`, the opcode definition is as follows: 302 *<testModeNumber> <cmeNumber><stim_CEs>:*	If the design includes vector compression logic and `MASK` logic, this line provides the `Channel Mask Enable` input. *stim_CEs* is the variable that is set. *cmeNumber* is an integer with `1` as the minimum value and the number of CME pins on the part as the maximum value.

303 *<stim_CMs>:* **Note:** If `write_vectors scanvariables=byregister`, the opcode definition is as follows: 303 *<regNumber><stim_CMs>:*	This entry provides the `Channel Mask` input data that Encounter Test ATPG creates while generating the test patterns. *stim_CMs* is the variable that is set. *regNumber* is an integer with `1` as the minimum value and the number of stim or measure registers as the maximum value.
304 *<experimentNumber> <stim_CLs>:* **Note:** If `write_vectors scanvariables=byregister`, the opcode definition is as follows: 304 *<experimentNumber> <regNumber><stim_CLs>:*	This line contains the scan input stimulus values for *stim_CLs*. These are the scan latch stimulus values from a `CORE` logic test pattern set. *experimentNumber* contains an experiment number that defines the experiment being used. If these scan stimulus values from a core test were skewed then the opcode will be `205`. *regNumber* is an integer with `1` as the minimum value and the number of stim or measure registers as the maximum value.
305 *<experimentNumber> <resp_CLs>:* **Note:** If `write_vectors scanvariables=byregister`, the opcode definition is as follows: 305 *<experimentNumber> <regNumber><resp_CLs>:*	This line contains the expected response values from a set of core test scan latches. *resp_CLs* is the variable that is set. This also includes different experiments. *regNumber* is an integer with `1` as the minimum value and the number of stim or measure registers as the maximum value.
306 *<stim_OIs>:* **Note:** If `write_vectors scanvariables=byregister`, the opcode definition is as follows: 306*<regNumber><stim_OIs>:*	This line contains the stimulus `OPCG` values that are loaded via OLE. *stim_OIs* is the variable that is set. *regNumber* is an integer with `1` as the minimum value and the number of stim or measure registers as the maximum value.

`400:`	This line invokes the `test_cycle` task. This task applies the functional capture sequence.
`401:`	This line invokes the `test_cycle_flush` task. This is an LSSD scan chain flush test.
`403:`	This line invokes the `scan_cycle` task. This task applies the scan chain shift pattern sequence.
`500:`	Start of a repeat loop.
`501:`	End of a repeat loop.
`600` *`<testModeNumber>`* *`<sequenceNumber>`*`<`*`MAX`*`>:`	This line invokes any `SCAN preconditioning` or `SCAN exit` sequences required by the design. Every test mode (*`testModeNumber`*) in the design can have a unique sequence (*`sequenceNumber`*) to enter or exit scan shift mode. A verilog task is defined for mode and the tasks are invoked as needed. In addition, there can be other special sequences such as a `MISR RESET` sequence for `OPMISRPLUS` compression logic. If used, *`MAX`* defines the number of cycles or the cycle number.
`601` *`<testModeNumber>`* *`<sequenceNumber>`* *`<MAX>:`*	Similar to `600`, this line invokes `SCAN preconditioning` or `SCAN exit` sequences but for OPCG logic.
`700` *`<experimentNumber>`* *`<scanNumber>`* *`<sequenceNumber>`* *`<MAX>:`*	Similar to `600`, this line is the preconditioning and exit sequencing for any embedded `CORE` test. Every experiment (*`experimentNumber`*) in the design can have a unique *`scanNumber`* and a unique sequence (*`sequenceNumber`*) to enter or exit scan shift mode. If used, *`MAX`* defines the number of cycles or the cycle number.

`800:`	This line invokes a diagnostic observe sequence is case the design contains `OPMISRPLUS` compression logic.
`801:`	This line will invoke a diagnostic return sequence.
`850 <taskNum>:`	This line contains the special event sequences for dynamic test pattern application that the design requires. *taskNum* is the special event sequence to be invoked.
`900 <patternNumber>:`	This line contains the ATPG pattern number.
`901 <patternNumber>:`	This line contains the measure ATPG pattern number.

Test Vectors

This section represents the actual test vectors in Verilog format. As can be seen below, this section starts with the Verilog “initial” construct and the opening of a “begin” block to contain the vector data and ends with two “end” constructs which close out the “begin” and “module” constructs to end the file.

```
//*************************************************************************//
//                        INCLUDE COMMON TASK DEFINITIONS                  //
//*************************************************************************//

  `include "VER.lssd.taskdef"

//*************************************************************************//
//  TESTER LOOP..............1         PROCEDURES HAVE MEMORY....no        //
//*************************************************************************//

  initial
     begin

        simulation_setup;

//*************************************************************************//
//  TEST PROCEDURE...........1         TYPE......................init      //
//  SLOW TO TURN OFF.........false     SEQUENCES HAVE MEMORY.....no        //
//*************************************************************************//

  PATTERN = "2.1.1.1.1.1";
        stim_PIs[01:19] = 19’bXXXXXXXXXXXXXXXZZXX;
        task_cycle;

  PATTERN = "2.1.1.1.1.2";
        stim_PIs[01:19] = 19’b000XXXXXXXXXXXXZZXX;
```

```
        task_cycle;

//*************************************************************************//
//  TEST PROCEDURE............2         TYPE.......................normal     //
//  SLOW TO TURN OFF..........false     SEQUENCES HAVE MEMORY.....no         //
//  STATIC FAULTS.............21        PERCENT STATIC FAULTS.....58.870968  //
//*************************************************************************//

  PATTERN = "2.1.1.2.1.1";
        stim_SLs[01:09] = 09'b010010011;
        stim_SLs[17:32] = 16'b0111000100000001;
        Scan_Preconditioning_Sequence_lssd;
        Scan_Sequence_lssd;

  PATTERN = "2.1.1.2.1.2";
        stim_PIs[01:19] = 19'b000011010000111ZZ00;
        task_cycle;

  PATTERN = "2.1.1.2.1.3";
        task_cycle;

  PATTERN = "2.1.1.2.1.4";
        resp_POs[01:08] = 08'b00100000;
        task_cycle;

        ...

  PATTERN = "2.1.1.5.9.2";
        stim_PIs[01:19] = 19'b000010110001101ZZ10;
        resp_POs[01:08] = 08'b00000000;
        task_cycle;

  PATTERN = "2.1.1.5.9.3";
        resp_POs[01:08] = 08'b00000000;
        task_cycle;

  PATTERN = "2.1.1.5.9.4";
        stim_CIs[01:19] = 19'b001XXXXXXXXXXXXXXX;
        task_cycle;

  PATTERN = "2.1.1.5.9.5";
        resp_MLs[01:09] = 09'b100011011;
        resp_MLs[17:32] = 16'b1001111011001101;

  PATTERN = "2.1.1.5.10.1";
        stim_SLs[01:09] = 09'b110001011;
        stim_SLs[17:32] = 16'b0010100100010011;
        Scan_Preconditioning_Sequence_lssd;
        Skewed_Unload_Sequence_lssd;
        Scan_Sequence_lssd;

  PATTERN = "2.1.1.5.10.2";
        stim_PIs[01:19] = 19'b000111110000111ZZ10;
        resp_POs[01:08] = 08'b00000000;
        task_cycle;

  PATTERN = "2.1.1.5.10.3";
        resp_POs[01:08] = 08'b00000000;
        task_cycle;

  PATTERN = "2.1.1.5.10.4";
```

```
        stim_CIs[01:19] = 19'b001XXXXXXXXXXXXXXX;
        task_cycle;

  PATTERN = "2.1.1.5.10.5";
        resp_MLs[01:09] = 09'b110100011;
        resp_MLs[17:32] = 16'b0111001000111111;
        Scan_Preconditioning_Sequence_lssd;
        Skewed_Unload_Sequence_lssd;
        Scan_Sequence_lssd;

//*************************************************************************//
//                              END_OF_VECTORS                             //
//*************************************************************************//

     simulation_wrapup ("logic.ex2.ts1.tl1");

     end

   endmodule
```

As patterns are encountered, various Verilog statements are produced as follows:

- `Stim_PI` events create an assignment of the PI values to be applied to the `stim_PIs` variable, including bidirectional PIs,
- `Stim_Clock` events create an assignment of the clock values to be applied to the respective clock index of the `stim_PIs` and `stim_CIs` variables,
- `Pulse clock` events create an assignment of the inactive clock value to the respective clock index of the `stim_PIs` variable and the active value to the `stim_CIs` variable,
- `Measure_PO` events create an assignment of the expected PO values to the `resp_POs` variable,
- `Scan_Load` or `scan_load` events create an assignment of stim values, potentially adjusted for inversion, to the `stim_SLs` variable,
- `Skewed_Scan_Load` events create an assignment of the latch stim values, potentially adjusted for inversion, to be applied to the `stim_SLs` and `stim_SSs` variables, and
- `Scan_Unload, Skewed_Scan_Unload,` and `scan_unload` events create an assignment of the expected values, potentially adjusted for inversion, to the `resp_MLs` variable.

Since Verilog register variables hold their values until explicitly changed, the above assignments are only produced if their values are different from the last set of values assigned to the variable. This reduces redundant assignments which, in turn, reduces the overall size of each vector file.

One or more (for scan events) of the tasks discussed previously are then invoked, depending on the type of events encountered in the vectors.

Equation-based Timing for Verilog Patterns

If you specify `write_vectors equationbasedtimings=yes,` Encounter Test defines the timing information in variables, which you can use to modify the timing information in the Verilog file.

The following is the sample equation-based timing definitions added to the `mainsim.v` file when specifying `equationbasedtimings=yes`:

```
//*************************************************************************//
//                     EQUATION BASED TIMINGS DEFINITIONS                  //
//*************************************************************************//

 real  cycle_time, cycle_delay, scaling_percent, scaling_factor;
 real  testperiod, testpulsewidth, testpioffset, testbidioffset, teststrobeoffset;
 real  scanperiod, scanpulsewidth, scanpioffset, scanbidioffset, scanstrobeoffset;
 integer  scan_length;

 real  test_A;
 real  test_B;
 real  test_C;

 real  scan_A;
 real  scan_B;
 real  testpioffset_end, testbidioffset_end, scanpioffset_end, scanbidioffset_end;
 real  testperiod_flush, teststrobeoffset_flush;
 real  scanperiod_flush, scanstrobeoffset_flush;
 real  test_A_end;
 real  scan_A_end;
 real  test_B_end;
 real  scan_B_end;
 real  test_C_end;
```

Verilog Variations

The following sections discuss variations on the basic Verilog file structure described above.

Overlapped/Non-Overlapped Scans

The default is to overlap scan-outs with scan-ins. This overlap reduces the number of cycles required for scanning of test data into and out of the scan chains. However, overlap is not always possible. For instance, overlap cannot be done between tester loops since tester loops are intended to be independent. In addition, overlap cannot be done where there are intervening events which must take place between the scan-out and the subsequent scan-in. Therefore, for those cases where overlap is not possible a non-overlapped scan sequence is produced even though the user has requested scan overlap.

The test vector example above shows overlapped scans. Notice that pattern 2.1.1.5.9.5, which resulted from a `Skewed_Scan_Unload` event, sets the values to be measured into the

`resp_MLs` variable. However, no scan procedures are invoked at this point, i.e., scan is "deferred." The following pattern, number 2.1.1.5.10.1, which resulted from a `Scan_Load` event, sets the scan values into `stim_SLs` as well as invokes the necessary scan procedures to do the scan.

Optionally, the user may request a non-overlap scan style, whether or not overlap is possible. By not overlapping scans the number of tester cycles required for scan effectively doubles. This is not usually desired as it increases the time to test the product and thus increases the cost of test. However, non-overlap may be required in some cases.

The Verilog example below shows a subset of the test vector section which implements a non-overlapped scan.

```
        ...

PATTERN = "2.1.1.5.9.5";
      resp_MLs[01:09] = 09'b100011011;
      resp_MLs[17:32] = 16'b1001111011001101;
      Scan_Preconditioning_Sequence_lssd;
      Skewed_Unload_Sequence_lssd;
      Scan_Sequence_lssd;

PATTERN = "2.1.1.5.10.1";
      stim_SLs[01:09] = 09'b110001011;
      stim_SLs[17:32] = 16'b0010100100010011;
      resp_MLs[17:32] = 16'bXXXXXXXXXXXXXXXX;
      Scan_Preconditioning_Sequence_lssd;
      Scan_Sequence_lssd;

      ...
```

Notice that each scan event invokes the required scan tasks. Also, to ensure that no measures are done during the scan-in operation, the `resp_MLs` variable is set to all X's. Effectively, this are the only differences for the non-overlap mode of operation. The scan sequence itself is not changed.

Macro Test Types

Support for Macro tests is very similar to Logic tests with the following exceptions:

- Macro tests typically contain loops to perform a repetitive application of clocks or to increment macro counters or address values, etc. Therefore, the loop and end loop events must be represented in the Verilog.
- Macro tests may also contain pulses on non-clock PIs due to selecting a non-clock PI correspondence point for a macro clock.

The above macro test considerations are reflected in the test vector section by using the Verilog "repeat" and "begin/end" constructs to represent the pattern loop events and by

changing pulse events on non-clock PIs into a sequence of two vectors: the first to turn the non-clock PI to its “on” state, the second to return it to its “off” state. If the non-clock PI is not originally off prior to the pulse, a third vector is added to establish this state.

```
        ...

PATTERN = "1.4.1.5.1.2";
         stim_PIs[01:19] = 19’b000100000000100ZZ00;
         task_cycle;

         stim_PIs[01:19] = 19’b000100000000100ZZ10;
         resp_POs[01:08] = 08’b00000000;
         task_cycle;

        ...
```

The above example shows how a negative-active pulse on a non-clock PI is treated in the Verilog using two consecutive vectors, the first a stim to 0 and the second a stim to 1.

LBIST Test Types

Support for LBIST tests involves consideration of the following LBIST test concepts:

- Scanning in the LBIST mode is somewhat different than normal scanning. The vectors will contain `channel_scan` events in lieu of the deterministic scan events. These `channel_scan` events also use attributes, as opposed to unique scan event types, to differentiate between normal and skewed scans. Therefore, unique LBIST “channel scan” sequences must be supported.
- The core of the LBIST test sequence involves cycling product LBIST clocks. These clocks cycle the on-product PRPGs and MISRs and move test values along the LBIST channels. Therefore, looping on the LBIST test sequence is required.
- LBIST test results are represented by “signatures.” These are the values that remain in the PRPGs and MISRs after the LBIST test sequence has completed. Signatures are represented as hexadecimal values by signature events. These must be converted to parent mode values and compared to the product values by a scan-out after the final sequence cycle has completed. Note, only the final signatures for each LBIST test sequence will be compared (see below).

These considerations are reflected in the task definition and test vector sections. In an LBIST task definition all parent mode and the child mode scan tasks are defined. Each collection of tasks has been derived from the appropriate mode's sequence definition file. Each mode has a preconditioning, skewed loading, skewed unloading, and scan sequence. These tasks are invoked in the test vector section whenever a scan sequence is required for the respective mode. Note that while the sequences themselves are different, the timings used within each are common, i.e., the “static” timings are used since scan is not timed.

An LBIST test vector section is very similar to deterministic test in construction. However, the test sequence uses Verilog “repeat” and “begin/end” constructs to model the LBIST test sequence cycle. As shown below, the patterns contained within the begin/end block reflect the patterns contained within the LBIST test sequence.

```
//*************************************************************************//
//  TEST PROCEDURE............1         TYPE......................init       //
//  SLOW TO TURN OFF..........false     SEQUENCES HAVE MEMORY.....no         //
//*************************************************************************//

   PATTERN = "1.2.1.1.1.1";
         stim_PIs[01:19] = 19’bXXXXXXXXXXXXXXXZZXX;
         task_cycle;

         stim_PIs[01:19] = 19’b000XXXXXXXXXXXXZZXX;
         stim_CIs[01:19] = 19’b000XXXXXXXXXXXXXXXX;
         task_cycle;

   PATTERN = "1.2.1.1.1.1";
         stim_SLs[01:09] = 09’b111110000;
         stim_SLs[17:32] = 16’bXXXXXXXXXXXXXXXX;
         Scan_Preconditioning_Sequence_lssd;
         Scan_Sequence_lssd;

   PATTERN = "1.2.1.1.1.2";
         stim_PIs[01:19] = 19’b000XXXXXXXX0XXXZZ1X;
         task_cycle;

//*************************************************************************//
//  TEST PROCEDURE............2         TYPE......................normal     //
//  SLOW TO TURN OFF..........false     SEQUENCES HAVE MEMORY.....no         //
//*************************************************************************//

   PATTERN = "1.2.1.2.1.1";
         Scan_Preconditioning_Sequencebsr_lbist;
         Scan_Sequence_lbist;

         stim_PIs[01:19] = 19’b000100000110000ZZ10;
         task_cycle;

repeat(256) begin
   PATTERN = "1.2.1.2.2.1";
         stim_CIs[01:19] = 19’b001XXXXXXXXXXXXXXX;
         task_cycle;

   PATTERN = "1.2.1.2.2.2";
         Scan_Preconditioning_Sequence_lbist;
         Skewed_Unload_Sequence_lbist;
         Scan_Sequence_lbist;

   PATTERN = "1.2.1.2.2.3";
end
         resp_MLs[01:09] = 09’b0100XXXXX;
         resp_MLs[17:32] = 16’bXXXXXXXXXXXXXXXX;
         Scan_Preconditioning_Sequence_lssd;
         Scan_Sequence_lssd;
```

At the completion of the sequence the final PRPG and MISR signature events return the product to the parent mode and a scan-out/compare is performed (as in example pattern 1.2.1.2.2.3) for the final signature values. In addition, "`begin_test_mode`" events encountered in any modeinit sequence cause the inclusion of the target mode's initialization sequence from its sequence definition file to be instantiated in-line in the vectors (see pattern 1.2.1.1.1.1 in the example).

The following limitations apply to the Verilog LBIST support:

- LBIST allows mixing of on-product and tester supplied PRPGs and MISRs. However, there is no reasonable way to represent tester PRPGs or MISRs in Verilog. Therefore, support will be limited to on-product PRPGs and MISRs only. If tester supplied PRPGs or MISRs are indicated in the test data, a message will be issued indicating that this type of test data cannot be supported and the test section will be skipped.
- LBIST allows the determination of "intermediate" signatures for PRPG and MISR values at various points within the test cycle. While Verilog support for intermediate signatures is possible, the requirement for it is not clear at this time. Thus, intermediate signatures will not be supported, however no messages will be issued if intermediate signatures are encountered.
- LBIST supports a "fast forward" mode of operation where ineffective (non-fault detecting) cycles are skipped. Again, while Verilog support is possible, the requirement for it is not clear at this time. Thus, fast forward cycles will not be supported.

Timed Test Types

Support for timed tests involves consideration of the following:

- "Timing Data" objects exist within the vectors file which specify explicit timings for test period, strobe offset, and primary signal I/Os.
- Each timing data object applies a "dynamic" pattern within one or more test sequences.
- One or more test cycles may be required for each dynamic pattern.

The above considerations are reflected in the Verilog by defining a unique task for each timing data object and then invoking the appropriate task for each dynamic pattern found within the test data. The following is an example of a dynamic task.

```
//*************************************************************************//
//                        DEFINE TIMED PARALLEL PROCEDURE                  //
//*************************************************************************//

  task TBautoLogicSeq39_36_0;
    begin

      #0.125000;
```

```
      buss_PIs[0021] = stim_CIs[0021];
     #6.625000;
      buss_PIs[0021] = stim_PIs[0021];
     #6.250000;
      buss_PIs[0196] = stim_CIs[0196];
     #6.625000;
      buss_PIs[0196] = stim_PIs[0196];
     #5.375000;
      stim_CIs       = 0200'bXXXXXX...0XX1XX00XXX;XXXX...XXX0XX1XX00XXX;

   end
 endtask
```

The task is given a unique name which is constructed from its sequence name and timing ID appended with a cycle number. Within each task the various values are applied to the PI pins at the times calculated from the timing data objects. When a test sequence containing a dynamic pattern is encountered in the vectors, the converter produces a statement which invokes the appropriate task as shown below.

```
      ...

   PATTERN = "2.1.1.85.1.4";
      stim_CIs[0001:0200] = 0200'bXXXXX...XXXXXX0XX1XX10XXX;
      task_cycle;

   PATTERN = "2.1.1.85.1.5";
      stim_CIs[0001:0200] = 0200'bXXXX1XXX...XXX0XX1XX10XXX;
      TBautoLogicSeq39_36_0;


      ...
```

Multiple task invocation statements are produced if there are multiple cycles required, each referencing the appropriate task. For untimed “static” patterns, the tasks which use the user specified timings are invoked as usual.

Refer to "Default Timings for Clocks" in the *Encounter Test: Guide 6: Test Vectors* for related information.

Scan Formats

As discussed above, two scan forms of scan are supported.

When the *parallel* format is selected, several changes are required in Verilog files from what has been described earlier. The most obvious, and most significant, change involves the scan task which must now provide the Verilog which manipulates the flops and latches directly instead of via a shifting process. There are also several minor changes required to other sections. The following example shows how the chain variables are defined for these scan formats.

```
//*************************************************************************//
//                     DEFINE VARIABLES FOR ALL SHIFT CHAINS               //
//*************************************************************************//

  reg    [1:32] stim_SLs;

  reg    [1:02] stim_SSs;

  reg    [1:32] resp_MLs;

  reg    [1:32] buss_SLs;

  wire   [1:32] buss_MLs;

     ...
```

The differences here are the addition of the new `buss_SLs` and `buss_MLs` variables which are added to provide a source for manipulating flop and latch inputs and outputs. These variables are connected to the structure by the following assignments.

```
//*************************************************************************//
//                         MAKE SOME OTHER CONNECTIONS                     //
//*************************************************************************//

   assign(supply0,supply1) // Control Register 1
      lbistchip_inst.SI1 = buss_SLs[16]!==1'bZ ?  buss_SLs[16] : 1'bZ ,
      lbistchip_inst.fvsrlout = buss_SLs[15]!==1'bZ ?  buss_SLs[15] : 1'bZ ,
      lbistchip_inst.M00.Y1 = buss_SLs[14]!==1'bZ ?  buss_SLs[14] : 1'bZ ,
      lbistchip_inst.M00.Y2 = buss_SLs[13]!==1'bZ ?  buss_SLs[13] : 1'bZ ,
      lbistchip_inst.M00.Y3 = buss_SLs[12]!==1'bZ ?  buss_SLs[12] : 1'bZ ,
      lbistchip_inst.Y4out = buss_SLs[11]!==1'bZ ?  buss_SLs[11] : 1'bZ ,

        ...

      lbistchip_inst.M05.SlaveOut1 = buss_SLs[19]!==1'bZ ?  buss_SLs[19] : 1'bZ ,
      lbistchip_inst.M05.SlaveOut2 = buss_SLs[18]!==1'bZ ?  buss_SLs[18] : 1'bZ ,
      lbistchip_inst.M05.SlaveOut3 = buss_SLs[17]!==1'bZ ?  buss_SLs[17] : 1'bZ ;

   assign(supply0,supply1) // Observe Register 1
      buss_MLs[01] =  lbistchip_inst.SO1_BIDI ,
      buss_MLs[02] =  lbistchip_inst.M01.srl3Out ,
      buss_MLs[03] =  lbistchip_inst.M01.srl2Out ,

        ...

      buss_MLs[29] =  lbistchip_inst.ch1slo4 ,
      buss_MLs[30] =  lbistchip_inst.M02.SlaveOut3 ,
      buss_MLs[31] =  lbistchip_inst.M02.SlaveOut2 ,
      buss_MLs[32] =  lbistchip_inst.M02.SlaveOut1 ;
```

Values are applied to the flops or latches by setting values into `buss_SLs`. Simulation values from each scan cell are collected by `buss_MLs` which can then be compared to values in `resp_MLs` just as `buss_POs` values are compared to values in `resp_POs`.

Analysis of Verilog Simulation Miscompares

The Encounter Test Verilog vectors, in combination with a user-provided Verilog model, can be used to drive a Verilog simulation. To facilitate the use of the vectors in this manner, the taskdef files contain statements which compare the expected design responses predicted by Encounter Test to those generated by the Verilog model during the Verilog simulation. These statements attempt to mimic the comparisons done by test equipment as if the vectors were actually applied to the product. They are therefore inserted into the cycle at the user specified strobe offset time.

When a miscompare occurs, a message is issued to the Verilog log file indicating the type of miscompare, the pattern number, the expected responses, and the simulated responses.

Parallel Cycle Miscompares

For a parallel test cycle, comparisons are done at product POs. An example miscompare message of this type is shown below.

```
PO miscompare(s) at pattern:  2.1.1.2.1.4 at Time:  72
     Expected: 0     Simulated: 1     On Output: DO2
     Expected: 0     Simulated: X     On Output: DO4
```

This example indicates that a PO miscompare occurred at pattern number 2.1.1.2.1.4 (this is interpreted as experiment 2, test section 1, tester loop 1, test procedure 2, test sequence 1, and pattern 4). Also indicated are the expected values, as calculated by Encounter Test simulation, the simulated values, as calculated by Verilog and the output pin name in error.

Values are retained in each variable (`stim_PIs`, `resp_POs`, `stim_SLs`, etc.) until explicitly reset. For example, the values for a `stim_PI` come from the prior assignment. Note that some of the scan tasks (preconditioning is one example) will change the values in `stim_PIs` to establish the scan state. These values are left in `stim_PIs`, i.e., not reset back to their value prior to scan. Encounter Test leaves the scan state values on PIs after a scan operation i.e., the `stim_PI` is permanently changed by the preconditioning assignments.

Scan Cycle Miscompares

For scan cycle miscompares, comparisons are performed at scan-out POs for the *serial* format. For the *parallel* format, comparisons are performed at the flop or latch outputs however, the failure is presented as if it took place at the scan-out POs. The provides a common miscompare message format. An example of a scan miscompare is shown below.

```
SO miscompare(s) at pattern: 2.1.1.4.1.1, Bit: 5, Scan Section: Scan_Sequence at Time:  8
     Expected: 0     Simulated: 1     On Output: SO1
```

This example shows that a miscompare occurred at pattern 2.1.1.4.1.1. It also shows that bit 5 was the failing bit position, relative to the scan-out PO and the applicable scan section name. Again, the expected value, simulated value, and failing scan-out PO are shown.

A

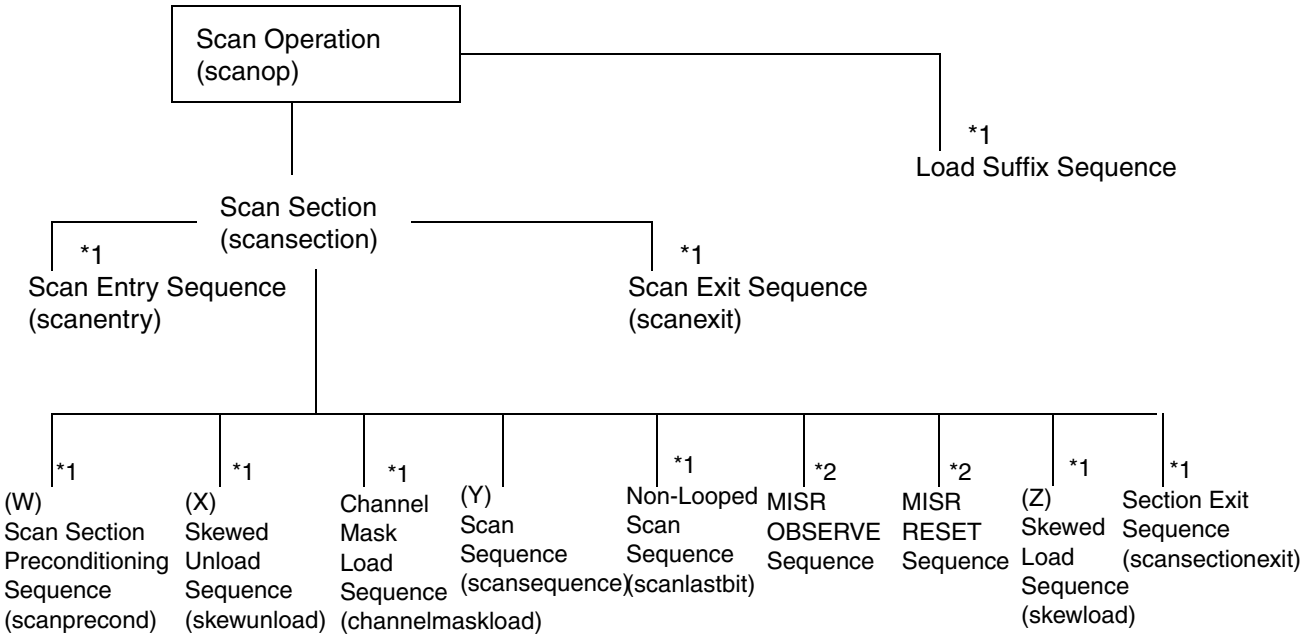
Scan Operation

This chapter covers Scan Operation Structure on page 197.

Scan Operation Structure

The scan operation has the following structure:

Figure A-1 Scan Operation Structure



NOTES: *1 denotes the sequnce is optional.

*2 denotes that this sequence is present only for an On-Product MISR test mode

Scan_Load event= (W) +(Y)
Skewed_Scan_Load event= (W) + (Y) + (Z)
Scan_Unload event= (W) + (Y)
Skewed_Scan_Unload event= (W) + (X) + (Y)
Channel_Scan 0 event= (W) + (Y)
Channel_Scan (skewed_load) event= (W) + (Y)
Channel_Scan (skewed_unload) event=(W) + (X) + (Y)
Channel_Scan (skewed_unload,skewed_load) event= (W) + (X) + (Y) + (Z)

B

Encounter Test Pattern Data Examples

Examples of pattern data are included for these formats:

- Vector
- Node List

Examples of these types of test pattern data are included:

- LSSD Sequence Definition
- WRPT
- LBIST

Vector Format Example

Following is an example Vector Format. The commented lines at the beginning of the file are the vector correspondence data.

```
TBDpatt_Format (mode=vector, model_entity_form=name);
#
#Vector_Correspondence
#
# Note:
#    If the original position of any latch is changed within the Scan_Load or
#    Meas_Latch vectors below, it is possible that some of the information
#    included in the Controllable/Observable Register commentary will no longer be
#    true. The potentially incorrect fields would include First_Stim_Bit,
#    Last_Stim_Bit, First_Meas_Bit and Last_Meas_Bit.
#
# Legend:
#
# tf    => the test function for the corresponding primary input/output pin
#            AC => A shift clock
#            AS => A shift and system clock
#            BC => B shift clock
#            BDY => boundary (test pin -- data)
#            BI => bi-directional inhibit
#            BS => B shift and system clock
#            CI => clock isolation
#            CTL => boundary (test pin -- control)
#            EC => shift clock for edge-sensitive flip-flops
```

```
#            ES => clock for both shift and system function of edge-sensitive
#                  flip-flops.
#            LH => linehold
#            ME => MISR enable
#            OI => output inhibit
#            PC => P clock
#            PS => P and system clock
#            SC => system clock
#            SG => scan gate
#            SI => scan-in
#            SO => scan-out
#            TI => test inhibit
#
#  index  => TestBench model pin index
#
#
# PI:
#     (PI 1 = "Pin.f.l.lsrfh0b.nl.a0",   #  index = 0
#      PI 2 = "Pin.f.l.lsrfh0b.nl.a1",   #  index = 1
#      PI 3 = "Pin.f.l.lsrfh0b.nl.a2",   #  index = 2
#      PI 4 = "Pin.f.l.lsrfh0b.nl.a3",   #  index = 3  tf =  SI
#      PI 5 = "Pin.f.l.lsrfh0b.nl.a4",   #  index = 4  tf = +SG
#      PI 6 = "Pin.f.l.lsrfh0b.nl.a5",   #  index = 5
#      PI 7 = "Pin.f.l.lsrfh0b.nl.a6",   #  index = 6  tf = -AS
#      PI 8 = "Pin.f.l.lsrfh0b.nl.a7",   #  index = 7  tf = +SG
#      PI 9 = "Pin.f.l.lsrfh0b.nl.a8",   #  index = 8  tf = -SG
#      PI 10 = "Pin.f.l.lsrfh0b.nl.a9")   #  index = 9  tf = -BC
#
# PO:
#      (PO 1 = "Pin.f.l.lsrfh0b.nl.00",   #  index = 10
#       PO 2 = "Pin.f.l.lsrfh0b.nl.01",   #  index = 11
#      PO 3 = "Pin.f.l.lsrfh0b.nl.02")   #  index = 12  tf =  SO

#
# Scan Chain Definition
#  Legend:
#    Load_Node    => Pin where logic values are placed for transfer into a latch
#                      via the load operation (e.g. a scan-in primary input).
#    Unload_Node  => Pin where latch logic values appear as the result of an unload
#                      operation (e.g. a scan-out primary output).
#    index        => TestBench model pin index for load/unload nodes or block index
#                      for stim/measure latches.
#    Load_Sect      => The id of the load section.
#    Unload_Sect    => The id of the unload section.
#    Bit_Length     => The number of bit positions in stim/observable register.
#    Number_Of_RSLs => The number of Representative Stim Latches in the
controllable register.
#    Number_Of_SSLs => The number of Skewed Stim Latches in the controllable
register.
#    Number_Of_RMLs => The number of Representative Measure Latches in the
observable register.
#    First_Stim_Bit => The bit position in the Scan_Load vector where RSL values
#                      for this scan chain begin.
#    Last_Stim_Bit  => The bit position in the Scan_Load vector where RSL values
#                      for this scan chain stop.
#    First_Meas_Bit => The bit position in the Meas_Latch vector where RML values
#                      for this scan chain begin.
#    Last_Meas_Bit  => The bit position in the Meas_Latch vector where RML values
#                       for this scan chain stop.
#
```

```
# Controllable Register 1:
#    Load_Node = "Pin.f.l.lsrfh0b.nl.a3"   index = 3
#    Load_sect = 6
#    Bit_Length = 2
#    Number_Of_RSLs = 2
#    Number_of_SSLs = 1
#    First_Stim_Bit = 1  Last_Stim_Bit = 2
#
# Observable Register 1:
#    Unload_Node = "Pin.f.l.lsrfh0b.nl.02"   index = 12
#    Unload_sect = 6
#    Bit_Length = 2
#    Number_Of_RMLs = 2
#    First_Meas_Bit = 1  Last_Meas_Bit = 2
#
#  RSL    => Representative Stim Latch
#  SSL    => Skewed Stim Latch
#  RML    => Representative Measure Latch
#  SR     => The id of the controllable register which includes this latch.
#            This id can be correlated to the scan chain definition
#            information listed above.
#  MR     => The id of the observable register which includes this latch.
#            This id can be correlated to the scan chain definition
#            information listed above.
#  pos    => Position in the scan chain which this latch occupies.
#             For a controllable register, the first latch which receives a
#            value from the load node is in position 1 (i.e., latch
#            closest to the load node).  For a observable register, the
#            latch whose value reaches the unload node first is in
#            position 1 (i.e., latch closest to the unload node).
#  invert => "Yes" means there is inversion in the scan chain
#            between this latch and the scan chain I/O pin.  For
#            a stim latch, the inversion is with respect to the scan
#            chain input pin;  for a measure latch, the inversion
#            is with respect to the scan chain output pin.
#            "No" means there is no inversion between the scan chain
#            I/O pin and the latch.
#
# Scan_Load:
# (RSL 1 = "Block.f.l.lsrfh0b.nl.srf10hm.021",  #  SR = 1  pos = 1  index = 48
invert = no
#    RSL 2 = "Block.f.l.lsrfh0b.nl.srf10hn.021",  #  SR = 1  pos = 2  index = 56
invert = no
#    SSL 3 = "Block.f.l.lsrfh0b.nl.srf10hm.014")  #  SR = 1  pos = 1  index = 47
invert = no
#
# Meas_Latch:
#    (RML 1 = "Block.f.l.lsrfh0b.nl.srf10hn.021",  #  MR = 1  pos = 1  index = 56
invert = no
#    RML 2 = "Block.f.l.lsrfh0b.nl.srf10hm.021")  #  MR = 1  pos = 2  index = 48
invert = no
#    ;                                            # End of vector correspondence

[ Experiment tg 1;
[ Test_Section 1.1 (tester_termination = 0,  termination_domination = tester,
        test_section_type = scan, test_type = static);
  [ Tester_Loop 1.1.1 ();
   [ Test_Procedure 1.1.1.1 (type = init);
    [ Test_Sequence 1.1.1.1.1 (type = init);
     [ Pattern 1.1.1.1.1.1 (pattern_type = static);
       Event 1.1.1.1.1.1.1   Stim_PI ():
```

```
     ] Pattern 1.1.1.1.1.1;
   ] Test_Sequence 1.1.1.1.1;
   ] Test_Procedure 1.1.1.1;
   [ Test_Procedure 1.1.1.2 (static_faults = 76, percent_static_faults =
46.060608);
     [ Test_Sequence 1.1.1.2.1 ();
     [ Pattern 1.1.1.2.1.1 (pattern_type = static);
        Event 1.1.1.2.1.1.1  Scan_Load ():
10;
     ] Pattern 1.1.1.2.1.1;
     [ Pattern 1.1.1.2.1.2 (pattern_type = static);
        Event 1.1.1.2.1.2.1  Stim_PI ():
0001100100;
     ] Pattern 1.1.1.2.1.2;
     [ Pattern 1.1.1.2.1.3 (pattern_type = static);
        Event 1.1.1.2.1.3.1  Pulse ():
"Pin.f.l.lsrfh0b.nl.a6"=+ ;
        Event 1.1.1.2.1.3.2  Pulse ():
"Pin.f.l.lsrfh0b.nl.a9"=+ ;
     ] Pattern 1.1.1.2.1.3;
     [ Pattern 1.1.1.2.1.4 (pattern_type = static);
        Event 1.1.1.2.1.4.1  Measure_PO ():
111;
     ] Pattern 1.1.1.2.1.4;
     [ Pattern 1.1.1.2.1.5 (pattern_type = static);
        Event 1.1.1.2.1.5.1  Stim_PI ():
0000100100;
     ] Pattern 1.1.1.2.1.5;
     [ Pattern 1.1.1.2.1.6 (pattern_type = static);
        Event 1.1.1.2.1.6.1  Pulse ():
"Pin.f.l.lsrfh0b.nl.a6"=+ ;
        Event 1.1.1.2.1.6.2  Pulse ():
"Pin.f.l.lsrfh0b.nl.a9"=+ ;
     ] Pattern 1.1.1.2.1.6;
     [ Pattern 1.1.1.2.1.7 (pattern_type = static);
        Event 1.1.1.2.1.7.1  Measure_PO ():
101;
     ] Pattern 1.1.1.2.1.7;
     [ Pattern 1.1.1.2.1.8 (pattern_type = static);
        Event 1.1.1.2.1.8.1  Stim_PI ():
1010100100;
     ] Pattern 1.1.1.2.1.8;
     [ Pattern 1.1.1.2.1.9 (pattern_type = static);
        Event 1.1.1.2.1.9.1  Pulse ():
"Pin.f.l.lsrfh0b.nl.a6"=+ ;
        Event 1.1.1.2.1.9.2  Pulse ():
"Pin.f.l.lsrfh0b.nl.a9"=+ ;
     ] Pattern 1.1.1.2.1.9;
     [ Pattern 1.1.1.2.1.10 (pattern_type = static);
        Event 1.1.1.2.1.10.1  Measure_PO ():
110;
     ] Pattern 1.1.1.2.1.10;
     [ Pattern 1.1.1.2.1.11 (pattern_type = static);
        Event 1.1.1.2.1.11.1  Stim_PI ():
1011100100;
     ] Pattern 1.1.1.2.1.11;
     [ Pattern 1.1.1.2.1.12 (pattern_type = static);
        Event 1.1.1.2.1.12.1  Pulse ():
"Pin.f.l.lsrfh0b.nl.a6"=+ ;
        Event 1.1.1.2.1.12.2  Pulse ():
"Pin.f.l.lsrfh0b.nl.a9"=+ ;
```

```
     ] Pattern 1.1.1.2.1.12;
     [ Pattern 1.1.1.2.1.13 (pattern_type = static);
        Event 1.1.1.2.1.13.1  Measure_PO ():
110;
     ] Pattern 1.1.1.2.1.13;
     [ Pattern 1.1.1.2.1.14 (pattern_type = static);
        Event 1.1.1.2.1.14.1  Scan_Unload ():
01;
     ] Pattern 1.1.1.2.1.14;
    ] Test_Sequence 1.1.1.2.1;
   ] Test_Procedure 1.1.1.2;
  ] Tester_Loop 1.1.1;
 ] Test_Section 1.1;
] Experiment tg 1;
# experiments = 1
# test sections = 1
# tester loops = 1
# test procedures = 2
# test sequences = 2
# patterns = 15
# events = 19
```

Node List Format Example

Following is an example Node List format.

```
TBDpatt_Format (mode=node, model_entity_form=name);
[ Experiment tg 1;
 [ Test_Section 1.1 (tester_termination = 0,  termination_domination = tester,
        test_section_type = scan, test_type = static);
  [ Tester_Loop 1.1.1 ();
   [ Test_Procedure 1.1.1.1 (type = init);
    [ Test_Sequence 1.1.1.1.1 (type = init);
    [ Pattern 1.1.1.1.1.1 (pattern_type = static);
      Event 1.1.1.1.1.1.1   Stim_PI ():
"Pin.f.l.lsrfh0b.nl.a6"=0
"Pin.f.l.lsrfh0b.nl.a9"=0 ;
    ] Pattern 1.1.1.1.1.1;
    ] Test_Sequence 1.1.1.1.1;
   ] Test_Procedure 1.1.1.1;
   [ Test_Procedure 1.1.1.2 ();
    [ Test_Sequence 1.1.1.2.1 ();
    [ Pattern 1.1.1.2.1.1 (pattern_type = static);
       Event 1.1.1.2.1.1.1   Scan_Load ():
"Block.f.l.lsrfh0b.nl.srf10hm.021"=1
"Block.f.l.lsrfh0b.nl.srf10hn.021"=0 ;
     ] Pattern 1.1.1.2.1.1;
     [ Pattern 1.1.1.2.1.2 (pattern_type = static);
        Event 1.1.1.2.1.2.1   Stim_PI ():
"Pin.f.l.lsrfh0b.nl.a4"=1
"Pin.f.l.lsrfh0b.nl.a6"=0
"Pin.f.l.lsrfh0b.nl.a7"=1
"Pin.f.l.lsrfh0b.nl.a9"=0 ;
     ] Pattern 1.1.1.2.1.2;
     [ Pattern 1.1.1.2.1.3 (pattern_type = static);
        Event 1.1.1.2.1.3.1   Pulse ():
"Pin.f.l.lsrfh0b.nl.a6"=+ ;
```

```
        Event 1.1.1.2.1.3.2   Pulse ():
"Pin.f.l.lsrfh0b.nl.a9"=+ ;
     ] Pattern 1.1.1.2.1.3;
     [ Pattern 1.1.1.2.1.4 (pattern_type = static);
        Event 1.1.1.2.1.4.1   Measure_PO ():
"Pin.f.l.lsrfh0b.nl.00"=1
"Pin.f.l.lsrfh0b.nl.01"=1
"Pin.f.l.lsrfh0b.nl.02"=1 ;
     ] Pattern 1.1.1.2.1.4;
     [ Pattern 1.1.1.2.1.5 (pattern_type = static);
        Event 1.1.1.2.1.5.1   Stim_PI ():
"Pin.f.l.lsrfh0b.nl.a4"=1
"Pin.f.l.lsrfh0b.nl.a6"=0
"Pin.f.l.lsrfh0b.nl.a7"=1
"Pin.f.l.lsrfh0b.nl.a9"=0 ;
     ] Pattern 1.1.1.2.1.5;
     [ Pattern 1.1.1.2.1.6 (pattern_type = static);
        Event 1.1.1.2.1.6.1   Pulse ():
"Pin.f.l.lsrfh0b.nl.a6"=+ ;
        Event 1.1.1.2.1.6.2   Pulse ():
"Pin.f.l.lsrfh0b.nl.a9"=+ ;
     ] Pattern 1.1.1.2.1.6;
     [ Pattern 1.1.1.2.1.7 (pattern_type = static);
        Event 1.1.1.2.1.7.1   Measure_PO ():
"Pin.f.l.lsrfh0b.nl.00"=1
"Pin.f.l.lsrfh0b.nl.01"=0
"Pin.f.l.lsrfh0b.nl.02"=1 ;
     ] Pattern 1.1.1.2.1.7;
     [ Pattern 1.1.1.2.1.8 (pattern_type = static);
        Event 1.1.1.2.1.8.1   Stim_PI ():
"Pin.f.l.lsrfh0b.nl.a4"=1
"Pin.f.l.lsrfh0b.nl.a6"=0
"Pin.f.l.lsrfh0b.nl.a7"=1
"Pin.f.l.lsrfh0b.nl.a9"=0 ;
     ] Pattern 1.1.1.2.1.8;
     [ Pattern 1.1.1.2.1.9 (pattern_type = static);
        Event 1.1.1.2.1.9.1   Pulse ():
"Pin.f.l.lsrfh0b.nl.a6"=+ ;
        Event 1.1.1.2.1.9.2   Pulse ():
"Pin.f.l.lsrfh0b.nl.a9"=+ ;
     ] Pattern 1.1.1.2.1.9;
     [ Pattern 1.1.1.2.1.10 (pattern_type = static);
        Event 1.1.1.2.1.10.1   Measure_PO ():
"Pin.f.l.lsrfh0b.nl.00"=1
"Pin.f.l.lsrfh0b.nl.01"=1
"Pin.f.l.lsrfh0b.nl.02"=0 ;
     ] Pattern 1.1.1.2.1.10;
     [ Pattern 1.1.1.2.1.11 (pattern_type = static);
        Event 1.1.1.2.1.11.1   Stim_PI ():
"Pin.f.l.lsrfh0b.nl.a4"=1
"Pin.f.l.lsrfh0b.nl.a6"=0
"Pin.f.l.lsrfh0b.nl.a7"=1
"Pin.f.l.lsrfh0b.nl.a9"=0 ;
     ] Pattern 1.1.1.2.1.11;
     [ Pattern 1.1.1.2.1.12 (pattern_type = static);
        Event 1.1.1.2.1.12.1   Pulse ():
"Pin.f.l.lsrfh0b.nl.a6"=+ ;
        Event 1.1.1.2.1.12.2   Pulse ():
"Pin.f.l.lsrfh0b.nl.a9"=+ ;
     ] Pattern 1.1.1.2.1.12;
     [ Pattern 1.1.1.2.1.13 (pattern_type = static);
```

```
        Event 1.1.1.2.1.13.1   Measure_PO ():
"Pin.f.l.lsrfh0b.nl.00"=1
"Pin.f.l.lsrfh0b.nl.01"=1
"Pin.f.l.lsrfh0b.nl.02"=0 ;
     ] Pattern 1.1.1.2.1.13;
     [ Pattern 1.1.1.2.1.14 (pattern_type = static);
        Event 1.1.1.2.1.14.1   Scan_Unload ():
"Block.f.l.lsrfh0b.nl.srf10hm.021"=1
"Block.f.l.lsrfh0b.nl.srf10hn.021"=0 ;
     ] Pattern 1.1.1.2.1.14;
    ] Test_Sequence 1.1.1.2.1;
   ] Test_Procedure 1.1.1.2;
  ] Tester_Loop 1.1.1;
 ] Test_Section 1.1;
 [ Test_Section 1.2 (tester_termination = 0,   termination_domination = tester,
        test_section_type = logic, test_type = dynamic);
  [ Tester_Loop 1.2.1 ();
   [ Test_Procedure 1.2.1.1 (type = init);
    [ Test_Sequence 1.2.1.1.1 (type = init);
     [ Pattern 1.2.1.1.1.1 (pattern_type = static);
        Event 1.2.1.1.1.1.1   Stim_PI ():
"Pin.f.l.lsrfh0b.nl.a6"=0
"Pin.f.l.lsrfh0b.nl.a9"=0 ;
     ] Pattern 1.2.1.1.1.1;
    ] Test_Sequence 1.2.1.1.1;
   ] Test_Procedure 1.2.1.1;
   [ Test_Procedure 1.2.1.2 ();
    [ Test_Sequence 1.2.1.2.1 ();
     [ Pattern 1.2.1.2.1.1 (pattern_type = static);
        Event 1.2.1.2.1.1.1   Skewed_Scan_Load ():
"Block.f.l.lsrfh0b.nl.srf10hm.021"=0
"Block.f.l.lsrfh0b.nl.srf10hn.021"=1
"Block.f.l.lsrfh0b.nl.srf10hm.014"=1 ;
     ] Pattern 1.2.1.2.1.1;
     [ Pattern 1.2.1.2.1.2 (pattern_type = static);
        Event 1.2.1.2.1.2.1   Stim_PI ():
"Pin.f.l.lsrfh0b.nl.a4"=1
"Pin.f.l.lsrfh0b.nl.a6"=0
"Pin.f.l.lsrfh0b.nl.a7"=1
"Pin.f.l.lsrfh0b.nl.a9"=0 ;
     ] Pattern 1.2.1.2.1.2;
     [ Pattern 1.2.1.2.1.3 (pattern_type = dynamic);
        Event 1.2.1.2.1.3.1   ulse (timed_type = release):
"Pin.f.l.lsrfh0b.nl.a9"=+ ;
        Event 1.2.1.2.1.3.2   Stim_PI (timed_type = release):
"Pin.f.l.lsrfh0b.nl.a4"=0
"Pin.f.l.lsrfh0b.nl.a6"=0
"Pin.f.l.lsrfh0b.nl.a7"=1
"Pin.f.l.lsrfh0b.nl.a9"=0 ;
        Event 1.2.1.2.1.3.3   Measure_PO (timed_type = capture):
"Pin.f.l.lsrfh0b.nl.00"=1
"Pin.f.l.lsrfh0b.nl.01"=1
"Pin.f.l.lsrfh0b.nl.02"=0 ;
     ] Pattern 1.2.1.2.1.3;
    ] Test_Sequence 1.2.1.2.1;
    [ Test_Sequence 1.2.1.2.2 ();
     [ Pattern 1.2.1.2.2.1 (pattern_type = static);
        Event 1.2.1.2.2.1.1   Skewed_Scan_Load ():
"Block.f.l.lsrfh0b.nl.srf10hm.021"=1
"Block.f.l.lsrfh0b.nl.srf10hn.021"=0
"Block.f.l.lsrfh0b.nl.srf10hm.014"=1 ;
```

```
     ] Pattern 1.2.1.2.2.1;
     [ Pattern 1.2.1.2.2.2 (pattern_type = static);
        Event 1.2.1.2.2.2.1   Stim_PI ():
"Pin.f.l.lsrfh0b.nl.a4"=1
"Pin.f.l.lsrfh0b.nl.a6"=0
"Pin.f.l.lsrfh0b.nl.a7"=1
"Pin.f.l.lsrfh0b.nl.a9"=0 ;
     ] Pattern 1.2.1.2.2.2;
     [ Pattern 1.2.1.2.2.3 (pattern_type = dynamic);
        Event 1.2.1.2.2.3.1   Pulse (timed_type = release):
"Pin.f.l.lsrfh0b.nl.a9"=+ ;
        Event 1.2.1.2.2.3.2   Stim_PI (timed_type = release):
"Pin.f.l.lsrfh0b.nl.a4"=1
"Pin.f.l.lsrfh0b.nl.a6"=0
"Pin.f.l.lsrfh0b.nl.a7"=1
"Pin.f.l.lsrfh0b.nl.a9"=0 ;
        Event 1.2.1.2.2.3.3   Measure_PO (timed_type = capture):
"Pin.f.l.lsrfh0b.nl.00"=1
"Pin.f.l.lsrfh0b.nl.01"=1
"Pin.f.l.lsrfh0b.nl.02"=1 ;
     ] Pattern 1.2.1.2.2.3;
    ] Test_Sequence 1.2.1.2.2;
   ] Test_Procedure 1.2.1.2;
  ] Tester_Loop 1.2.1;
 ] Test_Section 1.2;
] Experiment tg 1;
# experiments = 1
# test sections = 2
# tester loops = 2
# test procedures = 4
# test sequences = 5
# patterns = 22
# events = 30
```

AC Example

```
TBDpatt_Format (mode=vector, model_entity_form=name);
#
#Vector_Correspondence
#
# Legend:
#
#  tf     => the test function for the corresponding primary input/output pin
#            AC => A shift clock
#            AS => A shift and system clock
#            BC => B shift clock
#            BDY => boundary (test pin -- data)
#            BI => bi-directional inhibit
#            BS => B shift and system clock
#            CI => clock isolation
#            CTL => boundary (test pin -- control)
#            EC => shift clock for edge-sensitive flip-flops
#            ES => clock for both shift and system function of edge-sensitive
#                  flip-flops.
#            LH => linehold
#            ME => MISR enable
#            OI => output inhibit
#            PC => P clock
#            PS => P and system clock
```

```
#            SC => system clock
#            SG => scan gate
#            SI => scan-in
#            SO => scan-out
#            TI => test inhibit
#
#  index  => TestBench model pin index
#
#
# PI:
#      (PI 1 = "Pin.f.l.omniTest.nl.PIN0",   #  index = 0
#      PI 2 = "Pin.f.l.omniTest.nl.PIN1",   #  index = 1
#      PI 3 = "Pin.f.l.omniTest.nl.PIN2",   #  index = 2
#      PI 4 = "Pin.f.l.omniTest.nl.input1")   #  index = 3
#
# PO:
#      (PO 1 = "Pin.f.l.omniTest.nl.output")   #  index = 4
#      ;                                       # End of vector correspondence

# Test Pattern Audits Summary:
# ----------------------------
# Experiment Name =  acexp
#                                                                    Status
#                                                                    ------
# Model errors exist.....................................................No
# Test Mode errors exist which could result in pessimistic results......No
# Test Mode errors exist which could result in bad test data............No
# One or more test patterns had a Test Inhibit overridden...............No
# One or more test patterns had multiple clocks away from stability.....No
# A value of X propagated into a signature register.....................No
# The value specified for globalterm violated the TDR globalterm
specification....Yes
# Number of Hard 3-state conflicts.....................................0
# Number of Soft 3-state conflicts.....................................0
# Number of Unknown conflicts..........................................0
# SIMPATS value not between min/max for running signatures..............No
# LAPGS option specified in TDR overridden..............................No
# Orthogonal patterns found during simulation exist in test data........No
# Good Machine oscillations found during simulation.....................No
# Test Patterns were simulated with risky keepers (possible glitches)...No
# The Scan Chain test is included.......................................No
# The Driver and Receiver test is included..............................No
# The Iddq test is included.............................................No
# The Interconnect tests are included...................................No
# No orthogonality checks performed during simulation...................No
# Audit Code for Test Generation.......................................0
# Audit Code for Simulation............................................0
# Audit Code for WRP...................................................0
# Audit Code for LBIST.................................................0
# Test Section types included:
#   Logic............................Yes
#   Logic WRP........................No
#   Logic LBIST......................No
#   Flush............................No
#   Scan.............................No
#   Driver/Receiver..................No
#   Macro............................No
#   Iddq.............................No
#   IEEE 1149.1 Integrity Test.......No
#   ICT Stuck Driver.................No
```

```
#   ICT Stuck Driver Diagnostic......No
#   ICT log of N plus 2.............No
#   ICT 2 times log of N............No
#   ICT N plus 1....................No
#   IOWRAP Stuck Driver.............No
#   IOWRAP log of N plus 2..........No
#   IOWRAP 2 times log of N.........No
#   IOWRAP N plus 1.................No
#   Path............................No
#   Channel Scan....................No

[ Experiment acexp 1;
 [ Define_Sequence TBautoLogicSeq0 19951002204813 1.1 (test);
  [ Timing_Data  ( automatic, number_of_cycles = 1,
         early = ( 1.000000, 0.000000, 0.000000) ,
         late = ( 1.000000, 0.000000, 1.0000) )
   [ Pin_Timing:
    tester_cycle 0.000000 ps cycle 0;
    stim_PIs RorF 950.000000 ps cycle 0 event 1 "Pin.f.l.omniTest.nl.PIN0";
    stim_PIs RorF 950.000000 ps cycle 0 event 1 "Pin.f.l.omniTest.nl.PIN1";
    stim_PIs RorF 950.000000 ps cycle 0 event 1 "Pin.f.l.omniTest.nl.PIN2";
    stim_PIs RorF 950.000000 ps cycle 0 event 1 "Pin.f.l.omniTest.nl.input1";
    PO_strobe RorF 1330.000000 ps cycle 0 event 2 "Pin.f.l.omniTest.nl.output";
    tester_cycle 2280.000000 ps cycle 1;
   ] Pin_Timing;
  ] Timing_Data 1.1.1 ;
  [ Pattern 1.1.1 (pattern_type = static);
     Event 1.1.1.1   Stim_PI ():
  ] Pattern 1.1.1;
  [ Pattern 1.1.2 (pattern_type = dynamic);
     Event 1.1.2.1   Stim_PI (timed_type = release):
     Event 1.1.2.2   Measure_PO (timed_type = capture):
x;
  ] Pattern 1.1.2;
 ] Define_Sequence TBautoLogicSeq0 1.1;
 [ Test_Section 1.1 (tester_termination = 1,  termination_domination = tester,
         test_section_type = logic, test_type = dynamic);
  [ Tester_Loop 1.1.1 ();
   [ Test_Procedure 1.1.1.1 (type = init);
    [ Test_Sequence 1.1.1.1.1 ();
    ] Test_Sequence 1.1.1.1.1;
   ] Test_Procedure 1.1.1.1;
   [ Test_Procedure 1.1.1.2 (static_faults = 12, percent_static_faults = 54.545456,
   dynamic_faults = 8, percent_dynamic_faults = 28.571246);
    [ Test_Sequence 1.1.1.2.1 ();
     [ SeqDef=(TBautoLogicSeq0,"19951002204813") ] SeqDef;
     [ Timing_ID = 1 ];
     [ Pattern 1.1.1.2.1.1 (pattern_type = static);
        Event 1.1.1.2.1.1.1   Stim_PI ():
0100;
        Event 1.1.1.2.1.1.2   Measure_PO ():
0;
     ] Pattern 1.1.1.2.1.1;
     [ Pattern 1.1.1.2.1.2 (pattern_type = dynamic);
        Event 1.1.1.2.1.2.1   Stim_PI (timed_type = release):
1101;
        Event 1.1.1.2.1.2.2   Measure_PO (timed_type = capture):
1;
     ] Pattern 1.1.1.2.1.2;
    ] Test_Sequence 1.1.1.2.1;
   ] Test_Procedure 1.1.1.2;
```

```
  ] Tester_Loop 1.1.1;
 ] Test_Section 1.1;
] Experiment acexp 1;
# experiments = 1
# test sections = 1
# tester loops = 1
# test procedures = 6
# test sequences = 6
# patterns = 12
# events = 23
```

1149.1 Mode Initialization Example with User-Supplied Custom Scan Sequence

An example of what a user supplied custom scan sequence might look like for a test mode that scans using 1149.1 and uses the `Capture_DR` and `Update_DR` states:

```
TBDpatt_Format (mode=node, model_entity_form=name);

## This file is an example of custom sequence definitions that might be
## used for a test mode that is doing 1149.1 internal scan and uses the
## Capture_DR state to actually capture values into the scan latches and
## uses the Update_DR state to load parallel stable latches with values
## that were scanned in through the TDI-TDO path.  This is only an example
## and a real design might need to have additional latches included in
## the Force() event as well as other operations to correctly initialize
## the test mode.


# The Test Mode initialization sequence
#-------------------------------------------------------------
[Define_Sequence Mode_init (modeinit);
  [Pattern;
        Event Stim_PI ():      "enable"       =1
                                "tck"       =0
                                "tms"      =1
                                "trst"      =1                    ]Pattern;

  ## Reset the TAP state machine

  [Pattern;
        Event Stim_PI ():      "trst"      =1;
        Event Pulse ():      "tck"      =+;                      ]Pattern;
  [Pattern;
        Event Stim_PI ():      "trst"      =0;
        Event Pulse ():      "tck"      =+;                      ]Pattern;

  ## Be extra sure the state machine is in Test_Logic_Reset ;-)
  [Pattern (pattern_type=begin_loop); Event Repeat():5;          ]Pattern;
  [Pattern;
        Event Pulse ():         "tck"              =+ ;          ]Pattern;
  [Pattern (pattern_type=end_loop);                              ]Pattern;

  # We should be in Test_Logic_Reset state

  ## Move to Shift_IR and load internal scan instruction 10011
```

```
  # enter Run_Test_Idle state
  [Pattern;
        Event Stim_PI ():       "tms"             =0 ;
        Event Pulse ():         "tck"             =+ ;            ]Pattern;
  # enter Select_DR state
  [Pattern;
        Event Stim_PI ():       "tms"             =1 ;
        Event Pulse ():         "tck"             =+ ;            ]Pattern;
  # enter Select_IR state
  [Pattern;
        Event Pulse ():         "tck"             =+ ;            ]Pattern;
  # enter Capture_IR state
  [Pattern;
        Event Stim_PI ():       "tms"             =0 ;
        Event Pulse ():         "tck"             =+ ;            ]Pattern;
  # enter Shift_IR state
  [Pattern;
        Event Pulse ():         "tck"             =+ ;            ]Pattern;
  ## load internal scan instruction 10011  (loading right-most bits
first!!)
  [Pattern;
        Event Stim_PI ():       "tdi"             =1 ;
        Event Pulse ():         "tck"             =+ ;            ]Pattern;
  [Pattern;
        Event Stim_PI ():       "tdi"             =1 ;
        Event Pulse ():         "tck"             =+ ;            ]Pattern;
  [Pattern;
        Event Stim_PI ():       "tdi"             =0 ;
        Event Pulse ():         "tck"             =+ ;            ]Pattern;
  [Pattern;
        Event Stim_PI ():       "tdi"             =0 ;
        Event Pulse ():         "tck"             =+ ;            ]Pattern;
  [Pattern;
        Event Stim_PI ():       "tms"             =1 ;     # switch to
Exit1_IR state too!
        Event Stim_PI ():       "tdi"             =1 ;
        Event Pulse ():         "tck"             =+ ;            ]Pattern;
  # in Exit1_IR state

  # enter Update_IR state
  [Pattern;
        Event Pulse ():         "tck"             =+ ;            ]Pattern;
  # enter Select_DR state
  [Pattern;
        Event Pulse ():         "tck"             =+ ;            ]Pattern;
  # enter Capture_DR state
  [Pattern;
        Event Stim_PI ():       "tms"             =0 ;
        Event Pulse ():         "tck"             =+ ;            ]Pattern;
  # enter Shift-DR state
  [Pattern;
        Event Pulse ():         "tck"             =+ ;            ]Pattern;

  ## Note: The mode init sequence must leave the design in the same state
  ##       that every scan load/unload must start in.  That state is the
  ##       Shift_DR state whenever Capture_DR is being used!!!

]Define_Sequence Mode_init;
```

```
# Scan procedures
#-------------------------------------------------------------
[Define_Sequence Scan_setup (scanprecond);
   [Pattern;
        Event Stim_PI ():        "tms"             =0;

      ## Following Event forces machine into Shift-DR, which should
      ## match where we really are at the start of every scan!
      Event Force ():
        "chip.tap.state_out_ff3_slave"=1
        "chip.tap.state_out_ff2_slave"=0
        "chip.tap.state_out_ff1_slave"=1
        "chip.tap.state_out_ff0_slave"=0  ;
  ]Pattern;
]Define_Sequence Scan_setup;

  ## This sequence is executed once for each shift of the register, except
  ## for the last bit shift, which must be defined in the scanlastbit
sequence.
[Define_Sequence Shift-dr  (scansequence, repeat = 4);
   [Pattern;
        Event Measure_Scan_Data ():     "tdo";
        Event Set_Scan_Data ():         "tdi";
        Event Pulse ():                 "tck"    =+ ;            ]Pattern;
]Define_Sequence Shift-dr;

  ## This sequence shifts the last bit through the internal register and
moves
  ## from the Shift_DR state to the Exit1_DR state.
[Define_Sequence Exit1-dr (scanlastbit);
   [Pattern;
        Event Measure_Scan_Data ():     "tdo";
        Event Set_Scan_Data ():         "tdi";
        Event Stim_PI ():               "tms"     =1;
        Event Pulse ():                 "tck"     =+ ;            ]Pattern;
]Define_Sequence Exit1-dr;

  ## This sequence pretends to end the scan operation by exiting back
  ## to the TAP_TG_STATE of Capture_DR; however, we are really leaving
  ## the TAP state machine in the Update_DR state so that we can apply
  ## the Update_DR operation during the loadsuffix sequence.
[Define_Sequence Updt-dr (scansectionexit);
   [Pattern  (pattern_type= static);
#       Event Stim_PI ():       "tms"             =1 ;
        Event Pulse ():         "tck"             =+ ;            ]Pattern;
   [Pattern  (pattern_type= static);
        ##  Now in Update-DR State
        ## Following Event forces (jumpsp) the machine into Capture_DR
state
        ## We are actually leaving scan in the Update_DR state
      Event Force ():
        "chip.tap.state_out_ff3_slave"=1
        "chip.tap.state_out_ff2_slave"=0
        "chip.tap.state_out_ff1_slave"=0
        "chip.tap.state_out_ff0_slave"=0   ;
    Event Stim_PI ():                  "tms"     =0 ;            ]Pattern
;
```

```
  # Note that TMS must be left at its TestConstraint (TC) value of zero
(0) as
  # we exit the scan operation.

]Define_Sequence Updt-dr;


  ## We only support a single scan section when Update_DR is being used.
[Define_Sequence Scan_proc (scansection);
  [Pattern;
               Event Apply (): Scan_setup;
               Event Apply (): Shift-dr;
               Event Apply (): Exit1-dr;
               Event Apply (): Updt-dr;                        ]Pattern;
]Define_Sequence Scan_proc;

[Define_Sequence Scan_Op (scanop);
  [Pattern;    Event Apply (): Scan_proc;                      ]Pattern;
]Define_Sequence Scan_Op;



# The Load_S
uffix operation to load the Alternate Stim Latches (ASLs)
# Note: normally the loadsuffix sequence is optional, but when used to
#       apply the Update_DR operation of 1149.1, it must always be
#       included after every Scan_Load operation.
#-------------------------------------------------------------------
[Define_Sequence Load_Suffix  (loadsuffix);
   [Pattern;

      # First we must jump back to the Update_DR state which is where
      # the state machine really is...

      Event Force ():
        "chip.tap.state_out_ff3_slave"=0
        "chip.tap.state_out_ff2_slave"=0
        "chip.tap.state_out_ff1_slave"=1
        "chip.tap.state_out_ff0_slave"=0  ;

      # Now we pulse TCK and load the parallel latches and move to
Select_DR state.

      Event Stim_PI ():                 "tms"=1 ;
      Event Pulse ():                   "tck"=+ ;
]Pattern;

  # Now we pulse TCK and move to Capture_DR state.

  [Pattern;
        Event Stim_PI ():                 "tms"=0 ;
        Event Pulse ():                 "tck"=+ ;
]Pattern;

]Define_Sequence Load_Suffix;
```

C

WGL Pattern Data Examples

This section provides examples of WGL format which are typical for Deterministic Logic tests, Macro tests, LBIST tests, and ICT tests.

WGL Scan Vector Explanation and Examples

WGL Scanchain Definition

WGL language syntax requires all scan chains to be defined before being referenced. That is, either Scan_in or Scan_out pin should be defined though for some scan chains, both the pins may have both Scan_in and Scan_out pins defined as part of its structure.

Below is an example of a scan chain with both Scan_in and Scan_out defined:

```
scanchain
  "MREG_1_TB_EXTEST_CAP_UPDT_324BGA" [
    "JTAG_TDI",
    "top.io_CEN0_0.BSC_MERGED_364.TDO_net_reg.I0.dff",
     ...other scan flops...
    "top.io_IO_6_4.TBB_BSC_IO_6_4__0.TDO_net_reg.I0.dff",
    "JTAG_TDO"
  ];
end
```

Below is an example of scan chain definition with only scan_in pin:

```
scanchain
  "XOR_MASK_REG" [
    "Scan_In1",
    "M40.P01",
    ...other scan flops...
```

```
    "M01.M08.P01",
  ];
end
```

There can be one or multiple scan chains defined within a WGL scanchain declaration. It is important to note the following:

- Scan chains need not be of a uniform length.
- The defined scan chains may be a mixture of only scan_in, only scan_out, or both scan_in and scan_out.
- A scan chain can have an inversion represented by a ! between memory elements, as shown in "Deterministic WGL Pattern Data Examples" on page 217.

WGL Scanstate Definition

After a scan chain is defined, WGL allows the definition of scan input and scan output values that will be applied through scan vectors. These scan values are defined through the scanstate structure. Within each specific value entry, there is an explicit reference to an already defined scanchain. Below is an example that maps to the scanchain names defined above.

```
scanstate
  "SS.1.2.1.2.2.2.3" :=
     "MREG_1_TB_EXTEST_CAP_UPDT_324BGA" (
     111111111111110111111100001111111111111111100 ) ;
  "SS.1.2.1.2.3.2.3" :=
     "MREG_1_TB_EXTEST_CAP_UPDT_324BGA" (
     XXXXXX00XX00XX0000XX0000XXXX000XXXX000000000001 ) ;
  "SS.1.2.1.2.3.4.5" :=
     "XOR_MASK_REG" (
     110001000000000001 ) ;
end
```

There may be scan chains of different length and types existing within a design. Below are some things to consider:

- The number of logic values within each scanstate entry matches the number of flops/bits defined in the scanchain. The entries will be of different length.
- The values represent only the scan flops within the chain, and not the scan_in or scan_out pins (as one of these pins may be missing in the scan chain definition).
- Scanstate definition does not declares the values as scan shift input or scan shift output. This is done in the WGL scan vector.

- The values are defined as they appear in the latch or scanchain definition. Therefore if you want the value at the SI or SO pin then you need to look at the scanchain definition and consider any inversion.

WGL Scan Vectors

After a scanchain has been defined and scanstate values have been created, they are referenced within a WGL scan vector statement.

A sample scan vector statement is given below:

```
  ## NOTE: The "-" placeholder for the scan_in pin as the 5th entry
  ## of the second row (bidi pin as input row). Values for this pin
  ## come from the "input" vector.
  ##  NOTE the reference to the scanstate "SS.1.2.1.2.3.4.5". The values
  ##  in this variable are applied to the scan input pin of scanchain
  ##  XOR_MASK_REG.  That scan input pin is defined as
  ## "Scan_In1" in the scanchain  statement.
scan ( +, "scan_cycle" ) := [
  X X X 1 1 X X X X X 0 X X X 0
  Z Z Z 1 - Z 0 1 Z Z Z Z Z Z Z Z Z Z Z
  X X X X
  - - - - - - - - - - - - - - - - - - -  ],
  input  [ "XOR_MASK_REG" : "SS.1.2.1.2.3.4.5" ];

  ## NOT A scan vector thus all pins defined.  No "-" placeholder for
  ##  any of the scan_in or scan_out pins.
vector ( +, "scan_cycle" ) := [
  X X X 1 1 X X X X X 0 X X X 0
  Z Z Z 0 1 Z 0 1 Z Z Z Z Z Z Z Z Z Z Z
  X X X X
  - - - - - - - - - - - - - - - - - - - -];
## NOTE:  Even though the scanchain is defined with both an input
## and output scan pin there is only a scan shift output  in this vector.
## NOTE:  The scan_in pin (5th entry in second row) is now explicitly defined.
## IF this scan input pin was defined AND there was an "input" scan
## statement then there would be a conflict on what value to set this
## pin and thus an ERROR
## IF there was a Measure output value for the scan out pin this would
## conflict with the "output" scan statement and thus be an ERROR
scan   ( +, "scan_cycle" ) := [
  X X X 1 1 X X X X X 0 X X X 0
  Z Z Z 1 1 Z 0 1 Z Z Z Z Z Z Z Z Z Z Z
  X X X X
  - - - - - - - - - - - - - - - - - - -  ],
  output [ "MREG_1_TB_EXTEST_CAP_UPDT_324BGA" : "SS.1.2.1.2.3.2.3" ];
```

Scan vectors comprise of the following main parts:

- The parenthesis contains a reference to a timing template. The template referenced in the example is scan_cycle. This template contains the definition of the timing of input stimulus and output measures for every pin.

- The next section explicitly defines the value for every pin contacted by the tester. This is broken down by the following lines:
  - First line has the input pin values
  - Second line has the input value for bidi pins
  - Third line has output pin measure values
  - Fourth line has output measure values for bidi pins

  The order of contacted pins in this list is defined by the order in which they appear within the pattern MAIN statement.

  Note: Though every tester contacted pin is represented in the scan vector statement, a specific entry may not have an explicit logic value and may contain a "-" placeholder instead.

- The input and/or output scan statements. Each statement references a specific scan chain and scanstate variable.
  - The number of logic values in the scanstate variable must match the number of flops defined in the scanchain.
  - Any scan vector may have only input references, only output references, or both.
    - Chains referenced via input statement must have a scan_in pin defined.
    - Chains referenced via output statement must have a scan_out pin defined.
    - Within a scan vector, a chain with both scan_in and scan_out defined may be referenced by only an input statement, only an output statement, or both.
    - If a chain with both pins defined in the scanchain statement is referenced only via an input statement, then the scan_out pin value is provided in the scan vector pin section. If it is referenced only via an output statement then the scan_in pin value is provided in the scan vector pin section.

  Note: All defined scan chains need not be referenced in every scan vector statement. There will be chains that are not referenced because they are not relevant for the current operation. In such case, their scan in and scan out pins will be supplied logic values within the pin section of the scan vector. These pins are not undefined.

  The tester cycle count provided within comments in the example is incremented by the largest scan chain referenced within a specific scan vector. The cycle count is not expected to increase by the longest defined scan chain as in the XOR compression mask scan vector references. These are very short scan chains that are scanned separately from the functional logic scan chains.

Deterministic WGL Pattern Data Examples

This section provides examples of WGL format which are typical for all deterministically derived tests.

WGL Deterministic Signals File

The following is an example of a WGL deterministic signals file.

```
##************************************************************************##
##                              WGL SIGNALS FILE                          ##
##  Encounter(TM) Test and Diagnostics 3.0     Apr 01, 2005 (linux24 TDA30)  ##
##************************************************************************##
##                                                                        ##
##  FILE CREATED.............April 01, 2005 at 09:51:18                  ##
##                                                                        ##
##  PROJECT NAME.............lbc                                         ##
##                                                                        ##
##  TESTMODE.................lssd                                        ##
##                                                                        ##
##  TDR......................dummy_tester_lssd                            ##
##                                                                        ##
##  TEST PERIOD..............80       TEST STROBE TYPE..........edge      ##
##  TEST PULSE WIDTH.........8        TEST TIME UNITS...........ns        ##
##  TEST PI OFFSET...........0                                            ##
##  TEST BIDI OFFSET.........0                                            ##
##  TEST STROBE OFFSET.......72       X VALUE..................X         ##
##                                                                        ##
##  SCAN PERIOD..............80       SCAN STROBE TYPE..........edge      ##
##  SCAN PULSE WIDTH.........8        SCAN TIME UNITS...........ns        ##
##  SCAN PI OFFSET...........16                                           ##
##  SCAN BIDI OFFSET.........16                                           ##
##  SCAN STROBE OFFSET.......0        SCAN OVERLAP.............yes       ##
##                                                                        ##
##************************************************************************##

signal
      "A" : input;     ## pinName = A;  tf = -AC  ; testOffset = 8;  scanOffset = 24;
      "B" : input;    ## pinName = B;  tf = -BC  ; testOffset = 24;  scanOffset = 40;
      "C" : input;     ## pinName = C;  tf = -SC  ; testOffset = 8;  scanOffset = 16;
      "CS" : input;      ## pinName = CS; testOffset = 0;  scanOffset = 16;
      "DI1" : input;      ## pinName = DI1; testOffset = 0;  scanOffset = 16;
      "DI2" : input;      ## pinName = DI2; testOffset = 0;  scanOffset = 16;
      "DI3" : input;      ## pinName = DI3; testOffset = 0;  scanOffset = 16;
      "DI4" : input;      ## pinName = DI4; testOffset = 0;  scanOffset = 16;
      "ENABLE1" : input;      ## pinName = ENABLE1;  tf = +SE  ; testOffset = 0;
      "ENABLE2" : input;      ## pinName = ENABLE2;  tf = +SE  ; testOffset = 0;
      "ME" : input;      ## pinName = ME;  tf = +SE  ; testOffset = 0;
      "PS" : input;      ## pinName = PS; testOffset = 0;  scanOffset = 16;
      "SEL" : input;     ## pinName = SEL;  tf = -SE  ; testOffset = 0;  scanOffset
```

```
      "SI2" : input;      ## pinName = SI2;  tf =  SI  ; testOffset = 0;  scanOffset
      "SO1_BIDI" : bidir;       ## pinName = SO1_BIDI;  tf = ZSE   SO  BIDI ;
      "ST1" : input;      ## pinName = ST1;  tf = -SE  ; testOffset = 0;  scanOffset
      "ST2" : input;       ## pinName = ST2;  tf = +SE  ; testOffset
      "DO2" : output;       ## pinName = DO2;
      "DO3" : output;       ## pinName = DO3;
      "DO4" : output;       ## pinName = DO4;
      "SO1" : output;       ## pinName = SO1;
      "SO2" : output;       ## pinName = SO2;

 end

scancell
        "M00.M02.P01";
        "M00.M03.P01";
        "M00.M04.P01";
        "M00.M05.P01";
        "M01.M05.P01";
        "M01.M06.P01";
        "M01.M07.P01";
        "M01.M08.P01";
        "M02.M01.P01";
        "M02.M02.P01";
        "M02.M03.P01";
        "M02.M04.P01";
        "M03.M01.P01";
        "M03.M02.P01";
        "M03.M03.P01";
        "M03.M04.P01";
        "M04.M01.P01";
        "M04.M02.P01";
        "M04.M03.P01";
        "M04.M04.P01";
        "M05.M01.P01";
        "M05.M02.P01";
        "M05.M03.P01";
        "M05.M04.P01";
        "M40.P01";
end

scanchain
"MREG_1_FULLSCAN" [
        "SI1",
        "M40.P01",
        "M00.M02.P01",
        "M00.M03.P01",
        "M00.M04.P01",
        "M00.M05.P01",
        "M01.M05.P01",
        "M01.M06.P01",
        "M01.M07.P01",
        "M01.M08.P01",
        "SO1_BIDI"
        ];
 "MREG_2_FULLSCAN" [
        "SI2",
        "M02.M01.P01",
        "M02.M02.P01",
        "M02.M03.P01",
        "M02.M04.P01",
        "M03.M01.P01",
```

```
        "M03.M02.P01",
        "M03.M03.P01",
        "M03.M04.P01",
        "M04.M01.P01",
        "M04.M02.P01",
        "M04.M03.P01",
        "M04.M04.P01",
        "M05.M01.P01",
        "M05.M02.P01",
        "M05.M03.P01",
        "M05.M04.P01",
        "SO2_BIDI"
        ];
end
```

WGL LSSD Flush Test Example

The following is an example of a WGL LSSD flush test.

```
##*************************************************************************##
##                              WGL VECTOR FILE                            ##
##  Encounter(TM) Test and Diagnostics 3.0     Apr 01, 2005 (linux24 TDA30)  ##
##*************************************************************************##
##                                                                         ##
##  FILE CREATED.............April 01, 2005 at 09:51:17                   ##
##                                                                         ##
##  PROJECT NAME.............lbc                                          ##
##                                                                         ##
##  TESTMODE.................lssd                                         ##
##                                                                         ##
##  TDR......................dummy_tester_lssd                             ##
##                                                                         ##
##  TEST PERIOD..............80       TEST STROBE TYPE..........edge       ##
##  TEST PULSE WIDTH.........8        TEST TIME UNITS...........ns         ##
##  TEST PI OFFSET...........0                                             ##
##  TEST BIDI OFFSET.........0                                             ##
##  TEST STROBE OFFSET.......72       X VALUE..................X           ##
##                                                                         ##
##  SCAN PERIOD..............80       SCAN STROBE TYPE..........edge       ##
##  SCAN PULSE WIDTH.........8        SCAN TIME UNITS...........ns         ##
##  SCAN PI OFFSET...........16                                            ##
##  SCAN BIDI OFFSET.........16                                            ##
##  SCAN STROBE OFFSET.......0        SCAN OVERLAP.............yes        ##
##                                                                         ##
##  EXPERIMENT...............1                                             ##
##                                                                         ##
##  TEST SECTION.............1        TEST SECTION TYPE........flush       ##
##  TESTER TERMINATION.......0        TERMINATION DOMINATION....tester     ##
##                                                                         ##
##*************************************************************************##

waveform "WGL.lssd.flush.ex1.ts1"

        include "WGL.lssd.signals";
##*************************************************************************##
##                            TIMING DEFINITIONS                           ##
```

```
##*************************************************************************##

timeplate "test_cycle_stimclks" period 1280ns
        "A" := input [ 0ns:P, 8ns:S ];
        "B" := input [ 0ns:P, 24ns:S ];
        "C" := input [ 0ns:P, 8ns:S ];
        "CS" := input [ 0ns:S ];
        "DI1" := input [ 0ns:S ];
        "DI2" := input [ 0ns:S ];
        "DI3" := input [ 0ns:S ];
        "DI4" := input [ 0ns:S ];
        "ENABLE1" := input [ 0ns:S ];
        "ENABLE2" := input [ 0ns:S ];
        "ME" := input [ 0ns:S ];
        "PS" := input [ 0ns:S ];
        "SEL" := input [ 0ns:S ];
        "SI1" := input [ 0ns:S ];
        "SI2" := input [ 0ns:S ];
        "SO1_BIDI" := input [ 0ns:S ];
        "SO2_BIDI" := input [ 0ns:S ];
        "ST1" := input [ 0ns:S ];
        "ST2" := input [ 0ns:S ];
        "DO1" := output [ 0ns:X, 1152ns:Q'edge ];
        "DO2" := output [ 0ns:X, 1152ns:Q'edge ];
        "DO3" := output [ 0ns:X, 1152ns:Q'edge ];
        "DO4" := output [ 0ns:X, 1152ns:Q'edge ];
        "SO1" := output [ 0ns:X, 1152ns:Q'edge ];
        "SO1_BIDI" := output [ 0ns:X, 1152ns:Q'edge ];
        "SO2" := output [ 0ns:X, 1152ns:Q'edge ];
        "SO2_BIDI" := output [ 0ns:X, 1152ns:Q'edge ];
end
##*************************************************************************##
##                            DEFINE SCAN STATES                           ##
##*************************************************************************##

scanstate
end


##*************************************************************************##
##                              TEST VECTORS                               ##
##*************************************************************************##

pattern MAIN ( "A", "B", "C", "CS", "DI1", "DI2", "DI3", "DI4", "ENABLE1",
               "ENABLE2", "ME", "PS", "SEL", "SI1", "SI2", "ST1", "ST2",
               "SO1_BIDI":I, "SO2_BIDI":I,
               "DO1", "DO2", "DO3", "DO4", "SO1", "SO2",
               "SO1_BIDI":O, "SO2_BIDI":O )

##*************************************************************************##
##  TESTER LOOP...............1        PROCEDURES HAVE MEMORY....no        ##
##  TEST PROCEDURE............1        TYPE......................init      ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no        ##
##  TEST SEQUENCE.............1        TYPE......................init      ##
##*************************************************************************##

## Processing the Static: EVENT 1.1.1.1.1.1.1:  Stim_PI:
   vector ( +, "test_cycle_stimclks" ) := [
        0 0 0 X X X X X X X X X X X X X X
        Z Z
        X X X X X X
```

```
        - - ];

##*************************************************************************##
##  TEST PROCEDURE...........2        TYPE......................normal      ##
##  SLOW TO TURN OFF.........false    SEQUENCES HAVE MEMORY.....yes         ##
##  TEST SEQUENCE............1        TYPE......................normal      ##
##*************************************************************************##

## Processing the Static: EVENT 1.1.1.2.1.1.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.2.1.2.1:  Stim_PI:
##*************************************************************************##
##  TEST SEQUENCE............2        TYPE......................normal      ##
##*************************************************************************##

## Processing the Static: EVENT 1.1.1.2.2.1.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.2.2.2.1:  Stim_Clock:
## Processing the Static: EVENT 1.1.1.2.2.2.2:  Stim_Clock:
   vector ( +, "test_cycle_stimclks" ) := [
        1 1 0 0 0 0 0 0 1 1 1 0 0 X X 0 1
        Z Z
        X X X X X X
        - - ];
## Processing the Static: EVENT 1.1.1.2.2.2.3:  Stim_PI:
## Processing the Static: EVENT 1.1.1.2.2.2.4:  Measure_PO:
   vector ( +, "test_cycle_stimclks" ) := [
        1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1
        - -
        X X X X X X
        0 0 ];

##*************************************************************************##
##  TEST SEQUENCE............3        TYPE......................normal      ##
##*************************************************************************##

## Processing the Static: EVENT 1.1.1.2.3.1.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.2.3.1.2:  Measure_PO:
   vector ( +, "test_cycle_stimclks" ) := [
        1 1 0 0 0 0 0 0 1 1 1 0 0 1 1 0 1
        - -
        X X X X X X
        1 1 ];

##*************************************************************************##
##  TEST SEQUENCE............4        TYPE......................normal      ##
##*************************************************************************##

## Processing the Static: EVENT 1.1.1.2.4.1.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.2.4.1.2:  Measure_PO:
   vector ( +, "test_cycle_stimclks" ) := [
        1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1
        - -
        X X X X X X
        0 0 ];
##*************************************************************************##
##  TEST SEQUENCE............5        TYPE......................normal      ##
##*************************************************************************##

## Processing the Static: EVENT 1.1.1.2.5.1.1:  Stim_Clock:
## Processing the Static: EVENT 1.1.1.2.5.1.2:  Stim_Clock:
   vector ( +, "test_cycle_stimclks" ) := [
        0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1
```

```
        Z Z
        X X X X X X
        - - ];
## Inserted final non-scan Pattern
   vector ( +, "test_cycle_stimclks" ) := [
        0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1
        Z Z
        X X X X X X
        - - ];
end end
```

WGL Scan Chain Test Example

The following is an example of a WGL scan chain test.

```
    ****************************************************************************
##
##                                WGL VECTOR FILE                              ##
##  Encounter(TM) Test and Diagnostics 3.0     Apr 01, 2005 (linux24 TDA30)  ##
##*************************************************************************##
##                                                                           ##
##  FILE CREATED..............April 01, 2005 at 09:51:17                   ##
##                                                                           ##
##  PROJECT NAME..............lbc                                           ##
##                                                                           ##
##  TESTMODE..................lssd                                          ##
##                                                                           ##
##  TDR......................dummy_tester_lssd                              ##
##                                                                           ##
##  TEST PERIOD...............80       TEST STROBE TYPE..........edge       ##
##  TEST PULSE WIDTH..........8        TEST TIME UNITS...........ns         ##
##  TEST PI OFFSET............0                                             ##
##  TEST BIDI OFFSET..........0                                             ##
##  TEST STROBE OFFSET........72       X VALUE...................X          ##
##                                                                           ##
##  SCAN PERIOD...............80       SCAN STROBE TYPE..........edge       ##
##  SCAN PULSE WIDTH..........8        SCAN TIME UNITS...........ns         ##
##  SCAN PI OFFSET............16                                            ##
##  SCAN BIDI OFFSET..........16                                            ##
##  SCAN STROBE OFFSET........0        SCAN OVERLAP..............yes        ##
##                                                                           ##
##  EXPERIMENT................1                                             ##
##                                                                           ##
##  TEST SECTION..............2        TEST SECTION TYPE.........scan       ##
##  TESTER TERMINATION........0        TERMINATION DOMINATION....tester     ##
##                                                                           ##
##*************************************************************************##

waveform "WGL.lssd.scan.ex1.ts2"

        include "WGL.lssd.signals";
##*************************************************************************##
##                              TIMING DEFINITIONS                           ##
##*************************************************************************##

timeplate "scan_cycle" period 80ns
        "A" := input [ 0ns:P,  24ns:S, 32ns:D ];
```

```
        "B" := input [ 0ns:P,  40ns:S, 48ns:D ];
        "C" := input [ 0ns:P,  16ns:S, 24ns:D ];
        "CS" := input [ 0ns:P,  16ns:S ];
        "DI1" := input [ 0ns:P,  16ns:S ];
        "DI2" := input [ 0ns:P,  16ns:S ];
        "DI3" := input [ 0ns:P,  16ns:S ];
        "DI4" := input [ 0ns:P,  16ns:S ];
        "ENABLE1" := input [ 0ns:P,  16ns:S ];
        "ENABLE2" := input [ 0ns:P,  16ns:S ];
        "ME" := input [ 0ns:P,  16ns:S ];
        "PS" := input [ 0ns:P,  16ns:S ];
        "SEL" := input [ 0ns:P,  16ns:S ];
        "SI1" := input [ 0ns:P,  16ns:S ];
        "SI2" := input [ 0ns:P,  16ns:S ];
        "SO1_BIDI" := input [ 0ns:P,  16ns:S ];
        "SO2_BIDI" := input [ 0ns:P,  16ns:S ];
        "ST1" := input [ 0ns:P,  16ns:S ];
        "ST2" := input [ 0ns:P,  16ns:S ];
        "DO1" := output [ 0ns:Q'edge ];
        "DO2" := output [ 0ns:Q'edge ];
        "DO3" := output [ 0ns:Q'edge ];
        "DO4" := output [ 0ns:Q'edge ];
        "SO1" := output [ 0ns:Q'edge ];
        "SO1_BIDI" := output [ 0ns:Q'edge ];
        "SO2" := output [ 0ns:Q'edge ];
        "SO2_BIDI" := output [ 0ns:Q'edge ];
end
timeplate "test_cycle_lssd" period 80ns
        "A" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "B" := input [ 0ns:P, 24ns:S, 32ns:D ];
        "C" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "CS" := input [ 0ns:S ];
        "DI1" := input [ 0ns:S ];
        "DI2" := input [ 0ns:S ];
        "DI3" := input [ 0ns:S ];
        "DI4" := input [ 0ns:S ];
        "ENABLE1" := input [ 0ns:S ];
        "ENABLE2" := input [ 0ns:S ];
        "ME" := input [ 0ns:S ];
        "PS" := input [ 0ns:S ];
        "SEL" := input [ 0ns:S ];
        "SI1" := input [ 0ns:S ];
        "SI2" := input [ 0ns:S ];
        "SO1_BIDI" := input [ 0ns:S ];
        "SO2_BIDI" := input [ 0ns:S ];
        "ST1" := input [ 0ns:S ];
        "ST2" := input [ 0ns:S ];
        "DO1" := output [ 0ns:X, 72ns:Q'edge ];
        "DO2" := output [ 0ns:X, 72ns:Q'edge ];
        "DO3" := output [ 0ns:X, 72ns:Q'edge ];
        "DO4" := output [ 0ns:X, 72ns:Q'edge ];
        "SO1" := output [ 0ns:X, 72ns:Q'edge ];
        "SO1_BIDI" := output [ 0ns:X, 72ns:Q'edge ];
        "SO2" := output [ 0ns:X, 72ns:Q'edge ];
        "SO2_BIDI" := output [ 0ns:X, 72ns:Q'edge ];
end
##*****************************************************************************##
##                            DEFINE SCAN STATES                               ##
##*****************************************************************************##

scanstate
```

```
 "SS.1.2.1.2.1.1.1" :=
        "MREG_1_FULLSCAN" ( 100110011)
        "MREG_2_FULLSCAN" ( 1001100110011001) ;
 "SS.1.2.1.2.7.1.1" :=
        "MREG_1_FULLSCAN" ( 100110011)
        "MREG_2_FULLSCAN" ( 1001100110011001) ;
end


##*************************************************************************##
##                                TEST VECTORS                             ##
##*************************************************************************##

pattern MAIN ( "A", "B", "C", "CS", "DI1", "DI2", "DI3", "DI4", "ENABLE1",
               "ENABLE2", "ME", "PS", "SEL", "SI1", "SI2", "ST1", "ST2",
               "SO1_BIDI":I, "SO2_BIDI":I,
               "DO1", "DO2", "DO3", "DO4", "SO1", "SO2",
               "SO1_BIDI":O, "SO2_BIDI":O )

##*************************************************************************##
##  TESTER LOOP...............1        PROCEDURES HAVE MEMORY....no         ##
##  TEST PROCEDURE............1        TYPE......................init       ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no         ##
##  TEST SEQUENCE.............1        TYPE......................init       ##
##*************************************************************************##

## Processing the Static: EVENT 1.2.1.1.1.1.1:  Stim_PI:
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 X X X X X X X X X X X X X X
        Z Z
        X X X X X X
        - - ];
##*************************************************************************##
##  TEST PROCEDURE............2        TYPE......................normal     ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....yes        ##
##  STATIC FAULTS.............571      PERCENT STATIC FAULTS.....54.277565  ##
##  TEST SEQUENCE.............1        TYPE......................normal     ##
##*************************************************************************##

## Processing the Static: EVENT 1.2.1.2.1.1.1:  Scan_Load:
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 X X X X X 1 1 1 X 0 X X 0 1
        Z Z
        X X X X X X
        - - ];
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 X X X X X 1 1 1 X 0 X X 0 1
        Z Z
        X X X X X X
        - - ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        1 1 0 X X X X X 1 1 1 X 0 - - 0 1
        Z Z
        X X X X X X
        - - ],
        input  [ "MREG_1_FULLSCAN" :  "SS.1.2.1.2.1.1.1" ] ,
        input  [ "MREG_2_FULLSCAN" :  "SS.1.2.1.2.1.1.1" ] ;

## Processing the Static: EVENT 1.2.1.2.1.2.1:  Stim_PI:
```

```
##************************************************************************##
##  TEST SEQUENCE.............2        TYPE......................normal      ##
##************************************************************************##

## Processing the Static: EVENT 1.2.1.2.2.1.1:  Stim_PI:
## Processing the Static: EVENT 1.2.1.2.2.2.1:  Stim_PI:
##************************************************************************##
##  TEST SEQUENCE.............3        TYPE......................normal      ##
##************************************************************************##

## Processing the Static: EVENT 1.2.1.2.3.1.1:  Stim_PI:
## Processing the Static: EVENT 1.2.1.2.3.1.2:  Pulse:
## Processing the Static: EVENT 1.2.1.2.3.1.3:  Pulse:
## Processing the Static: EVENT 1.2.1.2.3.1.4:  Measure_PO:
   vector ( +, "test_cycle_lssd" ) := [
        1 1 0 0 0 0 0 0 1 1 1 0 0 1 1 0 1
        - -
        1 1 1 0 1 0
        1 0 ];
                        .
                        .
                        .

##************************************************************************##
##  TEST SEQUENCE.............7        TYPE......................normal      ##
##************************************************************************##
## Processing the Static: EVENT 1.2.1.2.7.1.1:  Scan_Unload:
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 0 0 1 0 1 1 1 1 1 0 1 1 0 1
        Z Z
        X X X X X X
        - - ];
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 0 0 1 0 1 1 1 1 1 0 1 1 0 1
        Z Z
        X X X X X X
        - - ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        1 1 0 0 0 1 0 1 1 1 1 1 0 1 1 0 1
        - -
        X X X X X X
        - - ],
        output [ "MREG_1_FULLSCAN" :  "SS.1.2.1.2.7.1.1" ] ,
        output [ "MREG_2_FULLSCAN" :  "SS.1.2.1.2.7.1.1" ] ;

## Inserted final non-scan Pattern
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 0 0 1 0 1 1 1 1 1 0 X X 0 1
        Z Z
        X X X X X X
        - - ];
end end
```

WGL Logic Test Example

The following is an example of a WGL logic test.

```
                                    ##*****************************************
*******************************##
##                              WGL VECTOR FILE                              ##
##  Encounter(TM) Test and Diagnostics 3.0     Apr 01, 2005 (linux24 TDA30)  ##
##*****************************************************************************##
##                                                                             ##
##  FILE CREATED.............April 01, 2005 at 09:51:17                     ##
##                                                                             ##
##  PROJECT NAME.............lbc                                             ##
##                                                                             ##
##  TESTMODE.................lssd                                            ##
##                                                                             ##
##  TDR......................dummy_tester_lssd                                ##
##                                                                             ##
##  TEST PERIOD..............80       TEST STROBE TYPE..........edge         ##
##  TEST PULSE WIDTH.........8        TEST TIME UNITS...........ns           ##
##  TEST PI OFFSET...........0                                                ##
##  TEST BIDI OFFSET.........0                                                ##
##  TEST STROBE OFFSET.......72       X VALUE..................X            ##
##                                                                             ##
##  SCAN PERIOD..............80       SCAN STROBE TYPE..........edge         ##
##  SCAN PULSE WIDTH.........8        SCAN TIME UNITS...........ns           ##
##  SCAN PI OFFSET...........16                                               ##
##  SCAN BIDI OFFSET.........16                                               ##
##  SCAN STROBE OFFSET.......0        SCAN OVERLAP.............yes          ##
##                                                                             ##
##  EXPERIMENT...............2                                                ##
##                                                                             ##
##  TEST SECTION.............1        TEST SECTION TYPE........logic        ##
##  TESTER TERMINATION.......0        TERMINATION DOMINATION....tester       ##
##                                                                             ##
##*****************************************************************************##

waveform "WGL.lssd.logic.ex2.ts1"

        include "WGL.lssd.signals";

##*****************************************************************************##
##                            TIMING DEFINITIONS                               ##
##*****************************************************************************##

timeplate "scan_cycle" period 80ns
        "A" := input [ 0ns:P,  24ns:S, 32ns:D ];
        "B" := input [ 0ns:P,  40ns:S, 48ns:D ];
        "C" := input [ 0ns:P,  16ns:S, 24ns:D ];
        "CS" := input [ 0ns:P,  16ns:S ];
        "DI1" := input [ 0ns:P,  16ns:S ];
        "DI2" := input [ 0ns:P,  16ns:S ];
        "DI3" := input [ 0ns:P,  16ns:S ];
        "DI4" := input [ 0ns:P,  16ns:S ];
        "ENABLE1" := input [ 0ns:P,  16ns:S ];
        "ENABLE2" := input [ 0ns:P,  16ns:S ];
        "ME" := input [ 0ns:P,  16ns:S ];
        "PS" := input [ 0ns:P,  16ns:S ];
        "SEL" := input [ 0ns:P,  16ns:S ];
        "SI1" := input [ 0ns:P,  16ns:S ];
```

```
        "SI2" := input [ 0ns:P,  16ns:S ];
        "SO1_BIDI" := input [ 0ns:P,  16ns:S ];
        "SO2_BIDI" := input [ 0ns:P,  16ns:S ];
        "ST1" := input [ 0ns:P,  16ns:S ];
        "ST2" := input [ 0ns:P,  16ns:S ];
        "DO1" := output [ 0ns:Q'edge ];
        "DO2" := output [ 0ns:Q'edge ];
        "DO3" := output [ 0ns:Q'edge ];
        "DO4" := output [ 0ns:Q'edge ];
        "SO1" := output [ 0ns:Q'edge ];
        "SO1_BIDI" := output [ 0ns:Q'edge ];
        "SO2" := output [ 0ns:Q'edge ];
        "SO2_BIDI" := output [ 0ns:Q'edge ];
end
timeplate "test_cycle_lssd" period 80ns
        "A" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "B" := input [ 0ns:P, 24ns:S, 32ns:D ];
        "C" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "CS" := input [ 0ns:S ];
        "DI1" := input [ 0ns:S ];
        "DI2" := input [ 0ns:S ];
        "DI3" := input [ 0ns:S ];
        "DI4" := input [ 0ns:S ];
        "ENABLE1" := input [ 0ns:S ];
        "ENABLE2" := input [ 0ns:S ];
        "ME" := input [ 0ns:S ];
        "PS" := input [ 0ns:S ];
        "SEL" := input [ 0ns:S ];
        "SI1" := input [ 0ns:S ];
        "SI2" := input [ 0ns:S ];
        "SO1_BIDI" := input [ 0ns:S ];
        "SO2_BIDI" := input [ 0ns:S ];
        "ST1" := input [ 0ns:S ];
        "ST2" := input [ 0ns:S ];
        "DO1" := output [ 0ns:X, 72ns:Q'edge ];
        "DO2" := output [ 0ns:X, 72ns:Q'edge ];
        "DO3" := output [ 0ns:X, 72ns:Q'edge ];
        "DO4" := output [ 0ns:X, 72ns:Q'edge ];
        "SO1" := output [ 0ns:X, 72ns:Q'edge ];
        "SO1_BIDI" := output [ 0ns:X, 72ns:Q'edge ];
        "SO2" := output [ 0ns:X, 72ns:Q'edge ];
        "SO2_BIDI" := output [ 0ns:X, 72ns:Q'edge ];
end
##************************************************************************##
##                            DEFINE SCAN STATES                          ##
##************************************************************************##

scanstate
 "SS.2.1.1 .2.1.1.1" :=
        "MREG_1_FULLSCAN" ( 000010100)
        "MREG_2_FULLSCAN" ( 1110011001001111) ;
 "SS.2.1.1.2.2.1.1" :=
        "MREG_1_FULLSCAN" ( 100000111)
        "MREG_2_FULLSCAN" ( 1101111010001110) ;
## The Following is StimLatchExtra A.
 "SS.2.1.1.3.1.1.1" :=
        "MREG_1_FULLSCAN" ( 101110110)
        "MREG_2_FULLSCAN" ( 0001011010010110) ;
 "SS.2.1.1.3.1.5.1" :=
        "MREG_1_FULLSCAN" ( 010110110)
        "MREG_2_FULLSCAN" ( 0000101101001011) ;
```

```
        .
        .
        .
 "SS.2.1.1.8.13.4.1" :=
        "MREG_1_FULLSCAN" ( 100011001)
        "MREG_2_FULLSCAN" ( 1001010011001111) ;
 "SS.2.1.1.8.14.1.1" :=
        "MREG_1_FULLSCAN" ( 000101101)
        "MREG_2_FULLSCAN" ( 0000010000101111) ;
 "SS.2.1.1.8.14.4.1" :=
        "MREG_1_FULLSCAN" ( 000101101)
        "MREG_2_FULLSCAN" ( 1100000010011100) ;
end
##*************************************************************************##
##                              TEST VECTORS                               ##
##*************************************************************************##

pattern MAIN ( "A", "B", "C", "CS", "DI1", "DI2", "DI3", "DI4", "ENABLE1",
               "ENABLE2", "ME", "PS", "SEL", "SI1", "SI2", "ST1", "ST2",
               "SO1_BIDI":I, "SO2_BIDI":I,
               "DO1", "DO2", "DO3", "DO4", "SO1", "SO2",
               "SO1_BIDI":O, "SO2_BIDI":O )

##*************************************************************************##
##  TESTER LOOP...............1        PROCEDURES HAVE MEMORY....no         ##
##  TEST PROCEDURE............1        TYPE......................init       ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no         ##
##  TEST SEQUENCE.............1        TYPE......................init       ##
##*************************************************************************##

## Processing the Static: EVENT 2.1.1.1.1.1.1:  Stim_PI:
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 X X X X X X X X X X X X X X
        Z Z
        X X X X X X
        - - ];

##*************************************************************************##
##  TEST PROCEDURE............2        TYPE......................normal     ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no         ##
##  STATIC FAULTS.............18       PERCENT STATIC FAULTS.....55.988594  ##
##  TEST SEQUENCE.............1        TYPE......................normal     ##
##*************************************************************************##
## Processing the Static: EVENT 2.1.1.2.1.1.1:  Scan_Load:
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 X X X X X 1 1 1 X 0 X X 0 1
        Z Z
        X X X X X X
        - - ];
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 X X X X X 1 1 1 X 0 X X 0 1
        Z Z
        X X X X X X
        - - ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        1 1 0 X X X X X 1 1 1 X 0 - - 0 1
        Z Z
        X X X X X X
        - - ],
```

```
        input  [ "MREG_1_FULLSCAN" :  "SS.2.1.1.2.1.1.1" ] ,
        input  [ "MREG_2_FULLSCAN" :  "SS.2.1.1.2.1.1.1" ] ;

## Processing the Static: EVENT 2.1.1.2.1.2.1:  Stim_PI:
## Processing the Static: EVENT 2.1.1.2.1.3.1:  Stim_PI:
## Processing the Static: EVENT 2.1.1.2.1.4.1:  Measure_PO:
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 1 0 1 1 1 0 0 0 0 1 1 1 1 0
        1 1
        0 0 0 0 0 0
        - - ];
##************************************************************************##
##  TEST SEQUENCE.............2        TYPE......................normal      ##
##************************************************************************##

## Processing the Static: EVENT 2.1.1.2.2.1.1:  Scan_Load:
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 1 0 1 1 1 1 1 1 0 0 1 1 0 1
        Z Z
        X X X X X X
        - - ];
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 1 0 1 1 1 1 1 1 0 0 1 1 0 1
        Z Z
        X X X X X X
        - - ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        1 1 0 1 0 1 1 1 1 1 1 0 0 - - 0 1
        Z Z
        X X X X X X
        - - ],
        input  [ "MREG_1_FULLSCAN" :  "SS.2.1.1.2.2.1.1" ] ,
        input  [ "MREG_2_FULLSCAN" :  "SS.2.1.1.2.2.1.1" ] ;

## Processing the Static: EVENT 2.1.1.2.2.2.1:  Stim_PI:
## Processing the Static: EVENT 2.1.1.2.2.3.1:  Stim_PI:
## Processing the Static: EVENT 2.1.1.2.2.4.1:  Measure_PO:
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 0 1 1 0 0 1 0 0 0 1 1 0 1 0
        - -
        0 1 1 1 0 0
        0 0 ];
        .
        .
        .
##************************************************************************##
##  TEST SEQUENCE.............13       TYPE......................normal      ##
##************************************************************************##

## Processing the Static: EVENT 2.1.1.8.13.1.1:  Scan_Load:  ( Overlap is in Effect
 ) ## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 0 1 0 1 0 1 1 1 1 0 1 1 0 1
        Z Z
        X X X X X X
        - - ];
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 0 1 0 1 0 1 1 1 1 0 1 1 0 1
        Z Z
```

```
        X X X X X X
        - - ];
## Inserted the Scan Sequence:  Skewed_Unload_Sequence
   vector ( +, "test_cycle_lssd" ) := [
        0 1 0 0 1 0 1 0 1 1 1 1 0 1 1 0 1
        Z Z
        X X X X X X
        - - ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        1 1 0 0 1 0 1 0 1 1 1 1 0 - - 0 1
        - -
        X X X X X X
        - - ],
        input  [ "MREG_1_FULLSCAN" :  "SS.2.1.1.8.13.1.1" ] ,
        input  [ "MREG_2_FULLSCAN" :  "SS.2.1.1.8.13.1.1" ] ,
        output [ "MREG_1_FULLSCAN" :  "SS.2.1.1.8.12.4.1" ] ,
        output [ "MREG_2_FULLSCAN" :  "SS.2.1.1.8.12.4.1" ] ;

## Processing the Static: EVENT 2.1.1.8.13.2.1:  Stim_PI:
## Processing the Static: EVENT 2.1.1.8.13.2.2:  Measure_PO:
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 0 1 0 0 1 0 0 0 1 1 1 1 1 0
        - -
        0 0 0 0 0 0
        0 0 ];
## Processing the Static: EVENT 2.1.1.8.13.3.1:  Pulse:
   vector ( +, "test_cycle_lssd" ) := [
        0 0 1 0 1 0 0 1 0 0 0 1 1 1 1 1 0
        Z Z
        X X X X X X
        - - ];
## Processing the Static: EVENT 2.1.1.8.13.4.1:  Skewed_Scan_Unload:

##*****************************************************************************##
##  TEST SEQUENCE.............14       TYPE......................normal      ##
##*****************************************************************************##

## Processing the Static: EVENT 2.1.1.8.14.1.1:  Scan_Load:  ( Overlap is in Effect
 ) ## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 0 1 0 0 1 1 1 1 1 0 1 1 0 1
        Z Z
        X X X X X X
        - - ];
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 0 1 0 0 1 1 1 1 1 0 1 1 0 1
        Z Z
        X X X X X X
        - - ];
## Inserted the Scan Sequence:  Skewed_Unload_Sequence
   vector ( +, "test_cycle_lssd" ) := [
        0 1 0 0 1 0 0 1 1 1 1 1 0 1 1 0 1
        Z Z
        X X X X X X
        - - ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        1 1 0 0 1 0 0 1 1 1 1 1 0 - - 0 1
        - -
        X X X X X X
```

```
        - - ],
        input  [ "MREG_1_FULLSCAN" :  "SS.2.1.1.8.14.1.1" ] ,
        input  [ "MREG_2_FULLSCAN" :  "SS.2.1.1.8.14.1.1" ] ,
        output [ "MREG_1_FULLSCAN" :  "SS.2.1.1.8.13.4.1" ] ,
        output [ "MREG_2_FULLSCAN" :  "SS.2.1.1.8.13.4.1" ] ;
## Processing the Static: EVENT 2.1.1.8.14.2.1:  Stim_PI:
## Processing the Static: EVENT 2.1.1.8.14.2.2:  Measure_PO:
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 0
        - -
        0 0 0 0 0 0
        0 0 ];
## Processing the Static: EVENT 2.1.1.8.14.3.1:  Pulse:
   vector ( +, "test_cycle_lssd" ) := [
        0 0 1 0 1 1 0 0 0 0 0 0 1 1 0 1 0
        Z Z
        X X X X X X
        - - ];
## Processing the Static: EVENT 2.1.1.8.14.4.1:  Skewed_Scan_Unload:
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 0 1 1 0 0 1 1 1 0 0 1 0 0 1
        Z Z
        X X X X X X
        - - ];
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 0 1 1 0 0 1 1 1 0 0 1 0 0 1
        Z Z
        X X X X X X
        - - ];
## Inserted the Scan Sequence:  Skewed_Unload_Sequence
   vector ( +, "test_cycle_lssd" ) := [
        0 1 0 0 1 1 0 0 1 1 1 0 0 1 0 0 1
        Z Z
        X X X X X X
        - - ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        1 1 0 0 1 1 0 0 1 1 1 0 0 1 0 0 1
        - -
        X X X X X X
        - - ],
        output [ "MREG_1_FULLSCAN" :  "SS.2.1.1.8.14.4.1" ] ,
        output [ "MREG_2_FULLSCAN" :  "SS.2.1.1.8.14.4.1" ] ;
## Inserted final non-scan Pattern
   vector ( +, "test_cycle_lssd" ) := [
        0 0 0 0 1 1 0 0 1 1 1 0 0 X X 0 1
        Z Z
        X X X X X X
        - - ];
end end
```

WGL Driver/Receiver Test Example

The following is an example of a WGL driver/receiver test.

```
                                        ##*****************************************
*******************************##
```

```
##                              WGL VECTOR FILE                              ##
##  Encounter(TM) Test and Diagnostics 3.0     Apr 01, 2005 (linux24 TDA30)  ##
##*************************************************************************##
##                                                                         ##
##  FILE CREATED.............April 01, 2005 at 09:52:35                   ##
##                                                                         ##
##  PROJECT NAME.............btv                                           ##
##                                                                         ##
##  TESTMODE.................DRVRCV                                       ##
##                                                                         ##
##  TDR......................TBLLCC                                       ##
##                                                                         ##
##  TEST PERIOD..............80       TEST STROBE TYPE..........edge       ##
##  TEST PULSE WIDTH.........8        TEST TIME UNITS...........ns         ##
##  TEST PI OFFSET...........0                                             ##
##  TEST BIDI OFFSET.........0                                             ##
##  TEST STROBE OFFSET.......72       X VALUE..................X           ##
##                                                                         ##
##  SCAN PERIOD..............80       SCAN STROBE TYPE..........edge       ##
##  SCAN PULSE WIDTH.........8        SCAN TIME UNITS...........ns         ##
##  SCAN PI OFFSET...........16                                            ##
##  SCAN BIDI OFFSET.........16                                            ##
##  SCAN STROBE OFFSET.......0        SCAN OVERLAP.............yes        ##
##                                                                         ##
##  EXPERIMENT...............1                                             ##
##                                                                         ##
##  TEST SECTION.............1        TEST SECTION TYPE.........driver_rece ##
##  TESTER TERMINATION.......none     TERMINATION DOMINATION....tester     ##
##                                                                         ##
##*************************************************************************##

waveform "WGL.DRVRCV.driver_receiver.ex1.ts1"

        include "WGL.DRVRCV.signals";
##*************************************************************************##
##                              TIMING DEFINITIONS                         ##
##*************************************************************************##

timeplate "scan_cycle" period 80ns
        "A2" := input [ 0ns:P,  16ns:S, 24ns:D ];
        "A4" := input [ 0ns:P,  16ns:S ];
        "A5" := input [ 0ns:P,  16ns:S ];
        "A6" := input [ 0ns:P,  16ns:S, 24ns:D ];
        "A7" := input [ 0ns:P,  16ns:S, 24ns:U ];
        "A8" := input [ 0ns:P,  16ns:S, 24ns:D ];
        "B0" := input [ 0ns:P,  24ns:S, 32ns:D ];
        "B1" := input [ 0ns:P,  40ns:S, 48ns:D ];
        "B3" := input [ 0ns:P,  16ns:S, 24ns:D ];
        "B4" := input [ 0ns:P,  16ns:S, 24ns:D ];
        "B6" := input [ 0ns:P,  16ns:S ];
        "B7" := input [ 0ns:P,  16ns:S, 24ns:D ];
        "B8" := input [ 0ns:P,  40ns:S, 48ns:D ];
        "E1" := input [ 0ns:P,  16ns:S ];
        "F3" := input [ 0ns:P,  16ns:S ];
        "F4" := input [ 0ns:P,  16ns:S ];
        "I0" := input [ 0ns:P,  16ns:S ];
        "I1" := input [ 0ns:P,  16ns:S ];
        "L0" := input [ 0ns:P,  16ns:S ];
        "L1" := input [ 0ns:P,  16ns:S ];
        "L2" := input [ 0ns:P,  24ns:S, 32ns:D ];
        "L3" := input [ 0ns:P,  40ns:S, 48ns:D ];
```

```
        "S0" := input [ 0ns:P,  16ns:S ];
        "S1" := input [ 0ns:P,  16ns:S ];
        "T0" := input [ 0ns:P,  16ns:S ];
        "60" := output [ 0ns:Q'edge ];
        "61" := output [ 0ns:Q'edge ];
        "70" := output [ 0ns:Q'edge ];
end
timeplate "test_cycle_DRVRCV" period 80ns
        "A2" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "A4" := input [ 0ns:S ];
        "A5" := input [ 0ns:S ];
        "A6" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "A7" := input [ 0ns:P, 8ns:S, 16ns:U ];
        "A8" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "B0" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "B1" := input [ 0ns:P, 24ns:S, 32ns:D ];
        "B3" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "B4" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "B6" := input [ 0ns:S ];
        "B7" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "B8" := input [ 0ns:P, 24ns:S, 32ns:D ];
        "E1" := input [ 0ns:S ];
        "F3" := input [ 0ns:S ];
        "F4" := input [ 0ns:S ];
        "I0" := input [ 0ns:S ];
        "I1" := input [ 0ns:S ];
        "L0" := input [ 0ns:S ];
        "L1" := input [ 0ns:S ];
        "L2" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "L3" := input [ 0ns:P, 24ns:S, 32ns:D ];
        "S0" := input [ 0ns:S ];
        "S1" := input [ 0ns:S ];
        "T0" := input [ 0ns:S ];
        "60" := output [ 0ns:X, 72ns:Q'edge ];
        "61" := output [ 0ns:X, 72ns:Q'edge ];
        "70" := output [ 0ns:X, 72ns:Q'edge ];
end
##*************************************************************************##
##                            DEFINE SCAN STATES                           ##
##*************************************************************************##

scanstate
 "SS.1.1.1.2.1.1.1" :=
        "MREG_1_DRVRCV" ( 11)
       "MREG_2_DRVRCV" ( 00101010001000111010001010001000110111000110010000 0100) ;
 "SS.1.1.1.2.1.6.1" :=
        "MREG_1_DRVRCV" ( 11)
       "MREG_2_DRVRCV" ( 00101010001000111010001010001000110111000110010001011) ;
 "SS.1.1.1.3.1.1.1" :=
        "MREG_1_DRVRCV" ( 10)
       "MREG_2_DRVRCV" ( 11111010110011001111001101100111100011110101101011 0111) ;
                            .
                            .
                            .
 "SS.1.1.1.25.19.1.1" :=
        "MREG_1_DRVRCV" ( 11)
       "MREG_2_DRVRCV" ( 01000110111111011000111011110111010011110000101011 1000) ;
 "SS.1.1.1.25.19.4.1" :=
        "MREG_1_DRVRCV" ( 11)
       "MREG_2_DRVRCV" ( 01000110111111011000111011110111010011110100010101 1100) ;
end
```

```
##************************************************************************##
##                              TEST VECTORS                              ##
##************************************************************************##

pattern MAIN ( "A2", "A4", "A5", "A6", "A7", "A8", "B0", "B1", "B3", "B4",
               "B6", "B7", "B8", "E1", "F3", "F4", "I0", "I1", "L0", "L1",
               "L2", "L3", "S0", "S1", "T0",
               "60", "61", "70" )

##************************************************************************##
##  TESTER LOOP...............1        PROCEDURES HAVE MEMORY....no        ##
##  TEST PROCEDURE............1        TYPE......................init      ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no        ##
##  TEST SEQUENCE.............1        TYPE......................init      ##
##************************************************************************##

## Processing the Static: EVENT 1.1.1.1.1.1.1:  Stim_PI:
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 X X 0 1 0 0 0 0 0 X 0 0 X X X X X X X 0 0 X X X
        X X X ];

##************************************************************************##
##  TEST PROCEDURE............2        TYPE......................normal    ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no        ##
##  STATIC FAULTS.............3553     PERCENT STATIC FAULTS.....8.743049  ##
##  DRI/REC FAULTS............54       PERCENT DRI/REC FAULTS....9.246575  ##
##  TEST SEQUENCE.............1        TYPE......................normal    ##
##************************************************************************##

## Processing the Static: EVENT 1.1.1.2.1.1.1:  Scan_Load:
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 0 1 0 1 0 0 0 0 0 1 0 0 1 X X X X 1 1 0 0 0 1 X
        X X X ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        0 0 1 0 1 0 1 1 0 0 1 0 0 1 X - - X 1 1 1 1 0 1 X
        X X X ],
        input  [ "MREG_1_DRVRCV" :  "SS.1.1.1.2.1.1.1" ] ,
        input  [ "MREG_2_DRVRCV" :  "SS.1.1.1.2.1.1.1" ] ;
## Processing the Static: EVENT 1.1.1.2.1.2.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.2.1.2.2:  Pulse:
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 0 1 0 1 0 0 0 0 0 1 0 1 1 X X X X 1 1 0 0 0 1 X
        X X X ];
## Processing the Static: EVENT 1.1.1.2.1.3.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.2.1.4.1:  Pulse:
## Processing the Static: EVENT 1.1.1.2.1.5.1:  Measure_PO:
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 1 0 0 1 0 0 0 0 0 0 1 0 0 1 1 1 0 0 0 0 0 1 0 1
        1 1 Z ];
## Processing the Static: EVENT 1.1.1.2.1.6.1:  Skewed_Scan_Unload:
                              .
                              .
                              .

##************************************************************************##
##  TEST SEQUENCE.............2        TYPE......................normal    ##
##************************************************************************##
```

```
## Processing the Static: EVENT 1.1.1.4.2.1.1:  Scan_Load:  ( Overlap is in Effect
 ) ## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 1 1 1 0 0 0 1 1
        X X X ];
## Inserted the Scan Sequence:  Skewed_Unload_Sequence
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 1 1 1 0 1 0 1 1
        X X X ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        0 0 1 0 1 0 1 1 0 0 1 0 0 1 0 - - 1 1 1 1 1 0 1 1
        - X - ],
        input  [ "MREG_1_DRVRCV" :  "SS.1.1.1.4.2.1.1" ] ,
        input  [ "MREG_2_DRVRCV" :  "SS.1.1.1.4.2.1.1" ] ,
        output [ "MREG_1_DRVRCV" :  "SS.1.1.1.4.1.5.1" ] ,
        output [ "MREG_2_DRVRCV" :  "SS.1.1.1.4.1.5.1" ] ;
## Processing the Static: EVENT 1.1.1.4.2.2.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.4.2.2.2:  Pulse:
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 X X 1 1 1 0 0 0 1 1
        X X X ];
## Processing the Static: EVENT 1.1.1.4.2.3.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.4.2.3.2:  Measure_PO:
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
        X X X ];
## Processing the Static: EVENT 1.1.1.4.2.4.1:  Pulse:
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
        X X X ];
## Processing the Static: EVENT 1.1.1.4.2.5.1:  Skewed_Scan_Unload:

##*****************************************************************************##
##  TEST SEQUENCE.............3        TYPE......................normal      ##
##*****************************************************************************##

## Processing the Static: EVENT 1.1.1.4.3.1.1:  Scan_Load:  ( Overlap is in Effect
 ) ## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0
        X X X ];
## Inserted the Scan Sequence:  Skewed_Unload_Sequence
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 1 0 1 0
        X X X ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        0 0 1 0 1 0 1 1 0 0 1 0 0 1 0 - - 0 1 1 1 1 0 1 0
        - X - ],
        input  [ "MREG_1_DRVRCV" :  "SS.1.1.1.4.3.1.1" ] ,
        input  [ "MREG_2_DRVRCV" :  "SS.1.1.1.4.3.1.1" ] ,
        output [ "MREG_1_DRVRCV" :  "SS.1.1.1.4.2.5.1" ] ,
        output [ "MREG_2_DRVRCV" :  "SS.1.1.1.4.2.5.1" ] ;
## Processing the Static: EVENT 1.1.1.4.3.2.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.4.3.2.2:  Pulse:
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 X X 0 1 1 0 0 0 1 0
        X X X ];
## Processing the Static: EVENT 1.1.1.4.3.3.1:  Stim_PI:
```

```
## Processing the Static: EVENT 1.1.1.4.3.3.2:  Measure_PO:
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
        X 0 X ];
## Processing the Static: EVENT 1.1.1.4.3.4.1:  Pulse:
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
        X X X ];
## Processing the Static: EVENT 1.1.1.4.3.5.1:  Skewed_Scan_Unload:
                              .
                              .
                              .

##*************************************************************************##
##  TEST SEQUENCE.............19       TYPE......................normal      ##
##*************************************************************************##

## Processing the Static: EVENT 1.1.1.25.19.1.1:  Scan_Load:  ( Overlap is in
 Effect ) ## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 1
        X X X ];
## Inserted the Scan Sequence:  Skewed_Unload_Sequence
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 1 0 1 1
        X X X ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        0 0 1 0 1 0 1 1 0 0 1 0 0 1 0 - - 0 1 1 1 1 0 1 1
        - X - ],
        input  [ "MREG_1_DRVRCV" :  "SS.1.1.1.25.19.1.1" ] ,
        input  [ "MREG_2_DRVRCV" :  "SS.1.1.1.25.19.1.1" ] ,
        output [ "MREG_1_DRVRCV" :  "SS.1.1.1.25.18.4.1" ] ,
        output [ "MREG_2_DRVRCV" :  "SS.1.1.1.25.18.4.1" ] ;
## Processing the Static: EVENT 1.1.1.25.19.2.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.25.19.2.2:  Measure_PO:
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 1 0 1
        X X X ];
## Processing the Static: EVENT 1.1.1.25.19.3.1:  Pulse:
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 1 0 0 1 0 1 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 1 0 1
        X X X ];
## Processing the Static: EVENT 1.1.1.25.19.4.1:  Skewed_Scan_Unload:
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 0 1 0 1 0 0 0 0 0 1 0 0 1 1 1 1 0 1 1 0 0 0 1 1
        X X X ];
## Inserted the Scan Sequence:  Skewed_Unload_Sequence
   vector ( +, "test_cycle_DRVRCV" ) := [
        0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 1 0 1 0 1 1
        X X X ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        0 0 1 0 1 0 1 1 0 0 1 0 0 1 1 1 1 0 1 1 1 1 0 1 1
        - X - ],
        output [ "MREG_1_DRVRCV" :  "SS.1.1.1.25.19.4.1" ] ,
        output [ "MREG_2_DRVRCV" :  "SS.1.1.1.25.19.4.1" ] ;

## Inserted final non-scan Pattern
   vector ( +, "test_cycle_DRVRCV" ) := [
```

```
        0 0 1 0 1 0 0 0 0 0 1 0 0 1 1 X X 0 1 1 0 0 0 1 1
        X X X ];
end end
```

WGL IDDq Test Example

The following is an example of a WGL IDDq test.

```
##*************************************************************************##
##                              WGL VECTOR FILE                            ##
##  Encounter(TM) Test and Diagnostics 3.0     Apr 01, 2005 (linux24 TDA30)  ##
##*************************************************************************##
##                                                                         ##
##  FILE CREATED.............April 01, 2005 at 09:52:32                   ##
##                                                                         ##
##  PROJECT NAME.............btv                                          ##
##                                                                         ##
##  TESTMODE.................IDDQ                                         ##
##                                                                         ##
##  TDR......................TBAdvent_new                                 ##
##                                                                         ##
##  TEST PERIOD..............80       TEST STROBE TYPE..........edge       ##
##  TEST PULSE WIDTH.........8        TEST TIME UNITS...........ns         ##
##  TEST PI OFFSET...........0                                             ##
##  TEST BIDI OFFSET.........0                                             ##
##  TEST STROBE OFFSET.......72       X VALUE..................X           ##
##                                                                         ##
##  SCAN PERIOD..............80       SCAN STROBE TYPE..........edge       ##
##  SCAN PULSE WIDTH.........8        SCAN TIME UNITS...........ns         ##
##  SCAN PI OFFSET...........16                                            ##
##  SCAN BIDI OFFSET.........16                                            ##
##  SCAN STROBE OFFSET.......0        SCAN OVERLAP.............yes         ##
##                                                                         ##
##  EXPERIMENT...............1                                             ##
##                                                                         ##
##  TEST SECTION.............1        TEST SECTION TYPE........IDDq        ##
##  TESTER TERMINATION.......0        TERMINATION DOMINATION....tester     ##
##                                                                         ##
##*************************************************************************##

waveform "WGL.IDDQ.IDDq.ex1.ts1"

        include "WGL.IDDQ.signals";
##*************************************************************************##
##                            TIMING DEFINITIONS                           ##
##*************************************************************************##

timeplate "scan_cycle" period 80ns
        "A2" := input [ 0ns:P,  16ns:S, 24ns:D ];
        "A4" := input [ 0ns:P,  16ns:S ];
        "A5" := input [ 0ns:P,  16ns:S ];
        "A6" := input [ 0ns:P,  16ns:S, 24ns:D ];
        "A7" := input [ 0ns:P,  16ns:S, 24ns:U ];
        "A8" := input [ 0ns:P,  16ns:S, 24ns:D ];
        "B0" := input [ 0ns:P,  24ns:S, 32ns:D ];
        "B1" := input [ 0ns:P,  40ns:S, 48ns:D ];
        "B3" := input [ 0ns:P,  16ns:S, 24ns:D ];
```

```
        "B4" := input [ 0ns:P,  16ns:S, 24ns:D ];
        "B6" := input [ 0ns:P,  16ns:S ];
        "B7" := input [ 0ns:P,  16ns:S, 24ns:D ];
        "B8" := input [ 0ns:P,  40ns:S, 48ns:D ];
        "E1" := input [ 0ns:P,  16ns:S ];
        "F3" := input [ 0ns:P,  16ns:S ];
        "F4" := input [ 0ns:P,  16ns:S ];
        "I0" := input [ 0ns:P,  16ns:S ];
        "I1" := input [ 0ns:P,  16ns:S ];
        "L0" := input [ 0ns:P,  16ns:S ];
        "L1" := input [ 0ns:P,  16ns:S ];
        "L2" := input [ 0ns:P,  24ns:S, 32ns:D ];
        "L3" := input [ 0ns:P,  40ns:S, 48ns:D ];
        "S0" := input [ 0ns:P,  16ns:S ];
        "S1" := input [ 0ns:P,  16ns:S ];
        "T0" := input [ 0ns:P,  16ns:S ];
        "60" := output [ 0ns:Q'edge ];
        "61" := output [ 0ns:Q'edge ];
        "70" := output [ 0ns:Q'edge ];
end
timeplate "test_cycle_stimclks" period 80ns
        "A2" := input [ 0ns:P, 8ns:S ];
        "A4" := input [ 0ns:S ];
        "A5" := input [ 0ns:S ];
        "A6" := input [ 0ns:P, 8ns:S ];
        "A7" := input [ 0ns:P, 8ns:S ];
        "A8" := input [ 0ns:P, 8ns:S ];
        "B0" := input [ 0ns:P, 8ns:S ];
        "B1" := input [ 0ns:P, 24ns:S ];
        "B3" := input [ 0ns:P, 8ns:S ];
        "B4" := input [ 0ns:P, 8ns:S ];
        "B6" := input [ 0ns:S ];
        "B7" := input [ 0ns:P, 8ns:S ];
        "B8" := input [ 0ns:P, 24ns:S ];
        "E1" := input [ 0ns:S ];
        "F3" := input [ 0ns:S ];
        "F4" := input [ 0ns:S ];
        "I0" := input [ 0ns:S ];
        "I1" := input [ 0ns:S ];
        "L0" := input [ 0ns:S ];
        "L1" := input [ 0ns:S ];
        "L2" := input [ 0ns:P, 8ns:S ];
        "L3" := input [ 0ns:P, 24ns:S ];
        "S0" := input [ 0ns:S ];
        "S1" := input [ 0ns:S ];
        "T0" := input [ 0ns:S ];
        "60" := output [ 0ns:X, 72ns:Q'edge ];
        "61" := output [ 0ns:X, 72ns:Q'edge ];
        "70" := output [ 0ns:X, 72ns:Q'edge ];
end
timeplate "test_cycle_IDDQ" period 80ns
        "A2" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "A4" := input [ 0ns:S ];
        "A5" := input [ 0ns:S ];
        "A6" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "A7" := input [ 0ns:P, 8ns:S, 16ns:U ];
        "A8" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "B0" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "B1" := input [ 0ns:P, 24ns:S, 32ns:D ];
        "B3" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "B4" := input [ 0ns:P, 8ns:S, 16ns:D ];
```

```
        "B6" := input [ 0ns:S ];
        "B7" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "B8" := input [ 0ns:P, 24ns:S, 32ns:D ];
        "E1" := input [ 0ns:S ];
        "F3" := input [ 0ns:S ];
        "F4" := input [ 0ns:S ];
        "I0" := input [ 0ns:S ];
        "I1" := input [ 0ns:S ];
        "L0" := input [ 0ns:S ];
        "L1" := input [ 0ns:S ];
        "L2" := input [ 0ns:P, 8ns:S, 16ns:D ];
        "L3" := input [ 0ns:P, 24ns:S, 32ns:D ];
        "S0" := input [ 0ns:S ];
        "S1" := input [ 0ns:S ];
        "T0" := input [ 0ns:S ];
        "60" := output [ 0ns:X, 72ns:Q'edge ];
        "61" := output [ 0ns:X, 72ns:Q'edge ];
        "70" := output [ 0ns:X, 72ns:Q'edge ];
end
##*************************************************************************##
##                            DEFINE SCAN STATES                           ##
##*************************************************************************##

scanstate
 "SS.1.1.1.2.1.1.1" :=
        "MREG_1_IDDQ" ( 00)
        "MREG_2_IDDQ" ( 1011111110111101000010011010110101110100111011000 10...) ;
 "SS.1.1.1.3.1.1.1" :=
        "MREG_1_IDDQ" ( 11)
        "MREG_2_IDDQ" ( 1100110001111101101100000101101100100011001011110 01...) ;
 "SS.1.1.1.4.1.1.1" :=
        "MREG_1_IDDQ" ( 00)
        "MREG_2_IDDQ" ( 1110011101110000000001110000000010111111101100000 0...) ;
                                .
                                .
                                .
 "SS.1.1.1.91.1.1.1" :=
        "MREG_1_IDDQ" ( 11)
        "MREG_2_IDDQ" ( 0010110011001001111110000011011010000010100000000 0...) ;
 "SS.1.1.1.92.1.1.1" :=
        "MREG_1_IDDQ" ( 00)
        "MREG_2_IDDQ" ( 1111111111111111111111111111111111111111111001100 100...) ;
end


##*************************************************************************##
##                              TEST VECTORS                               ##
##*************************************************************************##

pattern MAIN ( "A2", "A4", "A5", "A6", "A7", "A8", "B0", "B1", "B3", "B4",
               "B6", "B7", "B8", "E1", "F3", "F4", "I0", "I1", "L0", "L1",
               "L2", "L3", "S0", "S1", "T0", "60", "61", "70" )

##*************************************************************************##
##  TESTER LOOP...............1        PROCEDURES HAVE MEMORY....no         ##
##  TEST PROCEDURE............1        TYPE.......................init       ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no         ##
##  TEST SEQUENCE.............1        TYPE.......................init       ##
##*************************************************************************##

## Processing the Static: EVENT 1.1.1.1.1.1.1:  Stim_PI:
```

```
   vector ( +, "test_cycle_IDDQ" ) := [
        0 X X 0 1 0 0 0 0 0 X 0 0 X X X X X X X 0 0 X X X
        X X X ];

##*************************************************************************##
##  TEST PROCEDURE............2        TYPE.......................normal      ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          ##
##  IDDQ FAULTS...............8160     PERCENT IDDQ FAULTS.......20.079729   ##
##  TEST SEQUENCE.............1        TYPE.......................normal      ##
##*************************************************************************##

## Processing the Static: EVENT 1.1.1.2.1.1.1:  Scan_Load:
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_IDDQ" ) := [
        0 0 1 0 1 0 0 0 0 0 1 0 0 1 X X X X 1 1 0 0 0 1 X
        X X X ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        0 0 1 0 1 0 1 1 0 0 1 0 0 1 X - - X 1 1 1 1 0 1 X
        X X X ],
        input  [ "MREG_1_IDDQ" :  "SS.1.1.1.2.1.1.1" ] ,
        input  [ "MREG_2_IDDQ" :  "SS.1.1.1.2.1.1.1" ] ;

## Processing the Static: EVENT 1.1.1.2.1.2.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.2.1.2.2:  Pulse:
   vector ( +, "test_cycle_IDDQ" ) := [
        0 0 1 0 1 0 0 0 0 0 1 0 1 1 X X X X 1 1 0 0 0 1 X
        X X X ];
## Processing the Static: EVENT 1.1.1.2.1.3.1:  Stim_PI:
   vector ( +, "test_cycle_IDDQ" ) := [
        0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0
        X X X ];
## Processing the Static: EVENT 1.1.1.2.1.4.1:  Measure_Current:
        { MEASURE_CURRENT }
                            .
                            .
                            .
##*************************************************************************##
##  TEST PROCEDURE............92       TYPE.......................normal      ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          ##
##  IDDQ FAULTS...............41       PERCENT IDDQ FAULTS.......99.707169   ##
##  TEST SEQUENCE.............1        TYPE.......................normal      ##
##*************************************************************************##

## Processing the Static: EVENT 1.1.1.92.1.1.1:  Scan_Load:
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_IDDQ" ) := [
        0 0 1 0 1 0 0 0 0 0 1 0 0 1 1 0 1 0 1 1 0 0 0 1 0
        X X X ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        0 0 1 0 1 0 1 1 0 0 1 0 0 1 1 - - 0 1 1 1 1 0 1 0
        X X X ],
        input  [ "MREG_1_IDDQ" :  "SS.1.1.1.92.1.1.1" ] ,
        input  [ "MREG_2_IDDQ" :  "SS.1.1.1.92.1.1.1" ] ;

## Processing the Static: EVENT 1.1.1.92.1.2.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.92.1.2.2:  Pulse:
   vector ( +, "test_cycle_IDDQ" ) := [
        0 0 1 0 1 0 0 0 0 0 1 0 1 1 1 X X 0 1 1 0 0 0 1 0
        X X X ];
```

```
## Processing the Static: EVENT 1.1.1.92.1.3.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.92.1.4.1:  Stim_Clock:
   vector ( +, "test_cycle_stimclks" ) := [
        0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0
        X X X ];
## Processing the Static: EVENT 1.1.1.92.1.5.1:  Measure_Current:
        { MEASURE_CURRENT }
## Processing the Static: EVENT 1.1.1.92.1.6.1:  Stim_Clock:
   vector ( +, "test_cycle_IDDQ" ) := [
        0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0
        X X X ];
## Inserted final non-scan Pattern
   vector ( +, "test_cycle_IDDQ" ) := [
        0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0
        X X X ];
end end
```

WGL Stuck Driver Test Example

The following is an example of a WGL stuck driver test.

```
##*************************************************************************##
##                              WGL VECTOR FILE                            ##
##  Encounter(TM) Test and Diagnostics 3.0     Apr 01, 2005 (linux24 TDA30)  ##
##*************************************************************************##
##                                                                         ##
##  FILE CREATED.............April 01, 2005 at 09:52:40                   ##
##                                                                         ##
##  PROJECT NAME.............asp                                          ##
##                                                                         ##
##  TESTMODE.................IOWRAP                                       ##
##                                                                         ##
##  TDR......................TBAdvent                                     ##
##                                                                         ##
##  TEST PERIOD..............80       TEST STROBE TYPE..........edge       ##
##  TEST PULSE WIDTH.........8        TEST TIME UNITS...........ns         ##
##  TEST PI OFFSET...........0                                             ##
##  TEST BIDI OFFSET.........0                                             ##
##  TEST STROBE OFFSET.......72       X VALUE..................X          ##
##                                                                         ##
##  SCAN PERIOD..............80       SCAN STROBE TYPE..........edge       ##
##  SCAN PULSE WIDTH.........8        SCAN TIME UNITS...........ns         ##
##  SCAN PI OFFSET...........16                                            ##
##  SCAN BIDI OFFSET.........16                                            ##
##  SCAN STROBE OFFSET.......0        SCAN OVERLAP.............yes        ##
##                                                                         ##
##  EXPERIMENT...............1                                             ##
##                                                                         ##
##  TEST SECTION.............1        TEST SECTION TYPE........IOWRAP_stuc ##
##  TESTER TERMINATION.......0        TERMINATION DOMINATION....tester     ##
##                                                                         ##
##*************************************************************************##

waveform "WGL.IOWRAP.IOWRAP_stuck_driver.ex1.ts1"

        include "WGL.IOWRAP.signals";
```

```
##*************************************************************************##
##                            TIMING DEFINITIONS                           ##
##*************************************************************************##

timeplate "scan_cycle" period 80ns
        "A/00/00" := input [ 0ns:P,  16ns:S ];
        "A/00/01" := input [ 0ns:P,  16ns:S ];
        "A/00/02" := input [ 0ns:P,  16ns:S ];
        "A/00/03" := input [ 0ns:P,  16ns:S ];
        "A/00/04" := input [ 0ns:P,  16ns:S ];
        "A/00/05" := input [ 0ns:P,  16ns:S ];
        "A/00/06" := input [ 0ns:P,  16ns:S ];
        "A/00/07" := input [ 0ns:P,  16ns:S ];
        "A/00/08" := input [ 0ns:P,  16ns:S ];
        "A/00/09" := input [ 0ns:P,  16ns:S ];
        "A/00/10" := input [ 0ns:P,  16ns:S ];
        "A/00/11" := input [ 0ns:P,  16ns:S ];
                          .
                          .
                          .
        "A/01/46" := output [ 0ns:Q'edge ];
        "A/01/47" := output [ 0ns:Q'edge ];
        "A/01/48" := output [ 0ns:Q'edge ];
        "A/01/49" := output [ 0ns:Q'edge ];
        "A/01/50" := output [ 0ns:Q'edge ];
        "A/01/51" := output [ 0ns:Q'edge ];
        "A/01/52" := output [ 0ns:Q'edge ];
        "A/01/53" := output [ 0ns:Q'edge ];
        "A/01/54" := output [ 0ns:Q'edge ];
end
timeplate "test_cycle_IOWRAP" period 80ns
        "A/00/00" := input [ 0ns:S ];
        "A/00/01" := input [ 0ns:S ];
        "A/00/02" := input [ 0ns:S ];
        "A/00/03" := input [ 0ns:S ];
                          .
                          .
                          .
        "A/01/51" := output [ 0ns:X, 72ns:Q'edge ];
        "A/01/52" := output [ 0ns:X, 72ns:Q'edge ];
        "A/01/53" := output [ 0ns:X, 72ns:Q'edge ];
        "A/01/54" := output [ 0ns:X, 72ns:Q'edge ];
end


##*************************************************************************##
##                            DEFINE SCAN STATES                           ##
##*************************************************************************##

scanstate
 "SS.1.1.1.2.1.1.1" :=
        "MREG_1_IOWRAP" ( 000011010100100011000101100110011111111111...) ;
 "SS.1.1.1.2.1.5.1" :=
        "MREG_1_IOWRAP" ( 101100000000000000110001000101000000000000...) ;
 "SS.1.1.1.2.2.1.1" :=
        "MREG_1_IOWRAP" ( 101101000110000100111101100011100011110000...) ;
 "SS.1.1.1.2.2.5.1" :=
        "MREG_1_IOWRAP" ( 000000000000000000110011111101000000011111...) ;
end

##*************************************************************************##
```

```
##                                TEST VECTORS                                ##
##***************************************************************************##

pattern MAIN ( "A/00/A0", "A/00/A1", "A/00/A2", "A/00/A3", "A/00/A4", "A/00/A5",
               "A/00/A6", "A/00/A7", "A/00/A8", "A/00/A9", "A/00/B0", "A/00/B1",
             "A/00/B2", "A/00/B3", "A/00/B4", "A/00/B5", "A/00/00":I, "A/00/01":I,
               "A/00/02":I, "A/00/03":I, "A/00/04":I, "A/00/05":I, "A/00/06":I,
               "A/00/07":I, "A/00/08":I, "A/00/09":I, "A/00/10":I, "A/00/11":I,
               "A/00/12":I, "A/00/13":I, "A/00/14":I, "A/00/15":I, "A/00/16":I,
                                    .
                                    .
                                    .
               "A/01/37":O, "A/01/39":O, "A/01/40":O, "A/01/41":O, "A/01/42":O,
               "A/01/43":O, "A/01/44":O, "A/01/45":O, "A/01/46":O, "A/01/47":O,
               "A/01/48":O, "A/01/49":O, "A/01/50":O, "A/01/51":O, "A/01/52":O,
               "A/01/53":O, "A/01/54":O )

##***************************************************************************##
##  TESTER LOOP...............1        PROCEDURES HAVE MEMORY....no          ##
##  TEST PROCEDURE............1        TYPE......................init        ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          ##
##  TEST SEQUENCE.............1        TYPE......................init        ##
##***************************************************************************##

## Processing the Static: EVENT 1.1.1.1.1.1.1:  Stim_PI:
   vector ( +, "test_cycle_IOWRAP" ) := [
        X 0 1 1 0 X X X X X 1 0 0 X X X
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ...Z
        X
         - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  ];

##***************************************************************************##
##  TEST PROCEDURE............2        TYPE......................normal      ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          ##
##  TEST SEQUENCE.............1        TYPE......................normal      ##
##***************************************************************************##
## Processing the Static: EVENT 1.1.1.2.1.1.1:  Scan_Load:
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_IOWRAP" ) := [
        0 0 1 1 0 1 1 X X 0 1 0 0 X 0 X
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ...Z
        X
         - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
b        0 0 0 0 0 1 1 X X 0 1 0 0 X 0 -
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ...Z
        X
         - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  ];
        input  [ "MREG_1_IOWRAP" :  "SS.1.1.1.2.1.1.1" ] ;

## Processing the Static: EVENT 1.1.1.2.1.2.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.2.1.3.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.2.1.4.1:  Pulse:
   vector ( +, "test_cycle_IOWRAP" ) := [
        1 1 1 1 0 1 0 0 X 1 1 0 0 0 1 1
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
         - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  ];
## Processing the Static: EVENT 1.1.1.2.1.5.1:  Skewed_Scan_Unload:
```

```
##************************************************************************##
##  TEST SEQUENCE............2        TYPE......................normal      ##
##************************************************************************##

## Processing the Static: EVENT 1.1.1.2.2.1.1:  Scan_Load:  ( Overlap is in Effect
 ) ## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_IOWRAP" ) := [
        0 0 1 1 0 1 1 0 X 0 1 0 0 0 0 1
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
         - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  ];
## Inserted the Scan Sequence:  Skewed_Unload_Sequence
   vector ( +, "test_cycle_IOWRAP" ) := [
        0 0 0 1 0 1 1 0 X 0 1 0 0 0 0 1
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
         - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        0 0 0 0 0 1 1 0 X 0 1 0 0 0 0 -
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        -
         - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  ];
        input  [ "MREG_1_IOWRAP" :  "SS.1.1.1.2.2.1.1" ] ,
        output [ "MREG_1_IOWRAP" :  "SS.1.1.1.2.1.5.1" ] ;

## Processing the Static: EVENT 1.1.1.2.2.2.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.2.2.3.1:  Stim_PI:
## Processing the Static: EVENT 1.1.1.2.2.4.1:  Pulse:
   vector ( +, "test_cycle_IOWRAP" ) := [
        1 1 1 1 0 1 0 1 X 1 1 0 0 1 1 1
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
         - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  ];
## Processing the Static: EVENT 1.1.1.2.2.5.1:
Skewed_Scan_UnloadSkewed_Scan_Unload:
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_IOWRAP" ) := [
        0 0 1 1 0 1 1 1 X 0 1 0 0 1 0 1
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
         - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  ];
## Inserted the Scan Sequence:  Skewed_Unload_Sequence
   vector ( +, "test_cycle_IOWRAP" ) := [
        0 0 0 1 0 1 1 1 X 0 1 0 0 1 0 1
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ...Z
        X
         - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        0 0 0 0 0 1 1 1 X 0 1 0 0 1 0 1
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ...Z
        X
         - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  ];
        output [ "MREG_1_IOWRAP" :  "SS.1.1.1.2.2.5.1" ] ;
## Inserted final non-scan Pattern
   vector ( +, "test_cycle_IOWRAP" ) := [
        0 0 1 1 0 1 1 1 X 0 1 0 0 1 0 X
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ...Z
        X
         - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  ];
```

```
end end
```

WGL Shorted Nets Test Example

The following is an example of a WGL shorted nets test.

```
##************************************************************************##
##                              WGL VECTOR FILE                           ##
##  Encounter(TM) Test and Diagnostics 3.0     Apr 01, 2005 (linux24 TDA30)  ##
##************************************************************************##
##                                                                        ##
##  FILE CREATED.............April 01, 2005 at 09:52:40                  ##
##                                                                        ##
##  PROJECT NAME.............asp                                          ##
##                                                                        ##
##  TESTMODE.................IOWRAP                                      ##
##                                                                        ##
##  TDR......................TBAdvent                                    ##
##                                                                        ##
##  TEST PERIOD..............80       TEST STROBE TYPE..........edge      ##
##  TEST PULSE WIDTH.........8        TEST TIME UNITS...........ns        ##
##  TEST PI OFFSET...........0                                            ##
##  TEST BIDI OFFSET.........0                                            ##
##  TEST STROBE OFFSET.......72       X VALUE..................X         ##
##                                                                        ##
##  SCAN PERIOD..............80       SCAN STROBE TYPE..........edge      ##
##  SCAN PULSE WIDTH.........8        SCAN TIME UNITS...........ns        ##
##  SCAN PI OFFSET...........16                                           ##
##  SCAN BIDI OFFSET.........16                                           ##
##  SCAN STROBE OFFSET.......0        SCAN OVERLAP.............yes       ##
##                                                                        ##
##  EXPERIMENT...............1                                            ##
##                                                                        ##
##  TEST SECTION.............2        TEST SECTION TYPE........IOWRAP_shor ##
##  TESTER TERMINATION.......0        TERMINATION DOMINATION....tester    ##
##                                                                        ##
##************************************************************************##

waveform "WGL.IOWRAP.IOWRAP_shorted_nets_2*logn.ex1.ts2"

        include "WGL.IOWRAP.signals";
##************************************************************************##
##                             TIMING DEFINITIONS                         ##
##************************************************************************##

timeplate "scan_cycle" period 80ns
        "A/00/00" := input [ 0ns:P,  16ns:S ];
        "A/00/01" := input [ 0ns:P,  16ns:S ];
        "A/00/02" := input [ 0ns:P,  16ns:S ];
                        .
                        .
                        .
        "A/01/51" := output [ 0ns:Q'edge ];
        "A/01/52" := output [ 0ns:Q'edge ];
        "A/01/53" := output [ 0ns:Q'edge ];
        "A/01/54" := output [ 0ns:Q'edge ];
```

```
end

timeplate "test_cycle_IOWRAP" period 80ns
        "A/00/00" := input [ 0ns:S ];
        "A/00/01" := input [ 0ns:S ];
        "A/00/02" := input [ 0ns:S ];
        "A/00/03" := input [ 0ns:S ];
                          .
                          .
                          .
        "A/01/51" := output [ 0ns:X, 72ns:Q'edge ];
        "A/01/52" := output [ 0ns:X, 72ns:Q'edge ];
        "A/01/53" := output [ 0ns:X, 72ns:Q'edge ];
        "A/01/54" := output [ 0ns:X, 72ns:Q'edge ];
end


##*************************************************************************##
##                            DEFINE SCAN STATES                           ##
##*************************************************************************##

scanstate
 "SS.1.2.1.2.1.1.1" :=
        "MREG_1_IOWRAP" ( 011011100011011010111001111000111101000 01...) ;
        1100100110101010100010101001 0) ;
 "SS.1.2.1.2.1.5.1" :=
        "MREG_1_IOWRAP" ( X1XX01010011011010110001111001000000000XXX...) ;

 "SS.1.2.1.3.1.1.1" :=
        "MREG_1_IOWRAP" ( 011011011001110111001111001001101100111 00...) ;
                             .
                             .
                             .
 "SS.1.2.1.16.1.5.1" :=
        "MREG_1_IOWRAP" ( 000000000000000000110001100111000000000111...) ;
 "SS.1.2.1.17.1.1.1" :=
        "MREG_1_IOWRAP" ( 100010101111011001110100101100011101011 01...) ;
 "SS.1.2.1.17.1.5.1" :=
        "MREG_1_IOWRAP" ( X1XX10011111011001110011101100000000000XXX...) ;
end


##*************************************************************************##
##                                TEST VECTORS                             ##
##*************************************************************************##

pattern MAIN ( "A/00/A0", "A/00/A1", "A/00/A2", "A/00/A3", "A/00/A4", "A/00/A5",
               "A/00/A6", "A/00/A7", "A/00/A8", "A/00/A9", "A/00/B0", "A/00/B1",
             "A/00/B2", "A/00/B3", "A/00/B4", "A/00/B5", "A/00/00":I, "A/00/01":I,
               "A/00/02":I, "A/00/03":I, "A/00/04":I, "A/00/05":I, "A/00/06":I,
               "A/00/07":I, "A/00/08":I, "A/00/09":I, "A/00/10":I, "A/00/11":I,
                                         .
                                         .
                                         .
               "A/01/33":O, "A/01/34":O, "A/01/35":O, "A/01/36":O, "A/01/37":O,
               "A/01/39":O, "A/01/40":O, "A/01/41":O, "A/01/42":O, "A/01/43":O,
               "A/01/44":O, "A/01/45":O, "A/01/46":O, "A/01/47":O, "A/01/48":O,
               "A/01/49":O, "A/01/50":O, "A/01/51":O, "A/01/52":O, "A/01/53":O,
               "A/01/54":O )
##*************************************************************************##
##  TESTER LOOP...............1        PROCEDURES HAVE MEMORY....no         ##
```

```
##  TEST PROCEDURE............1        TYPE.......................init        ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          ##
##  TEST SEQUENCE.............1        TYPE.......................init        ##
##*************************************************************************##

## Processing the Static: EVENT 1.2.1.1.1.1.1:  Stim_PI:
   vector ( +, "test_cycle_IOWRAP" ) := [
        X 0 1 1 0 X X X X X 1 0 0 X X X
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];

##*************************************************************************##
##  TEST PROCEDURE............2        TYPE.......................normal      ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          ##
##  TEST SEQUENCE.............1        TYPE.......................normal      ##
##*************************************************************************##

## Processing the Static: EVENT 1.2.1.2.1.1.1:  Scan_Load:
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_IOWRAP" ) := [
        0 0 1 1 0 1 1 X X 0 1 0 0 X 0 X
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        0 0 0 0 0 1 1 X X 0 1 0 0 X 0 -
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];

## Processing the Static: EVENT 1.2.1.2.1.2.1:  Stim_PI:
## Processing the Static: EVENT 1.2.1.2.1.3.1:  Stim_PI:
## Processing the Static: EVENT 1.2.1.2.1.4.1:  Pulse:
   vector ( +, "test_cycle_IOWRAP" ) := [
        1 1 1 1 0 1 0 1 X 1 1 0 0 0 1 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
## Processing the Static: EVENT 1.2.1.2.1.5.1:  Skewed_Scan_Unload:
##*************************************************************************##
##  TEST PROCEDURE............3        TYPE.......................normal      ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          ##
##  TEST SEQUENCE.............1        TYPE.......................normal      ##
##*************************************************************************##
                                  .
                                  .
                                  .

## Processing the Static: EVENT 1.2.1.16.1.5.1:  Skewed_Scan_Unload:

##*************************************************************************##
##  TEST PROCEDURE............17       TYPE.......................normal      ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          ##
##  TEST SEQUENCE.............1        TYPE.......................normal      ##
##*************************************************************************##

## Processing the Static: EVENT 1.2.1.17.1.1.1:  Scan_Load:  ( Overlap is in Effect
 ) ## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_IOWRAP" ) := [
```

```
        0 0 1 1 0 1 1 1 X 0 1 0 0 0 0 1
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
## Inserted the Scan Sequence:  Skewed_Unload_Sequence
   vector ( +, "test_cycle_IOWRAP" ) := [
        0 0 0 1 0 1 1 1 X 0 1 0 0 0 0 1
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        0 0 0 0 0 1 1 1 X 0 1 0 0 0 0 -
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
        input  [ "MREG_1_IOWRAP" :  "SS.1.2.1.17.1.1.1" ] ,
        output [ "MREG_1_IOWRAP" :  "SS.1.2.1.16.1.5.1" ] ;

## Processing the Static: EVENT 1.2.1.17.1.2.1:  Stim_PI:
## Processing the Static: EVENT 1.2.1.17.1.3.1:  Stim_PI:
## Processing the Static: EVENT 1.2.1.17.1.4.1:  Pulse:
   vector ( +, "test_cycle_IOWRAP" ) := [
        1 1 1 1 0 1 0 1 X 1 1 0 0 1 1 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
## Processing the Static: EVENT 1.2.1.17.1.5.1:  Skewed_Scan_Unload:
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_IOWRAP" ) := [
        0 0 1 1 0 1 1 1 X 0 1 0 0 1 0 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
## Inserted the Scan Sequence:  Skewed_Unload_Sequence
   vector ( +, "test_cycle_IOWRAP" ) := [
        0 0 0 1 0 1 1 1 X 0 1 0 0 1 0 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        0 0 0 0 0 1 1 1 X 0 1 0 0 1 0 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
        output [ "MREG_1_IOWRAP" :  "SS.1.2.1.17.1.5.1" ] ;

## Inserted final non-scan Pattern
   vector ( +, "test_cycle_IOWRAP" ) := [
        0 0 1 1 0 1 1 1 X 0 1 0 0 1 0 X
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
end end
```

WGL Pattern Data Examples for Macro Tests

This section provides examples of WGL format which are typical for all Macro tests.

WGL Macro Signals File

The following is an example of the WGL Macro signals file.

```
##*************************************************************************##
##                              WGL SIGNALS FILE                           ##
##  Encounter(TM) Test and Diagnostics 3.0.Dev Apr 08, 2005 (linux24 TDA30)  ##
##*************************************************************************##
##                                                                         ##
##  FILE CREATED.............April 11, 2005 at 09:47:07                   ##
##                                                                         ##
##  PROJECT NAME.............asp                                           ##
##                                                                         ##
##  TESTMODE.................MACRO                                        ##
##                                                                         ##
##  TDR......................TBAdvent                                     ##
##                                                                         ##
##  TEST PERIOD..............80       TEST STROBE TYPE..........edge       ##
##  TEST PULSE WIDTH.........8        TEST TIME UNITS...........ns         ##
##  TEST PI OFFSET...........0                                             ##
##  TEST BIDI OFFSET.........0                                             ##
##  TEST STROBE OFFSET.......72       X VALUE..................X          ##
##                                                                         ##
##  SCAN PERIOD..............80       SCAN STROBE TYPE..........edge       ##
##  SCAN PULSE WIDTH.........8        SCAN TIME UNITS...........ns         ##
##  SCAN PI OFFSET...........16                                            ##
##  SCAN BIDI OFFSET.........16                                            ##
##  SCAN STROBE OFFSET.......0        SCAN OVERLAP.............yes        ##
##                                                                         ##
##*************************************************************************##

signal

"A/00/00" : bidir; ## pinName = A/00/00;  tf =  BDY   BDY  BIDI ; testOffset = 0;
 scanOffset = 16;
"A/00/01" : bidir; ## pinName = A/00/01;  tf = BIDI ; testOffset = 0;  scanOffset
= 16;
"A/00/02" : bidir; ## pinName = A/00/02;  tf = BIDI ; testOffset = 0;  scanOffset
= 16;
"A/00/03" : bidir; ## pinName
 = A/00/03; tf = BIDI ; testOffset = 0; scanOffset = 16;
                                        .
.                                       .
                                        .

"A/01/51" : bidir;     ## pinName = A/01/51;  tf = BIDI ; testOffset = 0;  scanOffset
 = 16;
"A/01/52" : bidir;      ## pinName = A/01/52;  tf = BIDI ; testOffset
 = 0;  scanOffset = 16;
"A/01/53" : bidir;      ## pinName = A/01/53;  tf
 = BIDI ; testOffset = 0;  scanOffset = 16;
```

```
"A/01/54" : bidir;     ## pinName = A/01/54;  tf = BIDI ; testOffset = 0;  scanOffset
= 16;
"A/01/38" : output;     ## pinName = A/01/38;  tf =  SO   BDY  ;
end
scancell
        "CYCLE_COUNT.REG.0000100.slave";
        "CYCLE_COUNT.REG.0000101.slave";
        "CYCLE_COUNT.REG.0000102.slave";
        "CYCLE_COUNT.REG.0000103.slave";
        "CYCLE_COUNT.REG.0000104.slave";
        "CYCLE_COUNT.REG.0000105.slave";
        "CYCLE_COUNT.REG.0000106.slave";
        "CYCLE_COUNT.REG.0000107.slave";
        "CYCLE_COUNT.REG.0000108.slave";
        "CYCLE_COUNT.REG.0000109.slave";
        "DATA_REG1.REG.0000100.slave";
        "DATA_REG1.REG.0000101.slave";
        "DATA_REG1.REG.0000102.slave";
                    .
                    .
                    .
        "SLEEP_REG2.REG.0000103.slave";
        "SLEEP_REG.REG.0000100.slave";
        "SLEEP_REG.REG.0000101.slave";
        "SLEEP_REG.REG.0000102.slave";
        "SLEEP_REG.REG.0000103.slave";
end
scanchain
 "MREG_1_MACRO" [
        "A/00/B5",
        "SLEEP_REG.REG.0000100.slave",
        "SLEEP_REG.REG.0000101.slave",
        "SLEEP_REG.REG.0000102.slave",
        "SLEEP_REG.REG.0000103.slave",
        "SLEEP_REG2.REG.0000100.slave",
                    .
                    .
                    .
        "DATA_REG3.REG.0000103.slave",
        "DATA_REG3.REG.0000104.slave",
        "DATA_REG3.REG.0000105.slave",
        "DATA_REG3.REG.0000106.slave",
        "DATA_REG3.REG.0000107.slave",
        "DATA_REG3.REG.0000108.slave",
        "A/01/38"
        ];
end
```

WGL Macro Test Example

The following is an example of a WGL Macro test.

```
##*************************************************************************##
##                              WGL VECTOR FILE                            ##
##  Encounter(TM) Test and Diagnostics 3.0     Apr 08, 2005 (linux24 TDA30)  ##
##*************************************************************************##
##                                                                         ##
```

```
##  FILE CREATED.............April 11, 2005 at 09:47:07                    ##
##                                                                          ##
##  PROJECT NAME.............asp                                           ##
##                                                                          ##
##  TESTMODE.................MACRO                                         ##
##                                                                          ##
##  TDR......................TBAdvent                                      ##
##                                                                          ##
##  TEST PERIOD..............80       TEST STROBE TYPE..........edge        ##
##  TEST PULSE WIDTH.........8        TEST TIME UNITS...........ns          ##
##  TEST PI OFFSET...........0                                              ##
##  TEST BIDI OFFSET.........0                                              ##
##  TEST STROBE OFFSET.......72       X VALUE...................X           ##
##                                                                          ##
##  SCAN PERIOD..............80       SCAN STROBE TYPE..........edge        ##
##  SCAN PULSE WIDTH.........8        SCAN TIME UNITS...........ns          ##
##  SCAN PI OFFSET...........16                                             ##
##  SCAN BIDI OFFSET.........16                                             ##
##  SCAN STROBE OFFSET.......0        SCAN OVERLAP..............yes         ##
##                                                                          ##
##  EXPERIMENT...............1                                             ##
##                                                                          ##
##  TEST SECTION.............1        TEST SECTION TYPE.........macro       ##
##  TESTER TERMINATION.......none     TERMINATION DOMINATION....unknown     ##
##                                                                          ##
##*************************************************************************##
##                                                                          ##
##  This Test Section has not been simulated.                               ##
##                                                                          ##
##*************************************************************************##

waveform "WGL.MACRO.macro.ex1.ts1"

        include "WGL.MACRO.signals";
##*************************************************************************##
##                            TIMING DEFINITIONS                           ##
##*************************************************************************##

timeplate "scan_cycle" period 80ns
        "A/00/00" := input [ 0ns:P,  16ns:S ];
        "A/00/01" := input [ 0ns:P,  16ns:S ];
        "A/00/02" := input [ 0ns:P,  16ns:S ];
        "A/00/03" := input [ 0ns:P,  16ns:S ];
                          .
                          .
                          .
        "A/01/51" := output [ 0ns:Q'edge ];
        "A/01/52" := output [ 0ns:Q'edge ];
        "A/01/53" := output [ 0ns:Q'edge ];
        "A/01/54" := output [ 0ns:Q'edge ];
end

timeplate "test_cycle_MACRO" period 80ns
        "A/00/00" := input [ 0ns:S ];
        "A/00/01" := input [ 0ns:S ];
        "A/00/02" := input [ 0ns:S ];
        "A/00/03" := input [ 0ns:S ];
                          .
                          .
                          .
        "A/01/51" := output [ 0ns:X, 72ns:Q'edge ];
```

```
        "A/01/52" := output [ 0ns:X, 72ns:Q'edge ];
        "A/01/53" := output [ 0ns:X, 72ns:Q'edge ];
        "A/01/54" := output [ 0ns:X, 72ns:Q'edge ];
end
##************************************************************************##
##                            DEFINE SCAN STATES                          ##
##************************************************************************##

scanstate
end


##************************************************************************##
##                               TEST VECTORS                             ##
##************************************************************************##

pattern MAIN ( "A/00/A0", "A/00/A1", "A/00/A2", "A/00/A3", "A/00/A4", "A/00/A5",
               "A/00/A6", "A/00/A7", "A/00/A8", "A/00/A9", "A/00/B0", "A/00/B1",
               "A/00/B2", "A/00/B3", "A/00/B4", "A/00/B5",
                          .
                          .
                          .
               "A/01/33":O, "A/01/34":O, "A/01/35":O, "A/01/36":O, "A/01/37":O,
               "A/01/39":O, "A/01/40":O, "A/01/41":O, "A/01/42":O, "A/01/43":O,
               "A/01/44":O, "A/01/45":O, "A/01/46":O, "A/01/47":O, "A/01/48":O,
               "A/01/49":O, "A/01/50":O, "A/01/51":O, "A/01/52":O, "A/01/53":O,
               "A/01/54":O )

##************************************************************************##
##  TESTER LOOP...............1        PROCEDURES HAVE MEMORY....yes       ##
##  TEST PROCEDURE............1        TYPE......................init      ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no        ##
##  TEST SEQUENCE.............1        TYPE......................init      ##
##************************************************************************##

## Processing the Static: EVENT 1.1.1.1.1.1.1:  Stim_PI:
   vector ( +, "test_cycle_MACRO" ) := [
        0 0 1 1 0 X X X X 0 X 0 0 X X X
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
##************************************************************************##
##  TEST PROCEDURE............2        TYPE......................normal    ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....yes       ##
##  TEST SEQUENCE.............1        TYPE......................normal    ##
##************************************************************************##

## Processing the Static: EVENT 1.1.1.2.1.1.1:  Stim_PI:
   vector ( +, "test_cycle_MACRO" ) := [
        0 0 1 1 0 0 X X X 0 X 0 0 X 1 X
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];

##************************************************************************##
##  TEST PROCEDURE............3        TYPE......................normal    ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....yes       ##
##  TEST SEQUENCE.............1        TYPE......................normal    ##
##************************************************************************##

## Processing the Dynamic Pattern as a Static: EVENT 1.1.1.3.1.1.1:  Stim_PI:
```

```
## Processing the Dynamic Pattern as a Static: EVENT 1.1.1.3.1.1.2:  Pulse:
   vector ( +, "test_cycle_MACRO" ) := [
        0 0 1 1 0 0 X̄ 1 X 0̄ X 0 0 X 0 X
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
   vector ( +, "test_cycle_MACRO" ) := [
        0 0 1 1 0 0 X̄ 1 X 0̄ X 0 0 X 1 X
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
   vector ( +, "test_cycle_MACRO" ) := [
        0 0 1 1 0 0 X̄ 1 X 0̄ X 0 0 X 0 X
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
   vector ( +, "test_cycle_MACRO" ) := [
        0 0 1 1 0 0 X̄ 1 X 0̄ X 0 0 X 1 X
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
                          .
                          .
                          .

##*************************************************************************##
##  TEST PROCEDURE............9        TYPE.......................normal      ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....yes         ##
##  TEST SEQUENCE.............1        TYPE.......................normal      ##
##*************************************************************************##

## Processing the Static: EVENT 1.1.1.9.1.1.1:  Measure_PO:
   vector ( +, "test_cycle_MACRO" ) := [
        0 0 1 1 0 0 X̄ 1 X 0̄ 1 0 0 X 1 X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        X
        0 X X X X X X X X X X X X X X X X X X X X X X X X X X X X ... ];

##*************************************************************************##
##  TEST PROCEDURE............10       TYPE.......................normal      ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....yes         ##
##  TEST SEQUENCE.............1        TYPE.......................normal      ##
##*************************************************************************##

## Processing the Dynamic Pattern as a Static: EVENT 1.1.1.10.1.1.1:  Stim_PI:
   vector ( +, "test_cycle_MACRO" ) := [
        0 0 1 1 0 0 X̄ 0 X 0̄ 1 0 0 X 1 X
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
## Inserted final non-scan Pattern
   vector ( +, "test_cycle_MACRO" ) := [
        0 0 1 1 0 0 X̄ 0 X 0̄ 1 0 0 X 1 X
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z ... Z
        X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ];
end end
```

Timed Dynamic WGL Pattern Data Examples

This section provides examples of WGL format which are typical for all timed dynamic tests.

WGL Timed Signals File

The following is an example of a WGL timed signals file.

```
                                  ##*****************************************
*******************************##
##                              WGL SIGNALS FILE                            ##
##  Encounter(TM) Test and Diagnostics 3.0     Apr 08, 2005 (linux24 TDA30)  ##
##*****************************************************************************##
##                                                                           ##
##  FILE CREATED..............April 11, 2005 at 09:47:45                   ##
##                                                                           ##
##  PROJECT NAME..............mbi                                           ##
##                                                                           ##
##  TESTMODE..................SP_WAFER_AC                                  ##
##                                                                           ##
##  TDR.......................TBAdvantdynamic                              ##
##                                                                           ##
##  TEST PERIOD...............80       TEST STROBE TYPE..........edge       ##
##  TEST PULSE WIDTH..........8        TEST TIME UNITS...........ns         ##
##  TEST PI OFFSET............0                                            ##
##  TEST BIDI OFFSET..........0                                            ##
##  TEST STROBE OFFSET........72       X VALUE...................X          ##
##                                                                           ##
b##  SCAN PERIOD...............80       SCAN STROBE TYPE..........edge       ##
##  SCAN PULSE WIDTH..........8        SCAN TIME UNITS...........ns         ##
##  SCAN PI OFFSET............16                                           ##
##  SCAN BIDI OFFSET..........16                                           ##
##  SCAN STROBE OFFSET........0        SCAN OVERLAP..............yes        ##
##                                                                           ##
##*****************************************************************************##

signal
"A0180" : bidir; ## pinName = A0180;  tf = BIDI ; testOffset = 0;  scanOffset = 16;
"A0181" : bidir; ## pinName = A0181;  tf = BIDI ; testOffset = 0;  scanOffset = 16;
"A0182" : bidir; ## pinName = A0182;  tf = BIDI ; testOffset = 0;  scanOffset = 16;
"A0183" : bidir; ## pinName = A0183;  tf = BIDI ; testOffset = 0;  scanOffset = 16;
"A0184" : bidir; ## pinName = A0184;  tf = BIDI ; testOffset = 0;  scanOffset = 16;
"A0185" : bidir; ## pinName = A0185;  tf = BIDI ; testOffset = 0;  scanOffset = 16;
"A0186" : bidir; ## pinName = A0186;  tf = BIDI ; testOffset = 0;  scanOffset = 16;
"A0187" : bidir; ## pinName = A0187;  tf = BIDI ; testOffset = 0;  scanOffset = 16;
"A0188" : bidir; ## pinName = A0188;  tf = BIDI ; testOffset = 0;  scanOffset = 16;
"A0189" : bidir; ## pinName = A0189;  tf = BIDI ; testOffset = 0;  scanOffset = 16;
"A0190" : bidir; ## pinName = A0190;  tf = BIDI ; testOffset = 0;  scanOffset = 16;
                                    .
                                    .
                                    .
 "A0175" : output;      ## pinName = A0175;  tf =  BDY  ;
        "A0176" : output;      ## pinName = A0176;  tf =  SO  ;
        "A0177" : output;      ## pinName = A0177;
        "A0178" : output;      ## pinName = A0178;
        "A0179" : output;      ## pinName = A0179;
end
```

```
scancell
        "$blk_MBAA0BO.$blk_0000100.$blk_slave";
        "$blk_MBAA1A4.$blk_0000100.$blk_slave";
        "$blk_MBAA1A6.$blk_0000100.$blk_slave";
        "$blk_MBAA1AX.$blk_0000100.$blk_slave";
        "$blk_MBAA1AY.$blk_0000100.$blk_slave";
        "$blk_MBAA1BN.$blk_0000100.$blk_slave";
                                        .
                                        .
                                        .
        "$blk_MBAENBE.$blk_0000100.$blk_slave";
        "$blk_MBAENBF.$blk_0000100.$blk_slave";
        "$blk_MBAENBG.$blk_0000100.$blk_slave";
        "$blk_MBAENBH.$blk_0000100.$blk_slave";
        "$blk_MBAENBI.$blk_0000100.$blk_slave";
end

scanchain
 "MREG_1_SP_WAFER_AC" [
        "C0170",
        "$blk_MBADUBS.$blk_0000100.$blk_slave",
        "$blk_MBAD0A7.$blk_0000100.$blk_slave",
        "$blk_MBAD6AH.$blk_0000100.$blk_slave",
        "$blk_MBADTAU.$blk_0000100.$blk_slave",
        "$blk_MBADVBP.$blk_0000100.$blk_slave",
        "$blk_MBADTAL.$blk_0000100.$blk_slave",
        "$blk_MBAD3BN.$blk_0000100.$blk_slave",
                                        .
                                        .
                                        .
        "$blk_MBAEIAC.$blk_0000100.$blk_slave",
        "$blk_MBAEDA3.$blk_0000100.$blk_slave",
        "$blk_MBAEGA6.$blk_0000100.$blk_slave",
        "A0174"
        ];
"MREG_2_SP_WAFER_AC" [
        "C0161",
        "$blk_MBAAIBT.$blk_0000100.$blk_slave",
        "$blk_MBAA1BN.$blk_0000100.$blk_slave",
        "$blk_MBAABBD.$blk_0000100.$blk_slave",
        "$blk_MBAA3A3.$blk_0000100.$blk_slave",
                                        .
                                        .
                                        .
        "$blk_MBAENA3.$blk_0000100.$blk_slave",
        "$blk_MBAENAK.$blk_0000100.$blk_slave",
        "$blk_MBAENAG.$blk_0000100.$blk_slave",
        "$blk_MBAEMBZ.$blk_0000100.$blk_slave",
        "$blk_MBAENAJ.$blk_0000100.$blk_slave",
        "$blk_MBAEMB1.$blk_0000100.$blk_slave",
        "A0176"
        ];
end
```

WGL Timed Logic Test Example

The following is an example of a WGL timed logic test.

```
##************************************************************************##
##                              WGL VECTOR FILE                           ##
##  Encounter(TM) Test and Diagnostics 3.0     Apr 08, 2005 (linux24 TDA30)  ##
##************************************************************************##
##                                                                        ##
##  FILE CREATED.............April 11, 2005 at 09:47:29                  ##
##                                                                        ##
##  PROJECT NAME.............mbi                                         ##
##                                                                        ##
##  TESTMODE.................SP_WAFER_AC                                 ##
##                                                                        ##
##  TDR......................TBAdvantdynamic                             ##
##                                                                        ##
##  TEST PERIOD..............80       TEST STROBE TYPE..........edge      ##
##  TEST PULSE WIDTH.........8        TEST TIME UNITS...........ns        ##
##  TEST PI OFFSET...........0                                            ##
##  TEST BIDI OFFSET.........0                                            ##
##  TEST STROBE OFFSET.......72       X VALUE..................X         ##
##                                                                        ##
##  SCAN PERIOD..............80       SCAN STROBE TYPE..........edge      ##
##  SCAN PULSE WIDTH.........8        SCAN TIME UNITS...........ns        ##
##  SCAN PI OFFSET...........16                                           ##
##  SCAN BIDI OFFSET.........16                                           ##
##  SCAN STROBE OFFSET.......0        SCAN OVERLAP.............yes       ##
##                                                                        ##
##  EXPERIMENT...............2                                            ##
##                                                                        ##
##  TEST SECTION.............1        TEST SECTION TYPE........logic      ##
##  TESTER TERMINATION.......0        TERMINATION DOMINATION....tester    ##
##                                                                        ##
##************************************************************************##

waveform "WGL.SP_WAFER_AC.logic.ex2.ts1"

        include "WGL.SP_WAFER_AC.signals";
##************************************************************************##
##                            TIMING DEFINITIONS                          ##
##************************************************************************##

timeplate "scan_cycle" period 80ns
        "A0180" := input [ 0ns:P,  16ns:S ];
        "A0181" := input [ 0ns:P,  16ns:S ];
        "A0182" := input [ 0ns:P,  16ns:S ];
        "A0183" := input [ 0ns:P,  16ns:S ];
                          .
                          .
                          .
        "C0168" := output [ 0ns:Q'edge ];
        "C0169" := output [ 0ns:Q'edge ];
        "C0170" := output [ 0ns:Q'edge ];
end

timeplate "test_cycle_stimclks" period 80ns
        "A0180" := input [ 0ns:S ];
        "A0181" := input [ 0ns:S ];
        "A0182" := input [ 0ns:S ];
```

```
        "A0183" := input [ 0ns:S ];
                            .
                            .
                            .
        "C0167" := output [ 0ns:X, 72ns:Q'edge ];
        "C0168" := output [ 0ns:X, 72ns:Q'edge ];
        "C0169" := output [ 0ns:X, 72ns:Q'edge ];
        "C0170" := output [ 0ns:X, 72ns:Q'edge ];
end
timeplate "test_cycle_stimclks" period 80ns
        "A0180" := input [ 0ns:S ];
        "A0181" := input [ 0ns:S ];
        "A0182" := input [ 0ns:S ];
        "A0183" := input [ 0ns:S ];
                            .
                            .
                            .
        "C0167" := output [ 0ns:X, 72ns:Q'edge ];
        "C0168" := output [ 0ns:X, 72ns:Q'edge ];
        "C0169" := output [ 0ns:X, 72ns:Q'edge ];
        "C0170" := output [ 0ns:X, 72ns:Q'edge ];
end

timeplate "test_cycle_SP_WAFER_AC" period 80ns
        "A0180" := input [ 0ns:S ];
        "A0181" := input [ 0ns:S ];
        "A0182" := input [ 0ns:S ];
        "A0183" := input [ 0ns:S ];
                            .
                            .
                            .
        "C0167" := output [ 0ns:X, 72ns:Q'edge ];
        "C0168" := output [ 0ns:X, 72ns:Q'edge ];
        "C0169" := output [ 0ns:X, 72ns:Q'edge ];
        "C0170" := output [ 0ns:X, 72ns:Q'edge ];
end


timeplate "TBautoLogicSeq1_20001221220437_0" period 25000.000000 ps
        "A0180" := input [ 0ps:S ];
        "A0181" := input [ 0ps:S ];
        "A0182" := input [ 0ps:S ];
        "A0183" := input [ 0ps:S ];
                            .
                            .
                            .
        "C0168" := output [ 0ps:X ];
        "C0169" := output [ 0ps:X ];
        "C0170" := output [ 0ps:X ];
end
##*************************************************************************##
##                            DEFINE SCAN STATES                           ##
##*************************************************************************##

scanstate
## The Following is StimLatchExtra A.
 "SS.2.1.1.2.1.1.1" :=
        "MREG_2_SP_WAFER_AC" ( 00101010001100011110010101101011000000110...1)
        "MREG_1_SP_WAFER_AC" ( 00111011010001000100101010010001000101001...1)
 "SS.2.1.1.2.1.4.1" :=
        "MREG_1_SP_WAFER_AC" ( 10101011101001011110101010001001000001000...1)
```

```
        "MREG_2_SP_WAFER_AC" ( 000001001000010011011001011000110101011 10...1)
## The Following is StimLatchExtra A.
 "SS.2.1.1.3.1.1.1" :=
        "MREG_2_SP_WAFER_AC" ( 011000010010000101010001100110010110011 10...1)
        "MREG_1_SP_WAFER_AC" ( 110010011100110101000100001010111010110 01...1)
 "SS.2.1.1.3.1.4.1" :=
        "MREG_1_SP_WAFER_AC" ( 111001001110001100100010000101111100011 00...1)
        "MREG_2_SP_WAFER_AC" ( 001100001001000010101000110011001011001 11...1)
                              .
                              .
                              .
## The Following is StimLatchExtra A.
 "SS.2.1.1.130.28.1.1" :=
        "MREG_2_SP_WAFER_AC" ( 000010001001111001010100110010001001110 00...1)
        "MREG_1_SP_WAFER_AC" ( 101100100011110000000101000000000000000 01...1)
 "SS.2.1.1.130.28.5.1" :=
        "MREG_1_SP_WAFER_AC" ( 110110011101001000000010100000000000000 00...1)
        "MREG_2_SP_WAFER_AC" ( 001010000001001100000010111001000110111 00...1)
end

##************************************************************************##
##                                TEST VECTORS                            ##
##************************************************************************##

pattern MAIN ( "A01A0", "A01A1", "A01A2", "A01A3", "A01A4", "A01A5", "A01A6",
               "A01A7", "A01A8", "A0180":I, "A0181":I, "A0182":I, "A0183":I,
               "A0184":I, "A0185":I, "A0186":I, "A0187":I, "A0188":I, "A0189":I,
                              .
                              .
                              .
               "C0147":O, "C0148":O, "C0149":O, "C0150":O, "C0151":O, "C0152":O,
               "C0153":O, "C0154":O, "C0155":O, "C0156":O, "C0157":O, "C0158":O,
               "C0159":O, "C0160":O, "C0161":O, "C0162":O, "C0163":O, "C0164":O,
              "C0165":O, "C0166":O, "C0167":O, "C0168":O, "C0169":O, "C0170":O )

##************************************************************************##
##  TESTER LOOP...............1        PROCEDURES HAVE MEMORY....no         ##
##  TEST PROCEDURE............1        TYPE......................init       ##
##  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no         ##
##  TEST SEQUENCE.............1        TYPE......................init       ##
##************************************************************************##

## Processing the Static: EVENT 2.1.1.1.1.1.1:  Stim_PI:
   vector ( +, "test_cycle_SP_WAFER_AC" ) := [
        0 X X X 0 X X 0 X
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z . . . Z
        X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X . . . X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -];

##************************************************************************##
##  TEST PROCEDURE............2        TYPE......................normal     ##
##  SLOW TO TURN OFF..........true     SEQUENCES HAVE MEMORY.....no         ##
##  STATIC FAULTS............10279    PERCENT STATIC FAULTS.....55.604004  ##
##  DYNAMIC FAULTS...........1871     PERCENT DYNAMIC FAULTS....31.914642  ##
##  TEST SEQUENCE.............1        TYPE......................normal     ##
##************************************************************************##
## Processing the Static: EVENT 2.1.1.2.1.1.1:  l froSkewed_Scan_Load:
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_SP_WAFER_AC" ) := [
        0 X X X 0 0 1 0 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z . . . Z
```

```
        X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X . . . X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -];
   vector ( +, "test_cycle_SP_WAFER_AC" ) := [
        0 X X X 0 0 1 0 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z . . . Z
        X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X . . . X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        1 X X X 0 0 1 0 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z . . . Z
        X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X . . . X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -];
        input  [ "MREG_2_SP_WAFER_AC" :  "SS.2.1.1.2.1.1.1" ] ,
        input  [ "MREG_1_SP_WAFER_AC" :  "SS.2.1.1.2.1.1.1" ] ;
## Inserted the Scan Sequence:  Skewed_Load_Sequence
   vector ( +, "test_cycle_SP_WAFER_AC" ) := [
        0 X X X 0 0 1 0 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z . . . Z
        X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X . . . X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -];

## Processing the Static: EVENT 2.1.1.2.1.2.1:  Stim_PI:
   vector ( +, "test_cycle_SP_WAFER_AC" ) := [
        0 X X X 0 0 1 0 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z . . . Z
        X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X . . . X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -];
## Processing the Dynamic Timed: EVENT 2.1.1.2.1.3.1:  Pulse:
## Processing the Dynamic Timed: EVENT 2.1.1.2.1.3.2:  Pulse:
   vector ( +, "TBautoLogicSeq43_20001221221553_0" ) := [
        0 X X X 0 0 1 0 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z . . . Z
        X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X . . . X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -];
## Processing the Static: EVENT 2.1.1.2.1.4.1:  Skewed_Scan_Unload:
                                .
                                .
                                .
##*************************************************************************##
##  TEST SEQUENCE.............28       TYPE......................normal      ##
##*************************************************************************##

## Processing the Static: EVENT 2.1.1.130.28.1.1:  Skewed_Scan_Load:
( Overlap is in Effect )
## Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   vector ( +, "test_cycle_SP_WAFER_AC" ) := [
        0 X X X 0 0 1 0 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z . . . Z
        X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X . . . X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -];
   vector ( +, "test_cycle_SP_WAFER_AC" ) := [
        0 X X X 0 0 1 0 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z . . . Z
        X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X . . . X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -];
## Inserted the Scan Sequence:  Skewed_Unload_Sequence
   vector ( +, "test_cycle_SP_WAFER_AC" ) := [
        1 X X X 0 0 1 0 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z . . . Z
        X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X . . . X
```

```
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -];
## Inserted the Scan Sequence:  Scan_Sequence
   scan   ( +, "scan_cycle" ) := [
        1 X X X 0 0 1 0 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z . . . Z
        X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X . . . X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -];
        input  [ "MREG_2_SP_WAFER_AC" :  "SS.2.1.1.130.28.1.1" ] ,
        input  [ "MREG_1_SP_WAFER_AC" :  "SS.2.1.1.130.28.1.1" ] ,
        output [ "MREG_1_SP_WAFER_AC" :  "SS.2.1.1.130.27.5.1" ] ,
        output [ "MREG_2_SP_WAFER_AC" :  "SS.2.1.1.130.27.5.1" ] ;
## Inserted the Scan Sequence:  Skewed_Load_Sequence
   vector ( +, "test_cycle_SP_WAFER_AC" ) := [
        0 X X X 0 0 1 0 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z . . . Z
        X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X . . . X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -];
                                .
                                .
                                .

## Inserted final non-scan Pattern
   vector ( +, "test_cycle_SP_WAFER_AC" ) := [
        0 X X X 0 0 1 0 0
        Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z . . . Z
        X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X . . . X
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -];
end end
```

D

STIL Pattern Data Examples

This section provides examples of STIL format which are typical for Deterministic Logic tests, Macro tests, LBIST tests, and ICT tests.

Deterministic STIL Pattern Data Examples

This section provides examples of STIL format which are typical for all deterministically derived tests.

STIL Deterministic Signals File

The following is an example of a STIL deterministic signals file.

```
STIL 1.0;


//*************************************************************************//
//                            STIL SIGNALS FILE                            //
//  Encounter(TM) Test   2.0.0 Feb 24, 2004 (aix43_64 TDA20)               //
//*************************************************************************  //
//                                                                         //
//  FILE CREATED.............February 24, 2004 at 10:33:48                //
//                                                                         //
//  PROJECT NAME.............lbc                                          //
//                                                                         //
//  TESTMODE.................lssd                                         //
//                                                                         //
//  TDR......................dummy_tester_lssd                            //
//                                                                         //
//  TEST PERIOD..............80       TEST STROBE TYPE..........edge      //
//  TEST PULSE WIDTH.........8        TEST TIME UNITS...........ns        //
//  TEST PI OFFSET...........0                                            //
//  TEST BIDI OFFSET.........0                                            //
//  TEST STROBE OFFSET.......72       X VALUE..................Z          //
//                                                                         //
//  SCAN PERIOD..............80       SCAN STROBE TYPE..........edge      //
//  SCAN PULSE WIDTH.........8        SCAN TIME UNITS...........ns        //
```

```
//  SCAN PI OFFSET............16                                          //
//  SCAN BIDI OFFSET..........16                                          //
//  SCAN STROBE OFFSET........0        SCAN OVERLAP..............yes      //
//                                                                        //
//************************************************************************//
//************************************************************************//
//                              DEFINE SIGNALS                            //
//************************************************************************//

Signals {
       "A" In;      // pinName = A;  tf = -AC  ;  piEntry = 1;  hierIndex = 0; . . .
       "B" In;      // pinName = B;  tf = -BC  ;  piEntry = 2;  hierIndex = 1; . . .
       "C" In;      // pinName = C;  tf = -SC  ;  piEntry = 3;  hierIndex = 2; . . .
       "CS" In;      // pinName = CS;  piEntry = 4;  hierIndex = 3;  flatIndex . . .
                                        .
                                        .
                                        .
        "DO3" Out;      // pinName = DO3;  poEntry = 3;  hierIndex = 19;  . . .
        "DO4" Out;      // pinName = DO4;  poEntry = 4;  hierIndex = 20;  . . .
        "SO1" Out;      // pinName = SO1;  poEntry = 5;  hierIndex = 21;  . . .
        "SO2" Out;      // pinName = SO2;  poEntry = 7;  hierIndex = 23;  . . .
}
//************************************************************************//
//                           DEFINE SIGNAL GROUPS                         //
//************************************************************************//

SignalGroups {
   "ALLPIs" = '"A"+"B"+"C"+"CS"+"DI1"+"DI2"+"DI3"+"DI4"+"ENABLE1"+"ENABLE2"
            +"ME"+"PS"+"SEL"+"SI1"+"SI2"+"ST1"+"ST2"';

   "ALLPOs" = '"DO1"+"DO2"+"DO3"+"DO4"+"SO1"+"SO2"';

   "ALLIOs" = '"SO1_BIDI"+"SO2_BIDI"';

   "SI0001_lssd" = '"SI1"' { ScanIn 9; }
   "SI0002_lssd" = '"SI2"' { ScanIn 16; }

   "SO0001_lssd" = '"SO1_BIDI"' { ScanOut 9; }
   "SO0002_lssd" = '"SO2_BIDI"' { ScanOut 16; }

   "ALLACs_lssd" = '"A"';

   "ALLBCs_lssd" = '"B"';

   "ALLSIs_lssd" = '"SI0001_lssd"+"SI0002_lssd"';

   "ALLSOs_lssd" = '"SO0001_lssd"+"SO0002_lssd"';

}
//************************************************************************//
//                              DEFINE MACROS                             //
//************************************************************************//

MacroDefs {
   "TEST" { WaveformTable "test_cycle";
      Vector {
         "ALLPIs" = %;
         "ALLPOs" = %;
         "ALLIOs" = %; } }
```

```
   "SCAN_lssd" { WaveformTable "scan_cycle_lssd";
      Condition {
         "ALLSIs_lssd" = 00;
         "ALLPOs" = XXXXXX;
         "ALLIOs" = XX; }
      Shift { Vector {
         "ALLSOs_lssd" = #;
         "ALLSIs_lssd" = #;
         "ALLACs_lssd" = P;
         "ALLBCs_lssd" = P; } } }

}
```

STIL LSSD Flush Test Example

The following is an example of a STIL LSSD flush test.

```
STIL 1.0;

//*************************************************************************//
//                              STIL VECTOR FILE                           //
//  Encounter(TM) Test                 2.0.0 Feb 24, 2004 (aix43_64 TDA20)  //
//*************************************************************************//
//                                                                         //
//  FILE CREATED.............February 24, 2004 at 10:33:48                 //
//                                                                         //
//  PROJECT NAME.............lbc                                          //
//                                                                         //
//  TESTMODE.................lssd                                         //
//                                                                         //
//  TDR......................dummy_tester_lssd                            //
//                                                                         //
//  TEST PERIOD..............80       TEST STROBE TYPE..........edge      //
//  TEST PULSE WIDTH.........8        TEST TIME UNITS...........ns        //
//  TEST PI OFFSET...........0                                            //
//  TEST BIDI OFFSET.........0                                            //
//  TEST STROBE OFFSET.......72       X VALUE..................Z          //
//                                                                         //
//  SCAN PERIOD..............80       SCAN STROBE TYPE..........edge      //
//  SCAN PULSE WIDTH.........8        SCAN TIME UNITS...........ns        //
//  SCAN PI OFFSET...........16                                           //
//  SCAN BIDI OFFSET.........16                                           //
//  SCAN STROBE OFFSET.......0        SCAN OVERLAP.............yes        //
//                                                                         //
//  EXPERIMENT...............1        DATA FORMAT..............binary     //
//                                                                         //
//  TEST SECTION.............1        TEST SECTION TYPE........flush      //
//  TESTER TERMINATION.......0        TERMINATION DOMINATION....tester    //
//                                                                         //
//*************************************************************************//

        Include "STIL.lssd.signals";

//*************************************************************************//
//                            TIMING DEFINITIONS                           //
//*************************************************************************//
```

```
Timing {

   WaveformTable "test_cycle" { Period '1280ns';
 Waveforms {
        "A" { 01ZP { '0ns' P/P/P/D; '8ns' D/U/Z/U; '16ns' D/U/Z/D; } }
        "B" { 01ZP { '0ns' P/P/P/D; '24ns' D/U/Z/U; '32ns' D/U/Z/D; } }
        "C" { 01ZP { '0ns' P/P/P/D; '8ns' D/U/Z/U; '16ns' D/U/Z/D; } }
        "CS" { 01Z  { '0ns' D/U/Z; } }
        "DI1" { 01Z  { '0ns' D/U/Z; } }
                        .
                        .
                        .
        "DO4" { LHTX { '0ns' X; '1152ns' L/H/T/X; } }
        "SO1" { LHTX { '0ns' X; '1152ns' L/H/T/X; } }
        "SO1_BIDI" { LHTX { '0ns' X; '1152ns' L/H/T/X; } }
        "SO2" { LHTX { '0ns' X; '1152ns' L/H/T/X; } }
        "SO2_BIDI" { LHTX { '0ns' X; '1152ns' L/H/T/X; } }
} }
}

//*************************************************************************//
//                              TEST VECTORS                               //
//*************************************************************************//

PatternBurst
   MAIN_BRST { Termination { "ALLPOs" TerminateLow; "ALLIOs" TerminateLow; }
               PatList { MAIN_TEST; } }
PatternExec
   MAIN_EXEC { PatternBurst MAIN_BRST; }

Pattern
   MAIN_TEST {

//*************************************************************************//
//  TESTER LOOP...............1        PROCEDURES HAVE MEMORY....no        //
//  TEST PROCEDURE............1        TYPE......................init      //
//  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no        //
//  TEST SEQUENCE.............1        TYPE......................init      //
//*************************************************************************//

// Processing the Static: EVENT 1.1.1.1.1.1.1:  StimPI:
 Macro "TEST" {
        "ALLPIs" = 000
14 Z ;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }

//*************************************************************************//
//  TEST PROCEDURE............2        TYPE......................normal    //
//  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....yes       //
//  TEST SEQUENCE.............1        TYPE......................normal    //
//*************************************************************************//

// Processing the Static: EVENT 1.1.1.2.1.1.1:  StimPI:
// Processing the Static: EVENT 1.1.1.2.1.2.1:  StimPI:

//*************************************************************************//
//  TEST SEQUENCE.............2        TYPE......................normal    //
//*************************************************************************//
```

```
// Processing the Static: EVENT 1.1.1.2.2.1.1:  StimPI:
// Processing the Static: EVENT 1.1.1.2.2.2.1:  StimClock:
// Processing the Static: EVENT 1.1.1.2.2.2.2:  StimClock:
   Macro "TEST" {
        "ALLPIs" = 1100000011100ZZ01;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
// Processing the Static: EVENT 1.1.1.2.2.2.3:  StimPI:
// Processing the Static: EVENT 1.1.1.2.2.2.4:  MeasurePO:
"1.1.1.2.2.2":
   Macro "TEST" {
        "ALLPIs" = 11000000111000001;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = LL; }

//*************************************************************************//
//  TEST SEQUENCE.............5        TYPE......................normal      //
//*************************************************************************//

// Processing the Static: EVENT 1.1.1.2.5.1.1:  StimClock:
// Processing the Static: EVENT 1.1.1.2.5.1.2:  StimClock:
   Macro "TEST" {
        "ALLPIs" =
8 0 111000001;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
// Inserted final non-scan Pattern
   Macro "TEST" {
        "ALLPIs" =
8 0 111000001;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
}
```

STIL Scan Chain Test Example

The following is an example of a STIL scan chain test.

```
STIL 1.0;


//*************************************************************************//
//                            STIL VECTOR FILE                             //
//  Encounter(TM) Test                 2.0.0 Feb 24, 2004 (aix43_64 TDA20)  //
//*************************************************************************//
//                                                                         //
//  FILE CREATED..............February 24, 2004 at 10:33:48                //
//                                                                         //
//  PROJECT NAME..............lbc                                          //
//                                                                         //
//  TESTMODE..................lssd                                         //
//                                                                         //
//  TDR.......................dummy_tester_lssd                            //
//                                                                         //
//  TEST PERIOD...............80       TEST STROBE TYPE..........edge      //
//  TEST PULSE WIDTH..........8        TEST TIME UNITS...........ns        //
//  TEST PI OFFSET............0                                            //
```

```
//  TEST BIDI OFFSET..........0                                          //
//  TEST STROBE OFFSET........72       X VALUE...................Z       //
//                                                                       //
//  SCAN PERIOD...............80       SCAN STROBE TYPE..........edge   //
//  SCAN PULSE WIDTH..........8        SCAN TIME UNITS...........ns     //
//  SCAN PI OFFSET............16                                         //
//  SCAN BIDI OFFSET..........16                                         //
//  SCAN STROBE OFFSET........0        SCAN OVERLAP..............yes    //
//                                                                       //
//  EXPERIMENT................1        DATA FORMAT...............binary  //
//                                                                       //
//  TEST SECTION..............2        TEST SECTION TYPE.........scan   //
//  TESTER TERMINATION........0        TERMINATION DOMINATION....tester //
//                                                                       //
//***********************************************************************//

        Include "STIL.lssd.signals";

//***********************************************************************//
//                            TIMING DEFINITIONS                         //
//***********************************************************************//

Timing {

   WaveformTable "test_cycle" { Period '80ns';
      Waveforms {
 "A" { 01ZP { '0ns' P/P/P/D; '8ns' D/U/Z/U; '16ns' D/U/Z/D; } }
        "B" { 01ZP { '0ns' P/P/P/D; '24ns' D/U/Z/U; '32ns' D/U/Z/D; } }
        "C" { 01ZP { '0ns' P/P/P/D; '8ns' D/U/Z/U; '16ns' D/U/Z/D; } }
        "CS" { 01Z  { '0ns' D/U/Z; } }
                                .
                                .
                                .
        "SO1" { LHTX { '0ns' X; '72ns' L/H/T/X; } }
        "SO1_BIDI" { LHTX { '0ns' X; '72ns' L/H/T/X; } }
        "SO2" { LHTX { '0ns' X; '72ns' L/H/T/X; } }
        "SO2_BIDI" { LHTX { '0ns' X; '72ns' L/H/T/X; } }
} }

   WaveformTable "scan_cycle_lssd" { Period '80ns';
      Waveforms {
        "A" { 01ZP { '0ns' P/P/P/D; '24ns' D/U/Z/U; '32ns' D/U/Z/D; } }
        "B" { 01ZP { '0ns' P/P/P/D; '40ns' D/U/Z/U; '48ns' D/U/Z/D; } }
        "C" { 01ZP { '0ns' P/P/P/D; '24ns' D/U/Z/U; '32ns' D/U/Z/D; } }
                                .
                                .
                                .
        "SO1" { LHTX { '0ns' L/H/T/X; } }
        "SO1_BIDI" { LHTX { '0ns' L/H/T/X; } }
        "SO2" { LHTX { '0ns' L/H/T/X; } }
        "SO2_BIDI" { LHTX { '0ns' L/H/T/X; } }
} }
}

//***********************************************************************//
//                              TEST VECTORS                             //
//***********************************************************************//

PatternBurst
 MAIN_BRST { Termination { "ALLPOs" TerminateLow; "ALLIOs" TerminateLow; }
              PatList { MAIN_TEST; } }
```

```
PatternExec
   MAIN_EXEC { PatternBurst MAIN_BRST; }

Pattern
   MAIN_TEST {

//*************************************************************************//
//  TESTER LOOP...............1        PROCEDURES HAVE MEMORY....no          //
//  TEST PROCEDURE............1        TYPE......................init        //
//  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          //
//  TEST SEQUENCE.............1        TYPE......................init        //
//*************************************************************************//

// Processing the Static: EVENT 1.2.1.1.1.1.1:  StimPI:
   Macro "TEST" {
        "ALLPIs" = 000
14 Z ;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }

//*************************************************************************//
//  TEST PROCEDURE............2        TYPE......................normal      //
//  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....yes         //
//  STATIC FAULTS.............571      PERCENT STATIC FAULTS.....54.277565   //
//  TEST SEQUENCE.............1        TYPE......................normal      //
//*************************************************************************//

// Processing the Static: EVENT 1.2.1.2.1.1.1:  StimLatch:
// Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   Macro "TEST" {
        "ALLPIs" = 000ZZZZZ111Z0ZZ01;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
// Inserted the Scan Sequence:  Scan_Sequence
   Macro "SCAN_lssd" {
        "SI0001_lssd" = 110011001;
        "SI0002_lssd" =
4 1001 ; }

// Processing the Static: EVENT 1.2.1.2.1.2.1:  StimPI:

//*************************************************************************//
//  TEST SEQUENCE.............2        TYPE......................normal      //
//*************************************************************************//

// Processing the Static: EVENT 1.2.1.2.2.1.1:  StimPI:
// Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   Macro "TEST" {
        "ALLPIs" = 000ZZZZZ111Z0ZZ01;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
// Inserted the Scan Sequence:  Scan_Sequence
   Macro "SCAN_lssd" {
        "SI0001_lssd" = 110011001;
        "SI0002_lssd" =
4 1001 ; }

// Processing the Static: EVENT 1.2.1.2.1.2.1:  StimPI:

//*************************************************************************//
//  TEST SEQUENCE.............2        TYPE......................normal      //
```

```
//*************************************************************************//

// Processing the Static: EVENT 1.2.1.2.2.1.1:  StimPI:
// Processing the Static: EVENT 1.2.1.2.2.2.1:  StimPI:

//*************************************************************************//
//  TEST SEQUENCE.............3        TYPE......................normal      //
//*************************************************************************//

// Processing the Static: EVENT 1.2.1.2.3.1.1:  StimPI:
// Processing the Static: EVENT 1.2.1.2.3.1.2:  Pulse:
// Processing the Static: EVENT 1.2.1.2.3.1.3:  Pulse:
// Processing the Static: EVENT 1.2.1.2.3.1.4:  MeasurePO:
"1.2.1.2.3.1":
   Macro "TEST" {
        "ALLPIs" = PP000000111001101;
        "ALLPOs" = HHHLHL;
        "ALLIOs" = HL; }
                              .
                              .
                              .

//*************************************************************************//
//  TEST SEQUENCE.............7        TYPE......................normal      //
//*************************************************************************//

// Processing the Static: EVENT 1.2.1.2.7.1.1:  MeasureLatch:
// Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
// Inserted the Scan Sequence:  Scan_Sequence
"1.2.1.2.7.1":
   Macro "SCAN_lssd" {
        "SO0001_lssd" = HHLLHHLLH;
 "SO0002_lssd" =
4 HLLH ; }

// Inserted final non-scan Pattern
   Macro "TEST" {
        "ALLPIs" = 0000010111110ZZ01;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
}
```

STIL Logic Test Example

The following is an example of a STIL logic test.

```
STIL 1.0;
//*************************************************************************//
//                              STIL VECTOR FILE                           //
//  Encounter(TM) Test                2.0.0 Feb 24, 2004 (aix43_64 TDA20)  //
//*************************************************************************//
//                                                                         //
//  FILE CREATED.............February 24, 2004 at 10:33:49                //
//                                                                         //
//  PROJECT NAME.............lbc                                          //
//                                                                         //
//  TESTMODE.................lssd                                         //
//                                                                         //
```

```
//  TDR.......................dummy_tester_lssd                              //
//                                                                            //
//  TEST PERIOD...............80       TEST STROBE TYPE..........edge        //
//  TEST PULSE WIDTH..........8        TEST TIME UNITS...........ns          //
//  TEST PI OFFSET............0                                             //
//  TEST BIDI OFFSET..........0                                             //
//  TEST STROBE OFFSET........72       X VALUE...................Z           //
//                                                                            //
//  SCAN PERIOD...............80       SCAN STROBE TYPE..........edge        //
//  SCAN PULSE WIDTH..........8        SCAN TIME UNITS...........ns          //
//  SCAN PI OFFSET............16                                            //
//  SCAN BIDI OFFSET..........16                                            //
//  SCAN STROBE OFFSET........0        SCAN OVERLAP..............yes         //
//                                                                            //
//  EXPERIMENT................2        DATA FORMAT...............binary      //
//                                                                            //
//  TEST SECTION..............1        TEST SECTION TYPE.........logic       //
//  TESTER TERMINATION........0        TERMINATION DOMINATION....tester      //
//                                                                            //
//****************************************************************************//

        Include "STIL.lssd.signals";

//****************************************************************************//
//                            TIMING DEFINITIONS                              //
//****************************************************************************//

Timing {

   WaveformTable "test_cycle" { Period '80ns';
      Waveforms {
        "A" { 01ZP { '0ns' P/P/P/D; '8ns' D/U/Z/U; '16ns' D/U/Z/D; } }
 "B" { 01ZP { '0ns' P/P/P/D; '24ns' D/U/Z/U; '32ns' D/U/Z/D; } }
        "C" { 01ZP { '0ns' P/P/P/D; '8ns' D/U/Z/U; '16ns' D/U/Z/D; } }
        "CS" { 01Z  { '0ns' D/U/Z; } }
                        .
                        .
                        .
        "SO1" { LHTX { '0ns' X; '72ns' L/H/T/X; } }
        "SO1_BIDI" { LHTX { '0ns' X; '72ns' L/H/T/X; } }
        "SO2" { LHTX { '0ns' X; '72ns' L/H/T/X; } }
        "SO2_BIDI" { LHTX { '0ns' X; '72ns' L/H/T/X; } }
} }

   WaveformTable "scan_cycle_lssd" { Period '80ns';
      Waveforms {
        "A" { 01ZP { '0ns' P/P/P/D; '24ns' D/U/Z/U; '32ns' D/U/Z/D; } }
        "B" { 01ZP { '0ns' P/P/P/D; '40ns' D/U/Z/U; '48ns' D/U/Z/D; } }
        "C" { 01ZP { '0ns' P/P/P/D; '24ns' D/U/Z/U; '32ns' D/U/Z/D; } }
                        .
                        .
                        .
        "SO1" { LHTX { '0ns' L/H/T/X; } }
        "SO1_BIDI" { LHTX { '0ns' L/H/T/X; } }
        "SO2" { LHTX { '0ns' L/H/T/X; } }
        "SO2_BIDI" { LHTX { '0ns' L/H/T/X; } }
} }
}

//****************************************************************************//
//                                TEST VECTORS                                //
```

```
//*************************************************************************//

PatternBurst
   MAIN_BRST { Termination { "ALLPOs" TerminateLow; "ALLIOs" TerminateLow; }
               PatList { MAIN_TEST; } }
PatternExec
   MAIN_EXEC { PatternBurst MAIN_BRST; }

Pattern
   MAIN_TEST {

//*************************************************************************//
//  TESTER LOOP...............1        PROCEDURES HAVE MEMORY....no          //
//  TEST PROCEDURE............1        TYPE......................init        //
//  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          //
//  TEST SEQUENCE.............1        TYPE......................init        //
//*************************************************************************//

// Processing the Static: EVENT 2.1.1.1.1.1.1:  StimPI:
   Macro "TEST" {
        "ALLPIs" = 000
14 Z ;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }

//*************************************************************************//
//  TEST PROCEDURE............2        TYPE......................normal      //
//  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          //
//  STATIC FAULTS.............18       PERCENT STATIC FAULTS.....55.988594   //
//  TEST SEQUENCE.............1        TYPE......................normal      //
//*************************************************************************//

// Processing the Static: EVENT 2.1.1.2.1.1.1:  StimLatch:
// Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   Macro "TEST" {
        "ALLPIs" = 000ZZZZZ111Z0ZZ01;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
// Inserted the Scan Sequence:  Scan_Sequence
   Macro "SCAN_lssd" {
        "SI0001_lssd" = 001010000;
        "SI0002_lssd" = 1111001001100111; }

// Processing the Static: EVENT 2.1.1.2.1.2.1:  StimPI:
// Processing the Static: EVENT 2.1.1.2.1.3.1:  StimPI:
// Processing the Static: EVENT 2.1.1.2.1.4.1:  MeasurePO:
"2.1.1.2.1.4":
   Macro "TEST" {
        "ALLPIs" = 00010111000011110;
        "ALLPOs" = LLLLLL;
        "ALLIOs" = 11; }

//*************************************************************************//
//  TEST SEQUENCE.............2        TYPE......................normal      //
//*************************************************************************//

// Processing the Static: EVENT 2.1.1.2.2.1.1:  StimLatch:
// Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   Macro "TEST" {
        "ALLPIs" = 00010111111001101;
        "ALLPOs" = XXXXXX;
```

```
        "ALLIOs" = ZZ; }
// Inserted the Scan Sequence:  Scan_Sequence
   Macro "SCAN_lssd" {
        "SI0001_lssd" = 111000001;
    "SI0002_lssd" = 0111000101111011; }

// Processing the Static: EVENT 2.1.1.2.2.2.1:  StimPI:
// Processing the Static: EVENT 2.1.1.2.2.3.1:  StimPI:
// Processing the Static: EVENT 2.1.1.2.2.4.1:  MeasurePO:
"2.1.1.2.2.4":
   Macro "TEST" {
        "ALLPIs" = 00001100100011010;
        "ALLPOs" = LHHHLL;
        "ALLIOs" = LL; }

//*************************************************************************//
//  TEST PROCEDURE............3        TYPE.......................normal      //
//  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          //
//  STATIC FAULTS.............4        PERCENT STATIC FAULTS.....56.368820   //
//  TEST SEQUENCE.............1        TYPE.......................normal      //
//*************************************************************************//

// Processing the Static: EVENT 2.1.1.3.1.1.1:  StimLatchextraAclock:
// Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   Macro "TEST" {
        "ALLPIs" = 00001100111001001;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
// Inserted the Scan Sequence:  Scan_Sequence
   Macro "SCAN_lssd" {
        "SI0001_lssd" = 011011101;
        "SI0002_lssd" = 0110100101101000; }
// Inserted the Scan Sequence:  Skewed_Load_Sequence
   Macro "TEST" {
        "ALLPIs" = P0001100111000001;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }

// Processing the Static: EVENT 2.1.1.3.1.2.1:  StimPI:
// Processing the Static: EVENT 2.1.1.3.1.2.2:  MeasurePO:
// Processing the Static: EVENT 2.1.1.3.1.3.1:  StimPI:
// Processing the Static: EVENT 2.1.1.3.1.3.2:  MeasurePO:
"2.1.1.3.1.3":
   Macro "TEST" {
        "ALLPIs" = 00001100000110110;
        "ALLPOs" = LLLLLL;
        "ALLIOs" = LL; }
// Processing the Static: EVENT 2.1.1.3.1.4.1:  Pulse:
   Macro "TEST" {
        "ALLPIs" = 0P001100000110110;
        "ALLPOs" = XXXXXX;
    "ALLIOs" = ZZ; }
// Processing the Static: EVENT 2.1.1.3.1.5.1:  MeasureLatch:
                        .
                        .
                        .

//*************************************************************************//
//  TEST SEQUENCE.............14       TYPE.......................normal      //
//*************************************************************************//
```

```
// Processing the Static: EVENT 2.1.1.6.14.1.1:  StimLatch:  ( Overlap is in Effect
 ) // Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
// Inserted the Scan Sequence:  Skewed_Unload_Sequence
   Macro "TEST" {
        "ALLPIs" = 0P000101111000001;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
// Inserted the Scan Sequence:  Scan_Sequence
"2.1.1.6.13.5":
   Macro "SCAN_lssd" {
        "SO0001_lssd" = LLHHHLHLL;
        "SO0002_lssd" = HLLHLLHHLLHLHHLH;
        "SI0001_lssd" = 011011110;
        "SI0002_lssd" = 1111101001100010; }

// Processing the Static: EVENT 2.1.1.6.14.2.1:  StimPI:
// Processing the Static: EVENT 2.1.1.6.14.2.2:  MeasurePO:
// Processing the Static: EVENT 2.1.1.6.14.3.1:  StimPI:
// Processing the Static: EVENT 2.1.1.6.14.3.2:  MeasurePO:
"2.1.1.6.14.3":
   Macro "TEST" {
        "ALLPIs" = 00011001001011110;
        "ALLPOs" = LLLLLL;
        "ALLIOs" = LL; }
// Processing the Static: EVENT 2.1.1.6.14.4.1:  Pulse:
   Macro "TEST" {
        "ALLPIs" = P0011001001011110;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
// Processing the Static: EVENT 2.1.1.6.14.5.1:  BclockMeasureLatch:

//*************************************************************************//
//  TEST PROCEDURE............7        TYPE.......................normal      //
//  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          //
//  STATIC FAULTS.............103      PERCENT STATIC FAULTS.....80.228134   //
//  TEST SEQUENCE.............1        TYPE.......................normal      //
//*************************************************************************//

// Processing the Static: EVENT 2.1.1.7.1.1.1:  StimLatch:  ( Overlap is in Effect )
// Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
// Inserted the Scan Sequence:  Skewed_Unload_Sequence
   Macro "TEST" {
        "ALLPIs" = 0P011001111001101;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
// Inserted the Scan Sequence:  Scan_Sequence
"2.1.1.6.14.5":
   Macro "SCAN_lssd" {
        "SO0001_lssd" = LLLLHHHLL;
        "SO0002_lssd" = HHHHLHLHHHLHLHLH;
        "SI0001_lssd" = 111111011;
        "SI0002_lssd" =
7 1 001011100; }

// Processing the Static: EVENT 2.1.1.7.1.2.1:  StimPI:
// Processing the Static: EVENT 2.1.1.7.1.2.2:  MeasurePO:
"2.1.1.7.1.2":
   Macro "TEST" {
        "ALLPIs" = 00000110000010110;
        "ALLPOs" = LLLLLL;
        "ALLIOs" = LL; }
```

```
// Processing the Static: EVENT 2.1.1.7.1.3.1:  Pulse:
   Macro "TEST" {
        "ALLPIs" = 00P00110000010110;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
// Processing the Static: EVENT 2.1.1.7.1.4.1:  BclockMeasureLatch:
                     .
                     .
                     .
//*************************************************************************//
//  TEST SEQUENCE.............14       TYPE......................normal      //
//*************************************************************************//

// Processing the Static: EVENT 2.1.1.8.14.1.1:  StimLatch:  ( Overlap is in Effect
 ) // Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
// Inserted the Scan Sequence:  Skewed_Unload_Sequence
   Macro "TEST" {
        "ALLPIs" = 0P001001111101101;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
// Inserted the Scan Sequence:  Scan_Sequence
"2.1.1.8.13.4":
   Macro "SCAN_lssd" {
  "SO0001_lssd" = HLLHHLLLH;
        "SO0002_lssd" = HHHHLLHHLLHLHLLH;
        "SI0001_lssd" = 101101000;
        "SI0002_lssd" = 1111010000100000; }

// Processing the Static: EVENT 2.1.1.8.14.2.1:  StimPI:
// Processing the Static: EVENT 2.1.1.8.14.2.2:  MeasurePO:
"2.1.1.8.14.2":
   Macro "TEST" {
        "ALLPIs" = 00001100000011010;
        "ALLPOs" = LLLLLL;
        "ALLIOs" = LL; }
// Processing the Static: EVENT 2.1.1.8.14.3.1:  Pulse:
   Macro "TEST" {
        "ALLPIs" = 00P01100000011010;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
// Processing the Static: EVENT 2.1.1.8.14.4.1:  BclockMeasureLatch:
// Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
// Inserted the Scan Sequence:  Skewed_Unload_Sequence
   Macro "TEST" {
        "ALLPIs" = 0P001100111001001;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
// Inserted the Scan Sequence:  Scan_Sequence
"2.1.1.8.14.4":
   Macro "SCAN_lssd" {
        "SO0001_lssd" = HLHHLHLLL;
        "SO0002_lssd" = LLHHHLLHLLLLLLHH; }

// Inserted final non-scan Pattern
   Macro "TEST" {
        "ALLPIs" = 0000110011100ZZ01;
        "ALLPOs" = XXXXXX;
        "ALLIOs" = ZZ; }
}
```

STIL IDDq Test Example

The following is an example of a STIL IDDq test.

```
STIL 1.0;
//*************************************************************************//
//                            STIL VECTOR FILE                             //
//  Encounter(TM) Test                2.0.0 Feb 24, 2004 (aix43_64 TDA20)   //
//*************************************************************************//
//                                                                         //
//  FILE CREATED.............February 24, 2004 at 10:36:41                 //
//                                                                         //
//  PROJECT NAME.............btv                                           //
//                                                                         //
//  TESTMODE.................IDDQ                                        //
//                                                                         //
//  TDR......................TBAdvent_new                                //
//                                                                         //
//  TEST PERIOD..............80       TEST STROBE TYPE..........edge      //
//  TEST PULSE WIDTH.........8        TEST TIME UNITS...........ns        //
//  TEST PI OFFSET...........0                                            //
//  TEST BIDI OFFSET.........0                                            //
//  TEST STROBE OFFSET.......72       X VALUE..................Z          //
//                                                                         //
//  SCAN PERIOD..............80       SCAN STROBE TYPE..........edge      //
//  SCAN PULSE WIDTH.........8        SCAN TIME UNITS...........ns        //
//  SCAN PI OFFSET...........16                                           //
//  SCAN BIDI OFFSET.........16                                           //
//  SCAN STROBE OFFSET.......0        SCAN OVERLAP.............yes       //
//                                                                         //
//  EXPERIMENT...............1        DATA FORMAT..............binary     //
//                                                                         //
//  TEST SECTION.............1        TEST SECTION TYPE........IDDq       //
//  TESTER TERMINATION.......0        TERMINATION DOMINATION....tester    //
//                                                                         //
//*************************************************************************//

        Include "STIL.IDDQ.signals";

//*************************************************************************//
//                           TIMING DEFINITIONS                            //
//*************************************************************************//

Timing {

   WaveformTable "test_cycle" { Period '80ns';
      Waveforms {
        "A2" { 01ZP { '0ns' P/P/P/D; '8ns' D/U/Z/U; '16ns' D/U/Z/D; } }
        "A4" { 01Z  { '0ns' D/U/Z; } }
        "A5" { 01Z  { '0ns' D/U/Z; } }
        "A6" { 01ZP { '0ns' P/P/P/D; '8ns' D/U/Z/U; '16ns' D/U/Z/D; } }
                              .
                              .
                              .
        "T0" { 01Z  { '0ns' D/U/Z; } }
        "60" { LHTX { '0ns' X; '72ns' L/H/T/X; } }
        "61" { LHTX { '0ns' X; '72ns' L/H/T/X; } }
        "70" { LHTX { '0ns' X; '72ns' L/H/T/X; } }
} }
```

```
   WaveformTable "scan_cycle_IDDQ" { Period '80ns';
      Waveforms {
        "A2" { 01ZP { '0ns' P/P/P/D; '24ns' D/U/Z/U; '32ns' D/U/Z/D; } }
        "A4" { 01Z  { '0ns' P;       '16ns' D/U/Z; } }
        "A5" { 01Z  { '0ns' P;       '16ns' D/U/Z; } }
        "A6" { 01ZP { '0ns' P/P/P/D; '24ns' D/U/Z/U; '32ns' D/U/Z/D; } }
                              .
                              .
                              .
        "T0" { 01Z  { '0ns' P;       '16ns' D/U/Z; } }
        "60" { LHTX { '0ns' L/H/T/X; } }
        "61" { LHTX { '0ns' L/H/T/X; } }
        "70" { LHTX { '0ns' L/H/T/X; } }
} }
}

//*************************************************************************//
//                              TEST VECTORS                               //
//*************************************************************************//

PatternBurst
   MAIN_BRST { Termination { "ALLPOs" TerminateLow; }
               PatList { MAIN_TEST; } }
PatternExec
   MAIN_EXEC { PatternBurst MAIN_BRST; }

Pattern
   MAIN_TEST {

//*************************************************************************//
//  TESTER LOOP...............1        PROCEDURES HAVE MEMORY....no        //
//  TEST PROCEDURE............1        TYPE......................init      //
//  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no        //
//  TEST SEQUENCE.............1        TYPE......................init      //
//*************************************************************************//

// Processing the Static: EVENT 1.1.1.1.1.1.1:  StimPI:
   Macro "TEST" {
        "ALLPIs" = 0ZZ0100000Z00ZZZZZZZ00ZZZ;
        "ALLPOs" = XXX; }

//*************************************************************************//
//  TEST PROCEDURE............2        TYPE......................normal    //
//  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no        //
//  IDDQ FAULTS...............8160     PERCENT IDDQ FAULTS.......20.079729 //
//  TEST SEQUENCE.............1        TYPE......................normal    //
//*************************************************************************//

// Processing the Static: EVENT 1.1.1.2.1.1.1:  StimLatch:
// Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   Macro "TEST" {
        "ALLPIs" = 00101000001001ZZZZ110001Z;
        "ALLPOs" = XXX; }
// Inserted the Scan Sequence:  Scan_Sequence
   Macro "SCAN_IDDQ" {
        "SI0001_IDDQ" = 00;
        "SI0002_IDDQ" = 1000100010100000011100100010000101000001001010 1 . . .:

// Processing the Static: EVENT 1.1.1.2.1.2.1:  StimPI:
// Processing the Static: EVENT 1.1.1.2.1.2.2:  Pulse:
```

```
   Macro "TEST" {
        "ALLPIs" = 001010000010P1ZZZZ110001Z;
        "ALLPOs" = XXX; }
// Processing the Static: EVENT 1.1.1.2.1.3.1:  StimPI:
   Macro "TEST" {
        "ALLPIs" = 01001
9 0 11000000100;
        "ALLPOs" = XXX; }
// Processing the Static: EVENT 1.1.1.2.1.4.1:  MeasureCurrent:

//*************************************************************************//
//  TEST PROCEDURE............3        TYPE.......................normal      //
//  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          //
//  IDDQ FAULTS...............5364     PERCENT IDDQ FAULTS.......33.279198   //
//  TEST SEQUENCE.............1        TYPE.......................normal      //
//*************************************************************************//

// Processing the Static: EVENT 1.1.1.3.1.1.1:  StimLatch:
// Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   Macro "TEST" {
        "ALLPIs" = 00101000001001110011000 10;
        "ALLPOs" = XXX; }
        IddqTestPoint;
// Inserted the Scan Sequence:  Scan_Sequence
   Macro "SCAN_IDDQ" {
        "SI0001_IDDQ" = 11;
        "SI0002_IDDQ" = 1001100100110010010010011111011010101000010101 1 . . . ;
            110011; }

// Processing the Static: EVENT 1.1.1.3.1.2.1:  StimPI:
// Processing the Static: EVENT 1.1.1.3.1.2.2:  Pulse:
   Macro "TEST" {
        "ALLPIs" = 001010000010P11ZZ01100010;
        "ALLPOs" = XXX; }
// Processing the Static: EVENT 1.1.1.3.1.3.1:  StimPI:
   Macro "TEST" {
        "ALLPIs" = 01001
10 0 1010000100;
        "ALLPOs" = XXX; }
// Processing the Static: EVENT 1.1.1.3.1.4.1:  MeasureCurrent:
                              .
                              .
                              .

//*************************************************************************//
//  TEST PROCEDURE............91       TYPE.......................normal      //
//  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          //
//  IDDQ FAULTS...............45       PERCENT IDDQ FAULTS.......99.606277   //
//  TEST SEQUENCE.............1        TYPE.......................normal      //
//*************************************************************************//

// Processing the Static: EVENT 1.1.1.91.1.1.1:  StimLatch:
// Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   Macro "TEST" {
        "ALLPIs" = 00101000001001001111000 10;
        "ALLPOs" = XXX; }
// Inserted the Scan Sequence:  Scan_Sequence
   Macro "SCAN_IDDQ" {
        "SI0001_IDDQ" = 11;
        "SI0002_IDDQ" = 1100101111110001101101110011111101001111011000 1 . . . ;
```

```
// Processing the Static: EVENT 1.1.1.91.1.2.1:  StimPI:
// Processing the Static: EVENT 1.1.1.91.1.2.2:  Pulse:
   Macro "TEST" {
        "ALLPIs" = 001010000010P10ZZ11100010;
        "ALLPOs" = XXX; }
// Processing the Static: EVENT 1.1.1.91.1.3.1:  StimPI:
// Processing the Static: EVENT 1.1.1.91.1.4.1:  StimClock:
   Macro "TEST" {
        "ALLPIs" = 01001
9 0 10100010100;
        "ALLPOs" = XXX; }
// Processing the Static: EVENT 1.1.1.91.1.5.1:  MeasureCurrent:
// Processing the Static: EVENT 1.1.1.91.1.6.1:  StimClock:
   Macro "TEST" {
        "ALLPIs" = 01001
9 0 10100000100;
        "ALLPOs" = XXX; }
        IddqTestPoint;

//*************************************************************************//
//  TEST PROCEDURE............92       TYPE.......................normal      //
//  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          //
//  IDDQ FAULTS...............41       PERCENT IDDQ FAULTS.......99.707169   //
//  TEST SEQUENCE.............1        TYPE.......................normal      //
//*************************************************************************//

// Processing the Static: EVENT 1.1.1.92.1.1.1:  StimLatch:
// Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   Macro "TEST" {
        "ALLPIs" = 00101000001001101011000 10;
        "ALLPOs" = XXX; }
// Inserted the Scan Sequence:  Scan_Sequence
   Macro "SCAN_IDDQ" {
        "SI0001_IDDQ" = 00;
        "SI0002_IDDQ" = 1010001011110011101010011110000011000011011010 . . . ;

// Processing the Static: EVENT 1.1.1.92.1.2.1:  StimPI:
// Processing the Static: EVENT 1.1.1.92.1.2.2:  Pulse:
   Macro "TEST" {
        "ALLPIs" = 001010000010P11ZZ01100010;
        "ALLPOs" = XXX; }
// Processing the Static: EVENT 1.1.1.92.1.3.1:  StimPI:
// Processing the Static: EVENT 1.1.1.92.1.4.1:  StimClock:
   Macro "TEST" {
        "ALLPIs" = 00001
9 0 10000010100;
        "ALLPOs" = XXX; }
// Processing the Static: EVENT 1.1.1.92.1.5.1:  MeasureCurrent:
// Processing the Static: EVENT 1.1.1.92.1.6.1:  StimClock:
   Macro "TEST" {
        "ALLPIs" = 00001
9 0 10000000100;
        "ALLPOs" = XXX; }
        IddqTestPoint;
// Inserted final non-scan Pattern
   Macro "TEST" {
        "ALLPIs" = 00001
9 0 10000000100;
        "ALLPOs" = XXX; }
}
```

Timed Dynamic STIL Pattern Data Examples

This section provides examples of STIL format which are typical for all timed dynamic tests.

STIL Timed Signals File

The following is an example of a STIL timed signals file.

```
STIL 1.0;
//*************************************************************************//
//                              STIL SIGNALS FILE                          //
//  Encounter(TM) Test                2.0.0 Feb 24, 2004 (aix43_64 TDA20)   //
//*************************************************************************//
//                                                                         //
//  FILE CREATED.............February 24, 2004 at 10:37:39                 //
//                                                                         //
//  PROJECT NAME.............mbi                                          //
//                                                                         //
//  TESTMODE.................SP_WAFER_AC                                  //
//                                                                         //
//  TDR......................TBAdvantdynamic                              //
//                                                                         //
//  TEST PERIOD..............80       TEST STROBE TYPE..........edge       //
//  TEST PULSE WIDTH.........8        TEST TIME UNITS...........ns         //
//  TEST PI OFFSET...........0                                             //
//  TEST BIDI OFFSET.........0                                             //
//  TEST STROBE OFFSET.......72       X VALUE..................Z           //
//                                                                         //
//  SCAN PERIOD..............80       SCAN STROBE TYPE..........edge       //
//  SCAN PULSE WIDTH.........8        SCAN TIME UNITS...........ns         //
//  SCAN PI OFFSET...........16                                            //
//  SCAN BIDI OFFSET.........16                                            //
//  SCAN STROBE OFFSET.......0        SCAN OVERLAP.............yes        //
//                                                                         //
//*************************************************************************//

//*************************************************************************//
//                              DEFINE SIGNALS                             //
//*************************************************************************//

Signals {
        "A0180" InOut;      // pinName = A0180;  tf = BIDI ;  piEntry = 1; . . .
        "A0181" InOut;      // pinName = A0181;  tf = BIDI ;  piEntry = 2; . . .
        "A0182" InOut;      // pinName = A0182;  tf = BIDI ;  piEntry = 3; . . .
        "A0183" InOut;      // pinName = A0183;  tf = BIDI ;  piEntry = 4; . . .
        "A0176" Out;      // pinName = A0176;  tf =  SO  ;  poEntry = 77;  . . .
        "A0177" Out;      // pinName = A0177;  poEntry = 78;  hierIndex = 8. . .
        "A0178" Out;      // pinName = A0178;  poEntry = 79;  hierIndex = 8. . .
        "A0179" Out;      // pinName = A0179;  poEntry = 80;  hierIndex = 8. . .
}
//*************************************************************************//
//                            DEFINE SIGNAL GROUPS                         //
//*************************************************************************//

SignalGroups {
   "ALLPIs" = '"A01A0"+"A01A1"+"A01A2"+"A01A3"+"A01A4"+"A01A5"+"A01A6"
```

```
            +"A01A7"+"A01A8"';

   "ALLPOs" = '"A0100"+"A0101"+"A0102"+"A0103"+"A0104"+"A0105"+"A0106"
            +"A0107"+"A0108"+"A0109"+"A0110"+"A0111"+"A0112"+"A0113"+"A0114"
            +"A0115"+"A0116"+"A0117"+"A0118"+"A0119"+"A0120"+"A0121"+"A0122"
            +"A0123"+"A0124"+"A0125"+"A0126"+"A0127"+"A0128"+"A0129"+"A0130"
            +"A0131"+"A0132"+"A0133"+"A0134"+"A0135"+"A0136"+"A0137"+"A0138"
            +"A0139"+"A0140"+"A0141"+"A0142"+"A0143"+"A0144"+"A0145"+"A0146"
            +"A0147"+"A0148"+"A0149"+"A0150"+"A0151"+"A0152"+"A0153"+"A0154"
            +"A0155"+"A0156"+"A0157"+"A0158"+"A0159"+"A0160"+"A0161"+"A0162"
            +"A0163"+"A0164"+"A0165"+"A0166"+"A0167"+"A0168"+"A0169"+"A0170"
            +"A0171"+"A0172"+"A0173"+"A0174"+"A0175"+"A0176"+"A0177"+"A0178"
            +"A0179"';

   "ALLIOs" = '"A0180"+"A0181"+"A0182"+"A0183"+"A0184"+"A0185"+"A0186"
            +"A0187"+"A0188"+"A0189"+"A0190"+"A0191"+"A0192"+"A0193"+"A0194"
            +"A0195"+"A0196"+"A0197"+"A0198"+"A0199"+"B0100"+"B0101"+"B0102"
            +"B0103"+"B0104"+"B0105"+"B0106"+"B0107"+"B0108"+"B0109"+"B0110"
            +"B0111"+"B0112"+"B0113"+"B0114"+"B0115"+"B0116"+"B0117"+"B0118"
            +"B0119"+"B0120"+"B0121"+"B0122"+"B0123"+"B0124"+"B0125"+"B0126"
            +"B0127"+"B0128"+"B0129"+"B0130"+"B0131"+"B0132"+"B0133"+"B0134"
            +"B0135"+"B0136"+"B0137"+"B0138"+"B0139"+"B0140"+"B0141"+"B0142"
            +"B0143"+"B0144"+"B0145"+"B0146"+"B0147"+"B0148"+"B0149"+"B0150"
            +"B0151"+"B0152"+"B0153"+"B0154"+"B0155"+"B0156"+"B0157"+"B0158"
            +"B0159"+"B0160"+"B0161"+"B0162"+"B0163"+"B0164"+"B0165"+"B0166"
            +"B0167"+"B0168"+"B0169"+"B0170"+"B0171"+"B0172"+"B0173"+"B0174"
            +"B0175"+"B0176"+"B0177"+"B0178"+"B0179"+"B0180"+"B0181"+"B0182"
            +"B0183"+"B0184"+"B0185"+"B0186"+"B0187"+"B0188"+"B0189"+"B0190"
            +"B0191"+"B0192"+"B0193"+"B0194"+"B0195"+"B0196"+"B0197"+"B0198"
            +"B0199"+"C0100"+"C0101"+"C0102"+"C0103"+"C0104"+"C0105"+"C0106"
            +"C0107"+"C0108"+"C0109"+"C0110"+"C0111"+"C0112"+"C0113"+"C0114"
            +"C0115"+"C0116"+"C0117"+"C0118"+"C0119"+"C0120"+"C0121"+"C0122"
            +"C0123"+"C0124"+"C0125"+"C0126"+"C0127"+"C0128"+"C0129"+"C0130"
            +"C0131"+"C0132"+"C0133"+"C0134"+"C0135"+"C0136"+"C0137"+"C0138"
            +"C0139"+"C0140"+"C0141"+"C0142"+"C0143"+"C0144"+"C0145"+"C0146"
            +"C0147"+"C0148"+"C0149"+"C0150"+"C0151"+"C0152"+"C0153"+"C0154"
            +"C0155"+"C0156"+"C0157"+"C0158"+"C0159"+"C0160"+"C0161"+"C0162"
            +"C0163"+"C0164"+"C0165"+"C0166"+"C0167"+"C0168"+"C0169"+"C0170"';

   "SI0001_SP_WAFER_AC" = '"C0161"' { ScanIn 1190; }
   "SI0002_SP_WAFER_AC" = '"C0170"' { ScanIn 1278; }

   "SO0001_SP_WAFER_AC" = '"A0174"' { ScanOut 1278; }
   "SO0002_SP_WAFER_AC" = '"A0176"' { ScanOut 1190; }

   "ALLACs_SP_WAFER_AC" = '"C0167"';

   "ALLBCs_SP_WAFER_AC" = '"A01A0"';

   "ALLSIs_SP_WAFER_AC" = '"SI0001_SP_WAFER_AC"+"SI0002_SP_WAFER_AC"';

   "ALLSOs_SP_WAFER_AC" = '"SO0001_SP_WAFER_AC"+"SO0002_SP_WAFER_AC"';

}
//*************************************************************************//
//                              DEFINE MACROS                              //
//*************************************************************************//

MacroDefs {
   "TEST" { WaveformTable "test_cycle";
      Vector {
```

```
         "ALLPIs" = %;
         "ALLPOs" = %;
         "ALLIOs" = %; } }

   "SCAN_SP_WAFER_AC" { WaveformTable "scan_cycle_SP_WAFER_AC";
      Condition {
         "ALLSIs_SP_WAFER_AC" = 00;
         "ALLPOs" =
80 X;
         "ALLIOs" =
191 X; }
      Shift { Vector {
         "ALLSOs_SP_WAFER_AC" = #;
         "ALLSIs_SP_WAFER_AC" = #;
         "ALLACs_SP_WAFER_AC" = P;
         "ALLBCs_SP_WAFER_AC" = P; } } }


   "TBautoLogicSeq1_1_0" { WaveformTable "TBautoLogicSeq1_1_0_cycle";
      Vector {
         "ALLPIs" = %;
         "ALLPOs" = %;
         "ALLIOs" = %; } }
                              .
                              .
                              .

   "TBautoLogicSeq353_107_0" { WaveformTable "TBautoLogicSeq353_107_0_cycle";
      Vector {
         "ALLPIs" = %;
         "ALLPOs" = %;
         "ALLIOs" = %; } }
}
```

STIL Timed Logic Test Example

The following is an example of a STIL timed logic test.

```
STIL 1.0;
//*************************************************************************//
//                              STIL VECTOR FILE                           //
//  Encounter(TM) Test                2.0.0 Feb 24, 2004 (aix43_64 TDA20)   //
//*************************************************************************//
//                                                                         //
//  FILE CREATED.............February 24, 2004 at 10:37:51                 //
//                                                                         //
//  PROJECT NAME.............mbi                                           //
//                                                                         //
//  TESTMODE.................SP_WAFER_AC                                  //
//                                                                         //
//  TDR......................TBAdvantdynamic                              //
//                                                                         //
//  TEST PERIOD..............80       TEST STROBE TYPE..........edge       //
//  TEST PULSE WIDTH.........8        TEST TIME UNITS...........ns         //
//  TEST PI OFFSET...........0                                             //
//  TEST BIDI OFFSET.........0                                             //
//  TEST STROBE OFFSET.......72       X VALUE..................Z           //
//                                                                         //
```

```
//  SCAN PERIOD...............80       SCAN STROBE TYPE..........edge        //
//  SCAN PULSE WIDTH..........8        SCAN TIME UNITS...........ns          //
//  SCAN PI OFFSET............16                                             //
//  SCAN BIDI OFFSET..........16                                             //
//  SCAN STROBE OFFSET........0        SCAN OVERLAP..............yes         //
//                                                                           //
//  EXPERIMENT................2        DATA FORMAT...............binary      //
//                                                                           //
//  TEST SECTION..............1        TEST SECTION TYPE.........logic       //
//  TESTER TERMINATION........0        TERMINATION DOMINATION....tester      //
//                                                                           //
//***************************************************************************//
        Include "STIL.SP_WAFER_AC.signals";

//***************************************************************************//
//                            TIMING DEFINITIONS                             //
//***************************************************************************//

Timing {

   WaveformTable "test_cycle" { Period '80ns';
      Waveforms {
        "A0180" { 01Z  { '0ns' D/U/Z; } }
        "A0181" { 01Z  { '0ns' D/U/Z; } }
        "A0182" { 01Z  { '0ns' D/U/Z; } }
        "A0183" { 01Z  { '0ns' D/U/Z; } }
                      .
                      .
                      .
        "C0167" { LHTX { '0ps' X; } }
        "C0168" { LHTX { '0ps' X; } }
        "C0169" { LHTX { '0ps' X; } }
        "C0170" { LHTX { '0ps' X; } }
} }
}

//***************************************************************************//
//                               TEST VECTORS                                //
//***************************************************************************//

PatternBurst
   MAIN_BRST { Termination { "ALLPOs" TerminateLow; "ALLIOs" TerminateLow; }
               PatList { MAIN_TEST; } }
PatternExec
   MAIN_EXEC { PatternBurst MAIN_BRST; }

Pattern
   MAIN_TEST {

//***************************************************************************//
//  TESTER LOOP...............1        PROCEDURES HAVE MEMORY....no          //
//  TEST PROCEDURE............1        TYPE......................init        //
//  SLOW TO TURN OFF..........false    SEQUENCES HAVE MEMORY.....no          //
//  TEST SEQUENCE.............1        TYPE......................init        //
//***************************************************************************//

// Processing the Static: EVENT 2.1.1.1.1.1.1:  StimPI:
   Macro "TEST" {
        "ALLPIs" = 0ZZZ0ZZ0Z;
        "ALLPOs" = 80 X ;
        "ALLIOs" = 180 Z 0ZZ1Z100ZZZ; }
```

```
//*************************************************************************//
//  TEST PROCEDURE............2        TYPE.......................normal      //
//  SLOW TO TURN OFF..........true     SEQUENCES HAVE MEMORY.....no          //
//  STATIC FAULTS.............10279    PERCENT STATIC FAULTS.....55.604004   //
//  DYNAMIC FAULTS............1871     PERCENT DYNAMIC FAULTS....31.914642   //
//  TEST SEQUENCE.............1        TYPE.......................normal      //
//*************************************************************************//

// Processing the Static: EVENT 2.1.1.2.1.1.1:  StimLatchextraAclock:
// Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   Macro "TEST" {
        "ALLPIs" = 0ZZZ00100;
        "ALLPOs" = 80 X ;
        "ALLIOs" = 180 Z 0ZZ101001ZZ; }
// Inserted the Scan Sequence:  Scan_Sequence
   Macro "SCAN_SP_WAFER_AC" {
        "SI0001_SP_WAFER_AC" = 1011111001011001110101010110101110101 . . .; }
        "SI0002_SP_WAFER_AC" = 00010
3 0011 1101110100010100010111 . . .; }
// Inserted the Scan Sequence:  Skewed_Load_Sequence
   Macro "TEST" {
        "ALLPIs" = 0ZZZ00100;
        "ALLPOs" = 80 X ;
        "ALLIOs" = 180 Z 01Z1010P1Z1; }

// Processing the Static: EVENT 2.1.1.2.1.2.1:  StimPI:
   Macro "TEST" {
        "ALLPIs" = 0ZZZ00100;
        "ALLPOs" = 80 X ;
        "ALLIOs" = 180 Z 0ZZ101001ZZ; }
// Processing the Dynamic: EVENT 2.1.1.2.1.3.1:  Pulse:
// Processing the Dynamic: EVENT 2.1.1.2.1.3.2:  Pulse:
   Macro "TBautoLogicSeq43_40_0" {
        "ALLPIs" = 0ZZZ00100;
        "ALLPOs" = 80 X ;
        "ALLIOs" = 180 Z 0ZZP01P01ZZ; }
// Processing the Static: EVENT 2.1.1.2.1.4.1:  BclockMeasureLatch:
                        .
                        .
                        .
// Processing the Static: EVENT 2.1.1.130.28.3.1:  StimPI:
   Macro "TEST" {
        "ALLPIs" = 0ZZZ00100;
        "ALLPOs" = 80 X ;
        "ALLIOs" = 180 Z 0ZZ101001ZZ; }
// Processing the Dynamic: EVENT 2.1.1.130.28.4.1:  Pulse:
// Processing the Dynamic: EVENT 2.1.1.130.28.4.2:  StimPI:
// Processing the Dynamic: EVENT 2.1.1.130.28.4.3:  Pulse:
   Macro "TBautoLogicSeq1_1_0" {
        "ALLPIs" = PZZZ00001;
        "ALLPOs" = 80 X ;
        "ALLIOs" = 180 Z 0ZZ111P00ZZ; }
// Processing the Static: EVENT 2.1.1.130.28.5.1:  BclockMeasureLatch:
// Inserted the Scan Sequence:  Scan_Preconditioning_Sequence
   Macro "TEST" {
        "ALLPIs" = 0ZZZ00100;
        "ALLPOs" = 80 X ;
        "ALLIOs" = 180 Z 0ZZ1Z100ZZZ; }
// Inserted the Scan Sequence:  Skewed_Unload_Sequence
   Macro "TEST" {
```

```
        "ALLPIs" = PZZZ00100;
        "ALLPOs" = 80 X ;
        "ALLIOs" = 180 Z 0ZZ101001ZZ; }
// Inserted the Scan Sequence:  Scan_Sequence
"2.1.1.130.28.5":
   Macro "SCAN_SP_WAFER_AC" {
        "SO0001_SP_WAFER_AC" = LLHHLLLLLLHLHHHLHHLHLHHLLHHLHLLHL . . .; }
        "SO0002_SP_WAFER_AC" = LLHHHHHLHHHHHHLHHLLLLLHHLHLLLHHLL . . .; }

// Inserted final non-scan Pattern
   Macro "TEST" {
        "ALLPIs" = 0ZZZ00100;
        "ALLPOs" = 80 X ;
        "ALLIOs" = 180 Z 0ZZ101001ZZ; }
}
```

E

Verilog Pattern Data Examples

This section provides examples of Verilog format which are typical for deterministically derived tests and LBIST tests, as produced by Encounter Test. All examples use a serial scan format and a window strobe.

Deterministic Test Verilog Examples

This section provides examples of Verilog format which are typical for all deterministically derived tests such as flush, scan, logic, and driver and receiver tests.

Verilog IDDq Test Main Simulation File

The following is an example of a Verilog IDDq test.

```
//*************************************************************************//
//                           VERILOG MAINSIM FILE                          //
// Encounter(TM) Test and Diagnostics 3.1.Dev Jan 04, 2006 (linux24_64 ET31) //
//*************************************************************************//
//                                                                         //
//  FILE CREATED..............January 04, 2006 at 15:43:24                //
//                                                                         //
//  PROJECT NAME..............btv                                         //
//                                                                         //
//  TESTMODE..................IDDQ                                        //
//                                                                         //
//  TDR.......................TBAdvent_new                                //
//                                                                         //
//  TEST PERIOD...............80.000   TEST TIME UNITS...........ns       //
//  TEST PULSE WIDTH..........8.000                                        //
//  TEST STROBE OFFSET........72.000   TEST STROBE TYPE..........edge     //
//  TEST BIDI OFFSET..........0.000                                        //
//  TEST PI OFFSET............0.000    X VALUE...................X       //
//                                                                         //
//  TEST PI OFFSET for pin "A2" (PI # 1) is ......................8.000   //
//  TEST PI OFFSET for pin "A6" (PI # 4) is ......................8.000   //
//  TEST PI OFFSET for pin "A7" (PI # 5) is ......................8.000   //
//  TEST PI OFFSET for pin "A8" (PI # 6) is ......................8.000   //
```

```
//  TEST PI OFFSET for pin "B0" (PI # 7) is ......................8.000       //
//  TEST PI OFFSET for pin "B1" (PI # 8) is ......................24.000      //
//  TEST PI OFFSET for pin "B3" (PI # 9) is ......................8.000       //
//  TEST PI OFFSET for pin "B4" (PI # 10) is .....................8.000       //
//  TEST PI OFFSET for pin "B7" (PI # 12) is .....................8.000       //
//  TEST PI OFFSET for pin "B8" (PI # 13) is .....................24.000      //
//  TEST PI OFFSET for pin "L2" (PI # 21) is .....................8.000       //
//  TEST PI OFFSET for pin "L3" (PI # 22) is .....................24.000      //
//                                                                            //
//  SCAN FORMAT...............serial   SCAN OVERLAP..............yes         //
//  SCAN PERIOD...............80.000   SCAN TIME UNITS...........ns          //
//  SCAN PULSE WIDTH..........8.000                                           //
//  SCAN STROBE OFFSET........0.000    SCAN STROBE TYPE..........edge        //
//  SCAN BIDI OFFSET..........16.000                                          //
//  SCAN PI OFFSET............16.000   X VALUE...................X           //
//                                                                            //
//  SCAN PI OFFSET for pin "B0" (PI # 7) is ......................24.000      //
//  SCAN PI OFFSET for pin "B1" (PI # 8) is ......................40.000      //
//  SCAN PI OFFSET for pin "B8" (PI # 13) is .....................40.000      //
//  SCAN PI OFFSET for pin "L2" (PI # 21) is .....................24.000      //
//  SCAN PI OFFSET for pin "L3" (PI # 22) is .....................40.000      //
//                                                                            //
//****************************************************************************//

 `timescale 1 ns / 1 ps

  module btv_IDDQ;

//****************************************************************************//
//                DEFINE VARIABLES FOR ALL PRIMARY I/O PORTS                  //
//****************************************************************************//

  reg      [1:0025] stim_PIs;

  reg      [1:0025] part_PIs;

  reg      [1:0025] stim_CIs;

  reg      [1:0003] resp_POs;

  wire     [1:0003] part_POs;


//****************************************************************************//
//                   DEFINE VARIABLES FOR ALL SHIFT CHAINS                    //
//****************************************************************************//

  reg      [1:2070] stim_SLs;

  reg      [1:0002] stim_SSs;

  reg      [1:2070] resp_MLs;

//****************************************************************************//
//                              OTHER DEFINITIONS                             //
//****************************************************************************//

  integer  CYCLE, PInum, POnum, ORnum, MODENUM, EXPNUM, SCANOPNUM, SEQNUM, TASK
  integer  ERR, CMD, FID, TID, CNT, TOT, CID, LIX, MAX, FAILSETID;
  integer  sim_start [1:15], sim_count [1:15];
  reg      [1:8185] name_POs [1:0003];
```

```
  reg      sim_trace, sim_heart, sim_range, failset, global_term;
  reg      [1:800] PATTERN, pattern, TESTFILE, SOD, EOD;
  reg      [1:8184] FILE, COMMENT;
  reg      no_Pin_Found;

//*************************************************************************//
//        INSTANTIATE THE STRUCTURE AND CONNECT TO VERILOG VARIABLES       //
//*************************************************************************//

  BTV2
    BTV2_inst (
       .\60          (part_POs[0001]),      // pinName = 60;  tf =  SO   BDY  ;
       .\61          (part_POs[0002]),      // pinName = 61;  tf =  BDY  ;
       .\70          (part_POs[0003]),      // pinName = 70;  tf =  SO   BDY  ;

       .A2           (part_PIs[0001]), //      pinName = A2;  tf = -SC  ; testO
       .A4           (part_PIs[0002]), //      pinName = A4;  tf = -SE  ; testO
       .A5           (part_PIs[0003]), //      pinName = A5;  tf = +SE  ; testO
       .A6           (part_PIs[0004]), //      pinName = A6;  tf = -SC  ; testO
       .A7           (part_PIs[0005]), //      pinName = A7;  tf = +SC  ; testO
       .A8           (part_PIs[0006]), //      pinName = A8;  tf = -SC  ; testO
       .B0           (part_PIs[0007]), //      pinName = B0;  tf = -AC  ; testO
       .B1           (part_PIs[0008]), //      pinName = B1;  tf = -BC  ; testO
       .B3           (part_PIs[0009]), //      pinName = B3;  tf = -SC  ; testO
       .B4           (part_PIs[0010]), //      pinName = B4;  tf = -SC  ; testO
       .B6           (part_PIs[0011]), //      pinName = B6;  tf = +SE  ; testO
       .B7           (part_PIs[0012]), //      pinName = B7;  tf = -SC  ; testO
       .B8           (part_PIs[0013]), //      pinName = B8;  tf = -PC  ; testO
       .E1           (part_PIs[0014]), //      pinName = E1;  tf = +SE  -BI  ;
       .F3           (part_PIs[0015]), //      pinName = F3;  tf =  BDY  ; test
       .F4           (part_PIs[0016]), //      pinName = F4;  tf =  SI   BDY  ;
       .I0           (part_PIs[0017]), //      pinName = I0;  tf =  SI   BDY  ;
       .I1           (part_PIs[0018]), //      pinName = I1;  tf =  BDY  ; test
       .L0           (part_PIs[0019]), //      pinName = L0;  tf = +SE  ; testO
       .L1           (part_PIs[0020]), //      pinName = L1;  tf = +SE  ; testO
       .L2           (part_PIs[0021]), //      pinName = L2;  tf = -AC  ; testO
       .L3           (part_PIs[0022]), //      pinName = L3;  tf = -BC  ; testO
       .S0           (part_PIs[0023]), //      pinName = S0;  tf = -SE  ; testO
       .S1           (part_PIs[0024]), //      pinName = S1;  tf = +SE  ; testO
       .T0           (part_PIs[0025])); //      pinName = T0;  tf =  BDY  ; tes

//*************************************************************************//
//                        MAKE SOME OTHER CONNECTIONS                      //
//*************************************************************************//

  assign ( weak0, weak1 ) // Termination
       part_POs[0001] = global_term,     // pinName = 60;  tf =  SO   BDY  ;
       part_POs[0002] = global_term,     // pinName = 61;  tf =  BDY  ;
       part_POs[0003] = global_term;      // pinName = 70;  tf =  SO   BDY  ;


//*************************************************************************//
//                      OPEN THE FILE AND RUN SIMULATION                   //
//*************************************************************************//

  initial
    begin

       FILE = 0;
       sim_setup;
```

```
      for ( TID = 1; TID <= 99; TID = TID + 1 ) begin
        $sformat ( TESTFILE, "TESTFILE%0d=", TID );
        if ( $value$plusargs ( TESTFILE, FILE )) begin
          FID = $fopen ( FILE, "r" );
          if ( FID )  sim_vector_file;
          else $display ( "\n951 ERROR (TVE-951): Failed to open the file: %0s
        end
      end

      if ( FAILSETID )  $fclose ( FAILSETID );
      if ( FILE )
        $display ( "\n203 INFO (TVE-203): The total number of miscomparing vec
      else
        $display ( "\n661 WARNING (TVE-661): No input data files found. The da
      $finish;

    end

//*************************************************************************//
//                      DEFINE SIMULATION SETUP PROCEDURE                  //
//*************************************************************************//

  task sim_setup;
    begin

      TOT = 0;
      SOD = "";
      EOD = "";
      MAX = 1;

      sim_trace = 1'b0;
      sim_heart = 1'b0;
      sim_range = 1'b1;

      global_term = 1'bZ;

      failset = 1'b0;
      FAILSETID = 0;

      name_POs[0001] = "60";      // pinName = 60;  tf =  SO   BDY  ;
      name_POs[0002] = "61";      // pinName = 61;  tf =  BDY  ;
      name_POs[0003] = "70";      // pinName = 70;  tf =  SO   BDY  ;

      if ( $test$plusargs ( "DEBUG" ) )  sim_trace = 1'b1;


      if ( $test$plusargs ( "HEARTBEAT" ) )  sim_heart = 1'b1;

      if ( $value$plusargs ( "START_RANGE=%s", SOD ) )  sim_range = 1'b0;

      if ( $value$plusargs ( "END_RANGE=%s", EOD ) );

      if ( $test$plusargs ( "FAILSET" ) )  failset = 1'b1;

    end
  endtask

//*************************************************************************//
//                           FAILSET SETUP PROCEDURE                       //
//*************************************************************************//
```

```
  task failset_setup;
    begin

       FAILSETID = $fopen ( "VER.IDDQ.failset", "w" );
       if ( ! FAILSETID )
         $display ( "\n951 ERROR (TVE-951): Failed to open the file: VER.IDDQ.f

    end
  endtask

//*************************************************************************//
//                  READ COMMANDS AND DATA AND RUN SIMULATION              //
//*************************************************************************//

  task sim_vector_file;
    begin

       ERR = 0;
       LIX = 0;

       stim_CIs = 0025'b0XX0100000X00XXXXXXX00XXX;
       stim_SLs[0001:2070] = 2070'b0;
       resp_MLs[0001:2070] = 2070'bX;
       resp_POs = 0003'bX;

       $display ( "\n200 INFO (TVE-200): Reading vector file: %0s   [end TVE_20

       CNT = $fscanf ( FID, "%d", CMD );
       while ( CNT > 0 ) begin

         if ( sim_trace )  $display ( "\nCommand code:  %d ", CMD );

         case ( CMD )

           100: begin
             CNT = $fgets ( COMMENT, FID );
           end

           200: begin
             CNT = $fscanf ( FID, "%b", stim_PIs[1:25] );
           end

           201: begin
             CNT = $fscanf ( FID, "%b", stim_CIs[1:25] );
           end

           202: begin
             CNT = $fscanf ( FID, "%b", resp_POs[1:3] );
           end

           203: begin
             CNT = $fscanf ( FID, "%b", global_term );
           end

           204: begin
             CNT = $fscanf ( FID, "%b", stim_SSs[1:2] );
           end

           300: begin
             CNT = $fscanf ( FID, "%d", MODENUM );
             case ( MODENUM )
```

```
    3: begin
      CNT = $fscanf ( FID, "%b", stim_SLs[1034:1035] );
      CNT = $fscanf ( FID, "%b", stim_SLs[1036:2035] );
      CNT = $fscanf ( FID, "%b", stim_SLs[2036:2070] );
    end

  endcase
end

301: begin
  CNT = $fscanf ( FID, "%d", MODENUM );
  case ( MODENUM )

    3: begin
      CNT = $fscanf ( FID, "%b", resp_MLs[1:2] );
      CNT = $fscanf ( FID, "%b", resp_MLs[1036:2035] );
      CNT = $fscanf ( FID, "%b", resp_MLs[2036:2070] );
    end

  endcase
end

400: begin
  if ( sim_range )  test_cycle;
end

401: begin
  if ( sim_range )  test_cycle_flush;
end

403: begin
  if ( sim_range )  scan_cycle;
end

500: begin
  LIX = LIX + 1;
  CNT = $fscanf ( FID, "%d", sim_count[LIX] );
  if ( sim_count[LIX] )  sim_start[LIX] = $ftell ( FID );
end

501: begin
  sim_count[LIX] = sim_count[LIX] - 1;
  if ( sim_count[LIX] )  CNT = $fseek ( FID, sim_start[LIX], 0 );
  else  LIX = LIX - 1;
end

600: begin
  CNT = $fscanf ( FID, "%d", MODENUM );
  case ( MODENUM )

    3: begin
      CNT = $fscanf ( FID, "%d", SEQNUM );
      case ( SEQNUM )

        1: begin
          CNT = $fscanf ( FID, "%d", MAX );
          if ( sim_range )  Scan_Preconditioning_Sequence_IDDQ;
        end

        2: begin
```

```
                    CNT = $fscanf ( FID, "%d", MAX );
                    if ( sim_range )  Scan_Preconditioning_Sequencebsr_IDDQ;
                  end

                  3: begin
                    CNT = $fscanf ( FID, "%d", MAX );
                    if ( sim_range )  Skewed_Unload_Sequence_IDDQ;
                  end

                  4: begin
                    CNT = $fscanf ( FID, "%d", MAX );
                    if ( sim_range )  Scan_Sequence_IDDQ;
                  end

                  5: begin
                    CNT = $fscanf ( FID, "%d", MAX );
                    if ( sim_range )  Skewed_Load_Sequence_IDDQ;
                  end

                endcase
              end

            endcase
          end

          900, 901: begin
            CNT = $fscanf ( FID, "%s", pattern );
            if ( CMD == 901 )  PATTERN = pattern;
            if ( SOD == pattern )  begin
              sim_range = 1'b1;
            end
            if (( CMD == 900 ) & sim_range & sim_heart )  $display ( "\n202 IN
          end

          default: begin
            $display ( "\n999 ERROR (TVE-999): Internal Program Error occurred
            $display ( "           Unrecognized command code = %0d \n", CMD );
          end

        endcase

        if ( EOD == pattern )  begin
          sim_range = 1'b0;
        end
        CNT = $fscanf ( FID, "%d", CMD );

      end

      $display ( "\n201 INFO (TVE-201): Simulation complete on vector file: %0
      $fclose ( FID );

      TOT = TOT + ERR;

    end
  endtask

//*************************************************************************//
//                        DEFINE TEST FLUSH PROCEDURE                      //
//*************************************************************************//

  task test_cycle_flush;
```

```
begin

   if ( sim_trace )  $display ( "\nRunning task:  test_cycle_flush " );

  #0.000000;        // 0.000000 ns;  From the start of the cycle.
   part_PIs[0002] = stim_PIs[0002]; //      pinName = A4;  tf = -SE  ; test
   part_PIs[0003] = stim_PIs[0003]; //      pinName = A5;  tf = +SE  ; test
   part_PIs[0011] = stim_PIs[0011]; //      pinName = B6;  tf = +SE  ; test
   part_PIs[0014] = stim_PIs[0014]; //      pinName = E1;  tf = +SE  -BI  ;
   part_PIs[0015] = stim_PIs[0015]; //      pinName = F3;  tf =  BDY  ; tes
   part_PIs[0016] = stim_PIs[0016]; //      pinName = F4;  tf =  SI   BDY
   part_PIs[0017] = stim_PIs[0017]; //      pinName = I0;  tf =  SI   BDY
   part_PIs[0018] = stim_PIs[0018]; //      pinName = I1;  tf =  BDY  ; tes
   part_PIs[0019] = stim_PIs[0019]; //      pinName = L0;  tf = +SE  ; test
   part_PIs[0020] = stim_PIs[0020]; //      pinName = L1;  tf = +SE  ; test
   part_PIs[0023] = stim_PIs[0023]; //      pinName = S0;  tf = -SE  ; test
   part_PIs[0024] = stim_PIs[0024]; //      pinName = S1;  tf = +SE  ; test
   part_PIs[0025] = stim_PIs[0025]; //      pinName = T0;  tf =  BDY  ; tes
  #8.000000;        // 8.000000 ns;  From the start of the cycle.
   part_PIs[0001] = stim_PIs[0001]; //      pinName = A2;  tf = -SC  ; test
   part_PIs[0004] = stim_PIs[0004]; //      pinName = A6;  tf = -SC  ; test
   part_PIs[0005] = stim_PIs[0005]; //      pinName = A7;  tf = +SC  ; test
   part_PIs[0006] = stim_PIs[0006]; //      pinName = A8;  tf = -SC  ; test
   part_PIs[0007] = stim_PIs[0007]; //      pinName = B0;  tf = -AC  ; test
   part_PIs[0009] = stim_PIs[0009]; //      pinName = B3;  tf = -SC  ; test
   part_PIs[0010] = stim_PIs[0010]; //      pinName = B4;  tf = -SC  ; test
   part_PIs[0012] = stim_PIs[0012]; //      pinName = B7;  tf = -SC  ; test
   part_PIs[0021] = stim_PIs[0021]; //      pinName = L2;  tf = -AC  ; test
  #8.000000;        // 16.000000 ns;  From the start of the cycle.
   part_PIs[0001] = stim_CIs[0001]; //      pinName = A2;  tf = -SC  ; test
   part_PIs[0004] = stim_CIs[0004]; //      pinName = A6;  tf = -SC  ; test
   part_PIs[0005] = stim_CIs[0005]; //      pinName = A7;  tf = +SC  ; test
   part_PIs[0006] = stim_CIs[0006]; //      pinName = A8;  tf = -SC  ; test
   part_PIs[0007] = stim_CIs[0007]; //      pinName = B0;  tf = -AC  ; test
   part_PIs[0009] = stim_CIs[0009]; //      pinName = B3;  tf = -SC  ; test
   part_PIs[0010] = stim_CIs[0010]; //      pinName = B4;  tf = -SC  ; test
   part_PIs[0012] = stim_CIs[0012]; //      pinName = B7;  tf = -SC  ; test
   part_PIs[0021] = stim_CIs[0021]; //      pinName = L2;  tf = -AC  ; test
  #8.000000;        // 24.000000 ns;  From the start of the cycle.
   part_PIs[0008] = stim_PIs[0008]; //      pinName = B1;  tf = -BC  ; test
   part_PIs[0013] = stim_PIs[0013]; //      pinName = B8;  tf = -PC  ; test
   part_PIs[0022] = stim_PIs[0022]; //      pinName = L3;  tf = -BC  ; test
  #8.000000;        // 32.000000 ns;  From the start of the cycle.
   part_PIs[0008] = stim_CIs[0008]; //      pinName = B1;  tf = -BC  ; test
   part_PIs[0013] = stim_CIs[0013]; //      pinName = B8;  tf = -PC  ; test
   part_PIs[0022] = stim_CIs[0022]; //      pinName = L3;  tf = -BC  ; test
  #74488.000000;        // 74520.000000 ns;  From the start of the cycle.
   for ( POnum = 1; POnum <= 3; POnum = POnum + 1 ) begin
     if ((part_POs[POnum] !== resp_POs[POnum]) & (resp_POs[POnum] !== 1'bX)
       ERR = ERR + 1;
       $display ( "\n650 WARNING (TVE-650): PO miscompare at pattern: %0s a
       $display ( "           Expected: %b    Simulated: %b    On Output: %

       if (( failset ) & ( FAILSETID == 0 ))  failset_setup;
       if ( FAILSETID ) begin
         $fdisplay ( FAILSETID, " Chip %0s pad %0s pattern %0s position %0d
       end
     end
   end
  #8280.000000;        // 82800.000000 ns;  From the start of the cycle.
   resp_POs = 0003'bX;
```

```
    end
  endtask

//*************************************************************************//
//                          DEFINE TEST PROCEDURE                          //
//*************************************************************************//

  task test_cycle;
    begin

      if ( sim_trace )  $display ( "\nRunning task:  test_cycle " );

      #0.000000;        // 0.000000 ns;  From the start of the cycle.
       part_PIs[0002] = stim_PIs[0002]; //      pinName = A4;  tf = -SE  ; test
       part_PIs[0003] = stim_PIs[0003]; //      pinName = A5;  tf = +SE  ; test
       part_PIs[0011] = stim_PIs[0011]; //      pinName = B6;  tf = +SE  ; test
       part_PIs[0014] = stim_PIs[0014]; //      pinName = E1;  tf = +SE  -BI  ;
       part_PIs[0015] = stim_PIs[0015]; //      pinName = F3;  tf =  BDY  ; tes
       part_PIs[0016] = stim_PIs[0016]; //      pinName = F4;  tf =  SI   BDY
       part_PIs[0017] = stim_PIs[0017]; //      pinName = I0;  tf =  SI   BDY
       part_PIs[0018] = stim_PIs[0018]; //      pinName = I1;  tf =  BDY  ; tes
       part_PIs[0019] = stim_PIs[0019]; //      pinName = L0;  tf = +SE  ; test
       part_PIs[0020] = stim_PIs[0020]; //      pinName = L1;  tf = +SE  ; test
       part_PIs[0023] = stim_PIs[0023]; //      pinName = S0;  tf = -SE  ; test
       part_PIs[0024] = stim_PIs[0024]; //      pinName = S1;  tf = +SE  ; test
       part_PIs[0025] = stim_PIs[0025]; //      pinName = T0;  tf =  BDY  ; tes
      #8.000000;        // 8.000000 ns;  From the start of the cycle.
       part_PIs[0001] = stim_PIs[0001]; //      pinName = A2;  tf = -SC  ; test
       part_PIs[0004] = stim_PIs[0004]; //      pinName = A6;  tf = -SC  ; test
       part_PIs[0005] = stim_PIs[0005]; //      pinName = A7;  tf = +SC  ; test
       part_PIs[0006] = stim_PIs[0006]; //      pinName = A8;  tf = -SC  ; test
       part_PIs[0007] = stim_PIs[0007]; //      pinName = B0;  tf = -AC  ; test
       part_PIs[0009] = stim_PIs[0009]; //      pinName = B3;  tf = -SC  ; test
       part_PIs[0010] = stim_PIs[0010]; //      pinName = B4;  tf = -SC  ; test
       part_PIs[0012] = stim_PIs[0012]; //      pinName = B7;  tf = -SC  ; test
       part_PIs[0021] = stim_PIs[0021]; //      pinName = L2;  tf = -AC  ; test
      #8.000000;        // 16.000000 ns;  From the start of the cycle.
       part_PIs[0001] = stim_CIs[0001]; //      pinName = A2;  tf = -SC  ; test
       part_PIs[0004] = stim_CIs[0004]; //      pinName = A6;  tf = -SC  ; test
       part_PIs[0005] = stim_CIs[0005]; //      pinName = A7;  tf = +SC  ; test
       part_PIs[0006] = stim_CIs[0006]; //      pinName = A8;  tf = -SC  ; test
       part_PIs[0007] = stim_CIs[0007]; //      pinName = B0;  tf = -AC  ; test
       part_PIs[0009] = stim_CIs[0009]; //      pinName = B3;  tf = -SC  ; test
       part_PIs[0010] = stim_CIs[0010]; //      pinName = B4;  tf = -SC  ; test
       part_PIs[0012] = stim_CIs[0012]; //      pinName = B7;  tf = -SC  ; test
       part_PIs[0021] = stim_CIs[0021]; //      pinName = L2;  tf = -AC  ; test
      #8.000000;        // 24.000000 ns;  From the start of the cycle.
       part_PIs[0008] = stim_PIs[0008]; //      pinName = B1;  tf = -BC  ; test
       part_PIs[0013] = stim_PIs[0013]; //      pinName = B8;  tf = -PC  ; test
       part_PIs[0022] = stim_PIs[0022]; //      pinName = L3;  tf = -BC  ; test
      #8.000000;        // 32.000000 ns;  From the start of the cycle.
       part_PIs[0008] = stim_CIs[0008]; //      pinName = B1;  tf = -BC  ; test
       part_PIs[0013] = stim_CIs[0013]; //      pinName = B8;  tf = -PC  ; test
       part_PIs[0022] = stim_CIs[0022]; //      pinName = L3;  tf = -BC  ; test
      #40.000000;        // 72.000000 ns;  From the start of the cycle.
       for ( POnum = 1; POnum <= 3; POnum = POnum + 1 ) begin
         if ((part_POs[POnum] !== resp_POs[POnum]) & (resp_POs[POnum] !== 1'bX)
           ERR = ERR + 1;
           $display ( "\n650 WARNING (TVE-650): PO miscompare at pattern: %0s a
           $display ( "           Expected: %b    Simulated: %b    On Output: %
```

```
          if (( failset ) & ( FAILSETID == 0 ))  failset_setup;
          if ( FAILSETID ) begin
            $fdisplay ( FAILSETID, " Chip %0s pad %0s pattern %0s position %0d
          end
        end
      end
      #8.000000;        // 80.000000 ns;  From the start of the cycle.
       resp_POs = 0003'bX;

    end
  endtask

//*************************************************************************//
//                           DEFINE SCAN PROCEDURE                         //
//*************************************************************************//

  task scan_cycle;
    begin

       if ( sim_trace )  $display ( "\nRunning task:  scan_cycle " );

      #0.000000;        // 0.000000 ns;  From the start of the cycle.
       for ( POnum = 1; POnum <= 3; POnum = POnum + 1 ) begin
         if ((part_POs[POnum] !== resp_POs[POnum]) & (resp_POs[POnum] !== 1'bX)
           ERR = ERR + 1;
           $display ( "\n650 WARNING (TVE-650): PO miscompare at pattern: %0s a
           $display ( "           Expected: %b    Simulated: %b    On Output: %

           if (( failset ) & ( FAILSETID == 0 ))  failset_setup;
           if ( FAILSETID ) begin
             $fdisplay ( FAILSETID, " Chip %0s pad %0s pattern %0s position %0d
           end
         end
       end
      #16.000000;        // 16.000000 ns;  From the start of the cycle.
       part_PIs[0001] = stim_PIs[0001]; //      pinName = A2;  tf = -SC  ; test
       part_PIs[0002] = stim_PIs[0002]; //      pinName = A4;  tf = -SE  ; test
       part_PIs[0003] = stim_PIs[0003]; //      pinName = A5;  tf = +SE  ; test
       part_PIs[0004] = stim_PIs[0004]; //      pinName = A6;  tf = -SC  ; test
       part_PIs[0005] = stim_PIs[0005]; //      pinName = A7;  tf = +SC  ; test
       part_PIs[0006] = stim_PIs[0006]; //      pinName = A8;  tf = -SC  ; test
       part_PIs[0009] = stim_PIs[0009]; //      pinName = B3;  tf = -SC  ; test
       part_PIs[0010] = stim_PIs[0010]; //      pinName = B4;  tf = -SC  ; test
       part_PIs[0011] = stim_PIs[0011]; //      pinName = B6;  tf = +SE  ; test
       part_PIs[0012] = stim_PIs[0012]; //      pinName = B7;  tf = -SC  ; test
       part_PIs[0014] = stim_PIs[0014]; //      pinName = E1;  tf = +SE  -BI  ;
       part_PIs[0015] = stim_PIs[0015]; //      pinName = F3;  tf =  BDY  ; tes
       part_PIs[0016] = stim_PIs[0016]; //      pinName = F4;  tf =  SI   BDY
       part_PIs[0017] = stim_PIs[0017]; //      pinName = I0;  tf =  SI   BDY
       part_PIs[0018] = stim_PIs[0018]; //      pinName = I1;  tf =  BDY  ; tes
       part_PIs[0019] = stim_PIs[0019]; //      pinName = L0;  tf = +SE  ; test
       part_PIs[0020] = stim_PIs[0020]; //      pinName = L1;  tf = +SE  ; test
       part_PIs[0023] = stim_PIs[0023]; //      pinName = S0;  tf = -SE  ; test
       part_PIs[0024] = stim_PIs[0024]; //      pinName = S1;  tf = +SE  ; test
       part_PIs[0025] = stim_PIs[0025]; //      pinName = T0;  tf =  BDY  ; tes
      #8.000000;        // 24.000000 ns;  From the start of the cycle.
       part_PIs[0001] = stim_CIs[0001]; //      pinName = A2;  tf = -SC  ; test
       part_PIs[0004] = stim_CIs[0004]; //      pinName = A6;  tf = -SC  ; test
       part_PIs[0005] = stim_CIs[0005]; //      pinName = A7;  tf = +SC  ; test
       part_PIs[0006] = stim_CIs[0006]; //      pinName = A8;  tf = -SC  ; test
```

```
      part_PIs[0009] = stim_CIs[0009]; //      pinName = B3;  tf = -SC  ; test
      part_PIs[0010] = stim_CIs[0010]; //      pinName = B4;  tf = -SC  ; test
      part_PIs[0012] = stim_CIs[0012]; //      pinName = B7;  tf = -SC  ; test
      part_PIs[0007] = stim_PIs[0007]; //      pinName = B0;  tf = -AC  ; test
      part_PIs[0021] = stim_PIs[0021]; //      pinName = L2;  tf = -AC  ; test
     #8.000000;        // 32.000000 ns;  From the start of the cycle.
      part_PIs[0007] = stim_CIs[0007]; //      pinName = B0;  tf = -AC  ; test
      part_PIs[0021] = stim_CIs[0021]; //      pinName = L2;  tf = -AC  ; test
     #8.000000;        // 40.000000 ns;  From the start of the cycle.
      part_PIs[0008] = stim_PIs[0008]; //      pinName = B1;  tf = -BC  ; test
      part_PIs[0013] = stim_PIs[0013]; //      pinName = B8;  tf = -PC  ; test
      part_PIs[0022] = stim_PIs[0022]; //      pinName = L3;  tf = -BC  ; test
     #8.000000;        // 48.000000 ns;  From the start of the cycle.
      part_PIs[0008] = stim_CIs[0008]; //      pinName = B1;  tf = -BC  ; test
      part_PIs[0013] = stim_CIs[0013]; //      pinName = B8;  tf = -PC  ; test
      part_PIs[0022] = stim_CIs[0022]; //      pinName = L3;  tf = -BC  ; test
     #32.000000;        // 80.000000 ns;  From the start of the cycle.
      resp_POs = 0003'bX;

    end
  endtask

//*************************************************************************//
//                       DEFINE SCAN PRECOND PROCEDURE                     //
//*************************************************************************//

  task Scan_Preconditioning_Sequence_IDDQ;
    begin

      if ( sim_trace )  $display ( "\nRunning task:  Scan_Preconditioning_Sequ

      stim_PIs[0002] = 1'b0; //      pinName = A4;  tf = -SE  ; testOffset = 0
      stim_PIs[0003] = 1'b1; //      pinName = A5;  tf = +SE  ; testOffset = 0
      stim_PIs[0011] = 1'b1; //      pinName = B6;  tf = +SE  ; testOffset = 0
      stim_PIs[0014] = 1'b1; //      pinName = E1;  tf = +SE  -BI  ; testOffse
      stim_PIs[0019] = 1'b1; //      pinName = L0;  tf = +SE  ; testOffset = 0
      stim_PIs[0020] = 1'b1; //      pinName = L1;  tf = +SE  ; testOffset = 0
      stim_PIs[0023] = 1'b0; //      pinName = S0;  tf = -SE  ; testOffset = 0
      stim_PIs[0024] = 1'b1; //      pinName = S1;  tf = +SE  ; testOffset = 0

      test_cycle;

    end
  endtask

//*************************************************************************//
//                     DEFINE SCAN PRECOND BSR PROCEDURE                   //
//*************************************************************************//

  task Scan_Preconditioning_Sequencebsr_IDDQ;
    begin

      if ( sim_trace )  $display ( "\nRunning task:  Scan_Preconditioning_Sequ

    end
  endtask

//*************************************************************************//
//                       DEFINE SKEWED UNLOAD PROCEDURE                    //
//*************************************************************************//
```

```
  task Skewed_Unload_Sequence_IDDQ;
    begin

       if ( sim_trace )  $display ( "\nRunning task:  Skewed_Unload_Sequence_ID

       stim_PIs[0008] = 1'b1; //      pinName = B1;  tf = -BC  ; testOffset = 2
       stim_PIs[0022] = 1'b1; //      pinName = L3;  tf = -BC  ; testOffset = 2

       scan_cycle;

       stim_PIs[0008] = 1'b0; //      pinName = B1;  tf = -BC  ; testOffset = 2
       stim_PIs[0022] = 1'b0; //      pinName = L3;  tf = -BC  ; testOffset = 2

    end
  endtask

//*************************************************************************//
//                      DEFINE SCAN SEQUENCE PROCEDURE                     //
//*************************************************************************//

  task Scan_Sequence_IDDQ;
    begin

       if ( sim_trace )  $display ( "\nRunning task:  Scan_Sequence_IDDQ " );

       for ( CYCLE = 1; CYCLE <= MAX; CYCLE = CYCLE + 1 ) begin
      #0.000000;        // 0.000000 ns;  From the start of the cycle.
         if ((part_POs[1] !== resp_MLs[0+CYCLE]) & (resp_MLs[0+CYCLE] !== 1'bX)
           ERR = ERR + 1;
           $display ( "\n660 WARNING (TVE-660): SO miscompare at pattern: %0s a
           $display ( "           Expected: %0b    Simulated: %0b   On Output:

           if (( failset ) & ( FAILSETID == 0 ))  failset_setup;
           if ( FAILSETID ) begin
             $fdisplay ( FAILSETID, " Chip %0s pad %0s pattern %0s position %0d
           end
         end
         if ((part_POs[3] !== resp_MLs[1035+CYCLE]) & (resp_MLs[1035+CYCLE] !==
           ERR = ERR + 1;
           $display ( "\n660 WARNING (TVE-660): SO miscompare at pattern: %0s a
           $display ( "           Expected: %0b    Simulated: %0b   On Output:

           if (( failset ) & ( FAILSETID == 0 ))  failset_setup;
           if ( FAILSETID ) begin
             $fdisplay ( FAILSETID, " Chip %0s pad %0s pattern %0s position %0d
           end
         end
      #16.000000;        // 16.000000 ns;  From the start of the cycle.
       part_PIs[0016] = stim_SLs[0000+CYCLE]; //      pinName = F4;  tf =  SI
       part_PIs[0017] = stim_SLs[1035+CYCLE]; //      pinName = I0;  tf =  SI
      #8.000000;        // 24.000000 ns;  From the start of the cycle.
       part_PIs[0007] = 1'b1; //      pinName = B0;  tf = -AC  ; testOffset = 8
       part_PIs[0021] = 1'b1; //      pinName = L2;  tf = -AC  ; testOffset = 8
      #8.000000;        // 32.000000 ns;  From the start of the cycle.
       part_PIs[0007] = 1'b0; //      pinName = B0;  tf = -AC  ; testOffset = 8
       part_PIs[0021] = 1'b0; //      pinName = L2;  tf = -AC  ; testOffset = 8
      #8.000000;        // 40.000000 ns;  From the start of the cycle.
       part_PIs[0008] = 1'b1; //      pinName = B1;  tf = -BC  ; testOffset = 2
       part_PIs[0022] = 1'b1; //      pinName = L3;  tf = -BC  ; testOffset = 2
      #8.000000;        // 48.000000 ns;  From the start of the cycle.
       part_PIs[0008] = 1'b0; //      pinName = B1;  tf = -BC  ; testOffset = 2
```

```
      part_PIs[0022] = 1'b0; //      pinName = L3;  tf = -BC  ; testOffset = 2
     #32.000000;        // 80.000000 ns;  From the start of the cycle.
      end
      resp_MLs[0001:1035] = 1035'bX;
      resp_MLs[1036:2070] = 1035'bX;
      resp_POs = 0003'bX;
      stim_SLs[0001:1035] = 1035'b0;
      stim_SLs[1036:2070] = 1035'b0;
      stim_PIs = part_PIs;
      CYCLE = 0;

    end
  endtask

//*************************************************************************//
//                       DEFINE SKEWED LOAD PROCEDURE                      //
//*************************************************************************//

  task Skewed_Load_Sequence_IDDQ;
    begin

      if ( sim_trace )  $display ( "\nRunning task:  Skewed_Load_Sequence_IDDQ

      stim_PIs[0016] = stim_SSs[0001]; //      pinName = F4;  tf =  SI   BDY
      stim_PIs[0017] = stim_SSs[0002]; //      pinName = I0;  tf =  SI   BDY

      stim_PIs[0007] = 1'b1; //      pinName = B0;  tf = -AC  ; testOffset = 8
      stim_PIs[0021] = 1'b1; //      pinName = L2;  tf = -AC  ; testOffset = 8

      scan_cycle;

      stim_PIs[0007] = 1'b0; //      pinName = B0;  tf = -AC  ; testOffset = 8
      stim_PIs[0021] = 1'b0; //      pinName = L2;  tf = -AC  ; testOffset = 8

    end
  endtask

  endmodule
```

Timed Dynamic Verilog Pattern Data Examples

This section provides an example of Verilog format which is typical for all timed dynamic tests.

Verilog Timed Dynamic Pattern Main Simulation File

The following is an example of a truncated main simulation file for timed tests using a serial scan format. This file provides the structural connect and task definitions that are common to each Verilog vector file.

```
//*************************************************************************//
//                           VERILOG MAINSIM FILE                          //
// Encounter(TM) Test and Diagnostics 3.1.Dev Jan 04, 2006 (linux24_64 ET31) //
//*************************************************************************//
//                                                                         //
```

```
//  FILE CREATED.............January 04, 2006 at 15:44:04                  //
//                                                                          //
//  PROJECT NAME.............mbi                                           //
//                                                                          //
//  TESTMODE.................SP_WAFER_AC                                  //
//                                                                          //
//  TDR......................TBAdvantdynamic                              //
//                                                                          //
//  TEST PERIOD..............80.000   TEST TIME UNITS...........ns         //
//  TEST PULSE WIDTH.........8.000                                         //
//  TEST STROBE OFFSET.......72.000   TEST STROBE TYPE..........edge       //
//  TEST BIDI OFFSET.........0.000                                         //
//  TEST PI OFFSET...........0.000    X VALUE..................X          //
//                                                                          //
//  TEST PI OFFSET for pin "A01A0" (PI # 21) is .................24.000     //
//  TEST PI OFFSET for pin "C0160" (PI # 190) is ................8.000     //
//  TEST PI OFFSET for pin "C0163" (PI # 193) is ................8.000     //
//  TEST PI OFFSET for pin "C0166" (PI # 196) is ................8.000     //
//  TEST PI OFFSET for pin "C0167" (PI # 197) is ................8.000     //
//                                                                          //
//  SCAN FORMAT..............serial   SCAN OVERLAP.............yes        //
//  SCAN PERIOD..............80.000   SCAN TIME UNITS...........ns         //
//  SCAN PULSE WIDTH.........8.000                                         //
//  SCAN STROBE OFFSET.......0.000    SCAN STROBE TYPE..........edge       //
//  SCAN BIDI OFFSET.........16.000                                        //
//  SCAN PI OFFSET...........16.000   X VALUE..................X          //
//                                                                          //
//  SCAN PI OFFSET for pin "A01A0" (PI # 21) is .................40.000     //
//  SCAN PI OFFSET for pin "C0167" (PI # 197) is ................24.000     //
//                                                                          //
//**************************************************************************//

 `timescale 1 ns / 1 ps

  module mbi_SP_WAFER_AC;

//**************************************************************************//
//                DEFINE VARIABLES FOR ALL PRIMARY I/O PORTS                //
//**************************************************************************//

  reg      [1:0200] stim_PIs;

  reg      [1:0200] part_PIs;

  reg      [1:0200] stim_CIs;

  reg      [1:0271] resp_POs;

  wire     [1:0271] part_POs;


//**************************************************************************//
//                    DEFINE VARIABLES FOR ALL SHIFT CHAINS                 //
//**************************************************************************//

  reg      [1:2556] stim_SLs;

  reg      [1:0002] stim_SSs;

  reg      [1:2556] resp_MLs;
```

```
//*************************************************************************//
//                            OTHER DEFINITIONS                            //
//*************************************************************************//

  integer  CYCLE, PInum, POnum, ORnum, MODENUM, EXPNUM, SCANOPNUM, SEQNUM, TASK
  integer  ERR, CMD, FID, TID, CNT, TOT, CID, LIX, MAX, FAILSETID;
  integer  sim_start [1:15], sim_count [1:15];
  reg      [1:8185] name_POs [1:0271];
  reg      sim_trace, sim_heart, sim_range, failset, global_term;
  reg      [1:800] PATTERN, pattern, TESTFILE, SOD, EOD;
  reg      [1:8184] FILE, COMMENT;
  reg      no_Pin_Found;

//*************************************************************************//
//        INSTANTIATE THE STRUCTURE AND CONNECT TO VERILOG VARIABLES       //
//*************************************************************************//

  IE06303
    IE06303_inst (
       .A0100        (part_POs[0001]),      // pinName = A0100;
       .A0101        (part_POs[0002]),      // pinName = A0101;
       .A0102        (part_POs[0003]),      // pinName = A0102;
                                      .
                                      .
                                      .
       .A01A5        (part_PIs[0026]), //      pinName = A01A5;  tf = -BI  -OI
       .A01A6        (part_PIs[0027]), //      pinName = A01A6;  tf = +SE  -BI
       .A01A7        (part_PIs[0028]), //      pinName = A01A7;  tf = -TI  ; te
       .A01A8        (part_PIs[0029])); //      pinName = A01A8;  tf = -SE  ; t

//*************************************************************************//
//                        MAKE SOME OTHER CONNECTIONS                      //
//*************************************************************************//

  assign // BiDi Connections
       part_POs[0081] = part_PIs[0001],//      pinName = A0180;  tf = BIDI ; te
       part_POs[0082] = part_PIs[0002],//      pinName = A0181;  tf = BIDI ; te
       part_POs[0083] = part_PIs[0003],//      pinName = A0182;  tf = BIDI ; te
       part_POs[0084] = part_PIs[0004],//      pinName = A0183;  tf = BIDI ; te
                                         .
                                         .
                                         .
       part_POs[0268] = part_PIs[0197],//      pinName = C0167;  tf = -AC   BDY
       part_POs[0269] = part_PIs[0198],//      pinName = C0168;  tf = +CI   BDY
       part_POs[0270] = part_PIs[0199],//      pinName = C0169;  tf = BIDI ; te
       part_POs[0271] = part_PIs[0200]; //      pinName = C0170;  tf =  SI   BD

  assign ( weak0, weak1 ) // Termination
       part_POs[0001] = global_term,     // pinName = A0100;
       part_POs[0002] = global_term,     // pinName = A0101;
       part_POs[0003] = global_term,     // pinName = A0102;
       part_POs[0004] = global_term,     // pinName = A0103;
       part_POs[0005] = global_term,     // pinName = A0104;
                                    .
                                    .
                                    .
       part_POs[0269] = global_term,     // pinName = C0168;  tf =  BDY  +CI  B
       part_POs[0270] = global_term,     // pinName = C0169;  tf = BIDI ; testO
       part_POs[0271] = global_term;      // pinName = C0170;  tf =  BDY   SI
```

```
//*************************************************************************//
//                      OPEN THE FILE AND RUN SIMULATION                   //
//*************************************************************************//

  initial
    begin

       FILE = 0;
       sim_setup;

       for ( TID = 1; TID <= 99; TID = TID + 1 ) begin
         $sformat ( TESTFILE, "TESTFILE%0d=", TID );
         if ( $value$plusargs ( TESTFILE, FILE )) begin
           FID = $fopen ( FILE, "r" );
           if ( FID )  sim_vector_file;
           else $display ( "\n951 ERROR (TVE-951): Failed to open the file: %0s
         end
       end

       if ( FAILSETID )  $fclose ( FAILSETID );
       if ( FILE )
         $display ( "\n203 INFO (TVE-203): The total number of miscomparing vec
       else
         $display ( "\n661 WARNING (TVE-661): No input data files found. The da
       $finish;

    end

//*************************************************************************//
//                      DEFINE SIMULATION SETUP PROCEDURE                  //
//*************************************************************************//

  task sim_setup;
    begin

       TOT = 0;
       SOD = "";
       EOD = "";
       MAX = 1;

       sim_trace = 1'b0;
       sim_heart = 1'b0;
       sim_range = 1'b1;

       global_term = 1'bZ;

       failset = 1'b0;
       FAILSETID = 0;

       name_POs[0001] = "A0100";      // pinName = A0100;
       name_POs[0002] = "A0101";      // pinName = A0101;
       name_POs[0003] = "A0102";      // pinName = A0102;
                                .
                                .
                                .
       name_POs[0268] = "C0167";      // pinName = C0167;  tf =  BDY  -AC  BIDI
       name_POs[0269] = "C0168";      // pinName = C0168;  tf =  BDY  +CI  BIDI
       name_POs[0270] = "C0169";      // pinName = C0169;  tf = BIDI ; testOffs
       name_POs[0271] = "C0170";      // pinName = C0170;  tf =  BDY   SI   BDY

       if ( $test$plusargs ( "DEBUG" ) )  sim_trace = 1'b1;
```

```
      if ( $test$plusargs ( "HEARTBEAT" ) )  sim_heart = 1'b1;

      if ( $value$plusargs ( "START_RANGE=%s", SOD ) )  sim_range = 1'b0;

      if ( $value$plusargs ( "END_RANGE=%s", EOD ) );

      if ( $test$plusargs ( "FAILSET" ) )  failset = 1'b1;

    end
  endtask

//*************************************************************************//
//                          FAILSET SETUP PROCEDURE                        //
//*************************************************************************//

  task failset_setup;
    begin

      FAILSETID = $fopen ( "VER.SP_WAFER_AC.failset", "w" );
      if ( ! FAILSETID )
        $display ( "\n951 ERROR (TVE-951): Failed to open the file: VER.SP_WAF

    end
  endtask

//*************************************************************************//
//                 READ COMMANDS AND DATA AND RUN SIMULATION               //
//*************************************************************************//

  task sim_vector_file;
    begin

      ERR = 0;
      LIX = 0;

      stim_CIs = 0200'bXXXXXXXXXXXXXXXXXXXX0XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
      stim_SLs[0001:2556] = 2556'b0;
      resp_MLs[0001:2556] = 2556'bX;
      resp_POs = 0271'bX;

      $display ( "\n200 INFO (TVE-200): Reading vector file: %0s   [end TVE_20

      CNT = $fscanf ( FID, "%d", CMD );
      while ( CNT > 0 ) begin

        if ( sim_trace )  $display ( "\nCommand code:  %d ", CMD );

        case ( CMD )

          100: begin
            CNT = $fgets ( COMMENT, FID );
          end

          200: begin
            CNT = $fscanf ( FID, "%b", stim_PIs[1:200] );
          end

          201: begin
            CNT = $fscanf ( FID, "%b", stim_CIs[1:200] );
          end
```

```
                                .
                                .
                                .
          900, 901: begin
            CNT = $fscanf ( FID, "%s", pattern );
            if ( CMD == 901 )  PATTERN = pattern;
            if ( SOD == pattern )  begin
              sim_range = 1'b1;
            end
            if (( CMD == 900 ) & sim_range & sim_heart )  $display ( "\n202 IN
          end

          default: begin
            $display ( "\n999 ERROR (TVE-999): Internal Program Error occurred
            $display ( "           Unrecognized command code = %0d \n", CMD );
          end

        endcase

        if ( EOD == pattern )  begin
          sim_range = 1'b0;
        end
        CNT = $fscanf ( FID, "%d", CMD );

      end

      $display ( "\n201 INFO (TVE-201): Simulation complete on vector file: %0
      $fclose ( FID );

      TOT = TOT + ERR;

    end
  endtask

//*************************************************************************//
//                        DEFINE TEST FLUSH PROCEDURE                      //
//*************************************************************************//

  task test_cycle_flush;
    begin

      if ( sim_trace )  $display ( "\nRunning task:  test_cycle_flush " );

      #0.000000;        // 0.000000 ns;  From the start of the cycle.
       part_PIs[0022] = stim_PIs[0022]; //      pinName = A01A1; testOffset = 0
       part_PIs[0023] = stim_PIs[0023]; //      pinName = A01A2; testOffset = 0
       part_PIs[0024] = stim_PIs[0024]; //      pinName = A01A3; testOffset = 0
                                          .
                                          .
                                          .
       part_PIs[0196] = stim_CIs[0196]; //      pinName = C0166;  tf = -SC   BD
       part_PIs[0197] = stim_CIs[0197]; //      pinName = C0167;  tf = -AC   BD
      #8.000000;        // 24.000000 ns;  From the start of the cycle.
       part_PIs[0021] = stim_PIs[0021]; //      pinName = A01A0;  tf = -BS  ; t
      #8.000000;        // 32.000000 ns;  From the start of the cycle.
       part_PIs[0021] = stim_CIs[0021]; //      pinName = A01A0;  tf = -BS  ; t
      #91984.000000;        // 92016.000000 ns;  From the start of the cycle.
       for ( POnum = 1; POnum <= 271; POnum = POnum + 1 ) begin
         if ((part_POs[POnum] !== resp_POs[POnum]) & (resp_POs[POnum] !== 1'bX)
```

```
          ERR = ERR + 1;
          $display ( "\n650 WARNING (TVE-650): PO miscompare at pattern: %0s a
          $display ( "           Expected: %b    Simulated: %b    On Output: %

          if (( failset ) & ( FAILSETID == 0 ))  failset_setup;
          if ( FAILSETID ) begin
            $fdisplay ( FAILSETID, " Chip %0s pad %0s pattern %0s position %0d
          end
        end
      end
      #10224.000000;        // 102240.000000 ns;  From the start of the cycle.
       resp_POs = 0271'bX;

    end
  endtask

//*************************************************************************//
//                           DEFINE TEST PROCEDURE                         //
//*************************************************************************//

  task test_cycle;
    begin

       if ( sim_trace )  $display ( "\nRunning task:  test_cycle " );

      #0.000000;        // 0.000000 ns;  From the start of the cycle.
       part_PIs[0022] = stim_PIs[0022]; //      pinName = A01A1; testOffset = 0
       part_PIs[0023] = stim_PIs[0023]; //      pinName = A01A2; testOffset = 0
       part_PIs[0024] = stim_PIs[0024]; //      pinName = A01A3; testOffset = 0
                                          .
                                          .
                                          .
       part_PIs[0197] = stim_CIs[0197]; //      pinName = C0167;  tf = -AC   BD
      #8.000000;        // 24.000000 ns;  From the start of the cycle.
       part_PIs[0021] = stim_PIs[0021]; //      pinName = A01A0;  tf = -BS  ; t
      #8.000000;        // 32.000000 ns;  From the start of the cycle.
       part_PIs[0021] = stim_CIs[0021]; //      pinName = A01A0;  tf = -BS  ; t
      #40.000000;        // 72.000000 ns;  From the start of the cycle.
       for ( POnum = 1; POnum <= 271; POnum = POnum + 1 ) begin
         if ((part_POs[POnum] !== resp_POs[POnum]) & (resp_POs[POnum] !== 1'bX)
          ERR = ERR + 1;
          $display ( "\n650 WARNING (TVE-650): PO miscompare at pattern: %0s a
          $display ( "           Expected: %b    Simulated: %b    On Output: %

          if (( failset ) & ( FAILSETID == 0 ))  failset_setup;
          if ( FAILSETID ) begin
            $fdisplay ( FAILSETID, " Chip %0s pad %0s pattern %0s position %0d
          end
        end
      end
      #8.000000;        // 80.000000 ns;  From the start of the cycle.
       resp_POs = 0271'bX;

    end
  endtask

//*************************************************************************//
//                           DEFINE SCAN PROCEDURE                         //
//*************************************************************************//

  task scan_cycle;
```

```
    begin

       if ( sim_trace )  $display ( "\nRunning task:  scan_cycle " );

      #0.000000;        // 0.000000 ns;  From the start of the cycle.
       for ( POnum = 1; POnum <= 271; POnum = POnum + 1 ) begin
         if ((part_POs[POnum] !== resp_POs[POnum]) & (resp_POs[POnum] !== 1'bX)
           ERR = ERR + 1;
           $display ( "\n650 WARNING (TVE-650): PO miscompare at pattern: %0s a
           $display ( "           Expected: %b    Simulated: %b    On Output: %

           if (( failset ) & ( FAILSETID == 0 ))  failset_setup;
           if ( FAILSETID ) begin
             $fdisplay ( FAILSETID, " Chip %0s pad %0s pattern %0s position %0d
           end
         end
       end
      #16.000000;        // 16.000000 ns;  From the start of the cycle.
       part_PIs[0022] = stim_PIs[0022]; //      pinName = A01A1; testOffset = 0
b       part_PIs[0023] = stim_PIs[0023]; //      pinName = A01A2; testOffset = 0
       part_PIs[0024] = stim_PIs[0024]; //      pinName = A01A3; testOffset = 0
                                          .
                                          .
                                          .
       part_PIs[0200] = stim_PIs[0200]; //      pinName = C0170;  tf =  SI   BD
      #8.000000;        // 24.000000 ns;  From the start of the cycle.
       part_PIs[0190] = stim_CIs[0190]; //      pinName = C0160;  tf = -SC   BD
       part_PIs[0193] = stim_CIs[0193]; //      pinName = C0163;  tf = +SC   BD
       part_PIs[0196] = stim_CIs[0196]; //      pinName = C0166;  tf = -SC   BD
       part_PIs[0197] = stim_PIs[0197]; //      pinName = C0167;  tf = -AC   BD
      #8.000000;        // 32.000000 ns;  From the start of the cycle.
       part_PIs[0197] = stim_CIs[0197]; //      pinName = C0167;  tf = -AC   BD
      #8.000000;        // 40.000000 ns;  From the start of the cycle.
       part_PIs[0021] = stim_PIs[0021]; //      pinName = A01A0;  tf = -BS  ; t
      #8.000000;        // 48.000000 ns;  From the start of the cycle.
       part_PIs[0021] = stim_CIs[0021]; //      pinName = A01A0;  tf = -BS  ; t
      #32.000000;        // 80.000000 ns;  From the start of the cycle.
       resp_POs = 0271'bX;

    end
  endtask

//*************************************************************************//
//                        DEFINE SCAN PRECOND PROCEDURE                    //
//*************************************************************************//

  task Scan_Preconditioning_Sequence_SP_WAFER_AC;
    begin

       if ( sim_trace )  $display ( "\nRunning task:  Scan_Preconditioning_Sequ

       stim_PIs[0026] = 1'b0; //      pinName = A01A5;  tf = -BI  -OI  ; testOf
       stim_PIs[0027] = 1'b1; //      pinName = A01A6;  tf = +SE  -BI  ; testOf
       stim_PIs[0029] = 1'b0; //      pinName = A01A8;  tf = -SE  ; testOffset
       stim_PIs[0191] = 1'bZ; //      pinName = C0161;  tf =  SI   BDY   BDY  B
       stim_PIs[0192] = 1'bZ; //      pinName = C0162;  tf =  BDY   BDY  BIDI ;
       stim_PIs[0194] = 1'bZ; //      pinName = C0164;  tf = -SE   BDY  BIDI ;
       stim_PIs[0198] = 1'bZ; //      pinName = C0168;  tf = +CI   BDY  BIDI ;
       stim_PIs[0200] = 1'bZ; //      pinName = C0170;  tf =  SI   BDY   BDY  B

       test_cycle;
```

```
      stim_PIs[0194] = 1'b0; //      pinName = C0164;  tf = -SE   BDY  BIDI ;
      stim_PIs[0198] = 1'b1; //      pinName = C0168;  tf = +CI   BDY  BIDI ;

      test_cycle;

    end
  endtask

//*************************************************************************//
//                      DEFINE SCAN PRECOND BSR PROCEDURE                  //
//*************************************************************************//

  task Scan_Preconditioning_Sequencebsr_SP_WAFER_AC;
    begin

      if ( sim_trace )  $display ( "\nRunning task:  Scan_Preconditioning_Sequ

    end
  endtask

//*************************************************************************//
//                       DEFINE SKEWED UNLOAD PROCEDURE                    //
//*************************************************************************//

  task Skewed_Unload_Sequence_SP_WAFER_AC;
    begin

      if ( sim_trace )  $display ( "\nRunning task:  Skewed_Unload_Sequence_SP

      stim_PIs[0021] = 1'b1; //      pinName = A01A0;  tf = -BS  ; testOffset

      scan_cycle;

      stim_PIs[0021] = 1'b0; //      pinName = A01A0;  tf = -BS  ; testOffset

    end
  endtask

//*************************************************************************//
//                       DEFINE SCAN SEQUENCE PROCEDURE                    //
//*************************************************************************//

  task Scan_Sequence_SP_WAFER_AC;
    begin

      if ( sim_trace )  $display ( "\nRunning task:  Scan_Sequence_SP_WAFER_AC

      for ( CYCLE = 1; CYCLE <= MAX; CYCLE = CYCLE + 1 ) begin
     #0.000000;        // 0.000000 ns;  From the start of the cycle.
        if ((part_POs[75] !== resp_MLs[0+CYCLE]) & (resp_MLs[0+CYCLE] !== 1'bX
          ERR = ERR + 1;
          $display ( "\n660 WARNING (TVE-660): SO miscompare at pattern: %0s a
          $display ( "           Expected: %0b    Simulated: %0b   On Output:

          if (( failset ) & ( FAILSETID == 0 ))  failset_setup;
          if ( FAILSETID ) begin
            $fdisplay ( FAILSETID, " Chip %0s pad %0s pattern %0s position %0d
          end
        end
        if ((part_POs[77] !== resp_MLs[1278+CYCLE]) & (resp_MLs[1278+CYCLE] !=
```

```
          ERR = ERR + 1;
          $display ( "\n660 WARNING (TVE-660): SO miscompare at pattern: %0s a
          $display ( "           Expected: %0b    Simulated: %0b   On Output:

          if (( failset ) & ( FAILSETID == 0 ))  failset_setup;
          if ( FAILSETID ) begin
            $fdisplay ( FAILSETID, " Chip %0s pad %0s pattern %0s position %0d
          end
        end
      #16.000000;        // 16.000000 ns;  From the start of the cycle.
       part_PIs[0191] = stim_SLs[0000+CYCLE]; //      pinName = C0161;  tf =  S
       part_PIs[0200] = stim_SLs[1278+CYCLE]; //      pinName = C0170;  tf =  S
      #8.000000;        // 24.000000 ns;  From the start of the cycle.
       part_PIs[0197] = 1'b1; //      pinName = C0167;  tf = -AC   BDY  BIDI ;
      #8.000000;        // 32.000000 ns;  From the start of the cycle.
       part_PIs[0197] = 1'b0; //      pinName = C0167;  tf = -AC   BDY  BIDI ;
      #8.000000;        // 40.000000 ns;  From the start of the cycle.
       part_PIs[0021] = 1'b1; //      pinName = A01A0;  tf = -BS  ; testOffset
      #8.000000;        // 48.000000 ns;  From the start of the cycle.
       part_PIs[0021] = 1'b0; //      pinName = A01A0;  tf = -BS  ; testOffset
      #32.000000;        // 80.000000 ns;  From the start of the cycle.
       end
       resp_MLs[0001:1278] = 1278'bX;
       resp_MLs[1279:2556] = 1278'bX;
       resp_POs = 0271'bX;
       stim_SLs[0001:1278] = 1278'b0;
       stim_SLs[1279:2556] = 1278'b0;
       stim_PIs = part_PIs;
       CYCLE = 0;

    end
  endtask

//*************************************************************************//
//                        DEFINE SKEWED LOAD PROCEDURE                     //
//*************************************************************************//

  task Skewed_Load_Sequence_SP_WAFER_AC;
    begin

       if ( sim_trace )  $display ( "\nRunning task:  Skewed_Load_Sequence_SP_W

       stim_PIs[0191] = stim_SSs[0001]; //      pinName = C0161;  tf =  SI   BD
       stim_PIs[0200] = stim_SSs[0002]; //      pinName = C0170;  tf =  SI   BD

       stim_PIs[0197] = 1'b1; //      pinName = C0167;  tf = -AC   BDY  BIDI ;

       scan_cycle;

       stim_PIs[0197] = 1'b0; //      pinName = C0167;  tf = -AC   BDY  BIDI ;

    end
  endtask

//*************************************************************************//
//                         DEFINE TIMED TEST PROCEDURE                     //
//*************************************************************************//

  task TBautoLogicSeq1_20001221220437_0;
    begin
```

```
      if ( sim_trace )  $display ( "\nRunning task:  TBautoLogicSeq1_200012212

     #0.125000;        // 0.125000 ns;  From the start of the cycle.
      part_PIs[0021] = stim_PIs[0021];
      part_PIs[0027] = stim_PIs[0027];
      part_PIs[0200] = stim_PIs[0200];
     #6.625000;        // 6.750000 ns;  From the start of the cycle.
      part_PIs[0021] = stim_CIs[0021];
     #5.250000;        // 12.000000 ns;  From the start of the cycle.
      part_PIs[0029] = stim_PIs[0029];
     #0.375000;        // 12.375000 ns;  From the start of the cycle.
      part_PIs[0198] = stim_PIs[0198];
     #0.625000;        // 13.000000 ns;  From the start of the cycle.
      part_PIs[0196] = stim_PIs[0196];
     #0.375000;        // 13.375000 ns;  From the start of the cycle.
      part_PIs[0192] = stim_PIs[0192];
     #2.125000;        // 15.500000 ns;  From the start of the cycle.
      part_PIs[0191] = stim_PIs[0191];
     #1.500000;        // 17.000000 ns;  From the start of the cycle.
      part_PIs[0194] = stim_PIs[0194];
     #2.625000;        // 19.625000 ns;  From the start of the cycle.
      part_PIs[0196] = stim_CIs[0196];
     #5.375000;        // 25.000000 ns;  From the start of the cycle.
      stim_PIs[0021] = 1'b0;
      stim_PIs[0190] = 1'b0;
      stim_PIs[0193] = 1'b1;
      stim_PIs[0196] = 1'b0;
      stim_PIs[0197] = 1'b0;

    end
  endtask

                                  .
                                  .
                                  .

//*************************************************************************//
//                        DEFINE TIMED TEST PROCEDURE                      //
//*************************************************************************//

  task TBautoLogicSeq226_20001221224029_0;
    begin

      if ( sim_trace )  $display ( "\nRunning task:  TBautoLogicSeq226_2000122

     #0.125000;        // 0.125000 ns;  From the start of the cycle.
      part_PIs[0196] = stim_PIs[0196];
     #12.375000;        // 12.500000 ns;  From the start of the cycle.
      part_PIs[0196] = stim_CIs[0196];
     #5.875000;        // 18.375000 ns;  From the start of the cycle.
      part_PIs[0021] = stim_PIs[0021];
     #6.625000;        // 25.000000 ns;  From the start of the cycle.
      part_PIs[0021] = stim_CIs[0021];
     #0.125000;        // 25.125000 ns;  From the start of the cycle.
      stim_PIs[0021] = 1'b0;
      stim_PIs[0190] = 1'b0;
      stim_PIs[0193] = 1'b1;
      stim_PIs[0196] = 1'b0;
      stim_PIs[0197] = 1'b0;

    end
```

```
endtask

endmodule
```

F

TBDpatt Language Syntax

TBDpatt File Constructs

The following are the major constructs that constitute a TBDpatt file:

Keyword - Keywords are a basic unit of the TBDpatt syntax. Keywords imply a specific meaning to the TBDpatt parser. A keyword is an alphanumeric string that does not contain white space. Keywords may contain underscores.

Delimiter - Delimiters are single characters that are used to signify the beginning and/or end of a particular syntactic construct. For example, the end of a statement, the end of a list, or the end of a list item. Whether or not a particular character is acting as a delimiter may depend upon the context. For example, a blank, tab or newline serves as a delimiter among keywords, but not within a string value. TBDpatt uses the following characters as delimiters:

Delimiter Name	Character
Blank	
Tab	
Newline	
Semicolon	;
Colon	:
Parentheses	( )
Equal	=
Double quote	"
Quotation	’
Brackets	[ ]
Number sign	#

Value - A value is a delimited string that appears in TBDpatt that has no implicit meaning to the TBDpatt parser. Values typically appear on the right-hand side of an equal sign. Value delimiters vary, depending on the context.

Statement - A statement is a syntactically complete entity within the TBDpatt file. It is composed of keywords, values and delimiters. All statements end with a semicolon.

Attribute list - An attribute list is enclosed in parentheses, and the attributes are separated by commas. Each attribute is either a keyword or keyword=value. Attribute lists may appear at various locations within the TBDpatt file.

Vector - A vector is a series of logic values that fully specifies the value of a given set of pins or blocks in the model. A vector is numbered by position from left to right, with the first position being one. The correspondence between vector value position and the model entity to which the value belongs is established by use of a vector correspondence file.

Block - This is a container for other blocks and data. A block has start and stop statements. A block has scope, such that everything appearing after the start of the block and prior to the end of the block is considered to be contained within that block. Blocks have attributes that apply to the entire scope of the block. A block is started using the delimiter [, followed by the required block type, followed by an optional block id. For example,

```
[ Test_Section 1.5;
```

signifies the beginning of the fifth test section block within the first experiment in a TBDpatt file. Optionally, following the block id if it's present, else following the block type, is an attribute list associated with that block. For instance,

```
[ Test_Section 1.5 (tester_termination=0,
    termination_domination=tester);
```

specifies a pair of attribute values that are associated with Test Section 1.5. So, a complete block construct has the following form:

```
[ Test_Section 1.5 (tester_termination=1,
termination_domination=tester);
    [ Tester_Loop 1.5.1;
       [ Test_Procedure 1.5.1.1;
       ...
       ] Test_Procedure 1.5.1.1;
    ] Tester_Loop 1.5.1;
    ...
]Test_Section 1.5;
```

Block ids are present only for the convenience of the human reader. When the TBDpatt file is processed for input, block ids are ignored.

The following block types are currently supported:

```
Experiment
Test_Section
```

```
    Tester_Loop
    Test_Procedure
    Test_Sequence
    Pattern
```

Event - Events occur within pattern blocks, and the order in which the events appear correspond to the point in the simulation at which a given event occurred. Events have an optional associated id. As with block ids, event ids are useful only for the human reader, and are ignored by the TBDpatt parsing program.

Each event has an associated type that identifies the kind of test data it contains. Each event contains only one type of test data.

For more details, refer to "Event" on page 38.

The format of the data contained within the event is dependent on the event type and the global TBDpatt format.

Some sample events follow:

```
[ Pattern 1.1.1.1.1.2;
Event 1.1.1.1.1.2.1
      Scan_Load ():
      01001100011100001 11;
] Pattern 1.1.1.1.1.2;
[ Pattern 1.1.1.1.1.3;
  ...
  Event 1.1.1.1.1.3.5
        Pulse ():
        6=+;
...
] Pattern 1.1.1.1.1.3;
```

Comment - Comments are useful for annotating a TBDpatt file so that the file is easier to understand. Comments are not imported into the TBD binary file. Comments begin with the number sign **(#)** and end with the newline character **(**end of line**)**. A comment may appear on the same line following other non-commentary TBDpatt constructs, or on a line by itself.

Here are some examples of legal comments:

```
        [ Pattern 1.2.3.4.5.6;
        # Hey! There's some really neat stuff to follow...
          Event 1.2.3.4.5.6.1
                Stim_PI ():
                01100110;
        ] Pattern 1.2.3.4.5.6;


        [ Pattern 1.2.3.4.5.7;
          Event 1.2.3.4.5.7.1 # Hey! This is really neat stuff...
                     Stim_PI ():
                    01100110;
        ] Pattern 1.2.3.4.5.7;
```

TBDpatt Language Definition

Some earlier syntax will no longer be documented in this manual. Read Vectors will continue to accept the syntax even though it is not documented.

Syntax notation is defined as follows:

- `::=` is the definition symbol, read “is defined as”. The definition is everything following the `::=` symbol, up to the next blank line.
- variables of the language are italicized and are to be replaced by their definition For example,

  pattern

  is to be replaced by the definition for *pattern.*
- elements of the language which are to be typed as shown are in bold. For example,

  `[ Pattern`

  is to be typed exactly as shown when writing a TBDpatt file.
- Optional items are surrounded by braces, like this: `{optional item}`.
- An asterisk signifies that the preceding item may be repeated an indefinite number of times. *item** means a sequence of one or more occurrences of *item.* `{` *item* *`}` or `{` *item* `}`* signifies that *item* may be repeated 0 or more times.

  The vertical bar `("|")` separates mutually exclusive alternatives.

  - The first choice is from the end of the `::=` to the first `|`
  - The second choice is from the first `|` to the second `|` (or the end of the line)
  - The third choice is from the second `|` to the third `|` (or the end of the line), etc. For example,

    x `::=` **`ABC`** `|` **`XYZ`**

    means that item *x* may be either the character string "ABC" or the character string "XYZ".
- All other characters and symbols are to be interpreted literally.
- *blank* is defined to be an explicit blank character

The language definition is in a top-down order, high-level down to low-level; references precede definitions.

The TBDpatt language definition follows:

TBDpatt Language High-level Syntax

```
TBD_ascii_patterns_file ::= TBDpatt
| TBDseqPtt

asterisk ::= *

bit ::= 0 | 1


bit_data ::=
bit_position = logic_value

bit_position ::= positive_integer

bit_string ::= bit*

channel_scan_attribute ::=
        block_signature_register | skewed_load
        | skewed_unload | fast_forward
        | fast_forward_save

character_string ::= string_character*

comment ::= #
{comment_character*}

comment_character::=
string_character |
! | @ | # | $ |
& | asterisk | ( | ) | + |
& | [ | ] |
/ | : | ; |
, | " | '



component ::= positive_integer{.positive_integer}*

d ::= decimal_digit
```

```
data ::= quoted_character_string

datetime ::= yyyymmddtttttt

decimal_digit ::=
0 | positive_digit

decimal_fraction ::=
{unsigned_integer}.decimal_digit*

default_value ::=
0 | 1 | X |
scan_0 | scan_1

default_value_attribute ::= default_value=
default_value

diagnostic_pin_type ::=
Driver | Receiver

diagnostic_text ::=  stuck_driver_fault |
shorted_net_fault;

domination_type ::= tester | product

event_name_and_data ::=
          stim_measure_scan_pulse_response_or_expect_event |
          lbist_or_wrpt_event |
          miscellaneous_event |

event_user_object ::= keyed_data

expect_event_data ::= node_data;
{Detected_fault: diagnostic_text}*

experiment ::=
          [ Experiment name {component}
({TDM});
           {experiment_user_object*}
           {  [ Test_Section {component}(test_section_attribute=

test_section_attribute {,test_section_attribute});
           {test_section_user_object*}
           {  [ Tester_Loop {component}
```

```
({procedures_have_memory});
                    {tester_loop_user_object*}
                    { [ Test_Procedure {component}
({test_procedure_attribute{,test_procedure_attribute}*});
                       {test_procedure_user_object*}
                       { [Test_Sequence {component}
({test_sequence_attribute{,test_sequence_attribute}*});
                          {test_sequence_user_object*}
                          { [ Pattern {component}
({pattern_attribute{,pattern_attribute}*});
                               {pattern_user_object*}
                        {Event {component} event_name_and_data;}*
                               {event_user_object*}
                            ] {Pattern} {component}; }*
                            ] {Test_Sequence} {component}; }*
                       ] {Test_Procedure} {component}; }*
                    ] {Tester_Loop} {component}; }*
                 ] {Test_Section} {component}; }*
              ] {Experiment {name}} {component};

experiment_user_object ::= keyed_data
| sequence_definition

fault_type ::=
static_faults | iddq_faults |
dynamic_faults | driverReceiver_faults

flat_index ::= unsigned_integer

force_event_attribute ::= hold


form_type ::= name | hier_index | flat_index

hex_string ::= {decimal_digit |
a | b | c | d |
e | f | A | B |
C | D | E | F
}*

hierarchical_model_entity_name ::=
"character_string"
```

```
hierblock_index ::= unsigned integer

ignore_data ::= {ignore_value}*
| {node identifier = ignore value}*

Ignore_Measures ::= [Ignore_Measures;
                        {Ignore_Latches(default_value=1 | 0):
ignore_data;}
                        {Ignore_POs(default_value=1 | 0):
ignore data;}
                      ] {Ignore_Measures};
## (default_value=1 | 0) is an optional entry
## By default,default_value=0 that means latches and PO's will
be measured
## Example:  Mask all PO's and Mask all measurable Latches except
the 4 explicitly listed
## [ Test_Procedure 3.1.1.5 () ;
## [Ignore_Measures;
## Ignore_Latches(default_value=1):
## "Block.f.l.top.nl.SCO_NEWPINS.B002.I0.nlat_LATCH"=0
## "Block.f.l.top.nl.SCO_NEWPINS.B002.I1.nlat_LATCH"=0
## "Block.f.l.top.nl.SCO_NEWPINS.B002.I2.nlat_LATCH"=0
## "Block.f.l.top.nl.SCO_NEWPINS.B002.I3_nlat_LATCH"=0 ;
## Ignore_POs(default_value=1): ;
## ] Ignore_Measures;
ignore_value ::= 0 | 1

internal_logic_value ::=
0 | 1 | x | Z |
u | z | l | a |
b | c | d | m |
w | n | p | q |
H

internal_response_data ::=
node_identifier{time unsigned_integer
= internal logic value}*

iteration_attribute ::=
iteration count | fast_forward

iteration_count ::=
iteration =   positive integer
```

```
key ::= quoted_character_string

keyed_data ::= [ Keyed_Data ;
                    {key = data}*
                    ]{Keyed_Data};

lbist_or_wrpt_event ::=
        Channel_Scan ({channel_scan_attribute
{,channel_scan_attribute}*});
| Connect_Tester_PRPG ({timed_type_attribute}): node_identifier*
        | Effective_Cycle_Mask (): hex_string*
        | Latch_Values (signature_event_attribute_list):
hex_string*
        | Latch_Weight (): weight_data
        | PI_Weight (): weight_data
        | Product_PRPG_Signature (iteration_attribute):
Product_LFSR_State*
        | Product_MISR_Signature
(signature_event_attribute_list): Product_LFSR_State*
        | Pulse_Tester_PRPG_Clocks ({timed_type_attribute})
        | Pulse_Tester_SISR_Clocks ({timed_type_attribute})
        | Tester_PRPG_Seed (): tester_LFSR_state*
        | Tester_PRPG_Signature (iteration_attribute):
        tester_LFSR_state*
        | Tester_SISR_Signature
(signature_event_attribute_list):
        tester_LFSR_state*
        | Tester_SISR_Mask (): bit_string
        | Tester_SISR_Seed ({fast_forward}): tester_LFSR_state*

leading_zeros ::= 0*

Lineholds ::= [Lineholds:
            { node_identifier=logic_value(linehold_source); }*
            ] {Lineholds};

linehold_source ::= PTU_generated | app_generated

load_data ::=
stim_vector_data | bit_data

load_event_attribute_list ::=
```

```
stimregister_id , number_shifts

logic_test_type ::=
static | dynamic

logic_value ::= 0 | 1 | x | Z

loop_type ::= loop,

              repeat_attribute


lowercase_alphabetic ::=
a | b | c | d |
e | f | g | h |
i | j | k | l |
m | n | o | p |
q | r | s | t |
u | v | w | x |
y | z

m ::= decimal_digit

macro_index ::= unsigned_integer

macro_tester_loop_data ::= [Macro_Tester_Loop
                       (macro_algorithm=name,mic_name=name);
                        macros_in_group=(hierblock_index{,macro_
index}*);
                        ] {Macro_Tester_Loop};

macro_test_procedure_data ::=
[Macro_Test_Procedure(macro_operation=name);
                    ] {Macro_Test_Procedure};

macro_test_sequence_data ::= [Macro_Test_Sequence;
                    macros_in_subgroup = (hierblock_index{,
macro_index}*);
                    ] {Macro_Test_Sequence};

measure_event_data ::=
measure_vector_data | node_data*
```

```
measure_register_data ::= [Measure_Register
                    (Measure_Register=unsigned_integer,
                      Scan_Out=nodeID,
                      Scan_Length=unsigned_integer,
                    Load_Section_Preconditioning_Sequence=name,
                      Load_Sequence=name);
                       ] {Measure_Register};

measureregister_id ::=
real_integer

measure_vector_data ::= logic_value*

miscellaneous_event ::=
           Apply (): name |
           Begin_Test_Mode (): name |
           Repeat (): positive_integer

mode_type ::= node | vector

name ::= character_string |
"character_string"

node_data ::=
node_identifier = logic_value

nodeID ::= unsigned_integer

node_identifier ::= hierarchical_model_entity_name |
TB_hiermodel_pin_index | flat_index

node_type ::= PI | PO | PPI | cut_point


number_repeats ::= unsigned_integer

number_shifts ::=
real_integer

ObservePoints ::= [Observe_Points:
                     { node_identifier*};
                     ] {Observe_Points};
```

```
one_half ::= -1

4
optional_timing_data_attribute ::= maximum_path_length=time
                                 | minimum_path_length=time
                                 | cycles_to_repeat
                                 | early =
(unsigned_real_number,unsigned_real_number,unsigned_real_numbe
r)
                                 | late =
(unsigned_real_number,unsigned_real_number,unsigned_real_numbe
r)
                                 | delay_file_name="character_string"
                                 |
delay_file_date_time_stamp="character_string"
                                 |
delay_file_audit_string="character_string"
                                 | VDD = "character_string"
                                 | VTT = "character_string"
                                 | temp = "character_string"

optional_wait_osc_attribute ::= nostability | off


osc_cycles_attribute ::= cycles=positive_integer


osc_down_attribute ::= down=unsigned_real_number


osc_up_attribute ::= up=unsigned_real_number


pattern_attribute ::=
pattern_type_attribute | miscompare

pattern_type ::=
static | dynamic |
begin_loop | end_loop

pattern_type_attribute ::=
pattern_type = pattern_type
```

```
pattern_user_object ::= keyed data

pin_data ::=
pin_name=nodeID | {tf=polarity | test func}

pin_name ::= name

Pin_Timing ::= [Pin_Timing:
                {Pin_Timing_entry*}
                ] {Pin_Timing};

Pin_Timing_cycle ::=
tester_cycle time cycle info;

Pin_Timing_entry ::= Pin_Timing_list |
Pin_Timing_cycle

Pin_Timing_list ::= TimingType
TimingValues time cycle info event
positive_integer

node type timing node identifier;


pi_vector_data ::= [PI_Vector():pin_data*;
                   ] {PI_Vector};

polarity ::= + | - | Z

positive_digit ::=
1 | 2 | 3 | 4 |
5 | 6 | 7 | 8 |
9

positive_integer ::=
{leading_zeros}positive_digit{decimal_digit*}

po_vector_data ::= [PO_Vector():pin_data*;
                   ] {PO_Vector};

PPI_data ::= PPI_name = PPI_value
```

PPI_name ::= "*character string*"

PPI_value ::= *bit*

Product_LFSR_State ::= *hex string*

quoted_character_string ::=
"*comment character**"

real_integer ::=
unsigned integer{.}

repeat_attribute ::=

repeat=*positive_integer*

seq_audit_attribute ::= **Forces_unverified**
| **Forces_verified**
| **Forces_bad**
| **PPI_unverified**
| **PPI_verified**
| **PPI_bad**
| **Failed_verification**
| **Invalid_oscillator**

seq_audit_attribute_list ::= *seq_audit_attribute**

seq_def_attribute ::= *sequence type* | *repeat attribute* |
sequences_have_memory | *seq_audit_attribute_list*

seq_def_attribute_list ::= *seq def attribute**

sequence_def_id ::= **[SeqDef** = **(***name*,"*datetime*"**)]** {**SeqDef**} ;

seqdef_user_object ::= *keyed_data* | *timing_data*

| *setup_seq_name*

```
sequence_definition ::=
          [ Define_Sequence name {datetime}
{component}

            ({seq_def_attribute_list}) ;

            {seqdef_user_object*}
                       { [ Pattern {component}
({pattern_attribute{,pattern_attribute}*});
                            {pattern_user_object*}
                      {Event {component} event_name_and_data;}*
                            ] {Pattern} {component}; }*
          ] {Define_Sequence {name}}{component};

sequence_type ::=
                modeinit | scanop

               | scanentry | scansection
               | scanexit | scanprecond

               | skewunload | scansequence

               | scanlastbit | skewload
               | scansectionexit | loadsuffix

               | prpgsave | prpgrestore

               | test | setup


Setup_Patterns ::= [Setup_Patterns {block id};
                            { [Pattern {component}
({pattern_attribute{,pattern_attribute}*});
                            {pattern_user_object*}
                            {Event {component}
event_name_and_data;}*
                            ] {Pattern} {component}; }*
                      ] {Setup_Patterns} {block id};

setup_seq_name ::= [ SetupSeq = name
] ;
```

```
shorted_net_fault ::=
SNT on Net node_identifier

signature_event_attribute_list ::=
iteration_attribute{,final}

special_string_character ::=
. | _ | -
| $ | @

start_osc_attribute ::= osc_cycles_attribute | osc_up_attribute
| osc_down_attribute


start_osc_attribute_list ::= start_osc_attribute*


stim_data ::= stim_vector_data | node_data*

stim_measure_scan_pulse_response_or_expect_event ::=
          | Skewed_Scan_Unload ({default_value_attribute}):
measure_event_data
          | Expect ({timed_type_attribute}): expect_event_data

          | Force ({force_event_attribute}): node_data*

          | Internal_Response (): internal_response_data
          | Measure_Scan_Data (): node_identifier*
          | Scan_Unload ({default_value_attribute}):
{measure_event_data}
          | Measure_PO ({timed_type_attribute}):
{measure_event_data}
          | Measure_Current ()
          | Pulse ({timed_type_attribute}): {node_identifier =
polarity}*

          | Pulse_PPI ({timed_type_attribute}): {PPI_name =
polarity}*
          | Put_Stim_PI : stim_data
          | Release : node_identifier*

          | Set_Scan_Data (): node_identifier*
```

```
          | Start_Osc ({start_osc_attribute_list}):
{node_identifier = polarity}*

          | Stim_Clock ({timed_type_attribute}): node_data*
          | Scan_Load ({default_value_attribute}): stim_data
          | i ({default_value_attribute}): stim_data
          | Stim_PI ({timed_type_attribute}): stim_data
          | Stim_PI_Plus_Random ({timed_type_attribute}):
stim_data

          | Stim_PPI ({timed_type_attribute}):
PPI_data*
          | Stim_PPI_Clock ({timed_type_attribute}):
PPI_data*
          | Stop_Osc : PPI_data*

          | Load_SR ({load_event_attribute_list}): load_data
          | Skewed_Load_SR ({load_event_attribute_list}):
load_data
          | Unload_SR ({unload_event_attribute_list}):
unload_data
          | Skewed_Unload_SR ({unload_event_attribute_list}):
unload_data

          | Wait_Osc (wait_osc_attribute_list) PPI_name


stim_register_data ::= [Stim_Register
                        (Stim_Register=unsigned_integer,
                        (Scan_In=nodeID,
                        (Scan_Length=unsigned_integer,
                        Load_Section_Preconditioning_Sequence=name,
                        (Load_Sequence=name);
                         ] {Stim_Register};

stimregister_id ::=
real_integer

stim_vector_data ::=
{0 | 1 | x | Z |
.}*

string_character ::= decimal_digit |
```

```
lowercase_alphabetic |
            uppercase_alphabetic | special_string_character

stuck_driver_fault ::=
SDT on diagnostic_pin_type Pin
node_identifier

t ::= decimal_digit

TBDpatt ::=
                  TBDpatt_Format ({mode
=mode_type,}model_entity_form=form_type );
                  {vector_correspondence_data}
                  {test_pattern_audit_summary}
                  experiment*

TBDseqPatt ::=
                  TBDpatt_Format ({mode
=mode_type,}model_entity_form=form_type );
                  {vector_correspondence_data}
                  {test_pattern_audit_summary}
                  sequence_definition*

TB_hiermodel_pin_index ::= unsigned_integer

termination_type ::=
0 | 1 | none

test_coverage_attribute ::=
fault_type=unsigned_integer
| percent_fault_type=
unsigned_real_number

tester_LFSR_state ::=
node_identifier = hex_string

tester_loop_user_object ::= keyed_data
| macro_tester_loop_data

test_func ::=
AC | AS | BC | BDY |
BI | BS | CI | CTL |
EC | ES | LH | ME |
```

```
OI | PC | PS | SC |
SG | SI | SO | TI |
NIL | NIC

test_pattern_audit_summary ::= comment*

test_procedure_attribute ::= sequences_have_memory
                           | slow_to_turn_off
                           | type = test_procedure_type
                           | non-uniform_sequences
                           | test_coverage_attribute

test_procedure_type ::=
init | normal

test_procedure_user_object ::= keyed_data |
macro_test_procedure_data

test_section_attribute ::=
                       | tester_termination= termination_type
                       | termination_domination= domination_type
                       | test_type= logic_test_type
                       | test_section_type= test_section_type
                       | pin_timing
                       | tester_PRPGs
                       | tester_signatures
                       | product_PRPGs
                       | product_signatures
                       | fast_forward
                       | fast_forward_pins
                       | fast_forward_sequences

test_section_type ::=
                      logic | logic_WRP
                      | logic_LBIST | flush
                      | scan | driver_receiver
                      | macro | IDDq
                      | channel_scan
                      | IEEE_1149.1_integrity
                      | ICT_stuck_driver
                      | ICT_stuck_driver_diagnostic
                      | ICT_shorted_nets_log(n+2)
                      | ICT_shorted_nets_2*logn
```

```
                          | ICT_shorted_nets_n+1
                          | IOWRAP_stuck_driver
                          | IOWRAP_shorted_nets_log(n+2)
                          | IOWRAP_shorted_nets_2*logn
                          | IOWRAP_shorted_nets_n+1
                          | path

test_section_user_object ::= keyed_data

test_sequence_attribute ::=
                              | type = test_sequence_type
                              | miscompare

                              | seq_audit_attribute_list


test_sequence_type ::=
normal | init | setup |
loop_type

test_sequence_user_object ::= timing_id | sequence_def_id |
macro_test_sequence_data

time ::= unsigned_real_number
time_scale

time_cycle_info ::=
time cycle positive_integer

timed_type ::=
release | propagate |
capture | none

timed_type_attribute ::=
timed_type = timed_type

time_scale ::=
ps | ns | us
| ms | s

timing_data ::= [Timing_Data {name} {component}
                     (timing_data_attribute_list)
                     {Lineholds}
```

```
                    {ObservePoints}
                    {Ignore_Measures}
                    {Setup_Patterns}
                    Pin_Timing
                    ] {Timing_Data} {component};

timing_data_attribute_list ::= timing_data_type,
                               number_of_cycles=positive_integer
                               {,optional_timing_data_attribute}
*

timing_data_type ::=
manual | automatic

timing_id ::= [Timing_ID =
positive_integer ];

timing_node_identifier ::= node_identifier | PPI_name


TimingType ::=
stim_PIs | stim_clocks |

stim_PPIs | stim_PPI_clocks |

leading_edge_of_pulse |
trailing_edge_of_pulse |

leading_edge_of_PPI_pulse | trailing_edge_of_PPI_pulse |

PO_strobe

TimingValues ::=
Rising | Falling | RorF | to_Z

unload_data ::=
measure_vector_data | bit_data

unload_event_attribute_list ::=
measureregister_id , number_shifts

unsigned_integer ::= decimal_digit*
```

unsigned_real_number ::=*real_integer* |
decimal_fraction

unweighted ::= **-.**

uppercase_alphabetic ::=
A | **B** | **C** | **D** |
E | **F** | **G** | **H** |
I | **J** | **K** | **L** |
M | **N** | **O** | **P** |
Q | **R** | **S** | **T** |
U | **V** | **W** |
X | **Y** | **Z**

vector_correspondence_data ::= *comment**

wait_osc_attribute_list ::= *osc_cycles_attribute*
{*optional_wait_osc_attribute*}*

weight ::=
S0 | **S1** | **A1** | **A2** |
A3 | **A4** | **A5** | **A6** |
A7 | **O1** | **O2** | **O3** |
O4 | **O5** | **O6** | **O7** |
unweighted | *one_half*

weight_bit_data ::=
weight_vector | {*bit_position***=***weight*}*

weight_data ::=
weight_vector |
{*node_identifier***=***weight*}*

weight_vector ::= *weight**

y ::= *decimal_digit*

Index

Numerics

A

B

C

D

E

F

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y