Scan Insertion and ATPG Development via Synopsys™ Test Compiler

This application note presents Atmel's design guidelines, then gives specific recommendations for scan insertion and ATPG vector generation using the Synopsys™ Test Compiler™, version 3.2a. Example cases for both scan insertion and ATPG vector generation are given. These examples may serve as template scripts for your work. The basic methodology is planned to continue to apply on future releases of Test Compiler, but it should be noted that the specific syntax of the commands may change slightly between software releases.

Design Guidelines

Atmel uses the Synopsys Test Compiler software to insert scan and generate Automatic Test Pattern Generated (ATPG) vectors. Atmel's ATL60 and ATL50 macro cell libraries support the full-scan multiplexed-flip-flop style of scan. In order to obtain the best performance from Test Compiler under these conditions, the designer must have an extremely well controlled, fully synchronous design. Successful scan testing as implemented by Test Compiler is comprised of two steps: properly inserting the scan chains and properly generating ATPG vectors. Both steps are required but it is recommended that each step is performed separately. To that end, the following guidelines are provided.

General Guidelines for Scan Design - Design For Test (DFT) Rules

1. Perform testability analysis (check_test command) on each module of the design as it is compiled.
   - Highlights testability problems early in the design cycle.
   - Helps to isolate the cause of testability problems as you move up the hierarchy.
2. One additional input port is required for scan enable (TE).
3. Avoid using latches wherever possible.
4. Avoid combinational feedback loops.
5. Perform scan insertion at the top level prior to pad insertion.
6. Clock and Enable Rules:
   - Do not use multiple clocks within a scan chain.
   - Do not gate the clock.
   - Use the same clock edge for all clocking.
   - Dedicate an I/O port to reset and do not gate the Flip-Flop (FF) reset function.
7. Tri-state™ Nets Rules:
   - Avoid internal Tri-state busses.
   - Avoid using bidirectional ports for scan-in and scan-out ports.

ATL60
ATL50
Scan Insertion and ATPG

Application Note

0586A

Why are these guidelines listed?

In general, there are workarounds to all of the above problem areas. These workarounds involve trade-offs between the effort and cost to implement design fixes and the possibility of improving the achievable fault coverage. Design modifications for workarounds can be problematic in that the modifications can:

- take time to modify the design and verify the impact of the modifications (schedule impact)
- alter the performance of the functional design (timing impact)
- increase run time of Synopsys (schedule impact)

Additional constraints must be placed on the Test Compiler tool in order to program the tool to recognize these design modifications. These constraints can result in an additional schedule impact and may result in a reduced fault coverage.

In the context of this application note, the term Test Mode logic will refer to any design modification which allows for a design to be reconfigured for scan testing. Test Mode logic is used extensively to work around certain DFT rule violations. For most designs, this will result in a dedicated input port called TM or TEST_MODE, which when asserted, will reconfigure the design for scan testing. This input is held constant throughout the scan testing process. For some designs which may already have several different modes of operation, an additional “test mode” may be added to the internal logic without having to add an input port. The actual Test Mode logic added to a design can vary greatly by design. It typically will consist of multiplexing logic.

Guideline #1: check_test command

Designers should perform check_test commands on each module of a design as it is compiled. This has the advantage of quickly isolating potential testability problems very early in the design cycle and simplifying the debug process by limiting the amount of logic to be debugged at one time. One suggestion to minimize possible warnings is to follow Synopsys' Test Smart Compile methodology. Test Smart compile is invoked by specifying the scan style prior to compiling. To do this, execute the set_scan_style multiplexed_flip_flop command prior to executing any compile command. When this is done, the FFs which are available for consideration during optimization are limited to those that have scannable equivalents. This step can minimize the amount of optimization work that ultimately will be required for specific designs. Note: the set_register_type command takes precedence over a Test Smart compile strategy.

While running check_test on a design, error or warning messages may appear. Questions about these messages can usually be addressed by using the help command in conjunction with the warning or message code. If Design Analyzer is used, highlighting the error or warning message and selecting the SHOW button will highlight the problem area on the schematic. This feature is only valid while in Design Analyzer, not in Design Compiler Shell.

It should be noted that while this guideline is suggested, it does not guarantee that a design consisting only of 100% testable modules will be testable. At the top level of a design, module interconnections may cause other testability problems.

Prior to running check_test on whole or partial designs, all constraints must be set on Test Compiler (minimum condition). check_test is a generic command used by Test Compiler to reprogram its internal fault simulator which checks for warnings and errors. It also infers existing scan chains if proper switches are set. heck_test can be used more liberally if desired, but any constraints set after the last check_test command will have no effect on subsequent operations until check_test is re-executed.

Guideline #2: Test Enable Pin

For each design, Test Compiler needs to have a Test Enable (TE) input, and a Test Clock (usually a system clock) input. A Test Mode (TM) input is sometimes required. Test Compiler needs one Scan Data In (SDI) input and one Scan Data Out (SDO) output for each scan chain within the design, which can usually be mux'ed with other functional inputs and outputs.

Guideline #3: Avoid Latches

Since latches are considered a form of sequential cell, a scannable version of the cell must be available in order to obtain maximum results. Under the multiplexed_flip_flop style of scan, scannable versions of latches are not allowed. This is because scannable versions of latches cannot be functionally modeled. If no design workarounds are implemented, Test Compiler will classify all latches as black-box components. This designation basically implies that the tool doesn't know what to do with the cell. Without design modifications and/or software constraints (not always possible without design modifications), the tool will not be able to test any path in a design containing latches.

The workaround for designs containing latches is to force the latches into a ”transparent” mode during scan testing. This method forces the latch enables to a state that allows data to pass directly through the cell. The latch is essentially turned into a buffer and appears “transparent” to the tool. Based on the design, a set_test_hold command may be able to force this condition without any circuit modifications. This case tends to be the exception rather than the rule. In most designs, circuit modifications will be required. In these cases, the use of Test Mode logic is required to provide control of the latch enables so that they may be set to a transparent mode during scan.

Use of a “set_scan FALSE {cell_list} -transparent” command sets the latches to transparent mode. This command programs the fault simulator to expect that the latches have been set to a transparent mode. It does not automatically set them into a transparent mode. You need to follow the methods mentioned above to insure that the latches have been set to transparent mode.

Even when the latches are made transparent, the use of latches will reduce the overall fault coverage obtainable. This is because the enable nets will not be tested during scan as they are held to a constant logic level.

Guideline #4: Avoid Combinational Feedback

Combinational feedback loops introduce internal states in designs that cannot be synchronously controlled. In order for the ATPG algorithms to work properly, combinational feedback loops must be broken. Use of software constraints can break these loops by forcing unknowns somewhere along their paths. This is not always optimal, and a manual review of all feedback loops is highly recommended. This allows the designer to determine the optimal point at which to break the loop. Manual breaking of loops is carried out through use of the “set_test_isolate” command. Bidirectional ports configured as outputs result in combinational feedback loops.

Guideline #5: Insert Scan Before I/O

This guideline is stated because bidirectional ports tend to complicate the scan insertion process. If scan is inserted prior to instantiating the I/O ports, the process is much simpler. Once scan has been inserted, the I/O can be added to the design and ATPGs can be generated.

Guideline #6: CLOCKS and ENABLES

Test Compiler wants to have full control of the internal states of a design. The simplest way to achieve this is to have full external control of all clocks and enables in the design. Any FF not controllable via external clocks and enables (in scan mode) will not be included in the scan chains. This exclusion can have a major impact on the overall achievable fault coverage.

MULTIPLE CLOCK EDGES: If multiple clock edges are used to clock FFs, you run the risk of improperly passing through data on FFs in a given scan chain. The easiest solution here is to avoid using multiple clock edges altogether. Designs can be modified using Test Mode logic to achieve this condition during scan mode. If this methodology is not possible, you must manually order the scan chains to place the negative edge triggered FF at its beginning.

CLOCKS AND ENABLES AS DATA INPUTS: Clocks and enables should not be used as data inputs. If they are used in this manner, changes in the clock/enable signal may propagate invalid data changes throughout the circuit. Such use will result in reduced fault coverage on those data inputs. Test Mode logic can be used to correct these situations.

ASYNCHRONOUS RESETS: If external ports control the reset nets in the design, Test Compiler should automatically handle these conditions during ATPG generation. If there are gated resets in the design, they must be disabled using Test Mode logic. This will allow the tool to properly insert scan and generate vectors. Disabling these resets will reduce fault coverage on the reset nets.

Occasionally, resets can be handled through the use of custom initialization test protocols. This entails additional work and should not be initiated without first notifying Atmel of your intent. The custom initialization test protocol should be reviewed by an Atmel engineer to ensure that Atmel will be able to properly use it.

GATED CLOCKS: Gated clocks typically do not have a great impact on fault coverage, but they can entail additional work to make sure that the scan shift process works properly. Gated clocks also increase the risk of skewed clock problems.

When using gated clocks, pay close attention to the check_test reports. Within these reports, Test Compiler will report which nets are interpreted as being clocks. For gated clocks, Test Compiler will select one input only as a clock and consider all other inputs to be gated inputs. Test Compiler DOES NOT always pick the correct input to infer as a clock. Use the “create_test_clock” command to specify the correct clock port.

To disable clock gating during scan shift, use the “set_signal_type test_scan_enable CLK_ENABLE” command. CLK_ENABLE is the name of the signal which gates the clock. This will achieve the highest fault coverage, but requires additional work to specify all gated clocks in the design.

You can also use the set_test_hold command to effect the same change of disabling the gated clock. This is going to be a quicker software constraint, but it will result in lower fault coverage.

SEQUENTIAL GATED CLOCKS: If your design contains sequentially gated clocks, you must add Test Mode logic to bypass these clocks during scan testing. This will reduce the fault coverage.

MULTIPLE CLOCKS: Test Compiler can sometimes support multiple clocks. During check_test, Test Compiler will attempt to infer clock capture groups and allocate FFs to scan chains in the same clock domain. The risk for clock skew greatly increases. The resulting scan chains are not optimal. The goal is for Test Compiler to have full control of all clocks during scan testing. A much better method is to use Test Mode logic to gate a single test clock to all FFs.

Guideline #7: Internal Tri-state Nets

Internal Tri-state nets may lead to bus contention conditions. Should bus contention occur during a scan test, the device will fail test. Test Compiler does not handle internal Tri-stateable nets automatically. Circuit modifications are required to enable only one path on the Tri-state net at a time. While this allows the device to pass test, these circuit modifications may result in blockage of the remaining functional paths on the Tri-state nets. Disabling logic is not completely testable during scan. Fault coverage will be reduced.

If you decide to proceed with internal Tri-state busses, you need to verify that bus contention conditions do not exist during scan shifting. Test Compiler does not detect these conditions!

In addition, it is possible that disabling logic may not always enable the tool to properly generate ATPGs. To check this condition, always execute a “report_test - atpg_conflict” command while in Test Compiler. This command will check to see if any bus floats or bus contentions will prevent Test Compiler from generating a valid test.

In short, the use of internal Tri-state busses wll greatly impact fault coverage and also requires a lot of additional time modifying the design and constraining the test tool. The use of internal Tri-states is considered to be a high risk venture for scan design.

BIDIRECTIONAL PORTS: Test Compiler treats bidirectional ports as internal Tri-state nets, with the output driver and the tester (or simulator) as the two drivers on the net. Use of bidirectional ports as scan data inputs or outputs increases the risk of unsuccessfully implementing scan testing. In these cases, it is necessary to manually control the direction of the bidirectional port during scan, through the use of Test Mode logic. This will reduce the overall fault coverage obtainable.

When generating ATPGs on designs containing bidirectional ports, it is critical that the test_default_bidir _delay command is used. If this variable is not properly set, simulation mismatches or tester failures are likely. The default value for this variable is set to 55.0 ns.

To ensure valid data capture, set test_default_bidir_delay to be greater than the active edge of the clock. If bidirectional modes change from input to output, bus contention may occur at the time of the change, and remain in effect until the input data is released from the bidirectional port. Set test_default_bidir_delay to minimize this possibility. The only way to prevent this condition from happening is to use set atpg_bidirect_output_only to true within dc_shell. This forces the ATPG algorithms to force the bidirectional ports to be in the output mode during parallel cycles. This greatly reduces the achievable fault coverage.

Bidirectional ports CANNOT be used as clock ports.

Bidirectional ports configured as Scan Data In ports are supported only for existing scan designs. Restated, you can use bidirectional ports as Scan Data In ports only if you insert scan automatically on a design without I/O ports. I/O ports can be added later and then read into Test Compiler as a design with existing scan. This is not an automatic process.

Bidirectional ports configured as Scan Data Out ports is supported only for existing scan designs. Restated, you can use bidirectional ports as Scan Data Out ports only if you insert scan automatically on a design without I/O ports. I/O ports can be added later and then read into Test Compiler as a design with existing scan. This is not an automatic process.

OTHER CONSIDERATIONS: Specifying FFs not to be scanned. To specify FFs that should not be scanned (critical paths?), execute a ”set_scan FALSE” command on the FF. This will result in a loss of fault coverage.

Scan Insertion

The recommended methodology for scan insertion is to insert scan structures in one step from the top level of your design. This method is referred to as a top-down scan insertion. Alternatively, you can insert scan structures at the module level and add the appropriate connections as you move up the hierarchy. This method is referred to as bottom-up scan insertion. Bottom-up scan insertion requires a more thorough understanding of Test Compiler, takes more time to implement, and may result in an inefficient selection of scannable cells.

Scan insertion should only be attempted after all Test Compiler check_test warnings and errors have been reviewed and possibly corrected. Errors are catastrophic and prevent any further processing from occurring. Errors MUST be corrected. Warnings impact fault coverage. Warnings can usually be corrected via design modification and software constraints, although at the cost of a slightly reduced fault coverage. This also costs the designer added effort to implement the design modifications and may reduce normal functional operation performance. If warnings are not corrected, it may still be possible to insert scan and generate ATPGs, but the resulting fault coverage will be reduced. The resulting design will typically contain Test Mode logic as a result of correcting errors and warnings found when executing the check_test command.

Unless manually disabled via design modifications, Test Compiler will automatically insert Tri-state disabling logic. It is recommended that the designer handle his/her own disabling logic (Test Mode logic).

Scan chain routing can be done either automatically or it can be explicitly controlled. Test Compiler defaults to a hierarchical, alphanumeric routing order. This can result in an inefficient scan chain. It is recommended that a specific routing order be specified. Explicit routing orders cannot be

implemented on hierarchy. It can be implemented on a module by module basis, then stitched together, but this is a manually intensive effort.

In any event, Synopsys requires that any scan insertion and subsequent ATPG vectors that are generated be completely simulated (validated) at the gate level to insure proper operation. While generating ATPG vectors, Test Compiler uses a zero delay (timing), cycle based model, which forces the gate level simulation requirement. For this reason, all scan vectors should be generated and simulated prior to proceeding to physical place and route. Suggested simulators are Verilog-XL™, ViewSIM™, QuickSIM™, or Synopsys VSS™. Upon generation and simulation of ATPG vectors, you need to consult your Atmel technical design contact to determine compatibility with the target ATE platform prior to place and route.

Multiple scan chains are supported for both automatic insertion and for explicit insertion methods. Scan chain overhead can sometimes be reduced for shift registers. Since a shift register is already a form of a data chain, shift registers can be incorporated into scan chains without requiring the overhead of substituting scannable FFs for each bit of the register. Only the first bit needs to be a scannable FF. For some designs, this can result in a great reduction in the number of sites required for scannable designs. For most designs, it probably does not have a great impact.

On the following pages are some examples with specific commands to be used for inserting scan for various configurations. There are four examples, each is of increasing complexity. These examples can serve as templates for any scan insertion work which you may be performing. The basic circuit is assumed to be an ATL50 gate array (specify the specific library), contains some latches which will be made transparent, contains Test Mode logic and has combinational feedback loops.

CASE 1:

Automatic Insertion Of 1 Scan Chain, ATL60,5V typical Library, Contains Latches, Test Mode logic, and Combinational Feedback Loops

```
/*READ DESIGN*/
link_path = {atl60_5_typ.db, atl60_5_typ_io.db}
target_library = {atl60_5_typ.db, atl60_5_typ_io.db}
verilogout_no_tri = true
verilogout_single_bit = true
read -f verilog asic.v
current_design ASIC

/*CREATE OR DEFINE SCAN CLOCK*/
create_test_clock “SYSCLK” -period 1000 -waveform {450.0 600.0}
/*DEFINE SCAN STYLE*/
set_test_methodology full_scan
set_scan_style multiplexed_flip_flop
/*DEFINE TEST MODE*/
set_test_hold 1 TM

/*ISOLATE CELLS IN FEEDBACK LOOPS*/set_test_isolate OM32muxl0/I1
set_test_isolate OM32muxl1/I1
set_test_isolate OM32muxl2/I1
set_test_isolate OM32muxl3/I1
set_test_isolate OM32muxl4/I1
set_test_isolate OM32muxl5/I1
set_test_isolate OM32muxl6/I1
set_test_isolate OM32muxl7/I1

/*SET LATCHES TO TRANSPARENT MODE*/
set_scan FALSE {“$6I411\\$1I77\\$2I411”,”$6I411\\$1I77\\$2I412",”$6I411\\$1I77\\$2I413",”$6I411\\$1I77\\$2I414",
”$6I411\\$1I77\\$2I415",”$6I411\\$1I77\\$2I416",”$6I411\\$1I77\\$2I417",”$6I411\\$1I77\\$2I418",”$6I411\\$1I77\\$2I419",
”$6I411\\$1I77\\$2I420",”$6I411\\$1I77\\$2I421",”$6I411\\$1I77\\$2I422",”$6I411\\$1I77\\$2I423",”$6I411\\$1I77\\$2I424",
”$6I411\\$1I77\\$2I425",”$6I411\\$1I77\\$2I426",”$6I411\\$1I77\\$2I427",”$6I411\\$1I77\\$2I428",”$6I411\\$1I77\\$2I429",
”$6I411\\$1I77\\$2I430",”$6I411\\$1I77\\$2I431",”$6I411\\$1I77\\$2I432",”$6I411\\$1I77\\$2I433",”$6I411\\$1I77\\$2I434",
”$6I411\\$1I77\\$2I435",”$6I411\\$1I77\\$2I436",”$6I411\\$1I77\\$2I437",”$6I411\\$1I77\\$2I438",”$6I411\\$1I77\\$2I439",
”$6I411\\$1I77\\$2I440",”$6I411\\$1I77\\$2I441",”$6I411\\$1I77\$2I442",”$6I411\\$1I77\\$3I100",”$6I411\\$1I77\\$3I10",
”$6I411\\$1I77\\$3I102",”$6I411\\$1I77\\$3I103",”$6I411\\$1I77\\$3I104",”$6I411\\$1I77\\$4I98",”$6I411\\$1I77\\$4I99“}
-transparent
```

```
/*CHECK DESIGN RULES*/
check_test
/*For single scan chains, the following port definitions work */
/*DEFINE SCAN PORTS*/
set_signal_type test_scan_in SDI
set_signal_type test_scan_out SDO
set_signal_type test_scan_enable TE

/*INSERT SCAN CHAIN AUTOMATICALLY*/
insert_test insert_test.results

/*RECHECK DESIGN RULES AND SAVE DESIGN*/
check_test>check_test_after_insert.results
report_test -scan>scan_chain.rpt

/*PICK YOUR PREFERRED NETLIST FORMAT TO WRITE*/
write -f verilog -out asic_scan_1.v
write -f db -out asic_scan_1.db

quit
```

CASE 2:

Automatic Insertion Of 7 Scan Chains, Maximum Of 200 FFs Per Chain, ATL60, 5V, typical Library, Contains Latches, Test Mode logic, And Combinational Feedback Loops

```
/*READ DESIGN*/
link_path = {atl60_5_typ.db, atl60_5_typ_io.db}
target_library = {atl60_5_typ.db, atl60_5_typ_io.db}
verilogout_no_tri = true
verilogout_single_bit = true
read -f verilog asic.v
current_design ASIC

/*CREATE OR DEFINE SCAN CLOCK*/
create_test_clock “SYSCLK” -period 1000 -waveform {450.0 600.0}

/*DEFINE SCAN STYLE*/
set_test_methodology full_scan
set_scan_style multiplexed_flip_flop

/*DEFINE TEST MODE*/
set_test_hold 1 TM

/*ISOLATE CELLS IN FEEDBACK LOOPS*/
set_test_isolate OM32muxl0/I1
set_test_isolate OM32muxl1/I1
set_test_isolate OM32muxl2/I1
set_test_isolate OM32muxl3/I1
set_test_isolate OM32muxl4/I1
set_test_isolate OM32muxl5/I1
set_test_isolate OM32muxl6/I1
set_test_isolate OM32muxl7/I1

/*SET LATCHES TO TRANSPARENT MODE*/
set_scan FALSE {“$6I411\\$1I77\\$2I411”,”$6I411\\$1I77\\$2I412",”$6I411\\$1I77\\$2I413",”$6I411\\$1I77\\$2I414",
”$6I411\\$1I77\\$2I415",”$6I411\\$1I77\\$2I416",”$6I411\\$1I77\\$2I417",”$6I411\\$1I77\\$2I418",”$6I411\\$1I77\\$2I419",
”$6I411\\$1I77\\$2I420",”$6I411\\$1I77\\$2I421",”$6I411\\$1I77\\$2I422",”$6I411\\$1I77\\$2I423",”$6I411\\$1I77\\$2I424",
”$6I411\\$1I77\\$2I425",”$6I411\\$1I77\\$2I426",”$6I411\\$1I77\\$2I427",”$6I411\\$1I77\\$2I428",”$6I411\\$1I77\\$2I429",
”$6I411\\$1I77\\$2I430",”$6I411\\$1I77\\$2I431",”$6I411\\$1I77\\$2I432",”$6I411\\$1I77\\$2I433",”$6I411\\$1I77\\$2I434",
”$6I411\\$1I77\\$2I435",”$6I411\\$1I77\\$2I436",”$6I411\\$1I77\\$2I437",”$6I411\\$1I77\\$2I438",”$6I411\\$1I77\\$2I439",
”$6I411\\$1I77\\$2I440",”$6I411\\$1I77\\$2I441",”$6I411\\$1I77\$2I442",”$6I411\\$1I77\\$3I100",”$6I411\\$1I77\\$3I10",
”$6I411\\$1I77\\$3I102",”$6I411\\$1I77\\$3I103",”$6I411\\$1I77\\$3I104",”$6I411\\$1I77\\$4I98",”$6I411\\$1I77\\$4I99“}
-transparent
```

```
/*CHECK DESIGN RULES*/
check_test

/*SCAN ENABLE PORT*/
set_signal_type test_scan_enable TE

/*SCAN IN PORTS*/
set_signal_type test_scan_in SDI1   -index 1
set_signal_type test_scan_in SDI2   -index 2
set_signal_type test_scan_in SDI3   -index 3
set_signal_type test_scan_in SDI4   -index 4
set_signal_type test_scan_in SDI5   -index 5
set_signal_type test_scan_in SDI6   -index 6
set_signal_type test_scan_in SDI7   -index 7

/*SCAN OUT PORTS*/
set_signal_type test_scan_out SDO1   -index 1
set_signal_type test_scan_out SDO2   -index 2
set_signal_type test_scan_out SDO3   -index 3
set_signal_type test_scan_out SDO4   -index 4
set_signal_type test_scan_out SDO5   -index 5
set_signal_type test_scan_out SDO6   -index 6
set_signal_type test_scan_out SDO7   -index 7

/*CHECK DESIGN RULES & INSERT TEST*/
check_test>check_test_before_insert.results
insert_test -scan_chains 7 -max_scan_chain_length 200>insert_test.results

/*RECHECK DESIGN RULES AND SAVE DESIGN*/
check_test>check_test_after_insert.results
report_test -scan>scan_chain.rpt

/*PICK YOUR PREFERRED NETLIST FORMAT TO WRITE*/
write -f verilog -out asic_scan_1.v
write -f db -out asic_scan_1.db

quit
```

CASE 3:

7 Scan Chains, Explicit Routing Order, ATL60, 5V, typical Library, Contains Latches, Test Mode logic, And Combinational Feedback Loops

```
/*READ DESIGN*/
link_path = {atl60_5_typ.db, atl60_5_typ_io.db}
target_library = {atl60_5_typ.db, atl60_5_typ_io.db}
verilogout_no_tri = true
verilogout_single_bit = true
read -f verilog asic.v
current_design ASIC

/*CREATE OR DEFINE SCAN CLOCK*/
create_test_clock “SYSCLK” -period 1000 -waveform {450.0 600.0}
/*DEFINE SCAN STYLE*/
set_test_methodology full_scan
set_scan_style multiplexed_flip_flop

/*DEFINE TEST MODE*/
set_test_hold 1 TM

/*ISOLATE CELLS IN FEEDBACK LOOPS*/
set_test_isolate OM32muxl0/I1
set_test_isolate OM32muxl1/I1
set_test_isolate OM32muxl2/I1
set_test_isolate OM32muxl3/I1
set_test_isolate OM32muxl4/I1
set_test_isolate OM32muxl5/I1
set_test_isolate OM32muxl6/I1
set_test_isolate OM32muxl7/I

1/*SET LATCHES TO TRANSPARENT MODE*/
set_scan FALSE {“$6I411\\$1I77\\$2I411”,”$6I411\\$1I77\\$2I412",”$6I411\\$1I77\\$2I413",”$6I411\\$1I77\\$2I414",
”$6I411\\$1I77\\$2I415",”$6I411\\$1I77\\$2I416",”$6I411\\$1I77\\$2I417",”$6I411\\$1I77\\$2I418",”$6I411\\$1I77\\$2I419",
”$6I411\\$1I77\\$2I420",”$6I411\\$1I77\\$2I421",”$6I411\\$1I77\\$2I422",”$6I411\\$1I77\\$2I423",”$6I411\\$1I77\\$2I424",
”$6I411\\$1I77\\$2I425",”$6I411\\$1I77\\$2I426",”$6I411\\$1I77\\$2I427",”$6I411\\$1I77\\$2I428",”$6I411\\$1I77\\$2I429",
”$6I411\\$1I77\\$2I430",”$6I411\\$1I77\\$2I431",”$6I411\\$1I77\\$2I432",”$6I411\\$1I77\\$2I433",”$6I411\\$1I77\\$2I434",
”$6I411\\$1I77\\$2I435",”$6I411\\$1I77\\$2I436",”$6I411\\$1I77\\$2I437",”$6I411\\$1I77\\$2I438",”$6I411\\$1I77\\$2I439",
”$6I411\\$1I77\\$2I440",”$6I411\\$1I77\\$2I441",”$6I411\\$1I77\$2I442",”$6I411\\$1I77\\$3I100",”$6I411\\$1I77\\$3I10",
”$6I411\\$1I77\\$3I102",”$6I411\\$1I77\\$3I103",”$6I411\\$1I77\\$3I104",”$6I411\\$1I77\\$4I98",”$6I411\\$1I77\\$4I99“}
-transparent

/*CHECK DESIGN RULES*/
check_test

/*SCAN ENABLE PORT*/
set_signal_type test_scan_enable TE
```

```
/*SCAN IN PORTS*/
set_signal_type test_scan_in SDI1    -index 1
set_signal_type test_scan_in SDI2    -index 2
set_signal_type test_scan_in SDI3    -index 3
set_signal_type test_scan_in SDI4    -index 4
set_signal_type test_scan_in SDI5    -index 5
set_signal_type test_scan_in SDI6    -index 6
set_signal_type test_scan_in SDI7    -index 7

/*SCAN OUT PORTS*/
set_signal_type test_scan_out SDO1    -index 1
set_signal_type test_scan_out SDO2    -index 2
set_signal_type test_scan_out SDO3    -index 3
set_signal_type test_scan_out SDO4    -index 4
set_signal_type test_scan_out SDO5    -index 5
set_signal_type test_scan_out SDO6    -index 6
set_signal_type test_scan_out SDO7    -index 7

/*CHECK DESIGN RULES & INSERT TEST*/
check_test>check_test_before_insert.results

/SWAP FF"S FOR SCANNABLE VERSIONS*/
insert_test -no_route

/*SPECIFY SCAN CHAIN ROUTING ORDERS*/
set_test_routing_order -scan_chain 1 {sd3,sd2,sd1,sd0}
set_test_routing_order -scan_chain 2 {sd7,sd6,sd5,sd4}
set_test_routing_order -scan_chain 3 {sd11,sd10,sd9,sd8}
set_test_routing_order -scan_chain 4 {sd15,sd14,sd13,sd12}
set_test_routing_order -scan_chain 5 {sd19,sd18,sd17,sd16}
set_test_routing_order -scan_chain 6 {sd23,sd22,sd21,sd20}
set_test_routing_order -scan_chain 7 {sd27,sd26,sd25,sd24}

/*STITCH THE SCAN CHAINS UP*/
insert_test -no_insert
report -cell>cell.report

/*RECHECK DESIGN RULES AND SAVE DESIGN*/
check_test>check_test_after_insert.results
report_test -scan>scan_chain.rpt

/*PICK YOUR PREFERRED NETLIST FORMAT TO WRITE*/
write -f verilog -out asic_scan_1.v
write -f db -out asic_scan_1.db

quit
```

CASE 4:

7 Scan Chains, Explicit Routing Order, Including A 3-Bit Shift Register, ATL60, 5V, typical Library, Contains Latches, Test Mode logic, And Combinational Feedback Loops

```
/*READ DESIGN*/
link_path = {atl60_5_typ.db, atl60_5_typ_io.db}
target_library = {atl60_5_typ.db, atl60_5_typ_io.db}
verilogout_no_tri = true
verilogout_single_bit = true
read -f verilog asic.v
current_design ASIC

/*CREATE OR DEFINE SCAN CLOCK*/
create_test_clock “SYSCLK” -period 1000 -waveform {450.0 600.0}

/*DEFINE SCAN STYLE*/
set_testmethodology full_scan
set_scan_style multiplexed_flip_flop

/*DEFINE TEST MODE*/
set_test_hold 1 TM

/*ISOLATE CELLS IN FEEDBACK LOOPS*/
set_test_isolate OM32muxl0/I1
set_test_isolate OM32muxl1/I1
set_test_isolate OM32muxl2/I1
set_test_isolate OM32muxl3/I1
set_test_isolate OM32muxl4/I1
set_test_isolate OM32muxl5/I1
set_test_isolate OM32muxl6/I1
set_test_isolate OM32muxl7/I1

/*SET LATCHES TO TRANSPARENT MODE*/
set_scan FALSE {“$6I411\\$1I77\\$2I411”,”$6I411\\$1I77\\$2I412”,”$6I411\\$1I77\\$2I413”,”$6I411\\$1I77\\$2I414",
”$6I411\\$1I77\\$2I415”,”$6I411\\$1I77\\$2I416”,”$6I411\\$1I77\\$2I417”,”$6I411\\$1I77\\$2I418”,”$6I411\\$1I77\\$2I419",
”$6I411\\$1I77\\$2I420”,”$6I411\\$1I77\\$2I421”,”$6I411\\$1I77\\$2I422”,”$6I411\\$1I77\\$2I423”,”$6I411\\$1I77\\$2I424",
”$6I411\\$1I77\\$2I425”,”$6I411\\$1I77\\$2I426”,”$6I411\\$1I77\\$2I427”,”$6I411\\$1I77\\$2I428”,”$6I411\\$1I77\\$2I429",
”$6I411\\$1I77\\$2I430”,”$6I411\\$1I77\\$2I431”,”$6I411\\$1I77\\$2I432”,”$6I411\\$1I77\\$2I433”,”$6I411\\$1I77\\$2I434",
”$6I411\\$1I77\\$2I435”,”$6I411\\$1I77\\$2I436”,”$6I411\\$1I77\\$2I437”,”$6I411\\$1I77\\$2I438”,”$6I411\\$1I77\\$2I439",
”$6I411\\$1I77\\$2I440”,”$6I411\\$1I77\\$2I441”,”$6I411\\$1I77\$2I442”,”$6I411\\$1I77\\$3I100”,”$6I411\\$1I77\\$3I10",
”$6I411\\$1I77\\$3I102”,”$6I411\\$1I77\\$3I103”,”$6I411\\$1I77\\$3I104”,”$6I411\\$1I77\\$4I98”,”$6I411\\$1I77\\$4I99“}
-transparent

/*CHECK DESIGN RULES*/
check_test

/*SCAN ENABLE PORT*/
set_signal_type test_scan_enable TE

/*SCAN IN PORTS*/
set_signal_type test_scan_in SDI1   -index 1
set_signal_type test_scan_in SDI2   -index 2
set_signal_type test_scan_in SDI3   -index 3
set_signal_type test_scan_in SDI4   -index 4
set_signal_type test_scan_in SDI5   -index 5
```

```
set_signal_type test_scan_in SDI6    -index 6
set_signal_type test_scan_in SDI7    -index 7

/*SCAN OUT PORTS*/
set_signal_type test_scan_out SDO1    -index 1
set_signal_type test_scan_out SDO2    -index 2
set_signal_type test_scan_out SDO3    -index 3
set_signal_type test_scan_out SDO4    -index 4
set_signal_type test_scan_out SDO5    -index 5
set_signal_type test_scan_out SDO6    -index 6
set_signal_type test_scan_out SDO7    -index 7

/*CHECK DESIGN RULES & INSERT TEST*/
check_test>check_test_before_insert.results

/SWAP FF"S FOR SCANNABLE VERSIONS*/
insert_test -no_route

/*SPECIFY SCAN CHAIN ROUTING ORDERS*/
set_test_routing_order -scan_chain 1 {sd3,sd2,sd1,sd0}
set_test_routing_order -scan_chain 2 {sd7,sd6,sd5,sd4}
set_test_routing_order -scan_chain 3 {sd11,sd10,sd9,sd8}
set_test_routing_order -scan_chain 4 {sd15,sd14,sd13,sd12}
set_test_routing_order -scan_chain 5 {sd19,sd18,sd17,sd16}
set_test_routing_order -scan_chain 6 {sd23,sd22,sd21,sd20}
set_test_routing_order -scan_chain 7 {sd27,sd26,sd25,sd24}

/*STITCH THE SCAN CHAINS UP*/
insert_test -no_insert

report -cell cell.report

group {U2000,U2001,U2002} -design_name shift_register -cell_name sr1
current_design shift_register
set_dont_use {at24k5v/DFFC}
compile -incr
current_design ASIC
ungroup -all -flatten

/*RECHECK DESIGN RULES AND SAVE DESIGN*/
check_test>check_test_after_insert.results
report_test -scan>scan_chain.rpt

/*PICK YOUR PREFERRED NETLIST FORMAT TO WRITE*/
write -f verilog -out asic_scan_1.v
write -f db -out asic_scan_1.db

quit
```

ATPG Vector Generation

The general flow for generating ATPGs is:

- insert pads
- check_test (initializes test protocol)
- create_test_patterns
- write_test

If you have bidirectional ports in your design, you will have to insert the I/O after scan insertion and prior to generating ATPGs. If your design does not have bidirectional ports you may have inserted scan with the I/O ports already inserted and you are now ready to proceed with ATPGs and may bypass the next section.

Inserting I/O

While it is possible to allow Test Compiler to insert pads via the insert_pads command, Synopsys recommends that designers manually insert their pads to avoid possible conflicts.

Fault Coverage

If you need to specify certain faults not to be considered for ATPGs, use the “set_test_dont_fault” command to specify which faults to exclude from the fault list. Apply the set_test_dont_fault” command prior to executing the create_test_patterns command. To check the results of the create_test_patterns operation, you may create a variety of reports using the report_test command. Switches are available for assertions, coverage, dont_fault attributes, faults, and fault classes.

Faults are classified into 5 catagories:

Detected	A pattern may be generated to control and observe the fault
Abandoned	Test Compiler reached its CPU limit and has not found a pattern
Tied	The node is tied high/low, causing a controllability problem
Redundant	The design contains redundant logic and is untestable
Untestable	No pattern can be generated to control and observe the fault.

Fault Coverage is calculated as:

$$\frac{\text{detected\_faults}}{\text{detected\_faults} + \text{abandoned\_faults} + \text{untestable\_faults}} \times 100\%$$

Use of the set_test_dont_fault command affects the fault coverage calculations by eliminating the fault(s) from the fault list (reduces the denominator in the above calculation).

Generate ATPGs

Once the design has had scan inserted and has its I/O instantiated (the design should be flattened), ATPGs may be generated. To do this, you basically repeat the insert process to properly constrain Test Compilers fault simulator. The main difference is that you specify that the design has existing scan chains implemented. The same four example cases used for scan insertion are continued for ATPG vector generation. The optional timing parameters used are specific to Sentry testers. If other testers are targeted, delete the optional timing parameters section.

CASE 1:

Automatic Insertion of 1 Scan Chain, ATL60, 5V, typical Library, Contains Latches, Test Mode logic, and Combinational Feedback Loops

```
/*READ DESIGN*/
link_path = {atl60_5_typ.db, atl60_5_typ_io.db}
target_library = {atl60_5_typ.db, atl60_5_typ_io.db}
verilogout_no_tri = true
verilogout_single_bit = true
read -f verilog asic.v
current_design ASIC

/*SET TIMING PARAMETERS (OPTIONAL)*/
test_default_bidir_delay = 0
test_default_delay = 0
test_default_strobe = 950.0
test_default_period = 1000.0

/*CREATE OR DEFINE SCAN CLOCK*/
create_test_clock “SYSCLK” -period 1000 -waveform {450.0 600.0}

/*DEFINE TEST MODE*/
set_test_hold 1 TM

/*ISOLATE CELLS IN FEEDBACK LOOPS*/
set_test_isolate OM32muxl0/I1
set_test_isolate OM2muxI1/I1
set_test_isolate OM32muxl2/I1
set_test_isolate OM32muxl3/I1
set_test_isolate OM32muxl4/I1
set_test_isolate OM32muxl5/I1
set_test_isolate OM32muxl6/I1
set_test_isolate OM32muxl7/I1

/*SET LATCHES TO TRANSPARENT MODE*/
set_scan FALSE {“$6I411\\$1I77\\$2I411”,”$6I411\\$1I77\\$2I412",”$6I411\\$1I77\\$2I413",”$6I411\\$1I77\\$2I414",
”$6I411\\$1I77\\$2I415",”$6I411\\$1I77\\$2I416",”$6I411\\$1I77\\$2I417",”$6I411\\$1I77\\$2I418",”$6I411\\$1I77\\$2I419",
”$6I411\\$1I77\\$2I420",”$6I411\\$1I77\\$2I421",”$6I411\\$1I77\\$2I422",”$6I411\\$1I77\\$2I423",”$6I411\\$1I77\\$2I424",
”$6I411\\$1I77\\$2I425",”$6I411\\$1I77\\$2I426",”$6I411\\$1I77\\$2I427",”$6I411\\$1I77\\$2I428",”$6I411\\$1I77\\$2I429",
”$6I411\\$1I77\\$2I430",”$6I411\\$1I77\\$2I431",”$6I411\\$1I77\\$2I432",”$6I411\\$1I77\\$2I433",”$6I411\\$1I77\\$2I434",
”$6I411\\$1I77\\$2I435",”$6I411\\$1I77\\$2I436",”$6I411\\$1I77\\$2I437",”$6I411\\$1I77\\$2I438",”$6I411\\$1I77\\$2I439",
”$6I411\\$1I77\\$2I440",”$6I411\\$1I77\\$2I441",”$6I411\\$1I77\$2I442",”$6I411\\$1I77\\$3I100",”$6I411\\$1I77\\$3I10",
”$6I411\\$1I77\\$3I102",”$6I411\\$1I77\\$3I103",”$6I411\\$1I77\\$3I104",”$6I411\\$1I77\\$4I98",”$6I411\\$1I77\\$4I99“}
-transparent

/*DEFINE SCAN PORTS*/
set_signal_type test_scan_in SDI
set_signal_type test_scan_out SDO
set_signal_type test_scan_enable TE

/*DEFINE SCAN STYLE WITH EXISTING SCAN*/
set_test_methodology full_scan -existing_scan
set_scan_style multiplexed_flip_flop
/*REPROGRAM & INITIALIZE THE FAULT SIMULATOR & INFER THE SCAN PATHS*/
check_test
```

```
/*REPORT THE SCAN CHAINS*/
report_test -scan scan_chain.report

/*CREATE ATPGs*/
create_test_patterns -output asic_scan.vdb -compaction_effort med -max_cpu_per_fault 60
-backtrack_effort med -random_pattern_failure_limit 64 -sample 100

/*SAVE EVERYTHING*/
write -f verilog -output asic_scan.v
write -f db -output asic_scan.db

write_test_input_dont_care_value = 0
write_test_max_cycles = 2200

write_test -format verilog -input asic_scan.vdb -output scan_vectors -period 1000 -delay 0 -bidir_delay 0 -strobe 950

/*CREATE SOME REPORTS*/
report_test -faults -class untested>untested_faults.report
report_test -faults -class tied>tied_faults.report

quit
```

CASE 2:

Automatic Insertion of 7 Scan Chains, ATL60, 5V, typical Library, Contains Latches, Test Mode logic, and Combinational Feedback Loops

```
/*READ DESIGN*/
link_path = {atl60_5_typ.db, atl60_5_typ_io.db}
target_library = {atl60_5_typ.db, atl60_5_typ_io.db}
verilogout_no_tri = true
verilogout_single_bit = true
read -f verilog asic.v
current_design ASIC

/*SET TIMING PARAMETERS (OPTIONAL)*/
test_default_bidir_delay = 0
test_default_delay = 0
test_default_strobe = 950.0
test_default_period = 1000.0

/*CREATE OR DEFINE SCAN CLOCK*/
create_test_clock “SYSCLK” -period 1000 -waveform {450.0 600.0}

/*DEFINE TEST MODE*/
set_test_hold 1 TM

/*ISOLATE CELLS IN FEEDBACK LOOPS*/
set_test_isolate OM32muxl0/I1
set_test_isolate OM32muxl1/I1
set_test_isolate OM32muxl2/I1
set_test_isolate OM32muxl3/I1
set_test_isolate OM32muxl4/I1
set_test_isolate OM32muxl5/I1
set_test_isolate OM32muxl6/I1
set_test_isolate OM32muxl7/I1

/*SET LATCHES TO TRANSPARENT MODE*/
set_scan FALSE {“$6I411\\$1I77\\$2I411”,”$6I411\\$1I77\\$2I412",”$6I411\\$1I77\\$2I413",”$6I411\\$1I77\\$2I414",
”$6I411\\$1I77\\$2I415",”$6I411\\$1I77\\$2I416",”$6I411\\$1I77\\$2I417",”$6I411\\$1I77\\$2I418",”$6I411\\$1I77\\$2I419",
”$6I411\\$1I77\\$2I420",”$6I411\\$1I77\\$2I421",”$6I411\\$1I77\\$2I422",”$6I411\\$1I77\\$2I423",”$6I411\\$1I77\\$2I424",
”$6I411\\$1I77\\$2I425",”$6I411\\$1I77\\$2I426",”$6I411\\$1I77\\$2I427",”$6I411\\$1I77\\$2I428",”$6I411\\$1I77\\$2I429",
”$6I411\\$1I77\\$2I430",”$6I411\\$1I77\\$2I431",”$6I411\\$1I77\\$2I432",”$6I411\\$1I77\\$2I433",”$6I411\\$1I77\\$2I434",
”$6I411\\$1I77\\$2I435",”$6I411\\$1I77\\$2I436",”$6I411\\$1I77\\$2I437",”$6I411\\$1I77\\$2I438",”$6I411\\$1I77\\$2I439",
”$6I411\\$1I77\\$2I440",”$6I411\\$1I77\\$2I441",”$6I411\\$1I77\$2I442",”$6I411\\$1I77\\$3I100",”$6I411\\$1I77\\$3I10",
”$6I411\\$1I77\\$3I102",”$6I411\\$1I77\\$3I103",”$6I411\\$1I77\\$3I104",”$6I411\\$1I77\\$4I98",”$6I411\\$1I77\\$4I99“}
-transparent

/*SCAN ENABLE PORT*/
set_signal_type test_scan_enable TE

/*SCAN IN PORTS*/
set_signal_type test_scan_in SDI1   -index 1
set_signal_type test_scan_in SDI2   -index 2
set_signal_type test_scan_in SDI3   -index 3
set_signal_type test_scan_in SDI4   -index 4
set_signal_type test_scan_in SDI5   -index 5
set_signal_type test_scan_in SDI6   -index 6
set_signal_type test_scan_in SDI7   -index 7
```

```
/*SCAN OUT PORTS*/
set_signal_type test_scan_out SDO1    -index 1
set_signal_type test_scan_out SDO2    -index 2
set_signal_type test_scan_out SDO3    -index 3
set_signal_type test_scan_out SDO4    -index 4
set_signal_type test_scan_out SDO5    -index 5
set_signal_type test_scan_out SDO6    -index 6
set_signal_type test_scan_out SDO7    -index 7

/*DEFINE SCAN STYLE WITH EXISTING SCAN*/
set_test_methodology full_scan -existing_scan
set_scan_style multiplexed_flip_flop

/*REPROGRAM & INITIALIZE THE FAULT SIMULATOR & INFER THE SCAN PATHS*/
check_test

/*REPORT THE SCAN CHAINS*/
report_test -scan>scan_chain.report

/*CREATE ATPGs*/
create_test_patterns -output asic_scan.vdb -compaction_effort med -max_cpu_per_fault 60
-backtrack_effort med -random_pattern_failure_limit 64 -sample 100

/*SAVE EVERYTHING*/
write -f verilog -output asic_scan.v
write -f db -output asic_scan.db

write_test_input_dont_care_value = 0
write_test_max_cycles = 2200

write_test -format verilog -input asic_scan.vdb -output scan_vectors -period 1000 -delay 0 -bidir_delay 0 -strobe 950

/*CREATE SOME REPORTS*/
report_test -faults -class untested>untested_faults.report
report_test -faults -class tied>tied_faults.report

quit
```

CASE 3:

7 Scan Chains, Explicit Routing Order, ATL60, 5V, typical Library, Contains Latches, Test Mode logic, and Combinational Feedback Loops

```
/*READ DESIGN*/
link_path = {atl60_5_typ.db, atl60_5_typ_io.db}
target_library = {atl60_5_typ.db, atl60_5_typ_io.db}
verilogout_no_tri = true
verilogout_single_bit = true
read -f verilog asic.v
current_design ASIC

/*SET TIMING PARAMETERS (OPTIONAL)*/
test_default_bidir_delay = 0
test_default_delay = 0
test_default_strobe = 950.0
test_default_period = 1000.0

/*CREATE OR DEFINE SCAN CLOCK*/
create_test_clock “SYSCLK” -period 1000 -waveform {450.0 600.0}

/*DEFINE TEST MODE*/
set_test_hold 1 TM

/*ISOLATE CELLS IN FEEDBACK LOOPS*/
set_test_isolate OM32muxl0/I1
set_test_isolate OM32muxl1/I1
set_test_isolate OM32muxl2/I1
set_test_isolate OM32muxl3/I1
set_test_isolate OM32muxl4/I1
set_test_isolate OM32muxl5/I1
set_test_isolate OM32muxl6/I1
set_test_isolate OM32muxl7/I1

/*SET LATCHES TO TRANSPARENT MODE*/
set_scan FALSE {“$6I411\\$1I77\\$2I411”,”$6I411\\$1I77\\$2I412",”$6I411\\$1I77\\$2I413",”$6I411\\$1I77\\$2I414",
”$6I411\\$1I77\\$2I415",”$6I411\\$1I77\\$2I416",”$6I411\\$1I77\\$2I417",”$6I411\\$1I77\\$2I418",”$6I411\\$1I77\\$2I419",
”$6I411\\$1I77\\$2I420",”$6I411\\$1I77\\$2I421",”$6I411\\$1I77\\$2I422",”$6I411\\$1I77\\$2I423",”$6I411\\$1I77\\$2I424",
”$6I411\\$1I77\\$2I425",”$6I411\\$1I77\\$2I426",”$6I411\\$1I77\\$2I427",”$6I411\\$1I77\\$2I428",”$6I411\\$1I77\\$2I429",
”$6I411\\$1I77\\$2I430",”$6I411\\$1I77\\$2I431",”$6I411\\$1I77\\$2I432",”$6I411\\$1I77\\$2I433",”$6I411\\$1I77\\$2I434",
”$6I411\\$1I77\\$2I435",”$6I411\\$1I77\\$2I436",”$6I411\\$1I77\\$2I437",”$6I411\\$1I77\\$2I438",”$6I411\\$1I77\\$2I439",
”$6I411\\$1I77\\$2I440",”$6I411\\$1I77\\$2I441",”$6I411\\$1I77\$2I442",”$6I411\\$1I77\\$3I100",”$6I411\\$1I77\\$3I10",
”$6I411\\$1I77\\$3I102",”$6I411\\$1I77\\$3I103",”$6I411\\$1I77\\$3I104",”$6I411\\$1I77\\$4I98",”$6I411\\$1I77\\$4I99“}
-transparent

/*SCAN ENABLE PORT*/
set_signal_type test_scan_enable TE
```

```
/*SCAN IN PORTS*/
set_signal_type test_scan_in SDI1   -index1
set_signal_type test_scan_in SDI2   -index 2
set_signal_type test_scan_in SDI3   -index 3
set_signal_type test_scan_in SDI4   -index 4
set_signal_type test_scan_in SDI5   -index 5
set_signal_type test_scan_in SDI6   -index 6
set_signal_type test_scan_in SDI7   -index 7

/*SCAN OUT PORTS*/
set_signal_type test_scan_out SDO1   -index 1
set_signal_type test_scan_out SDO2   -index 2
set_signal_type test_scan_out SDO3   -index 3
set_signal_type test_scan_out SDO4   -index 4
set_signal_type test_scan_out SDO5   -index 5
set_signal_type test_scan_out SDO6   -index 6
set_signal_type test_scan_out SDO7   -index 7

/*DEFINE SCAN STYLE WITH EXISTING SCAN*/
set_test_methodology full_scan -existing_scan
set_scan_style multiplexed_flip_flop

/*REPROGRAM & INITIALIZE THE FAULT SIMULATOR & INFER THE SCAN PATHS*/
check_test

/*REPORT THE SCAN CHAINS*/
report_test -scan>scan_chain.report

/*CREATE ATPGs*/
create_test_patterns -output asic_scan.vdb -compaction_effort med -max_cpu_per_fault 60
-backtrack_effort med -random_pattern_failure_limit 64 -sample 100

/*SAVE EVERYTHING*/
write -f verilog -output asic_scan.v
write -f db -output asic_scan.db

write_test_input_dont_care_value = 0
write_test_max_cycles = 2200

write_test -format verilog -input asic_scan.vdb -output scan_vectors -period 1000 -delay 0 -bidir_delay 0 -strobe 950

/*CREATE SOME REPORTS*/
report_test -faults -class untested>untested_faults.report
report_test -faults -class tied>tied_faults.report

quit
```

CASE 4:

7 Scan Chains, Explicit Routing Order, Including a 3-bit Shift Register, ATL60, 5V, typical Library, Contains Latches, Test Mode logic, and Combinational Feedback Loops

```
/*READ DESIGN*/
link_path = {atl60_5_typ.db, atl60_5_typ_io.db}
target_library = {atl60_5_typ.db, atl60_5_typ_io.db}
verilogout_no_tri = true
verilogout_single_bit = true
read -f verilog asic.v
current_design ASIC

/*SET TIMING PARAMETERS (OPTIONAL)*/
test_default_bidir_delay = 0
test_default_delay = 0
test_default_strobe = 950.0
test_default_period = 1000.0

/*CREATE OR DEFINE SCAN CLOCK*/
create_test_clock “SYSCLK” -period 1000 -waveform {450.0 600.0}

/*DEFINE TEST MODE*/
set_test_hold 1 TM

/*ISOLATE CELLS IN FEEDBACK LOOPS*/
set_test_isolate OM32muxl0/I1
set_test_isolate OM32muxl1/I1
set_test_isolate OM32muxl2/I1
set_test_isolate OM32muxl3/I1
set_test_isolate OM32muxl4/I1
set_test_isolate OM32muxl5/I1
set_test_isolate OM32muxl6/I1
set_test_isolate OM32muxl7/I1

/*SET LATCHES TO TRANSPARENT MODE*/
set_scan FALSE {“$6I411\\$1I77\\$2I411”,”$6I411\\$1I77\\$2I412",”$6I411\\$1I77\\$2I413",”$6I411\\$1I77\\$2I414",
”$6I411\\$1I77\\$2I415",”$6I411\\$1I77\\$2I416",”$6I411\\$1I77\\$2I417",”$6I411\\$1I77\\$2I418",”$6I411\\$1I77\\$2I419",
”$6I411\\$1I77\\$2I420",”$6I411\\$1I77\\$2I421",”$6I411\\$1I77\\$2I422",”$6I411\\$1I77\\$2I423",”$6I411\\$1I77\\$2I424",
”$6I411\\$1I77\\$2I425",”$6I411\\$1I77\\$2I426",”$6I411\\$1I77\\$2I427",”$6I411\\$1I77\\$2I428",”$6I411\\$1I77\\$2I429",
”$6I411\\$1I77\\$2I430",”$6I411\\$1I77\\$2I431",”$6I411\\$1I77\\$2I432",”$6I411\\$1I77\\$2I433",”$6I411\\$1I77\\$2I434",
”$6I411\\$1I77\\$2I435",”$6I411\\$1I77\\$2I436",”$6I411\\$1I77\\$2I437",”$6I411\\$1I77\\$2I438",”$6I411\\$1I77\\$2I439",
”$6I411\\$1I77\\$2I440",”$6I411\\$1I77\\$2I441",”$6I411\\$1I77\$2I442",”$6I411\\$1I77\\$3I100",”$6I411\\$1I77\\$3I10",
”$6I411\\$1I77\\$3I102",”$6I411\\$1I77\\$3I103",”$6I411\\$1I77\\$3I104",”$6I411\\$1I77\\$4I98",”$6I411\\$1I77\\$4I99“}
-transparent

/*SCAN ENABLE PORT*/
set_signal_type test_scan_enable TE
```

```
/*SCAN IN PORTS*/
set_signal_type test_scan_in SDI1   -index 1
set_signal_type test_scan_in SDI2   -index 2
set_signal_type test_scan_in SDI3   -index 3
set_signal_type test_scan_in SDI4   -index 4
set_signal_type test_scan_in SDI5   -index 5
set_signal_type test_scan_in SDI6   -index 6
set_signal_type test_scan_in SDI7   -index 7

/*SCAN OUT PORTS*/
set_signal_type test_scan_out SDO1   -index 1
set_signal_type test_scan_out SDO2   -index 2
set_signal_type test_scan_out SDO3   -index 3
set_signal_type test_scan_out SDO4   -index 4
set_signal_type test_scan_out SDO5   -index 5
set_signal_type test_scan_out SDO6   -index 6
set_signal_type test_scan_out SDO7   -index 7

/*DEFINE SCAN STYLE WITH EXISTING SCAN*/
set_test_methodology full_scan -existing_scan
set_scan_style multiplexed_flip_flop

/*REPROGRAM & INITIALIZE THE FAULT SIMULATOR & INFER THE SCAN PATHS*/
check_test

/*REPORT THE SCAN CHAINS*/
report_test -scan>scan_chain.report

/*CREATE ATPGs*/
create_test_patterns -output asic_scan.vdb -compaction_effort med -max_cpu_per_fault 60
-backtrack_effort med -random_pattern_failure_limit 64 -sample 100

/*SAVE EVERYTHING*/
write -f verilog -output asic_scan.v
write -f db -output asic_scan.db

write_test_input_dont_care_value = 0
write_test_max_cycles = 2200

write_test -format verilog -input asic_scan.vdb -output scan_vectors -period 1000 -delay 0 -bidir_delay 0 -strobe 950

/*CREATE SOME REPORTS*/
report_test -faults -class untested>untested_faults.report
report_test -faults -class tied>tied_faults.report

quit
```

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

Europe
Atmel U.K., Ltd.
Coliseum Business Centre
Riverside Way
Camberley, Surrey GU15 3YL
England
TEL (44) 1276-686677
FAX (44) 1276-686697

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road
Tsimshatsui East
Kowloon, Hong Kong
TEL (852) 27219778
FAX (852) 27221369

Japan
Atmel Japan K.K.
Tonetsu Shinkawa Bldg., 9F
1-24-8 Shinkawa
Chuo-Ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Atmel Colorado Springs
1150 E. Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL (719) 576-3300
FAX (719) 540-1759

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex, France
TEL (33) 4 42 53 60 00
FAX (33) 4 42 53 60 01

Fax-on-Demand
North America:
1-(800) 292-8635

International:
1-(408) 441-0732

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

BBS
1-(408) 436-4309

0586A-9/95/5m