

# **Design-for-Test: Scan and ATPG Training**

Student Workbook

December 2003



Copyright © Mentor Graphics Corporation 2003. All rights reserved. This document contains information that is proprietary to Mentor Graphics Corporation and may not be duplicated in whole or in part in any form without written consent from Mentor Graphics. In accepting this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use of this information.

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS CORPORATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

**RESTRICTED RIGHTS LEGEND 03/97**

U.S. Government Restricted Rights. The SOFTWARE and documentation have been developed entirely at private expense and are commercial computer software provided with restricted rights. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement provided with the software pursuant to DFARS 227.7202-3(a) or as set forth in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, as applicable.

Contractor/manufacturer is:  
Mentor Graphics Corporation  
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.

A complete list of trademark names appears in a separate "[Trademark Information](#)" document.

This is an unpublished work of Mentor Graphics Corporation.

# Trademark Information

## Mentor Graphics Trademarks

The following are trademarks of Mentor Graphics Corporation:

3D Design, ABIST, Arithmetic BIST, Accelerated Technology®, AccuPARTner, AccuParts, AccuSim®, ADEPT, ADVance, ADVance MS, ADVance RFIC, AMPLE, Analog Analyst, Analog Station, Ares®, ARTgrid, ArtRouter, ARTshape, ASICPlan, ASICVector Interfaces, Aspire, AuthExpress, AutoActive®, AutoCells, AutoDissolve, AutoFilter, AutoFlow, AutoLib, AutoLinear, AutoLink, AutoLogic, AutoLogic BLOCKS, AutoLogic FPGA, AutoLogic VHDL®, AutomotiveLib, AutoPAR®, AutoTherm®, AutoTherm Duo, AutoTherm MCM, AutoView, Autowire Station, AXEL, AXEL Symbol Genie, BISTAchitect, BLAST, Blaze, BlazeRouter, Board Station Consumer, Board Architect, Board Designer, Board Layout, Board Link, Board Process Library, BoardSim®, Board Station®, BOLD Administator, BOLD Browser, BOLD Composer, BSDArchitect, BSPBuilder, Buy on Demand, Cable Analyzer, Cable Station, CAECO Designer, CAEFORM, Calibre®, Calibre DRC, Calibre DRC-H, Calibre DESIGNrev, Calibre CB, Calibre FRACTUREi, Calibre FRACTUREh, Calibre FRACTUREm, Calibre FRACTUREt, Calibre FRACTUREk, Calibre Flex MT, Calibre LITHOview, Calibre OPCbar, Calibre OPCpro, Calibre ORC, Calibre PRINTimage, Calibre PSMgate, Calibre PSMcheck, Calibre TDoc, Calibre WORKbench, Calibre RVE, Calibre MGC, Calibre Interactive, Calibre MDPview, Calibre xRC, Capital, Capital Analysis, Capital Archive, Capital Bridges, Capital Documents, Capital H®, Capital H the complete desktop engineer®, Capital Harness, Capital Harness Systems, Capital Insight, Capital Integration, Capital Manager, Capital Manufacture, Capital Support, Capital Systems, Capture Station®, Cell Builder, Cell Station®, CellFloor, CellGraph, CellPlace, CellPower, CellRoute, Centrity, CEOC, ChaseX, CheckMate, CHEOS, Chip Station®, ChipGraph, ChipLister, Circuit PathFinder, Co-Verification Environment, COLsim, CodeLab, CommLib, CommLib BMC, Concurrent Design Environment, Connectivity Dataport, Continuum, Continuum Power Analyst, CoreAlliance, CoreBIST, Core Builder, Core Factory, CTIntegrator, DataCentric Model, DataFusion, Datapath, Data Solvent, dBUG, Debug Detective, DC Analyzer, DeltaCore®, Design Architect®, Design Architect@-IC, Design Architect Elite, Design Capture, Design Exchange, Design Manager, Design Station®, DesignBook®, Design View, DesktopASIC, Destination PCB®, Destiny RE, DFTAdvisor, DFTArchitect, DFTInsight, DMS Xchange, DxAnalog, DxDataBook, DxDesigner, DxLibraryStudio, DxParts, DxPDF, DxViewOnly, DxVariantManager, Direct System Verification, DSV, Documentation Station, DSS (Decision Support System), DxAnalog, DxDataManager, DxDesigner, DxEnterprise for Agile, DMatrix, DxViewDraw, E3Lcable, EDT, Eldo, EldoNet, ePartners, ePlanner®, eProduct Designer, eProduct Services®, Empowering Solutions, Engineer's Desktop, EngineerView, Enterprise Librarian, ENRead, ENWrite, ESim, Exemplar, ExemplarLogic, Expedition, Explorer CAECO Layout, Explorer CheckMate, Explorer Datapath, Explorer Lsim, Explorer Lsim-C, Explorer Lsim-S, Explorer Ltime, Explorer Schematic, Explorer VHDLsim, ExpressIO, FabLink, Falcon®, Falcon Framework®, FastScan, FIRE, First-Pass Design Success, First-Pass Success, FlexSim, FlexTest, FDL (Flow Definition Language), FlowTabs, FlowXpert, FORMA, FormalPro, FPGA Advantage®, FPGAdvisor, FPGA BoardLink, FPGA Builder, FGPGASim, FPGA Station®, FrameConnect, Fusion, Galileo®, Gate Station®, GateGraph, GatePlace, GateRoute, Gemini®, GDT®, GDT Core, GDT Designer, GDT Developer, GENIE, GenWare, Geom Genie, HDL2Graphics, HDL Architect, HDL Architect Station, HDL Author, HDL Designer, HDL Designer Series, HDL Detective, HDL Inventor, HDL Link, HDL Pilot, HDL Processor, HDLSim, HDLWrite, Hierarchical Injection, Hierarchy Injection, HIC rules, Hardware Modeling Library, HotPlot®, Hybrid Designer, Hybrid Station®, HyperLynx®, HyperSuite®, IC Design Station, IC Designer, IC Layout Station, IC Station®, Streamview, ICBasic, ICblocks, ICcheck, ICcompact, ICdevice, ICextract, ICGen, ICgraph, ICLink, ICLister, ICplan, ICRT Controller, Compiler, ICrules, ICTrace, ICVerify, ICView, ICX, ICX Active, ICX Plan, ICX Pro, ICX Sentry, ICX Tau, ICX Verify, ICX Vision, ICX Custom Model, ICX Custom Modeling, ICX Project Modeling, ICX Standard Library, IDEA Series, Idea Station®, IKOS®, In All The Right Places, INFORM®, IFX, Inexia, Innovate PCB, Innoveda, Integrated Product Development, Integra Station, Integration Tool Kit, IT, INTELLITEST®, Interactive LAYOUT, Interconnect Table, Interface-Based Design, IB, Inventra, Inventra IPX, Inventra Soft Cores, IP Engine, IP Evaluation Kit, IP Factory, IP-PCB, IP QuickUse, IPSim, IS Analyzer, IS Floorplanner, IS MultiBoard, IS Optimizer, IS Synthesizer, iSolve, IV'locity, Language Neutral Licensing, Latium®, LAYOUT, LNL, LBIST, LBISTArchitect, Lc, Lcore, Leaf Cell Toolkit, Led, LED Layout, Leonardo®, LeonardoInsight, LeonardoSpectrum, Librarian, Library Builder, LineSim®, Logic Analyzer on a Chip, Logic Builder, Logical Cable, LogicLib, logio, Lsim, Lsim DSM, Lsim Gate, LsimNet, Lsim Power Analyst, Lsim Review, Lsim Switch, Lsim XL, Mach PA, Mach TA, Manufacture View, Manufacturing Advisor, Manufacturing Cable, MaskCompose, MaskPE®, MBIST, MBISTArchitecture, MBIST Full-Speed, MBIST Flex, MBIST In-Place, MBIST Manager, MCM Designer, MCM Station®, MDV, MegaFunction, Memory Builder, Memory Builder Conductor, Memory Builder Mozart, Memory Designer, Memory Model Builder, Mentor®, Mentor Graphics®, MicroPlan, MicroRoute, Microtec®, Mixed-Signal Pro, ModelEditor, ModelSim®, ModelSim LN, ModelSim VHDL, ModelSim VLOG, ModelSim SE, ModelStation®, Model Technology, Model Viewer, ModelViewerPlus, MODGEN, Monet®, MsLab, MsView, MS Analyzer, MS Architect, MS-Express, MSIMON, Nanokernal®, NetCheck, NETED, Nucleus, Opsim, OutNet, P&RIntegrator, PACKAGE, PADS®, PARADE, ParallelRoute-Autocells, ParallelRoute-MicroRoute, Parts SpecialList, PathLink, PCB-Gen, PCB-Generator, PCB IGES, PCB Mechanical Interface, PDLsim, Personal Learning Program, Physical Cable, Physical Test Manager: SITE, PLA Lcompiler, Platform Express, PX, PLDSynthesis, PLDSynthesis II, Power Analyst, Power Analyst Station, Power To Create®, PowerLogic, PowerPCB®, Precision, Pre-Silicon, ProjectXpert, ProtoBoard, ProtoView, QDS, QNet, Quality IBIS, QuickCheck, QuickFault, QuickConnect, QuickGrade, QuickHDL, QuickHDL Express, QuickHDL Pro, QuickPart Builder, QuickPart Tables, QuickParts, QuickPath, QuickSim, QuickStart, Quick Use, Quick Use Development System, QuickVHDL, QUIET, RAM Lcompiler, RC-Delay, RC-Reduction, RapidExpert, REAL Time Solutions!, Registrar, Reliability Advisor, Reliability Manager, REMEDI, Renoir®, RF Architect, RF Gateway, RISE, ROM Lcompiler, RTL X-Press, Satellite PCB Station, ScalableModels, Scalable Verification, SCAP, Scan-Sequential, Scepter, Scepter DFF, Schematic View Compiler SVC, Schemgen, SDF (Software Data Formatter), SDL2000 Lcompiler, Seamless®, Seamless Co-Designer, Seamless CVE, Seamless Express, Seamless C-Bridge, Selective Promotion, Sheet Planner, Signal Spy, Signal Vision, SignalMask OPC, Signature Synthesis, Simulation Manager, SimPlot, SimView, Smartgrid, SmartMask, SmartParts, SmartRouter, SmartScripts, Smartshape, SNX, SpeakPath Analyzer, SpeedGate, SpeedGate DSV, SpeedWave, SOS Initiative, Source Explorer, SpiceNet, SST Velocity®, Standard Power Model Format (SPMF), Structure Recovery, Super C, Super IC Station, Symbol Genie, Symbolscript, SYMED, SynthesisWizard, System Architect, System Design Station, System Modeling Blocks, Systems on Board Initiative, SystemVision, Target Manager, Tau®, TeraCell, TeraPlace, TeraPlace-GF, Tech Notes, TestKompress®, Test Station®, Test Structure Builder, The Ultimate Site For HDL Simulation, The Ultimate Tool For HDL Simulation, TimeCloser, Timing Builder, TNX, ToolBuilder, Transcable®, TrueTiming, Utopia, Vlog, V-Express, V-Net, VHDLnet, VHDLwrite, Verinex, ViewBase®, ViewDraw®, ViewCreator, ViewLogic®, ViewSim®, ViewWare®, Viking, Virtual Library, VirtualLogic, Virtual Target, Virtual Test Manager:TOP, Voyager®, VRTX®, VRTXme, VRTXsa, VRTX32®, VStation, VStation-30M, Waveform DataPort, We Make TMN Easy, Wiz-o-matic, WorkXpert, xCalibre®, xCalibre, Xconfig, XlibCreator, XML2AXEL, Xpert, Xpert API, XpertBuilder, Xpert Dialogs, Xpert Profiler, XRAY®, XRAY MasterWorks®, XSH®, Xtrace®, Xtrace Daemon, Xtrace Protocol, XTK, Zeelan®, Zero Tolerance Verification and ZLibs are trademarks or registered trademarks of Mentor Graphics Corporation or its affiliated companies in the United States and other countries.

The following are service marks of Mentor Graphics Corporation:

A World of Learning, ADAPT, AppNotes, Assess2000, BIST Compiler, BIST-In-Place, BIST-Ready, Concurrent Board Process, DirectConnect, ECO Immunity, EDGE (Engineering Design Guide for Excellence), Expert2000, FastTrack Consulting, IntraStep, ISD Creation, It's More than Just Tools, Knowledge Center, Knowledge-Sourcing, Mentor Graphics Support CD, Mentor Graphics SupportBulletin, Mentor Graphics SupportCenter, Mentor Graphics SupportFax, Mentor Graphics SupportNet-Email, Mentor Graphics SupportNet-FTP, Mentor Graphics SupportNet-Telnet, Mentor Graphics We Mean Business, MTPI, Online Knowledge Center, OpenDoor, Reinstatement 2000, SiteLine2000, SupportNet KnowledgeBase, Support Services BaseLine, Support Services ClassLine, Support Services Latitudes, Support Services OpenLine, Support Services PrivateLine, Support Services SiteLine, Support Services TechLine, Support Services RemoteLine, and VR-Process.

Emulation products not available for sale in the United States of America: 3G-Vivace, Celaro, Optima, Flexible Intellectual Property, SimExpress, Vivace®.

Mentor Graphics' trademarks may only be used with express written permission from Mentor Graphics. Fair use of Mentor Graphics' trademarks in advertising and promotion of Mentor Graphics products requires proper acknowledgement.

### Third-Party Trademarks

The following names are trademarks, registered trademarks, and service marks of other companies that appear in Mentor Graphics product publications:

Adobe, the Adobe logo, Acrobat, the Acrobat logo, Exchange, FrameMaker, FrameViewer, PostScript, and Reader are registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Altera, ByteBlaster, Excalibur, and Quartus are trademarks or registered trademarks of Altera Corporation in the United States and other countries.

AM1 88, AMD, AMD-K6, and AMD Athlon Processor are trademarks of Advanced Micro Devices, Inc.

Apple and Laserwriter are registered trademarks of Apple Computer, Inc.

ARC, ARCTangent, High C, MetaWare, and SeeCode are trademarks or registered trademarks of ARC International.

ARIES is a registered trademark of Aries Technology.

AMBA, ARM, ARMulator, ARM7TDMI, ARM7TDMI-S, ARM9TDMI, ARM9E-S, ARM946E-S, ARM966E-S, EmbeddedICE, StrongARM, TDMI, and Thumb are trademarks or registered trademarks of ARM Limited.

ASAP, Aspire, C-FAS, CMPI, Eldo-FAS, EldoHDL, Eldo-Opt, Eldo-UDM, EldoVHDL, Eldo-XL, Elga, Elib, Elib-Plus, ESim, Fidel, Fideldo, GENIE, GENLIB, HDL-A, MDT, MGS-MEMT, MixVHDL, Model Generator Series (MGS), Opsim, SimLink, SimPilot, SpecEditor, Success, SystemEldo, VHDeLDO and Xelga are registered trademarks of ANACAD Electrical Engineering Software, a unit of Mentor Graphics Corporation.

AutoCAD and DXF are registered trademarks of Autodesk, Inc.

Avant! is a registered trademark and Star-Hspice is a trademark of Avant! Corporation LLC, a subsidiary of Synopsys, Inc.

AVR is a registered trademark of Atmel Corporation.

Cadence, Afirma signalscan, Allegro, Analog Artist, Composer, Concept, Design Planner, Dracula, GDSII, GED, HLD Systems, Leapfrog, Logic DP, NC-Verilog, OCEAN, Physical DP, Pillar, Silicon Ensemble, SPECCTRA, Spectre, Verilog, Verilog XL, Veritime, and Virtuoso are trademarks or registered trademarks of Cadence Design Systems, Inc.

CAE+Plus and ArchGen are registered trademarks of Cynergy System Design.

CalComp is a registered trademark of CalComp, Inc.

Canon is a registered trademark of Canon, Inc. BJ-130, BJ-130e, BJ-330, and Bubble Jet are trademarks of Canon, Inc.

Centronics is a registered trademark of Centronics Data Computer Corporation.

Crosstalk Toolkit (XTK), Crosstalk Field Solver (XFX), Motive, PADS-Perform, Pre-Route Delay Quantifier (PDQ), and Mentor Graphics Board Station Translator (MBX) are trademarks or registered trademarks of Innoveda, Inc.

ColdFire and M-Core are registered trademarks of Motorola, Inc.

Ethernet is a registered trademark of Xerox Corporation.

Foresight and Foresight Co-Designer are trademarks of Nu Thena Systems, Inc.

FLEXIm is a trademark of Macrovision Corporation.

GenCAD is a trademark of Teradyne Inc.

Hewlett-Packard (HP), LaserJet, MDS, HP-UX, PA-RISC, APOLLO, DOMAIN and HP are registered trademarks of Hewlett-Packard Company.

HCL-eXceed and HCL-eXceed/W are registered trademarks of Hummingbird Communications, Ltd.

HyperHelp is a trademark of Bristol Technology Inc.

Installshield is a registered trademark and service mark of InstallShield Corporation.

IBM, PowerPC, and RISC Systems/6000 are trademarks of International Business Machines Corporation.

I-DEAS and UG/Wiring are registered trademarks of Electronic Data Systems Corporation.

IKON is a trademark of Tahoma Technology.

Imagen, QMS, QMS-PS 820, Innovator, and Real Time Rasterization are registered trademarks of MINOLTA-QMS Inc. imPRESS and UltraScript are trademarks of MINOLTA-QMS Inc.

ImageGear is a registered trademark of AccuSoft Corporation.

Infineon, TriCore, and C165 are trademarks of Infineon Technologies AG.

Intel, i960, i386, and i486 are registered trademarks of Intel Corporation.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc.

Linux is a registered trademark of Linus Torvalds.

MemoryModeler MemMaker are trademarks of Denali Software, Inc.

Microsoft, ActiveX, MS-DOS, VBScript, Visual Basic, Visual C++, Windows, Windows 95, Windows 98, Windows 2000, and Windows NT are trademarks or registered trademarks of Microsoft Corporation.

MIPS is a trademark of MIPS Technologies, Inc.

MULTI is a registered trademark of Green Hills Software, Inc.

NEC and NEC EWS4800 are trademarks of NEC Corp.

Netscape and Netscape Navigator are registered trademarks of Netscape Communications Corporation in the United States and other countries.

Novas, Debussy, and nWave are trademarks or registered trademarks of Novas Software, Inc.

OakDSPCore, TeakDSPCore, TeakLite, Teak, and PalmDSPCore are registered trademarks of ParthusCeva, Inc.  
Oracle, Oracle8i, and SQL\*Plus are trademarks or registered trademarks of Oracle Corporation.  
OSE is a registered trademark of OSE Systems. PartMiner and Free Trade Zone are registered trademarks of PartMiner, Inc.  
PKZIP is a registered trademark of PKWARE, Inc.  
PMC-Sierra and RM7000C are trademarks of PMC-Sierra, Inc.  
Pro/CABLING, Pro/ENGINEER, and HARNESSDESIGN are trademarks or registered trademarks of Parametric Technology Corporation.  
Quantic is a registered trademark of Quantic EMC Inc.  
QUASAR is a trademark of ASM Lithography Holding N.V.  
Red Hat is a registered trademark and CYGWIN is a trademark of Red Hat, Inc. in the United States and other countries.  
SAP and R/3 are registered trademarks of SAP AG.  
SCO and the SCO logo are trademarks or registered trademarks of Caldera International, Inc.  
Sneak Circuit Analysis Tool (SCAT) is a registered trademark of SoHaR Incorporated.  
SPARC is a registered trademark, and SPARCstation is a trademark, of SPARC International, Inc.  
Sun Microsystems, Sun Workstation, and NeWS are registered trademarks of Sun Microsystems, Inc. Sun, Sun-2, Sun-3, Sun-4, OpenWindows, SunOS, SunView, NFS, and NSE are trademarks of Sun Microsystems, Inc.  
SuperH is a trademark of Hitachi, Ltd.  
Symbian OS is a trademark of Symbian Ltd.  
Synopsys, Design Compiler, DesignWare, Library Compiler, LM-family, PrimeTime, SmartModel, Speed-Model, Speed Modeling, SimWave, and Chronologic VCS are trademarks or registered trademarks of Synopsys, Inc.  
SystemC is a registered trademark of Open SystemC Initiative, Inc. in the United States and other countries.  
TASKING is a registered trademark of Altium Limited.  
Teamwork is a registered trademark of Computer Associates International, Inc.  
Tensilica and Xtensa are registered trademarks of Tensilica, Inc.  
Times and Helvetica are registered trademarks of Linotype AG.  
TimingDesigner and QuickBench are registered trademarks of Forte Design Systems.  
Tri-State, Tri-State Logic, tri-state, and tri-state logic are registered trademarks of National Semiconductor Corporation.  
UNIX, Motif, and OSF/1 are registered trademarks of The Open Group in the United States and other countries.  
Versatec is a trademark of Xerox Engineering Systems, Inc.  
Visula is a registered trademark of Zuken-Redac.  
VxSim, VxWorks and Wind River Systems are trademarks or registered trademarks of Wind River Systems, Inc.  
XVision is a registered trademark of Tarantella, Inc.  
X Window System is a trademark of MIT (Massachusetts Institute of Technology).  
Z80 is a registered trademark of Zilog, Inc.  
ZSP and ZSP400 are trademarks of LSI Logic Corporation.

Other brand or product names that appear in Mentor Graphics product publications are trademarks or registered trademarks of their respective holders.

Updated 12/4/02

VLSIGOD Join Telegram @vlsigodofficial

## TABLE OF CONTENTS

<b>About This Training Workbook .....</b>	<b>xix</b>
Introduction .....	i-xix
Training Modules .....	i-xx
Audience .....	i-xxi
Prerequisite Knowledge .....	i-xxii
Acronyms Used in This Workbook .....	i-xxii
Customer Support Information .....	i-xxiii
<b>Module 1</b>	
<b>Basic Concepts .....</b>	<b>1-1</b>
Module Topics .....	1-2
Why ManufacturingTest? .....	1-3
What is Design-for-Test? .....	1-5
Why Design-for-Test? .....	1-6
Yield and Defect Levels .....	1-7
Testing and Cost .....	1-8
What is Testability? .....	1-9
Types of Test .....	1-12
Manufacturing Defects .....	1-13
Fault Models .....	1-14
Stuck-at Fault Model .....	1-15
Transition Fault Model .....	1-16
Path Delay Fault Model .....	1-19
IDDQ Fault Model .....	1-21
Scan Design .....	1-22
Scan Cell Types .....	1-23
Mux DFF Scan Cell .....	1-24
LSSD Scan Cell .....	1-25
Clocked Scan Cell .....	1-26
Scan Chains .....	1-27
Scan Based Designs .....	1-28

## TABLE OF CONTENTS (Cont.)

Design Flow .....	1-30
Test Flow .....	1-31
Tool Flow .....	1-32
DFTAdvisor Overview .....	1-33
FastScan Overview .....	1-34
Graphical User Interface .....	1-35
Getting Help .....	1-37
Unix and Kshell within the GUI .....	1-38
Accessing SupportNet Material .....	1-39
Customer Support .....	1-40
Lab: Basic Concepts and DFT Flow .....	1-41

### Module 2

<b>Full Scan DFT Flow .....</b>	<b>2-1</b>
---------------------------------	------------

Module Topics .....	2-2
Scan and ATPG Flow .....	2-3
Circuit Setup .....	2-4
Gate-Level Netlist .....	2-5
Auto Black Boxing for Incomplete Netlists .....	2-6
Black Boxes .....	2-7
DFT Library .....	2-8
Creating a DFT Library .....	2-9
Automatic Generation of DFT Libraries .....	2-10
Include File Handling .....	2-11
Invoking DFTAdvisor .....	2-12
DFTAdvisor Tool Flow: An Overview .....	2-13
Command Structure .....	2-14
DFTAdvisor Tool Flow .....	2-15
SETUP .....	2-16
Scan/Test Logic Configuration .....	2-19
Set Test Logic Configuration .....	2-20
Adding Test Logic .....	2-21

## TABLE OF CONTENTS (Cont.)

Set Test Logic Configuration (Defining Non-scan Areas) .....	2-23
Design Rule Checking (DRC) .....	2-24
DRC .....	2-25
DRC Basics .....	2-26
Types of DRCs .....	2-28
Scan Specific DRCs .....	2-29
DFTInsight .....	2-30
Troubleshooting DRC Violations: Reporting S1 Fails .....	2-31
Viewing the Problem: Analyzing S1 Violations .....	2-32
Troubleshooting DRC Violations: Adding Clocks .....	2-33
Troubleshooting DRC Violations: Reporting S2 Fails .....	2-34
Viewing the Problem: Analyzing S2 Violations .....	2-35
Troubleshooting DRC S2 Violation .....	2-36
Viewing the Added TestClock Logic .....	2-37
Scan Identification .....	2-38
Scan/Test Logic Insertion .....	2-39
Write Results .....	2-41
FastScan Dofile .....	2-42
Enhanced Procedure File .....	2-43
Invoking FastScan .....	2-45
FastScan Tool Flow an Overview .....	2-46
FastScan Tool Flow .....	2-47
SETUP .....	2-48
FastScan ATPG in a DC Scan Insertion Flow .....	2-50
ATPG Setup Files .....	2-51
DRC (FastScan) .....	2-52
Configuration .....	2-53
Generate Patterns .....	2-54
Create Patterns .....	2-55
Save Results .....	2-56
Saving Test Patterns .....	2-57
Scan and ATPG Tool Flow .....	2-58
Lab: Full Scan DFT Flow .....	2-59

## TABLE OF CONTENTS (Cont.)

### Module 3

#### Configuring Scan Chains/Test Logic and Full Scan Flow .....3-1

Module Topics .....	3-2
Scan Methodology: Scan Cells .....	3-3
Scan Methodology: Full Scan .....	3-4
Scan Methodology: Full Scan Versus Partial Scan .....	3-5
Scan Methodology: DFT library and Scan Identification .....	3-6
Test Logic .....	3-8
Test Logic: Defining Library Models .....	3-11
Pins .....	3-12
Defining Pins .....	3-13
Clocks .....	3-16
Multiple Clock Issues .....	3-17
Multiple Clocks: Minimizing Clock Skew .....	3-19
Multiple Clocks .....	3-22
Multiple Clocks: Merging Clock Edges .....	3-23
Multiple Clocks: Merging Different Clocks .....	3-24
Multiple Clocks: Using Lockup Latches .....	3-25
Balancing Scan Chains .....	3-26
Scan Chain Ordering and Stitching .....	3-28
Scan Chain Ordering and Stitching Flow .....	3-29
Scan Chain Stitching: Unstitched Scan Cells .....	3-30
Scan Chain Stitching: Stitching Existing Scan Cells .....	3-31
Lab: Configuring Scan Chains/Test Logic and Full Scan Flow .....	3-32

### Module 4

#### Understanding ATPGMessaging .....4-1

Module Topics .....	4-2
Messages at Invocation .....	4-3
Messages at Invocation: Warnings .....	4-4

## TABLE OF CONTENTS (Cont.)

Messages When Exiting Setup .....	4-5
ATPG Reporting .....	4-6
Special Messages in ATPG Reporting .....	4-8
Test Coverage Reporting .....	4-9
Test Coverage Reporting Fault Collapsing .....	4-12
Determining the Cause of Undetected Faults .....	4-13
Lab: Understanding ATPG Messaging .....	4-16
<b>Module 5</b>	
<b>Achieving High Test Coverage .....</b>	<b>5-1</b>
Module Topics .....	5-2
Methodologies: Initial Run (Fault Sampling) .....	5-3
Methodologies: External Fault List .....	5-4
Adding NOfaults .....	5-5
FastScan's Test Pattern Types .....	5-6
Basic Scan Patterns .....	5-7
Basic Scan Pattern Operation .....	5-9
Clock Primary Output Patterns .....	5-16
Clock Sequential Patterns .....	5-18
Clock Sequential Pattern Operation .....	5-21
RAM Sequential Patterns .....	5-28
RAM Sequential Patterns Example: To Test For Stuck-At-0 at the Output of U1 .....	5-32
RAM Sequential Pattern Operation .....	5-33
Multi Load Patterns .....	5-37
Multi Load Patterns Example .....	5-39
MacroTest Patterns .....	5-41
MacroTest .....	5-42
Memory BIST .....	5-43
Test Pattern Type Summary .....	5-44
Saving Patterns .....	5-45
Reuse, Debugging, and Diagnostics .....	5-46

## TABLE OF CONTENTS (Cont.)

Reuse, Debugging, and Diagnostics: ASCII and Binary Formats .....	5-47
Reuse, Debugging, and Diagnostics: Reading ASCII Files Back into FastScan.....	5-48
Time-Based Verification .....	5-49
Verification of Pattern Formats .....	5-51
Manufacturing Test .....	5-52
Lab: Achieving High Test Coverage .....	5-53

### Module 6

<b>Creating High Quality Patterns at Low Cost .....</b>	<b>6-1</b>
---	------------

Module Topics .....	6-2
Quality and Cost .....	6-3
Quality .....	6-4
Cost .....	6-5
At-Speed ATPG .....	6-6
At-Speed ATPG and the Transition Fault Model .....	6-8
At-Speed ATPG and the Path Delay Fault Model .....	6-9
The Path Delay Model .....	6-10
Transition Fault Patterns .....	6-16
Creating Transition Fault Patterns: Launch-Off Shift .....	6-17
Creating Transition Fault Patterns: Broadside .....	6-18
Timing for At-Speed Test .....	6-19
Path Delay Pattern Flow .....	6-21
Path Definition Files .....	6-23
Creating Path Delay Patterns .....	6-24
IDDQ Patterns .....	6-25
Creating IDDQ Patterns .....	6-26
Optimizing Quality and Cost .....	6-28
ATE Characteristics .....	6-30
Lab: Creating High Quality Patterns at Low Cost .....	6-31

### Module 7

<b>Advanced ATPG .....</b>	<b>7-1</b>
----------------------------	------------

## TABLE OF CONTENTS (Cont.)

Module Topics .....	7-2
Black Boxes .....	7-3
Black Box Examples .....	7-4
Testing Embedded Blocks .....	7-5
Testing Embedded Memories: MacroTest .....	7-7
At-Speed MacroTest .....	7-9
Testing Embedded Memories: Synchronous MacroTest .....	7-10
Built-In Self-Test Basics .....	7-12
Testing Embedded Memories: Memory BIST .....	7-13
Testing Embedded Memories: Memory BIST Bypass .....	7-14
Initialization Issues .....	7-15
Initialization Example .....	7-17
Auto Generate Test_Setup .....	7-18
Boundary Scan Basics .....	7-19
Boundary Scan Architecture .....	7-20
Connecting Boundary Scan with Internal Scan .....	7-21
Accessing Internal Scan Instructions .....	7-23
Connecting Internal Scan to Boundary Scan Using BSDArchitect .....	7-24
Top Up ATPG .....	7-25
Top Up Patterns From BIST .....	7-26
Diagnostics .....	7-27
Diagnostics: FastScan .....	7-28
Performing a Diagnosis .....	7-29
Diagnostic Commands .....	7-30
Diagnostics: Failure File .....	7-31
Diagnostics Report .....	7-32
Diagnostics Issues .....	7-34
Lab: Advanced ATPG .....	7-35
<b>Module 8</b>	
<b>Troubleshooting DRC and Simulation Mismatch .....</b>	<b>8-1</b>
Module Topics .....	8-2

## TABLE OF CONTENTS (Cont.)

Troubleshooting Areas of Low Coverage .....	8-3
Hierarchy Browser .....	8-4
Assessing the Problem .....	8-5
Faults Classified as ATPG Untestable .....	8-9
Faults Classified as Undetectable .....	8-10
Addressing Aborted Faults .....	8-11
Bus Contention .....	8-13
Addressing Bus Contention .....	8-14
Addressing Bus Contention: Types of Contention .....	8-15
Debugging Bus Contention .....	8-16
Fault-by-Fault AU Debugging: Report Testability Data Command .....	8-17
Report Testability Data Command .....	8-18
TieX (D5) .....	8-19
Fault-by-Fault AU Debugging: Set Gate Report Command .....	8-20
Set Gate Report Command -Constrain_Value .....	8-21
Fault-by-Fault AU Debugging: Analyze Fault Command .....	8-22
Fault-by-Fault AU Debugging: Report Test Stimulus Command .....	8-23
Lab: Troubleshooting Areas of Low Test Coverage .....	8-24

### Module 9

<b>Troubleshooting DRC and Simulation Mismatch .....</b>	<b>9-1</b>
Module Topics .....	9-2
Analyzing DRC Violations: Commands .....	9-3
Analyzing DRC Violations: Report Gates Command .....	9-4
DRC Violations: E4 - Procedure (Bus Contention) .....	9-5
Debugging E4 Violations .....	9-6
E4 Contention on Bidirectionals .....	9-7
Clocks .....	9-8
Clock Cones .....	9-9
Effect Cones .....	9-10
Both Cones .....	9-11
Clock Rules: C3 .....	9-12

## TABLE OF CONTENTS (Cont.)

DRC Violations: C3 .....	9-13
FastScan Event Simulation .....	9-14
Setting Event Simulation .....	9-15
Handling C3 Violations .....	9-16
Clock Rules: C6 .....	9-17
Handling C6 Violations .....	9-18
Data Rules: D5 .....	9-19
Data Rules: D6 .....	9-20
Handling D5 and D6 Violations .....	9-21
Scan Chain Trace Rules: T3 .....	9-22
Common Causes of T3 Errors .....	9-23
Scan Chain Trace Rules: T5 .....	9-24
Debugging T3 and T5 Violations .....	9-25
Testbenches .....	9-26
Serial Testbench .....	9-27
Parallel Testbench .....	9-28
Debugging Simulation Mismatches in FastScan .....	9-29
DRC Violations: Simulation Mismatches .....	9-30
When, Where, and How Many Mismatches .....	9-31
Clock Skew Problems .....	9-33
Timing Violations .....	9-34
Library Problems .....	9-35
Automatic Analysis of Simulation Mismatch .....	9-36
Debugging Serial Simulation Mismatches: Chain Test .....	9-38
Clock Skew in Chain Test .....	9-39
Lab: Troubleshooting DRC and Simulation Mismatch .....	9-40

## TABLE OF CONTENTS (Cont.)

## LIST OF FIGURES

Figure 1-1. The DFT Graphical User Interface .....	1-44
Figure 1-2. Using Query Help .....	1-51
Figure 2-1. DFTA Advisor Control Panel .....	2-62
Figure 2-2. FastScan Tool Flow.....	<b>2-66</b>
Figure 3-1. Connecting Existing Pins/Pads as Scan Inputs and Outputs .....	3-33
Figure 3-2. Balancing Scan Chains Using Lockup Latches .....	3-38
Figure 3-3. <b>Scan Cell Order File</b> .....	<b>3-48</b>
Figure 3-4. Scan Cell Reorder File .....	3-51
Figure 6-1. Copying the Timeplate.....	6-37
Figure 6-2. Renaming the Timeplate .....	6-38
Figure 6-3. Editing Timeplate .....	6-38

## LIST OF FIGURES (cont.)

## About This Training Workbook

### Introduction

The Scan and ATPG Training course is designed to be a three-day, basic to advanced level, instructor-led, “Design-for-Test process and Mentor Graphics tool training” course.

The following are the top level course goals:

- The student will understand Design-for-Test design processes
- The student will gain experience with Scan and ATPG tool flow
- The student will understand how to find information and problem-solve typical design issues

If taken in its entirety, this training course is intended to introduce design engineers both to the field of Design-for-Test and the Mentor Graphics Design-for- Test tool suite. It will educate designers to the basic theory of test, Design-for-Test processes and Scan and ATPG tool flow

## Training Modules

1. Understanding ATPGMessagingUnderstanding  
ATPGMessagingUnderstanding ATPGMessaging Basic Concepts and DFT Flow
2. Full Scan and DFT Flow
3. Configuring Scan Chains/Test Logic and Full Scan Flow
4. Understanding ATPG Messaging
5. Achieving High Test Coverage
6. Creating High Quality Patterns at Low Cost
7. Advanced ATPG
8. Troubleshooting Areas of Low Test Coverage
9. Troubleshooting DRC and Simulation Mismatch

## Audience

### Primary Audience

The target student profile is the Electronic Design Engineer using synthesis tools to develop synchronous digital designs. It is assumed that students will be using FastScan and DFTAdvisor in accordance with DFTInsight (as a debugging tool), and optionally, FlexTest. This type of student will comprise about 80% of the course attendees and will have the following characteristics:

- They have some limited familiarity with DFT terminology and concepts.
- They are required to use DFTAdvisor (or a similar product) to add full-scan circuitry to their design.
- As they work with these DFT tools, these engineers want to know “what is this tool doing to my design” (or my design flow) and “how do I control what the tool is doing to my design?”
- These engineers want to know how to analyze the tool-generated reports and modify the tool setup constraints to achieve the test goals that may be imposed on them by their organization.
- These engineers want to be well grounded in the basic tool process flow and be able to respond appropriately when the tools report “problems.”

### Secondary Audience

About 20% of the students will be “test engineers.” These engineers are typically members of a manufacturing test group or an internal CAD group that provides support for design engineers. Test engineers are typically well grounded in their understanding of DFT terms and concepts, but may not have had much exposure to running design synthesis tools or DFT test insertion and ATPG tools. In other words, they usually just get a set of test patterns handed to them by a designer and it is their job to create the final test set and program the tester.

Test engineers are typically in the course to get a better understanding of their customers (the design engineers), get experience running the tools, and get a greater appreciation of the problems that the design engineers face.

Although the class is not explicitly “tuned” to this audience, test engineers will profit greatly from this course and typically achieve their personal objectives in taking the course.

## Prerequisite Knowledge

Prerequisite knowledge in DFT fundamentals is required. The purpose of requiring prerequisites is to (1) reduce “learning overload” which can happen early in the course and (2) help the students move quickly toward learning tool concepts and best practices for getting results.

Generic DFT concepts and terminology can be learned from sources outside Mentor Graphics.

## Acronyms Used in This Workbook

ATE	Automatic Test Equipment
ATPG	Automatic Test Pattern Generation
DFT	Design for Test
BIST	Built-In Self Test
DRC	Design Rules Checking
GUI	Graphical User Interface

## Customer Support Information

Additional help is available from Mentor Graphics Customer Support using the following phone numbers, email address, and internet site:

**DirectConnect** (M-F: 6am-5:30pm, PST) 1-800-547-4303

**SupportCenter Fax** 1-800-684-1795

**SupportNet-Email** support\_net@mentor.com

**SupportNet-Web site** <http://www.mentor.com/supportnet>

**Mentor DFT Web site** <http://www.mentor.com/dft>

VLSIGOD Join Telegram @vlsigodofficial

# Module 1

## Basic Concepts

### Objectives

Upon completion of this module, you will be able to:

- Understand basic Design-for-Test (DFT) technology.
- Explain design flow.
- Invoke the ATPG graphical user interface (GUI) and use many of its features.
- Access reference information:
  - a. Using the help command.
  - b. Using online documentation to solve problem-solving issues.

## Module Topics

---

### Module Topics

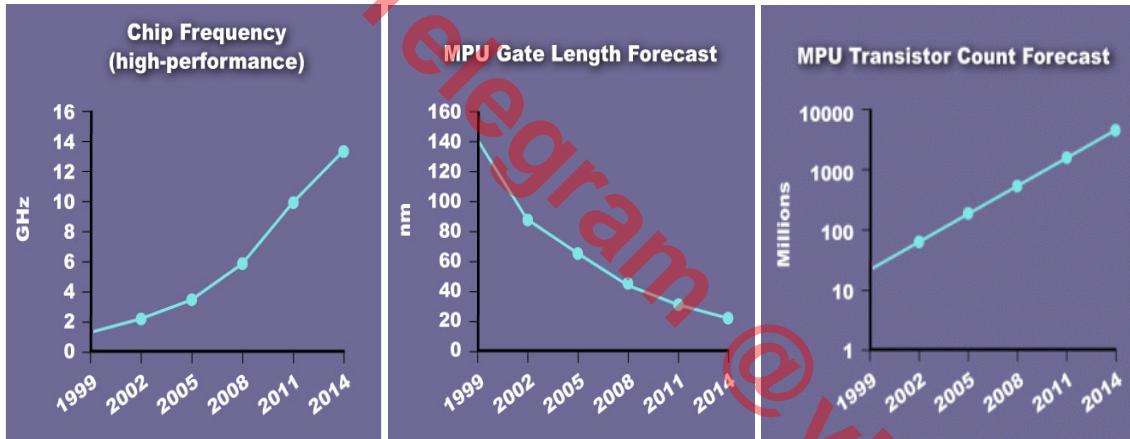
- ◆ This module addresses the following topics:
  - Test
  - Scan design
  - Design flow
  - DFT tools and tool flow
  - GUI and getting help
  - Accessing information

### Notes:

# Why Manufacturing Test?

## Why Manufacturing Test?

- ♦ Density and performance issues:
  - ITRS Roadmap 1999



1-3 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

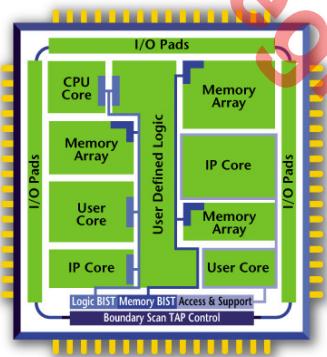
Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Why Manufacturing Test? (Cont.)

### Why Manufacturing Test? (Cont.)

- ◆ Design procedure challenges:
  - Core-based, IP-reuse designs
  - Large, distributed teams
  - Multiple design levels and constraints
  - Technology remapping and migration



1-4 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

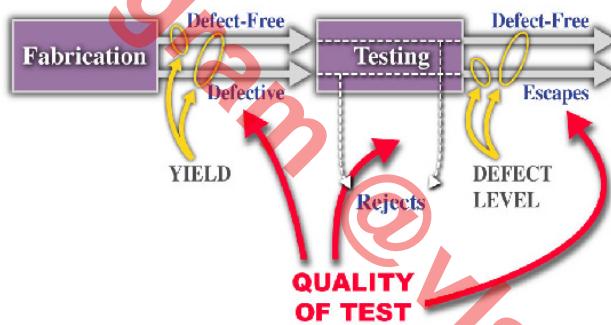
Copyright © 2003 Mentor Graphics Corporation

### Notes:

# What is Design-for-Test?

## What is Design-for-Test?

- ◆ DFT strategies that:
  - Improve quality by detecting defects
  - Make it easier to generate vectors
  - Reduce vector generation time
  - Reduce cost



1-5 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Why Design-for-Test?

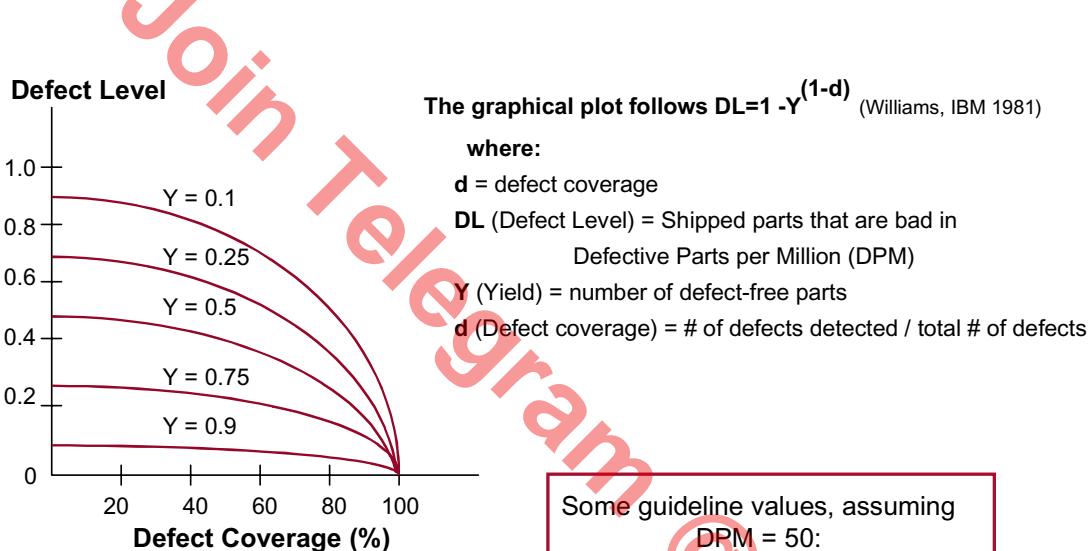
---

### Why Design-for-Test?

- ◆ To increase Productivity:
  - Shorter time-to-market
  - Reduced design cycle
  - Reduced cost
- ◆ To improve Quality:
  - Reduced Defects per million (DPM)
  - Improved quality of test

### Notes:

# Yield and Defect Levels



1-7 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

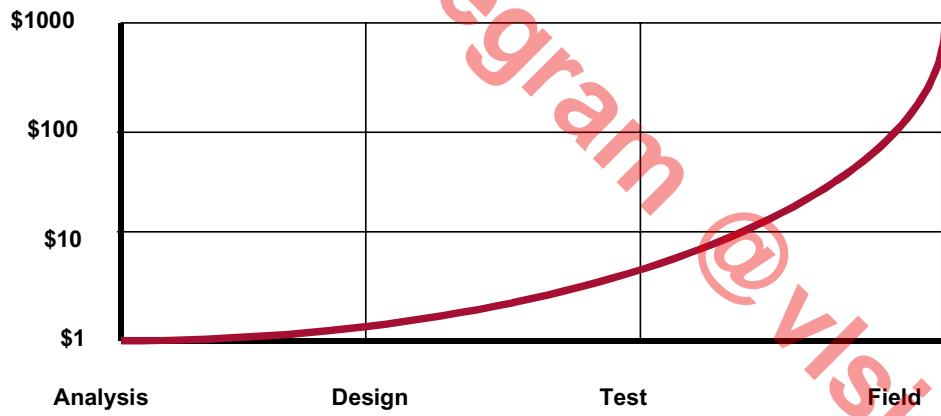
Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Testing and Cost

### Testing and Cost

- ♦ Low number of defective parts (DPM) is very critical
- ♦ Cost to replace parts grows exponentially throughout design cycle
- ♦ Cost of bad part in critical device (for example, a pacemaker or airplane) is immeasurable



1-8 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

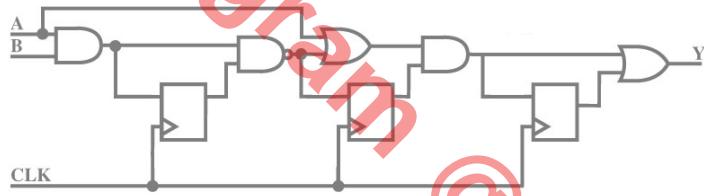
Copyright © 2003 Mentor Graphics Corporation

### Notes:

# What is Testability?

## What is Testability?

- ◆ The ability to put a design into a known initial state, and then control and observe internal signal values
- ◆ Circuit with DFFs:
  - Low testability



1-9 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

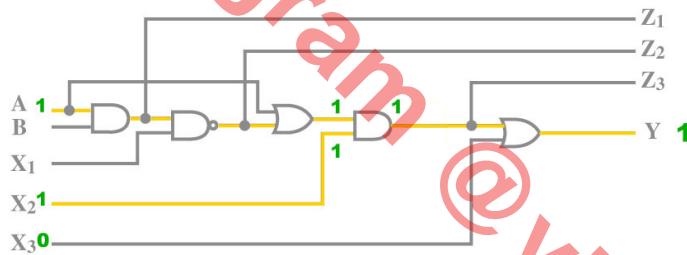
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# What is Testability? (Cont.)

## What is Testability? (Cont.)

- ◆ **Controllability:**
  - The ability to set a node to a specific value
- ◆ **Observability:**
  - The ability to observe a node's value
- ◆ **Circuit without DFFs:**
  - Circuit is controllable and observable



1-10 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

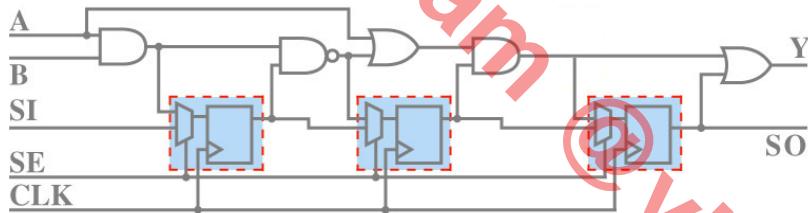
Copyright © 2003 Mentor Graphics Corporation

## Notes:

## What is Testability? (Cont.)

### What is Testability? (Cont.)

- ◆ A highly testable design:
  - A circuit that can be placed into a known initial state
  - PIs are controllable
  - POs are observable and measurable
- ◆ Circuit with DFFs replaced with MUX scan;
  - A highly testable design



1-11 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

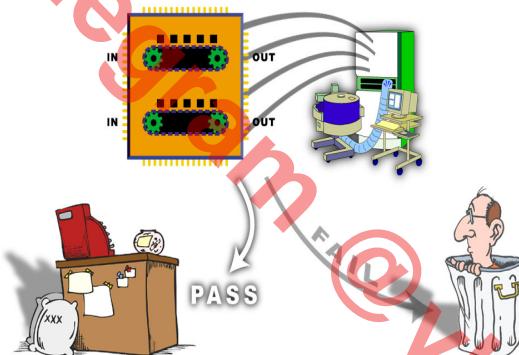
Copyright © 2003 Mentor Graphics Corporation

### Notes:

# Types of Test

## Types of Test

- ◆ **Functional tests**
  - Verify circuit functionality
  - Costly
- ◆ **Structural tests**
  - Target manufacturing defects



1-12 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Manufacturing Defects

## Manufacturing Defects

- ♦ Physical problems in silicon:
  - Contamination-causing opens
  - Extra metal-causing shorts
  - Insufficient doping
  - Process or mask errors
  - Trace bridges
  - Open vias
  - CMOS stuck-on
  - CMOS stuck-open
  - Slow transistors

Contamination



Extra metal



[Photos courtesy of SEMATECH & IBM. Nigh, et.al., 1998]

1-13 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Fault Models

### Fault Models

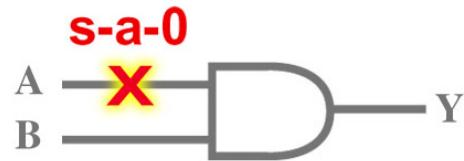
- ◆ Fault models:
  - Stuck-at-fault
  - Transition fault
  - Path delay
  - IDQ

## Notes:

# Stuck-at Fault Model

## Stuck-at Fault Model

- ◆ Fault models are logic targets for defects
- ◆ A fault is detected:
  - When a difference is observed between a “good” and “faulty” circuit
- ◆ Most common fault model:
  - Most defects are detected with the stuck-at fault model
  - A terminal of a gate is permanently stuck-at 0 or 1
  - Detects:
    - Opens or shorts in the interconnect
    - Bridging faults



A	B	Good	A s.a.0	A s.a.1
0	0	0	0	0
0	1	0	0	1
1	0	0	0	0
1	1	1	0	1

## Notes:

## Transition Fault Model

### Transition Fault Model

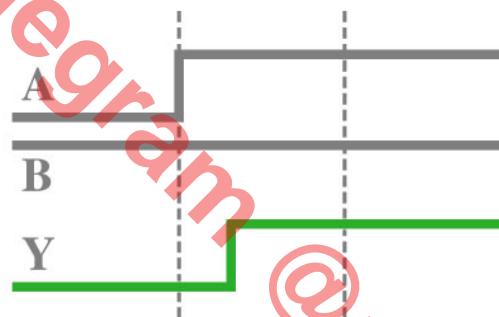
- ◆ Delay fault model
  - Slow-to-rise node
  - Slow-to-fall node
- ◆ Involves at-speed testing (using scan chains)
- ◆ application of two cycles:
  - Launch
  - Capture
- ◆ Tests for gross time delay
- ◆ Detects:
  - Partially conducting transistors
  - Interconnections

### Notes:

## Transition Fault Model (Cont.)

### Transition Fault Model (Cont.)

- ♦ Functional gate



1-17 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

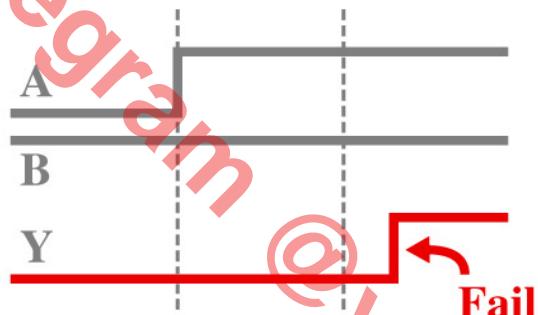
Copyright © 2003 Mentor Graphics Corporation

### Notes:

## Transition Fault Model (Cont.)

### Transition Fault Model (Cont.)

- ♦ Failing gate



### Notes:

## Path Delay Fault Model

### Path Delay Fault Model

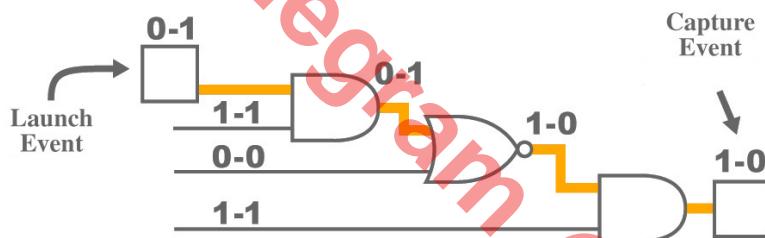
- ◆ Delay fault model:
  - Slow-to-rise path
  - Slow-to-fall path
- ◆ Involves at-speed application of two cycles:
  - Launch
  - Capture
- ◆ Tests for lumped time delay:
  - sum of time delays that stack up
- ◆ Detects:
  - Partially conducting transistors
  - Diffusions

### Notes:

## Path Delay Fault Model (Cont.)

### Path Delay Fault Model (Cont.)

- ◆ Test sequence:
  - Define a path
  - Launch
  - Capture

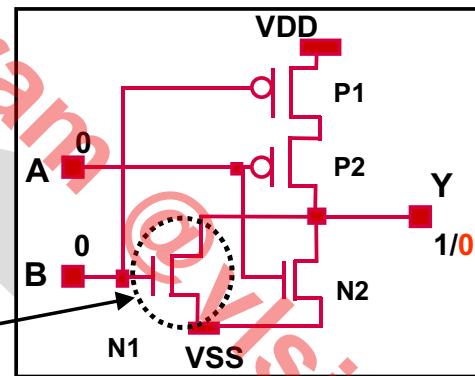
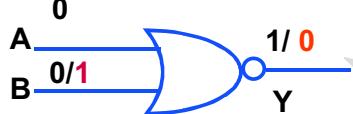


### Notes:

# IDDQ Fault Model

## IDDQ Fault Model

- ◆ Measures quiescent power supply current during the stable state
- ◆ Takes time, but attains 80-90% test coverage
- ◆ Detects:
  - CMOS transistor stuck-on/some stuck-open conditions
  - Bridging faults
  - Partially conducting transistors
  - ...



1-21 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Scan Design

### Scan Design

- ◆ Internal scan is a structured DFT technique that...
  - Replaces sequential storage elements with scan cells
  - Stitches scan cells into a serial scan register (scan chain)
  - Makes sequential circuitry appear combinational

### Notes:

## Scan Cell Types

### Scan Cell Types

- ◆ Scan cell types:

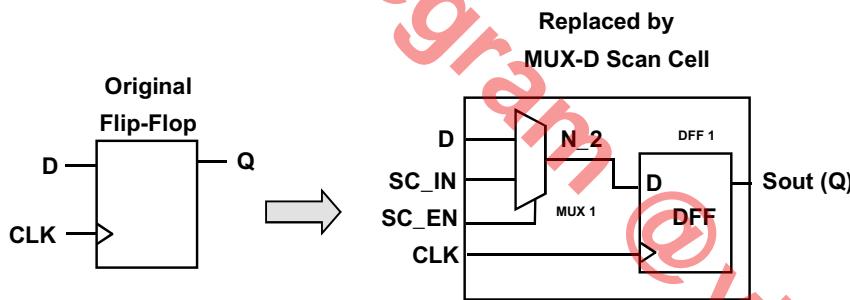
- Mux DFF
- Level-Sensitive Scan Design (LSSD)
- Clocked Scan

### Notes:

# Mux DFF Scan Cell

## Mux DFF Scan Cell

- ◆ Multiplexer used to select data input:
  - D in normal mode
  - SI in scan mode
- ◆ Scan enable (SE) selects mode of operation
- ◆ Increased propagation delay
- ◆ Adds 5-15% area overhead



1-24 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

# LSSD Scan Cell

## LSSD Scan Cell

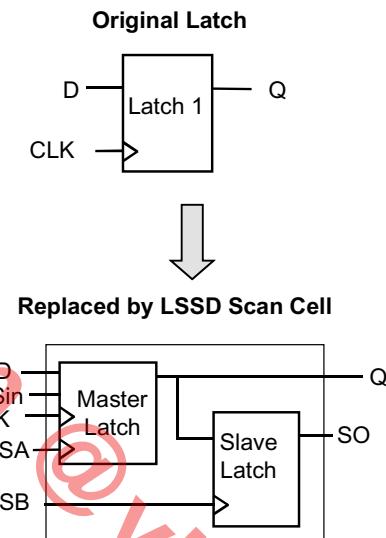
- ◆ Level-Sensitive Scan Design:

- Normal mode:

- The master latch captures system data D using the system (CLK) and outputs L1

- Test mode:

- The two non-overlapping clocks (ENSA and ENSB) shift data through the latches
    - The scan output is SO

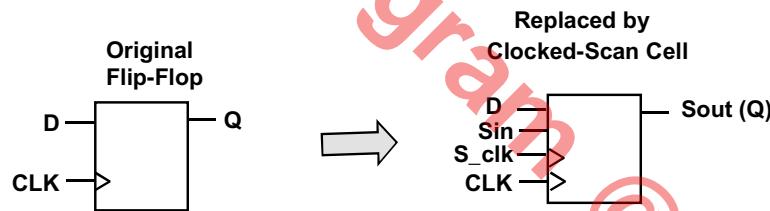


## Notes:

## Clocked Scan Cell

### Clocked Scan Cell

- ◆ Dedicated clock for shift
- ◆ Less propagation delay
- ◆ Non-scan cells are not disturbed during shift
- ◆ Reduced chance of clock skew

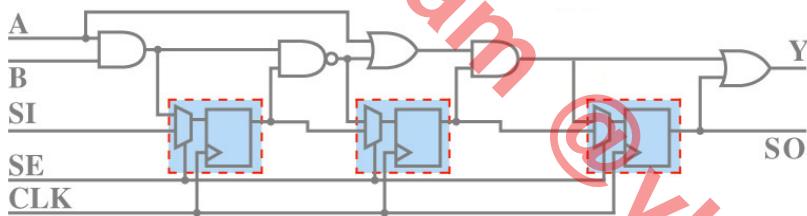


### Notes:

# Scan Chains

## Scan Chains

- ◆ Scan Enable:
  - When active allows scan data to enter the registers
- ◆ Scan Input port:
  - Data is loaded into scan cells
- ◆ Scan Output port:
  - Data is read by shifting data out



1-27 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

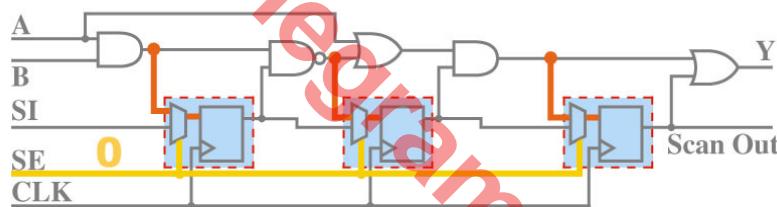
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Scan Based Designs

## Scan Based Designs

- ◆ Normal mode:
  - Sequential elements perform regular system functions



1-28 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

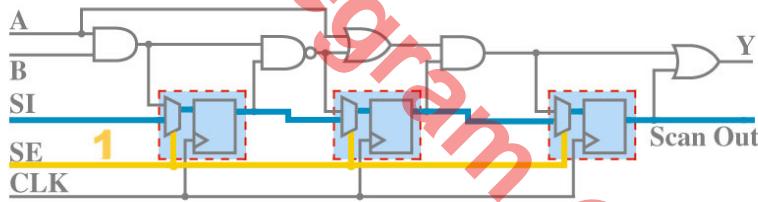
Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Scan Based Designs (Cont.)

### Scan Based Designs (Cont.)

- ◆ Scan mode:
  - Sequential elements are connected into one or more shift registers
  - Circuit appears combinational



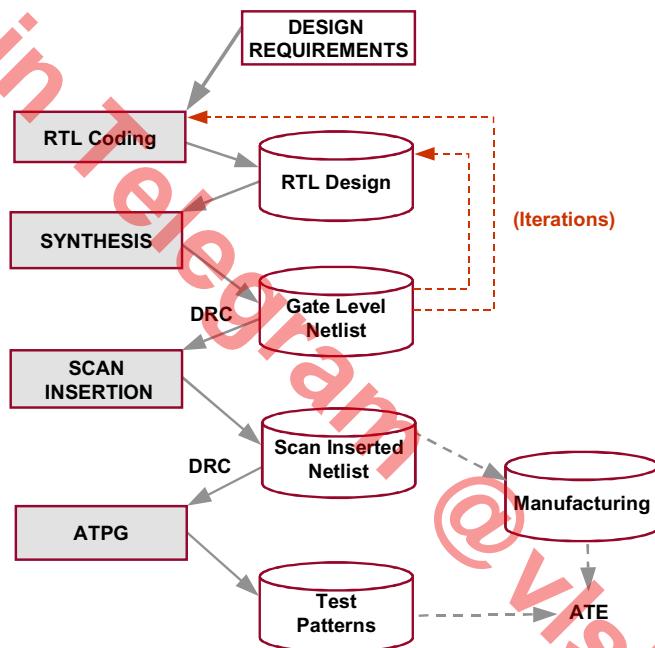
1-29 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

### Notes:

# Design Flow

## Design Flow

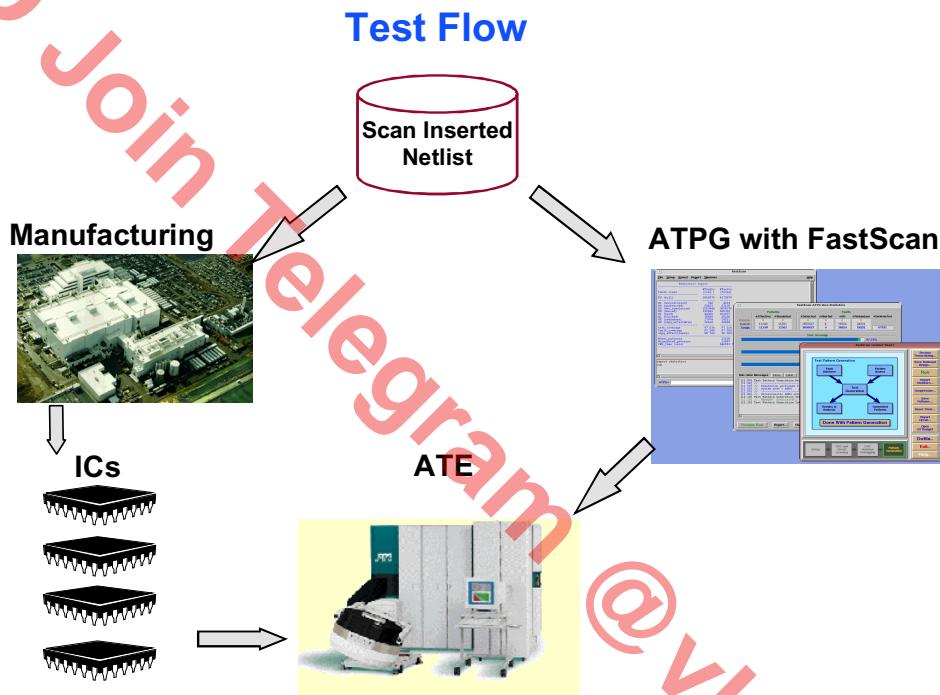


1-30 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Test Flow

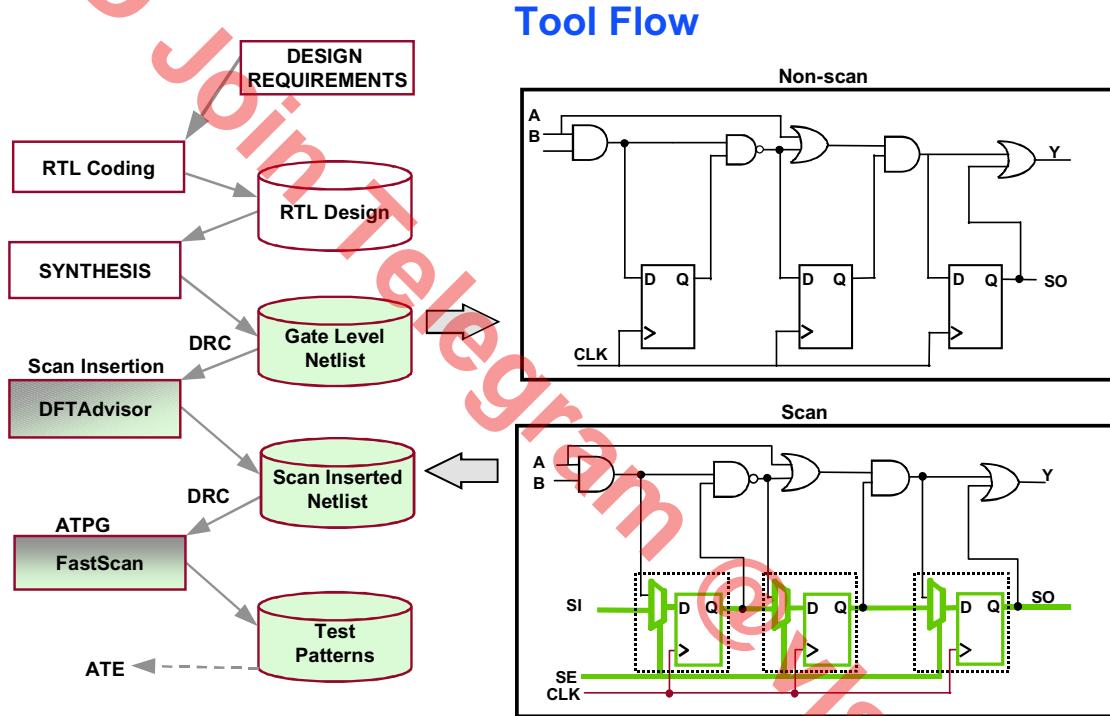


1-31 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

### Notes:

# Tool Flow



1-32 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

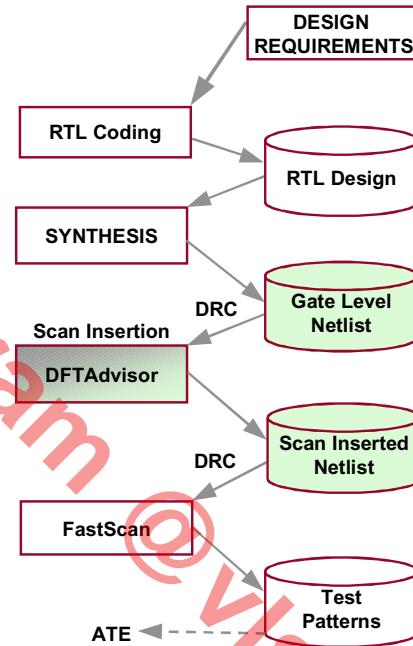
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# DFTAdvisor Overview

## DFTAdvisor Overview

- ◆ DFTAdvisor
  - Full scan insertion tool
- ◆ Abilities:
  - Testability analysis
  - Design Rules Checking (DRC)
  - Test logic synthesis



1-33 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

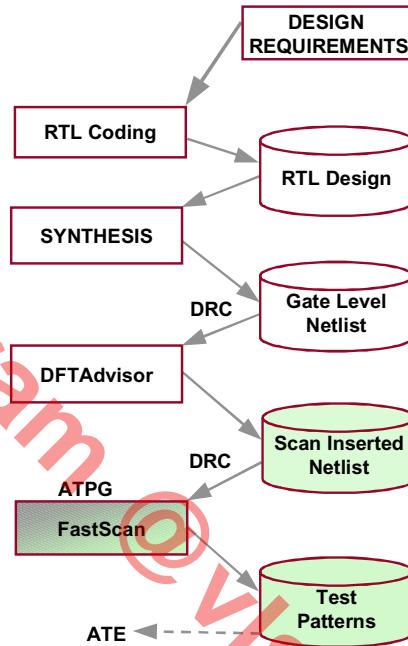
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# FastScan Overview

## FastScan Overview

- ◆ **FastScan**
  - Generates test patterns for production test
- ◆ **Abilities:**
  - Best suited for full-scan designs
  - Produces high test coverage
  - Generates compact pattern sets
  - Design rules checking (DRC)



1-34 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

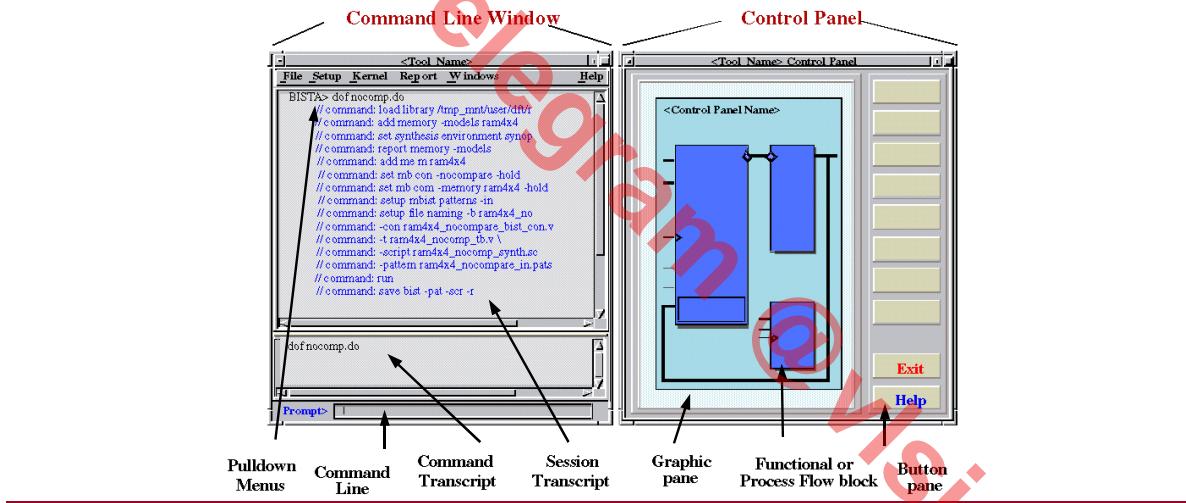
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Graphical User Interface

## Graphical User Interface

- ◆ All DFT tools use a similar Graphical User Interface (GUI)
- ◆ When you invoke a tool, it opens:
  - The Main (Command line) window
  - The Control Panel windows



1-35 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Graphical User Interface (Cont.)

### Graphical User Interface (Cont.)

- ◆ Main (Command line) window:
  - Executes commands using:
    - Pulldown menus
    - Popup menus
    - Command line
  - Command transcripting
  - Issues commands by following the 3-2-1 convention
- ◆ Control panel
  - Graphical link to:
    - Functional blocks
    - Process blocks

### Notes:

## Getting Help

### Getting Help

- ◆ **Types of Help:**
  - **Query Help:**
    - Text-based messages on dialog box objects
  - **Popup Help:**
    - Text-based messages on Control Panel objects
  - **Tool Guide:**
    - Tool information
  - **Help menu:**
    - Lists different help topics and tool specific PDF manuals
  - **Command Usage:**
    - SETUP> HELP [command\_name] [-manual]
    - Use -manual switch to automatically open command page in PDF file
  - **Acrobat Reader**
    - mgcdocs
    - shell> fastscan -manual OR
    - shell> dftadvisor -manual

### Notes:

# Unix and Kshell within the GUI

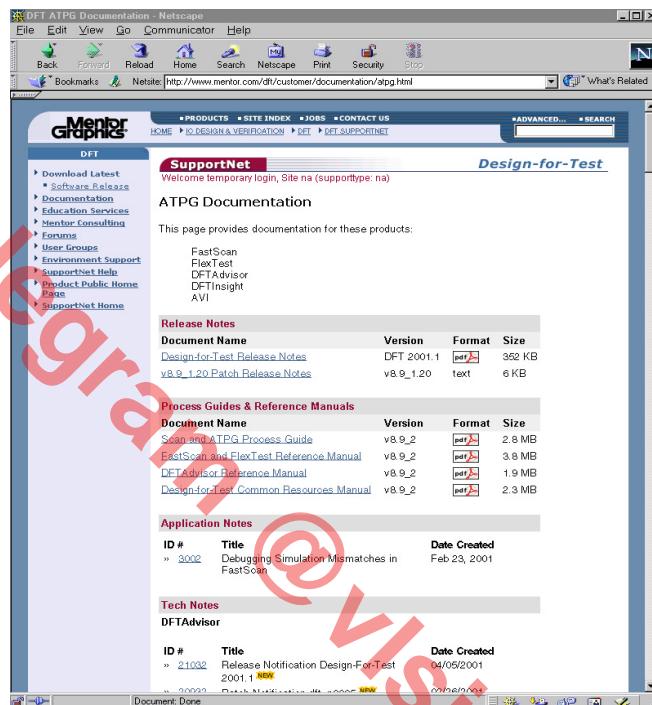
## Unix and Kshell within the GUI

- ◆ To run Unix operating system commands within the DFT session:
  - ATPG/SETUP/ ...> system [Unix command]
- ◆ To perform Kshell editing:
  - ATPG/SETUP ...> set command editing -vi

## Notes:

## Accessing SupportNet Material

- ◆ Login required
- ◆ Documentation
  - Release Notes
  - Process Guides and Reference Manuals
  - Application Notes
  - Tech Notes
- ◆ Software
  - Releases
  - Patches



1-39 • Design-for-Test: Scan and ATPG:  
Basic Concepts and DFT Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

### Notes:

## Customer Support

---

### Customer Support

- ◆ DirectConnect (M-F: 6am -5:30, PST)
  - 1-800-547-4303
- ◆ SupportCenter Fax
  - 1-800-684-1795
- ◆ SupportNet website
  - <http://www.mentor.com/supportnet>
- ◆ Mentor DFT website
  - <http://www.mentor.com/dft>
- ◆ User Group
  - Mentor Graphics User's Group (MUG)
- ◆ Forums
  - SupportNet-Email

### Notes:

# Lab: Basic Concepts and DFT Flow

## Objectives

- Invoke the ATPG Graphical User Interface (GUI) and use many of its features.
- Access information:
  - Find different ways of getting help.
  - Explore the help options in the Graphical User Interface (GUI) and command line.

## Lab Conventions

In order to make the lab instructions as simple and clear as possible, the instructions use the following conventions:

- You usually just click mouse buttons unless specifically told to do otherwise.
  - LMB — left mouse button
  - RMB — right mouse button
  - MMB — middle mouse button
- Numbered or lettered steps have you perform some action. Paragraphs without numbers only provide supplemental information or ask questions for you to think about.
- You should usually exit the tool after a lab. At the beginning of each lab is a “Getting Started” section that brings you to the correct state to perform the next exercise. Some of the exercises build on each other in which case you will be instructed to leave the tool up and running.

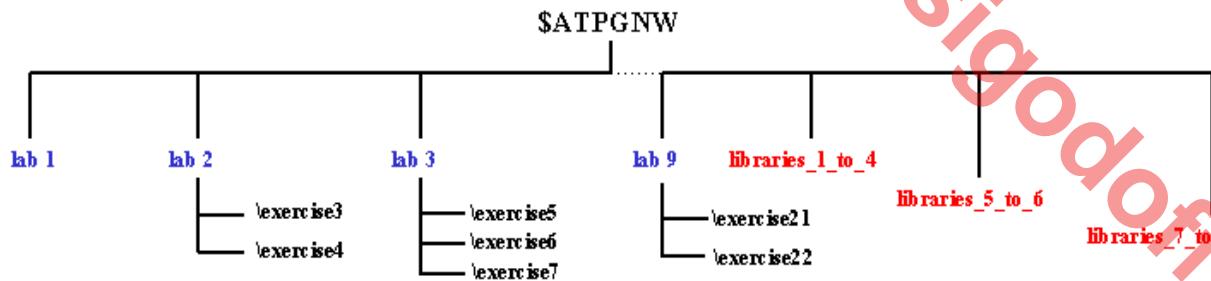
- If you have any problems or questions about a lab, ask your instructor for help.
- Experiment!

## List of Exercises

- Exercise 1: Using the Graphical User Interface (GUI)
- Exercise 2: Accessing Information

## Getting Started

1. Get your user name and login from your instructor.
2. Go to your lab workstation and log in. You should be in the \$HOME directory when you login.
3. The directory structure is:



\$ATPGNW is an environment variable set up under the home directory.

The libraries needed in the first four labs (exercises 1 — 10) are found in directories *libraries\_1\_to\_4*. The libraries needed for labs 5 and 6 (exercises 11 — 16) are found in directory *libraries\_5\_to\_6*. The libraries needed for the last three labs (exercises 17 — 22) are found in directory *libraries\_7\_to\_9*.

Change to the `$ATPGNW/lab1` directory.

```
shell> cd $ATPGNW/lab1
```

## Exercise 1: Using the Graphical User Interface (GUI)

Design-for-Test (DFT) tools use both the command line and the GUI. In this exercise, you invoke the FastScan GUI and gain experience using many of its common features.

1. Invoke FastScan.

```
shell> fastscan
```

This invokes the FastScan Invocation Arguments dialogue box. You must enter a design, a design format, and an ATPG library to invoke the GUI.

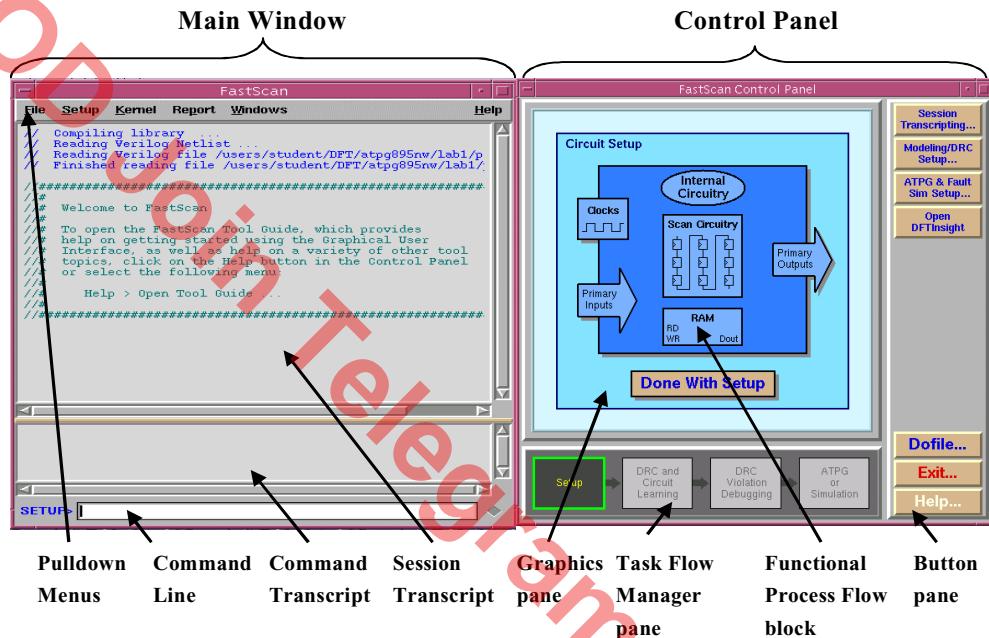
- a. Enter the following in the appropriate dialogue box. You must enter a design, a design format, and an ATPG library in order to invoke the GUI.

Design: pipe\_scan.v

Design Format: Verilog

Library: libraries\_1\_to\_4/adk.atpg

- b. Both the Main and Control Panel windows are now open.



**Figure 1-1. The DFT Graphical User Interface**

FastScan has compiled the DFT library and read in the Verilog netlist. Comments to this effect are displayed in the session transcript area.

A **SETUP>** prompt precedes the command line to indicate that you are in the **Setup** mode.

2. Examine the GUI.
  - a. Move both windows at the same time by clicking and holding the LMB in the title bar of the Main window.
  - b. Next move the cursor over a functional block in the Control Panel window. What happens?

---

What does that indicate? \_\_\_\_\_

3. Examine the Main window.

The Main window provides several ways to issue commands:

- Pulldown and popup menus
  - The command line
- a. Examine the pulldown menus. Use them to:
    - i. Open the *pipe\_scan.v* file in Read Only mode. (*File menu*)
    - ii. Open the **Setup Circuit Clocks** dialogue box. (*Setup menu*)

What does the **Turn on Query Help** button do?

---

Use the **Turn on Query Help** button to find the purpose of the **Automatically Identify and Add** button:

Click **Cancel** to exit; do not implement anything.

- iii. Investigate other **Setup** menu options using the **Turn on Query Help** button.

Click **Cancel** each time to exit; do not implement anything.

- iv. Select **Report > Environment** to identify the current chosen options. Where does the report appear?

- 
- v. Disable the tracking between the Control Panel and Main windows. (*Windows menu*)
  - vi. Load the Hierarchy Browser to explore the hierarchy of the design. (*Windows menu*)

What happens when you click on an instance in the Hierarchy Browser?

---

You can close the Design Hierarchy Browser when you are finished exploring the design.

- vii. Investigate the online manual to explore help options (*Help menu*).

Select **Help > On Commands > Open Reference Page**

What is the function of the **Add Clocks** command?

---

- viii. You also can get help with command usage by typing the Help command followed by the command name at the **SETUP >** prompt. Use this method to get information on the **Add Clocks** command:
- 
- ix. What happens when you type **Help Add <enter>** at the **SETUP >** prompt?
- 

Note that the help query is not case-sensitive.

- b. Examine the Session Transcript area.

The Session Transcript area lists all commands performed and tool messages in different colors. It provides a running log of your session.

- i. Use the *Scan and ATPG Process Guide* in the *DFT Bookcase* to find the answers to the following questions. (*Help menu*)

What colors are used for the following types of messages?

- Commands that have been issued: \_\_\_\_\_

- Error messages: \_\_\_\_\_
  - Warning messages: \_\_\_\_\_
  - Any other output message, e.g. information: \_\_\_\_\_
- ii. Investigate the popup menu in the Session Transcript using the RMB.

Try out the various menu options and fill in **Table 1-1**.

**Table 1-1. Session Transcript Popup Menu Items**

Menu Item	Description
Word Wrap	
Clear Transcript	
Save Transcript	
Font	
Copy Selection	
Open Selected File	
Exit	Terminates the application tool program — do not try yet!

- iii. Save the transcript as a file called *transcript1*.



**Helpful Tip:** In the session transcript, you re-enter a command in the command line by clicking and dragging the LMB to highlight the command, then click the MMB to place it in the command line.

- c. Examine the Command Transcript area.

This area lists all the commands you have performed, thus providing a running log of your session.

- i. Investigate the popup menu in the Command Transcript area.

Try out the various menu options and fill in [Table 1-2..](#)

**Table 1-2. Command Transcript Popup Menu Items**

Menu Item	Description
<b>Clear Command History</b>	
<b>Save Command History</b>	
<b>Previous Command Ctrl+P</b>	
<b>Next Command Ctrl+N</b>	
<b>Exit</b>	Terminates the application tool program — do not try yet!.



**Note**

Helpful Tip: You can repeat a command by double-clicking the command in the Command Transcript.

- d. Examine the command line.

If you are more command oriented, using the command line is another way for you to issue commands to the DFT product.

All of the commands follow the 3-2-1 minimum typing convention. That is, at a minimum you must specify the first three letters of the word, the first two letters of the second word, and the first letter of the last word(s). Minimum typing is not case sensitive, but is usually specified by capital letters. For example, the command Analyze Control Signals can be minimally entered in FastScan as “ANA CO S” or “ana co s”.

- i. Type **REPort DRC Rules** at the command line prompt to identify the DRC status of the design.

ii. Click once on the **REPort DRc Rules** in the Command Transcript area and note that the tool enters this at the command line prompt.

iii. Double click on the **REPort DRc Rules** in the Command Transcript area. What happens?

iv. Click the RMB in the command line to see the command line popup menu.

Try out the various menu options and fill in [Table 1-3](#).

**Table 1-3. Command Line Popup Menu Items**

Menu Item	Description
Cut	
Copy	
Paste	
Delete	
Select All	
Key Bindings	

4. Examine the Control Panel window.

The Control Panel window provides a graphical link to either the functional blocks pane where you can modify the setup, or the flow process pane from which you can modify your run. The window also presents a series of buttons that represent the actions most commonly performed.

a. Examine the Graphics pane area.

The Graphics pane is located on the left of the Control Panel window. The Graphics pane can either show the functional blocks that represent the typical relationship between a core design and the logic being manipulated by the DFT product, or show the process flow blocks that

represent the group of tasks that are part of the DFT product session. Some tools, such as DFTAdvisor or FastScan, have an additional graphics pane at the bottom of the window that changes based on your step in the process. It is called the Task Flow Manager pane.

Examine the Button pane area.

The Button pane is located on the right of the Control Panel window. It provides a list of actions commonly used while running the tool. You can click the LMB on a button to perform the task, or you can click the RMB on that button for **Popup Help** specific to that button.

Leave the tool running and go on to the next exercise.

## Exercise 2: Accessing Information

There are many different types of online help available. The different types include query help, search query, popup help, information messages, Tool Guide help, command usage, online manuals, and the **Help** menu.

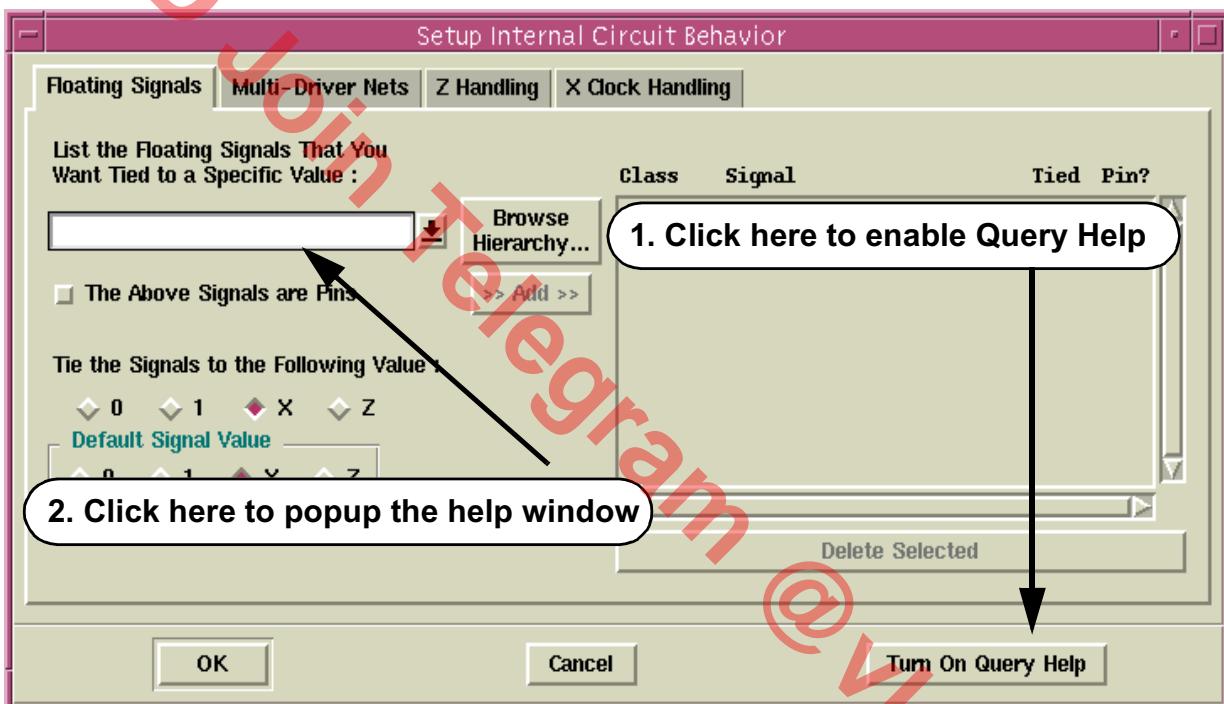
All DFT documentation is available online using Adobe Acrobat Reader. You can browse documents or have a document open to a specific page containing the information on a specific command.

Also, you can access Application and Tech Notes to problem solve specific DFT tool issues from the SupportNet website.

This exercise allows you to explore how to get help in many ways, but does not include accessing the SupportNet website. Access to the web is not provided on the student workstations.

1. FastScan should still be running from the last exercise. If not, invoke it by following the steps in the last exercise.
2. Accessing help:
  - a. **Query Help:** Setup options are available by clicking on any Functional Process Flow block in the Graphics pane as well as from the Setup

menu in the Main window. Use an alternative method to access Query Help in the Setup Internal Circuitry Behavior dialogue box.



**Figure 1-2. Using Query Help**

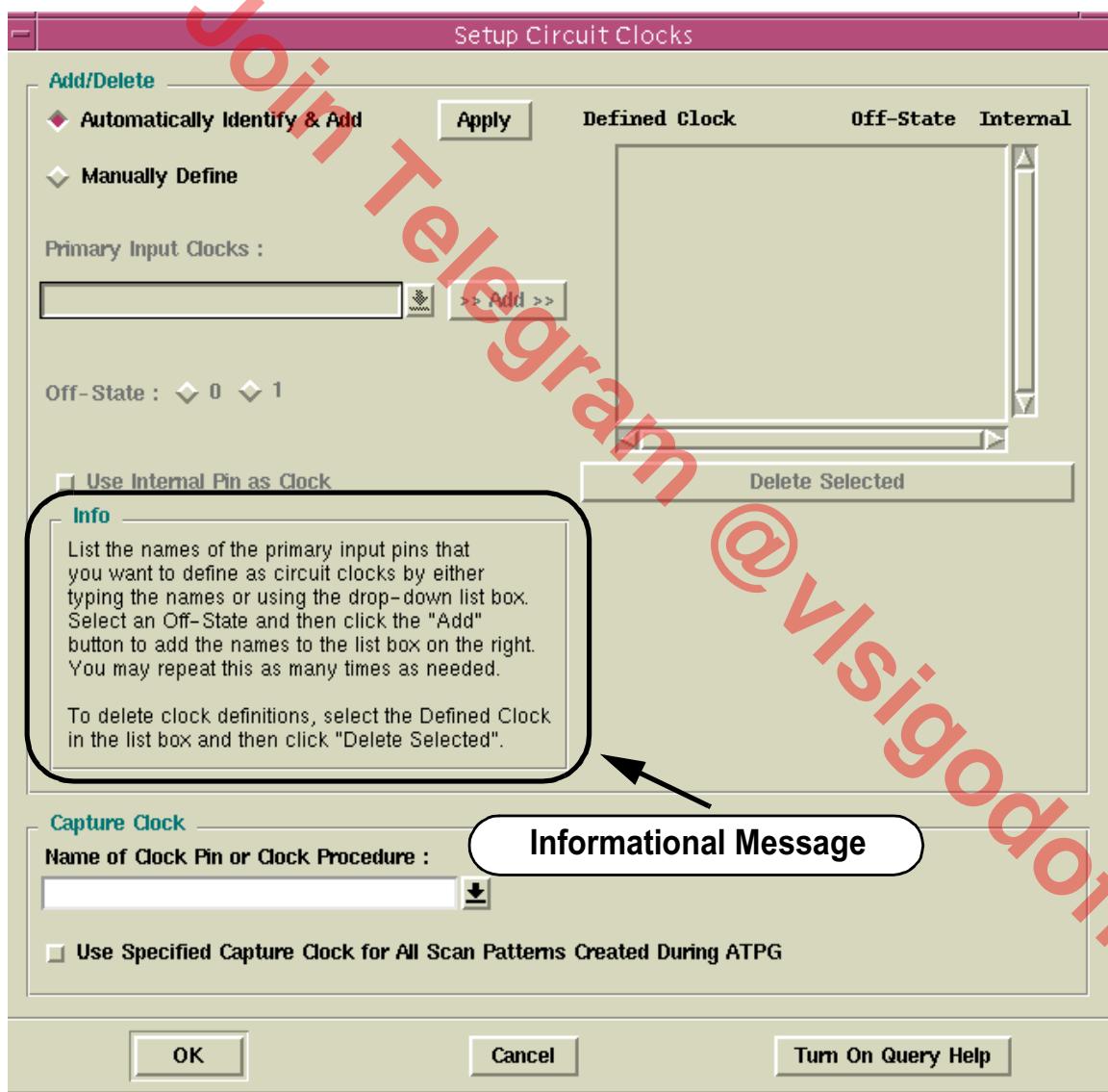
When you have finished investigating this mode, close the box by selecting Cancel.

- b. **Popup Help:** This is available on all active areas of the Control Panel. Using the RMB, click on a functional block, process block or button.

To remove the help window, click on the inside of it using the right or left mouse button. An alternative method to close the window is to press any key except the spacebar while the Control Panel window is active.

- c. **Informational Messages:** Some dialogue boxes contain these to help you understand more about the use of the box or its options.

Examine some of these to find out the types of messages provided.  
 Load the Setup Circuit Clocks dialogue box. (*LMB on Clocks functional block*)



Examine some other dialogue boxes. Always click Cancel when you are done.

- Tool Guide:** Only available for DFTAdvisor, FastScan and FlexTest. Find answers to certain questions on the flow within the tool.

Open the tool guide. (*Help menu*) Explore the guide. Which topic from *What Would You Like Help On?* do you find the topic *What is DFT?* (Hint: you have to click on one of the topics in the ..*Help On?* pane to find it.)

- 
- b. **Command Usage:** Get command syntax from the command line using the **Help** command followed by the full or partial command name.
- Examples:

**SETUP> Help add**

(Lists all add commands.)

**SETUP> Help add clocks**

(Gives command syntax for this command.)

**SETUP> Help add clocks -manual**

(Opens the reference page for this command.)

Try this out on some commands.

- c. **Online Help:** Application documentation is provided online in PDF format. Open manuals using the **Help** menu or the **Go To Manual** in the Query Help messages.

You also can open a separate shell window to open a PDF viewer:

**shell> mgcdocs**

This command brings up the Mentor Graphics Bookcase. There is a search facility available in the viewer under the **MGC** menu. (*Unix only. At present, Adobe does not support multiple-document, full-text search in Acrobat on the Linux platform. Linux users may have to use the Find function in individual documents.*)

Use online help and the search facility to answer the following questions.

- i. What is the purpose of the enhanced procedure file?

---

---

- ii. Find out what the **-Auto\_fix** switch does when used with the **Analyze Control** signals command.

---

---

- iii. Investigate other documentation and close the PDF reader when you are finished.

- d. If you have not exited the tool, do so now.

- e. You can invoke DFT documentation quickly from the shell prompt before the tool is up and running by using its shell invocation along with the **-manual** switch.

```
shell> fastscan -manual
```

You may continue to investigate the documentation or close the reader.

- f. **SupportNet:** This option is available with Internet access. (Not in this exercise.) Visit [www.mentor.com/supportnet/](http://www.mentor.com/supportnet/) when you have the opportunity.

## Test Your Knowledge (Optional)

1. List two ways to issue commands in the command line window.

---

2. How do you disable main window tracking?

---

3. What is the significance of commands and tool messages in red text?

---

4. How do you get the Command Transcript Popup box to appear?

---

5. How can you get help with commands using the command line?

---

6. What is the difference between functional blocks and process blocks in the Graphic Pane?

---

7. List various ways to access online help.

---

---

---

## **Lab Summary**

Now that you have completed the Basic Concepts and DFT flow lab, you should know:

- How to invoke FastScan.
- How to use many of the features of the Graphical User Interface. (GUI)
- How to get help with commands and tool usage.

## Module 2

# Full Scan DFT Flow

### Objectives

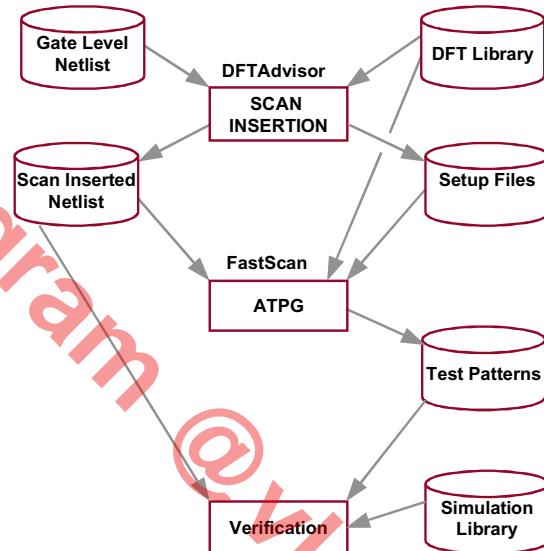
Upon completion of this module, you will be able to:

- Explain the basic process of scan insertion and pattern generation.
- Describe how DFTAdvisor changes the design.
- Use DFTAdvisor to insert full scan.
- Write a scan-inserted netlist file.
- Write ATPG setup files.
- Use DFTInsight to troubleshoot minor DRC violations.
- Use FastScan to create, compress, and save patterns.

# Module Topics

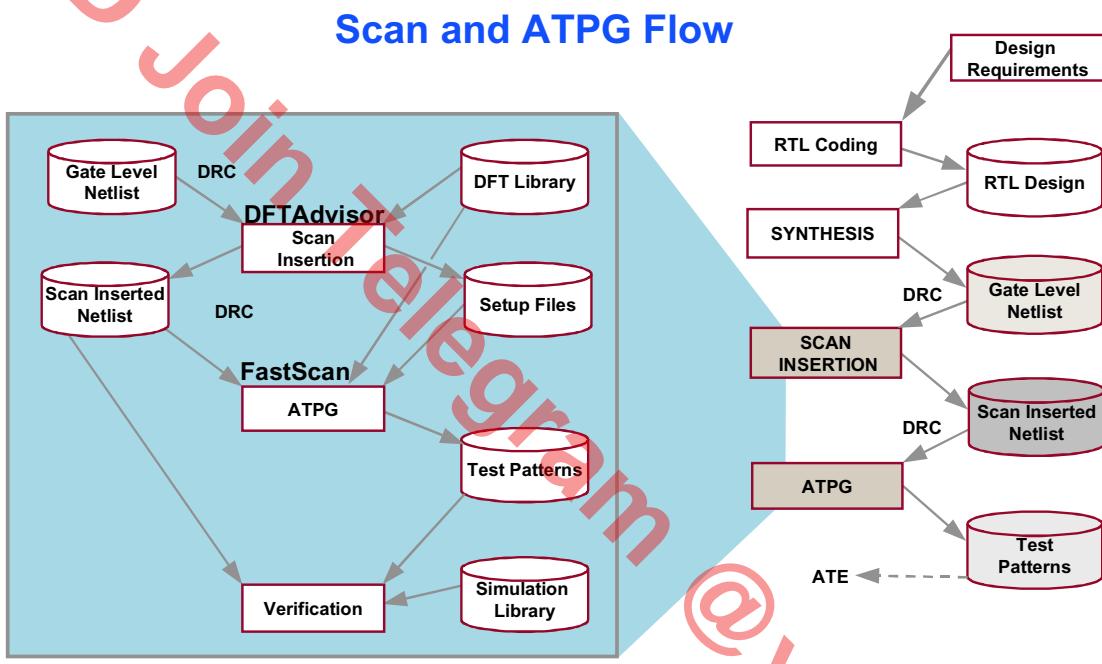
## Module Topics

- ◆ This module addresses following topics:
  - Scan and ATPG flow
  - DFTAdvisor tool flow
  - DRC and DFTInsight
  - FastScan tool flow



## Notes:

# Scan and ATPG Flow



2-3 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

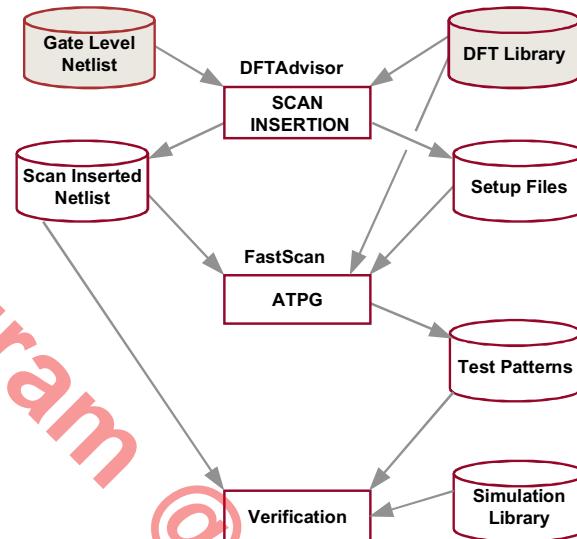
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Circuit Setup

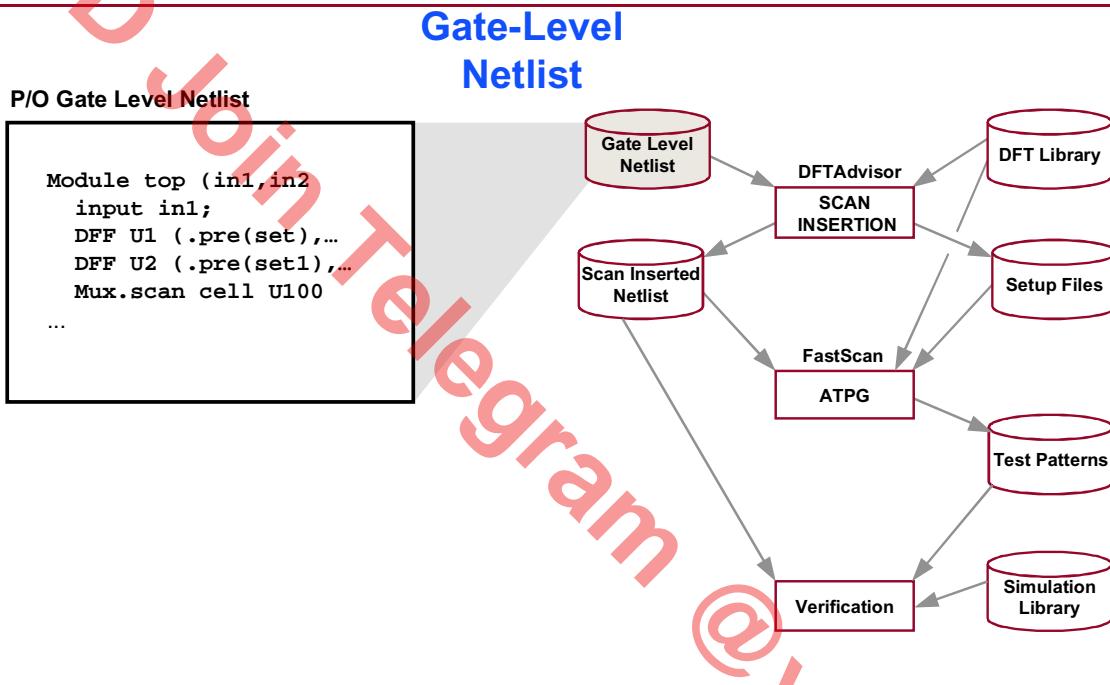
## Circuit Setup

- ◆ What you have:
  - A gate-level (non-scan) netlist
  - A DFT library:
    - Used by DFTAdvisor
    - Used by FastScan



## Notes:

# Gate-Level Netlist



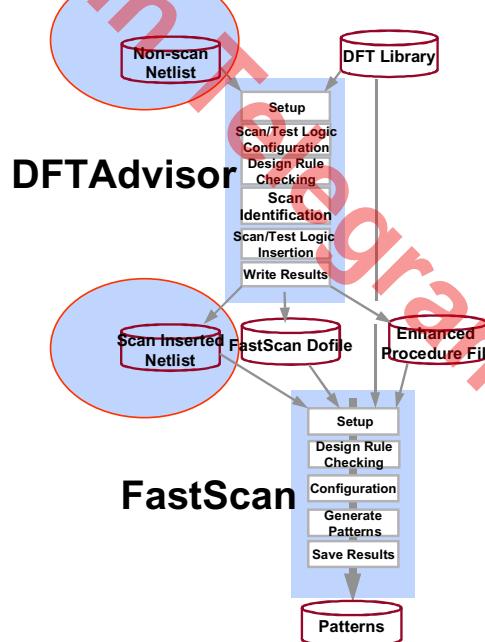
2-5 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Auto Black Boxing for Incomplete Netlists

## Auto Black Boxing for Incomplete Netlists



2-6 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Black Boxes

### Black Boxes

```
// FastScan v8.9_3.10 Fri Jun 8 13:13:39 PDT 2001
// Copyright (c) Mentor Graphics Corporation, 1992-2001, All Rights Reserved.
// UNPUBLISHED, LICENSED SOFTWARE.
// CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
// PROPERTY OF MENTOR GRAPHICS CORP OR ITS LICENSORS.
//
// USE OF THIS SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS.
//
// Mentor Graphics software executing under Sun SPARC SunOS.
// 32 bit version
//
// Compiling library ...
// Reading Verilog Netlist ...
// Reading Verilog file top.v
// Finished reading file top.v
// Warning: Following modules are undefined:
// BU110
// TDN1J
// DPL61
// AN220
// Use "add black box -auto" to treat as black boxes
// command: set sys m atpg
// Error: Instance of undefined model found, check black
// box warnings and use Add Black Box -auto to
// make these default black box models
//
// command: report black box -undefined
// Undefined Modules:
// BU110
// TDN1J
// DPL61
// AN220
//
// command: ADD BLack Box -Auto
...
```

- ◆ Allow analysis of incomplete designs.
    - Isolates analog blocks.
    - Isolates proprietary IP.
  - ◆ Warns of undefined model instances.
    - The undefined model is not black boxed.
- Use the **ADD BLack Box -Auto** command to black box all undefined models.

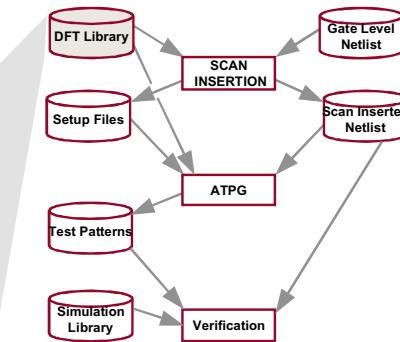
### Notes:

# DFT Library

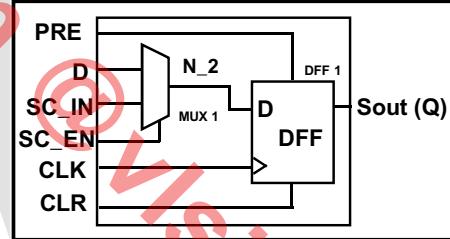
**DFT Model Description of Scan Cell (DFF)**

```

=====
Model: DFF
=====
model DFF ( PRE,CLR,CLK,D, Q, QB) (
  input ( PRE, CLR, D ) ()
    input(CLK) (clock = rise_edge)
    output(Q QB) (primitive = _dff(PRE,CLR,CLK,D,QB);
)
=====
Model: MUX_SCAN_CELL
=====
model MUX_SCAN_CELL (PRE, CLR, SC_IN, SC_EN, CLK, D, Q,
QB) (
  scan_definition (
    type = mux_scan
    scan_in = SC_IN;
    scan_enable = SC_EN;
    scan_out = Q, QB;
    non_scan_model = DFF ( PRE, CLR, CLK, D, Q, QB);
  )
  input (PRE, CLR, SC_IN, SC_EN, CLK ())
  intern(N_2) (primitive = _mux mux1 (D, SC_IN,
SC_EN,N_2))
  output(Q, QB) (primitive = _dff dff1(PRE, CLR, CLK,
N_2, Q, QB));
)
=====
```



Which maps non-scan models (DFF)  
to library replacements:  
**Mux\_Scan\_Cell Library Model**

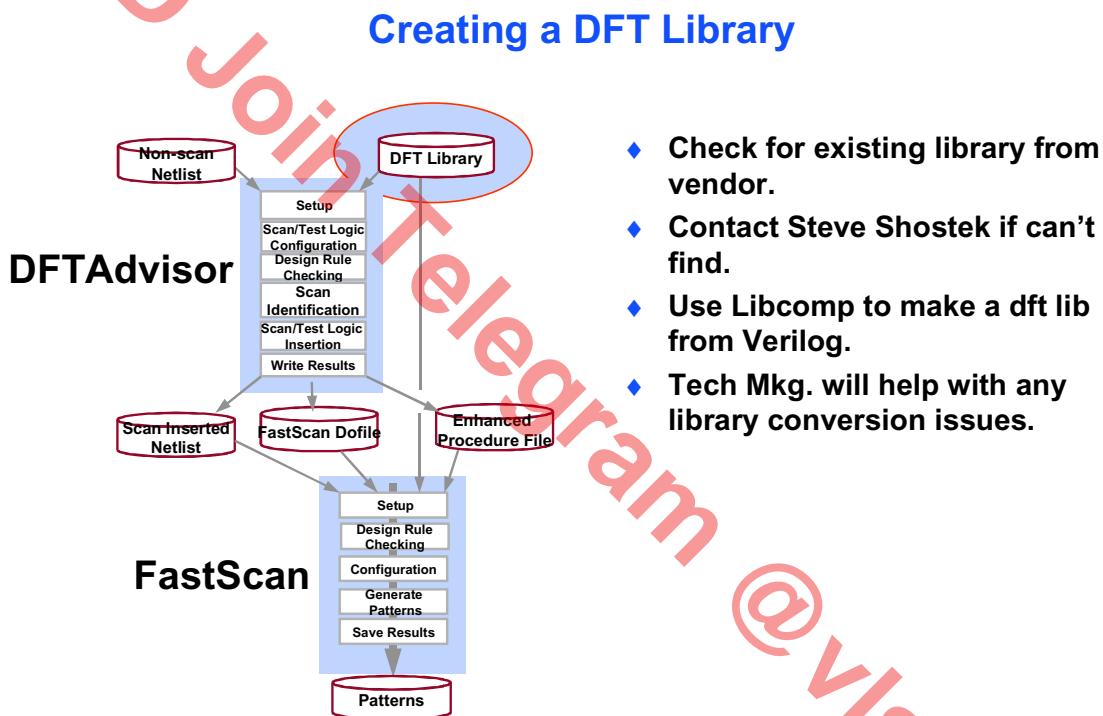


2-8 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Creating a DFT Library



2-9 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Automatic Generation of DFT Libraries

## Automatic Generation of DFT Libraries

- Automatically converts Verilog simulation library
  - Translates and optimizes UDPs
  - Automated verification of translated models & reports coverage per model
  - Reduces the effort in creating an ATPG library



```
libcomp <verilog_file> -dofile <file_name>
```

## Notes:

## Include File Handling

### Include File Handling

- Relative paths resolved from including file
- Allows locating including and included files in a directory using simple include text
- Backward compatibility still finds files in CWD
- Warning given if file found in both locations
- Applies to netlists, DFT Libraries, dofiles
- Example:      *fastscan vlog/topfile.v*

If *topfile.v* contains ‘*include “block1.v”* it will be found in *vlog* directory as expected.

### Notes:

# Invoking DFTAdvisor

## Invoking DFTAdvisor

- ◆ The DFTAdvisor executable resides in the Mentor Graphics tree at:
  - \$MGC\_HOME/bin/dftadvisor
- ◆ Invocation requirements:
  - A design netlist in one of the following formats:
    - Verilog
    - VHDL
    - EDIF
    - TDL
  - DFT library

### Invocation:

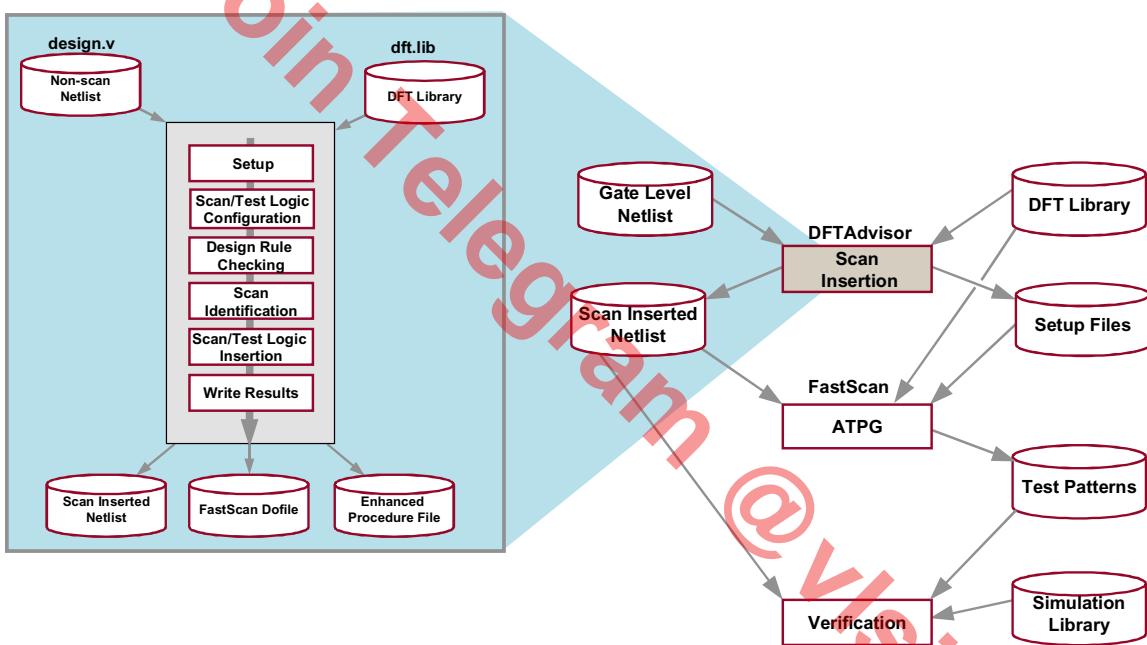
```
shell> dftadvisor design.v -verilog \  
-lib dft.lib -log transcript.log -replace
```

Use the -Log <filename> option:  
to write detailed session information to a file

Helpful tip

## Notes:

# DFTAdvisor Tool Flow: An Overview



2-13 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Command Structure

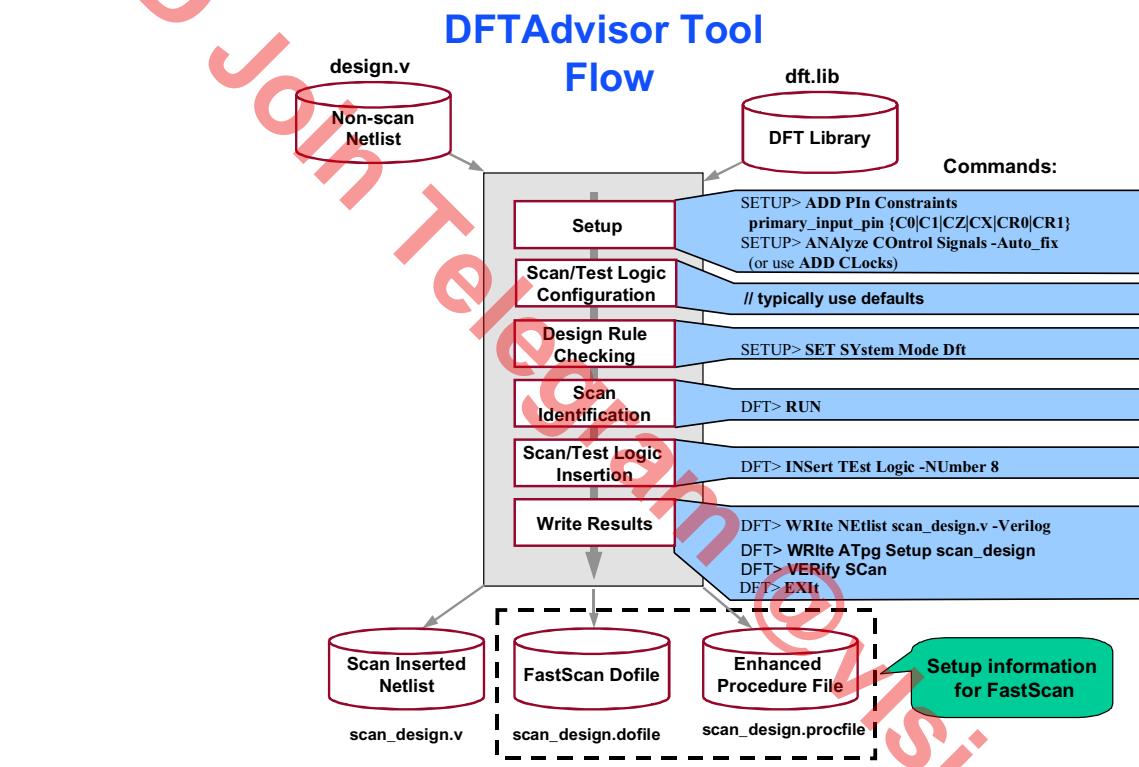
## Command Structure

- ◆ DFTAdvisor and FastScan use this command structure:
  - >ADD  
(to define a condition)
  - >REPort  
(to display anything added)
  - >DElete  
(to remove a condition)
  - >SET/SETup  
(to define global settings)
  - >WRItE/SAVe  
(to generate files)

```
> ADD PIn Constraint Input1 C1
> ANALyze COntrol Signals -Auto_fix
> REPort CLocks
```

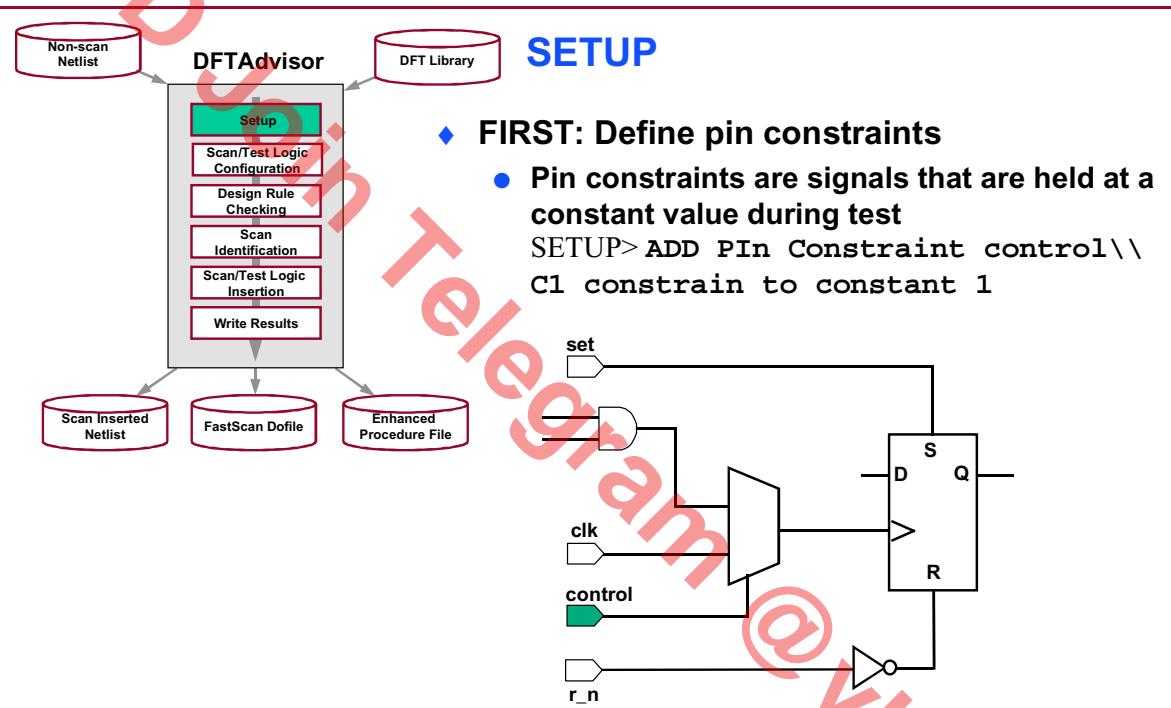
## Notes:

# DFTAdvisor Tool Flow

2-15 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

**SETUP**

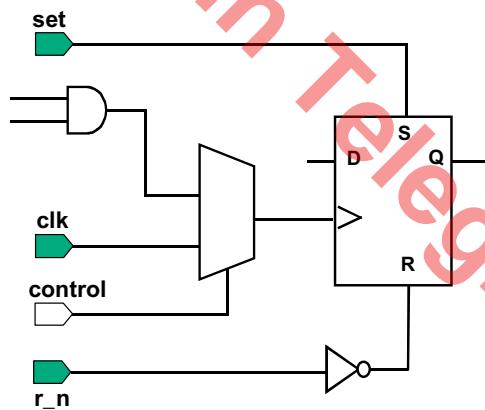
2-16 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

**Notes:**

## SETUP (Cont.)

### SETUP (Cont.)



#### ♦ SECOND: Define clocks

- Clocks are primary input signals that asynchronously change the state of sequential logic elements
  - clocks
  - sets
  - resets
  - RAM read/write clocks

SETUP> ADD Clock 0 clk set  
SETUP> ADD Clock 1 r\_n

Off state

Primary input pin

### Notes:

## SETUP (Cont.)

### SETUP (Cont.)

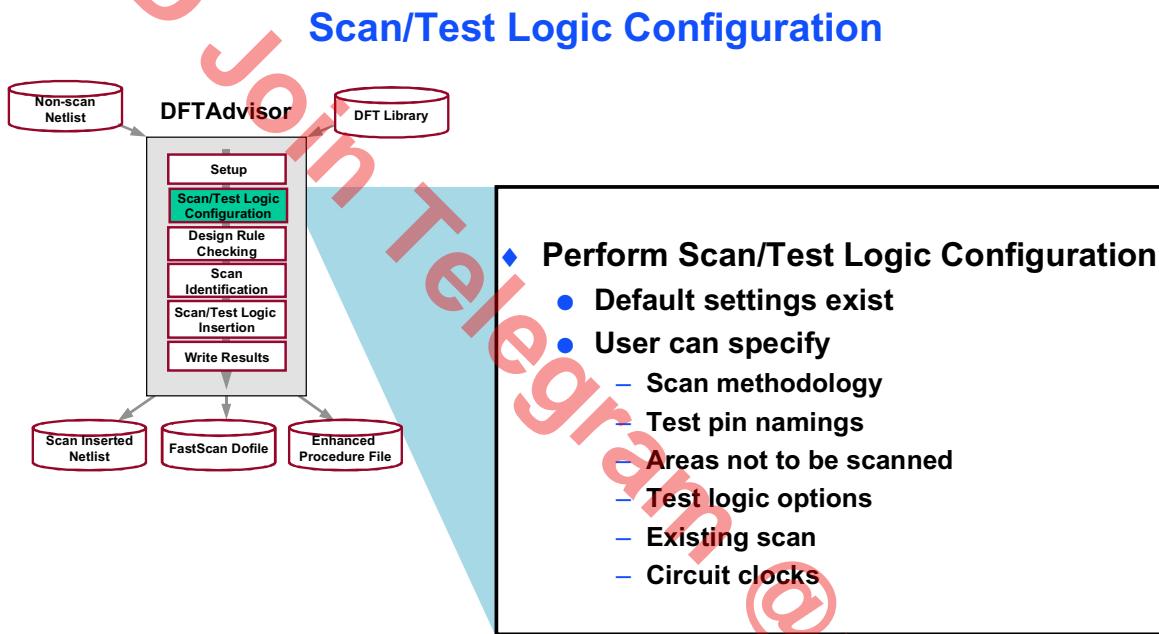
- ◆ The ANALyze COnrol Signals command identifies and optionally defines primary inputs of control signals
  - Use the -Auto\_fix option to automatically define all identified primary inputs

```
SETUP>ANALyze Control Signals -Auto_fix
```

Helpful tip

### Notes:

# Scan/Test Logic Configuration



2-19 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Set Test Logic Configuration

## Set Test Logic Configuration

- ♦ User defined setting options in SETUP mode:

- Scan Methodology

```
SET SCan Type {Mux_scan | Lssd | Clocked_scan}
```

- Test Pin namings:

```
SETUP SCan Insertion -SEN -TEN
```

```
SETUP SCan Pins {Input | Output}
```

```
ADD SScan Pins chain_name scan_input_pin scan_output_pin  
[Clock pin_name]
```

- Test Logic Options

```
SET TEST Logic
```

```
{ -Set {ON | OFF} | -Reset {ON | OFF} |  
-Clock {ON | OFF} | -Tristate {ON | OFF | Decoded} |  
-Bidi {ON | Scan | OFF} |  
-Ram {ON | OFF} |  
} ...
```

- Areas not to scan

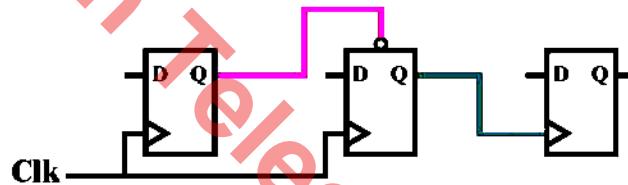
```
ADD NONscan Instances pathname... {-INstance |  
-Control_signal | -Module}
```

## Notes:

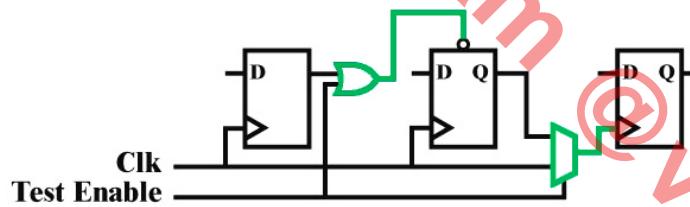
## Adding Test Logic

### Adding Test Logic

- Some designs contain uncontrollable clock circuitry



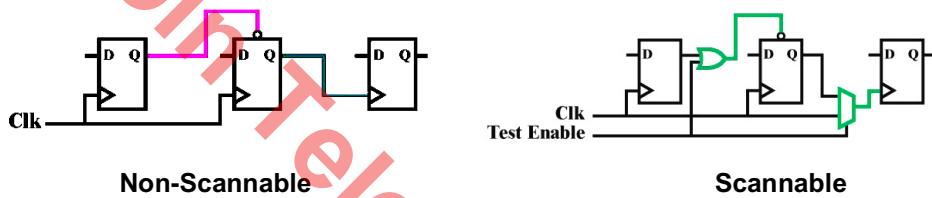
- Test logic is added to make the circuit scannable



### Notes:

## Adding Test Logic (Cont.)

### Adding Test Logic (Cont.)



- ◆ Required modifications:

- Specify which types of control lines are controllable

```
SETUP> SET Test Logic -Set ON -Clock on
```

- Disable set/reset

```
SETUP> ADD CELL Models -Type OR <cell name>
```

- Activate test mode with “test enable” signal

```
SETUP> ADD CELL Models -Type MUX selector data0 data  
<cell name>
```

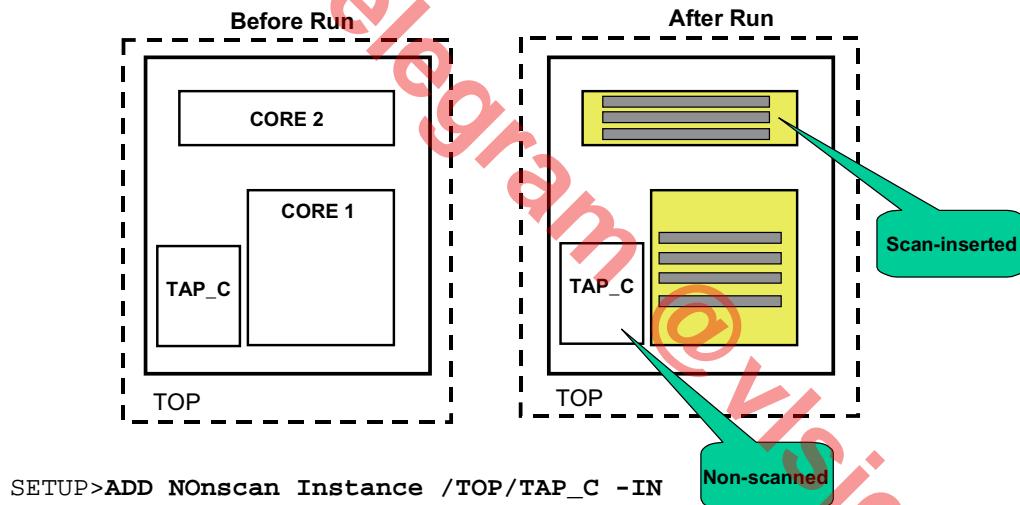
— Or: gates for test logic can be defined by “cell-type” in the library model

### Notes:

## Set Test Logic Configuration (Defining Non-scan Areas)

### Set Test Logic Configuration (Defining Non-Scan Areas)

- Excluding the TAP controller from scan

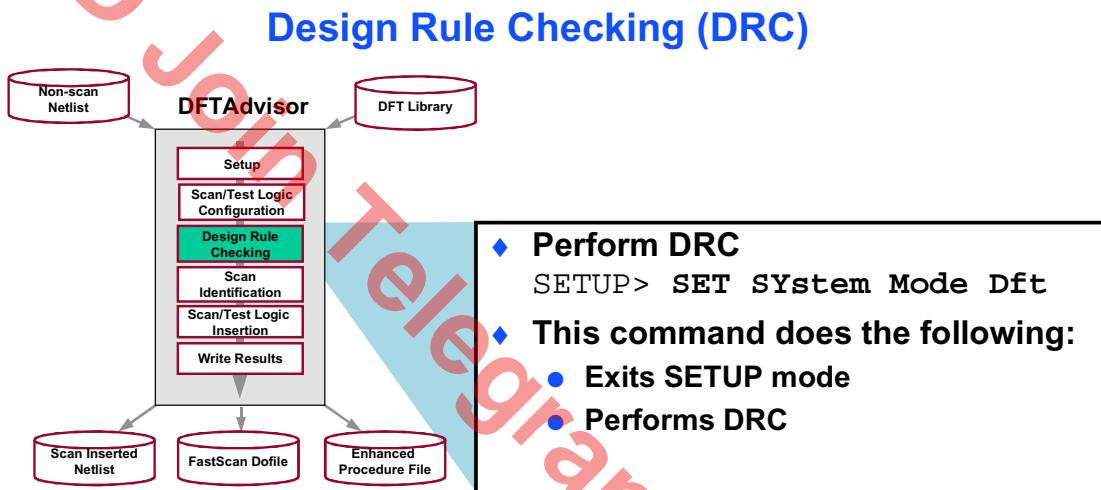


2-23 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

### Notes:

# Design Rule Checking (DRC)



## Notes:

## DRC

### DRC

- ◆ DFT tools perform DRC to ensure the following:
  - Correct scan operation
  - Proper handling of special design situations
  - Correct syntax and operation of test procedure file
- ◆ 8 types of rules check over 150 DRC violations
  - DRCs
  - Scan-specific DRCs

## Notes:

## DRC Basics

### DRC Basics

- ◆ Passing rules checking is vital for the successful use of the DFT tools
- ◆ Rule violations are handled in four ways:
  1. **Error** message:
    - rules checking is terminated and tool remains in SETUP mode
  2. **Warning** message:
    - Indicates the number of violations
  3. **Note**:
    - Summary message that displays number of violations
  4. **Ignore**:
    - No message given

SETUP> SET DRC Handling

User-defined handling  
of displayed messages

### Notes:

## DRC Basics (Cont.)

### DRC Basics (Cont.)

- Each rule is assigned a rule and violation identification number

For example:

S2-3

Occurrence:  
3rd occurrence  
of violation

Rule type:  
Scannability Rule

Specific rule:  
Ability to capture data  
with defined clocks

### Notes:

## Types of DRCs

### Types of DRCs

- ◆ **Procedure rules (P rules)**
  - Ensure test procedure file is syntactically and functionally correct
- ◆ **Data rules (D rules)**
  - Check the stability of the data in the scan chains
- ◆ **Clock rules (C rules)**
  - Verify clock operation
- ◆ **RAM rules (A rules)**
  - Check for testability conditions of embedded RAMs
- ◆ **Extra rules (E rules)**
  - Check for potential problems-usually ignored by the tool
    - Bus contention or data shifted through scan chains

### Notes:

## Scan Specific DRCs

### Scan Specific DRCs

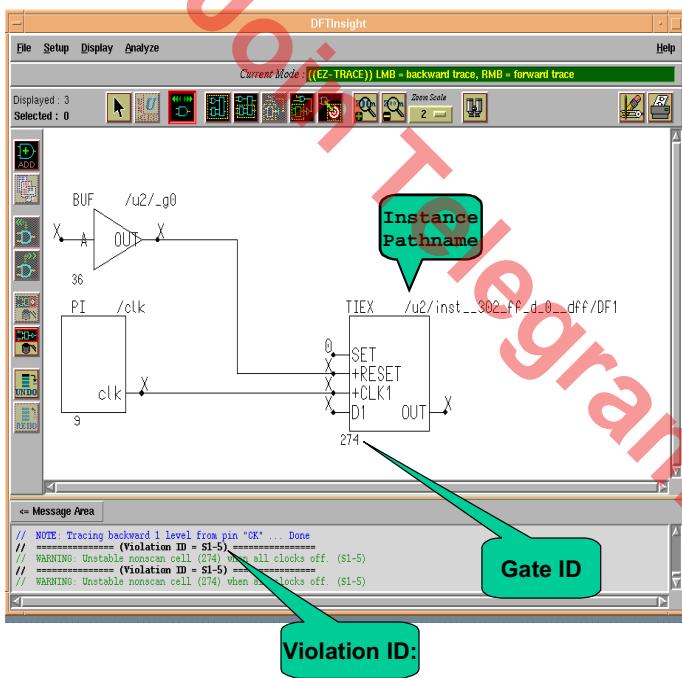
- ◆ General rules (G rules)
  - Check for gross scan definition errors
- ◆ Trace rules (T rules)
  - Use test procedure files to trace scan chains
- ◆ Scannability rules (S rules)
  - Ensure that DFTAdvisor can safely convert a sequential element into a scan element
  - Check scannability during DRC
    - S1 rule checking:
      - Ensure when all clocks off- sequential elements are stable and inactive
    - S2 rule checking:
      - Ensure that defined clocks capture data when all other clocks are off

DFT> REPORT DRC Rules

Displays all DRC rules or data for a specific violation

### Notes:

# DFTInsight



A DFT tool that provides an interactive, graphical debugging environment

Displays a partial netlist containing:

- Pertinent instances based on the following:
  - Paths
  - Scan circuitry
  - Circuitry involved in rules violations
- Simulation and analysis data

## Notes:

# Troubleshooting DRC Violations: Reporting S1 Fails

## Troubleshooting DRC Violations: Reporting S1 Fails

- ◆ Problem: DFTAdvisor does not pass DRC

### SETUP> SET SYSTEM Mode Dft

```
// Flattening process completed, design_cells=51 library_primitives=64 sim_gates=282 PIs=9 POs=6.  
// -----  
// Begin circuit learning analyses.  
// -----  
// Equivalent gates=6 classes=3 CPU time=0.00 sec.  
// Learning completed, implications=0, tied_gates=7, CPU time=0.00 sec.  
// -----  
// Begin scan chain identification process, memory elements = 7.  
// -----  
// -----  
// Begin scannable cell rules checking for 7 nonscan memory elements.  
// -----  
// WARNING: There were 7 scannability rule S1 fails (unstable nonscan cells when clocks off).  
// 7 non-scan memory elements identified as non-scannable.  
// 0 non-scan memory elements identified as scannable.
```

### DFT> REPort DRC Rule S1

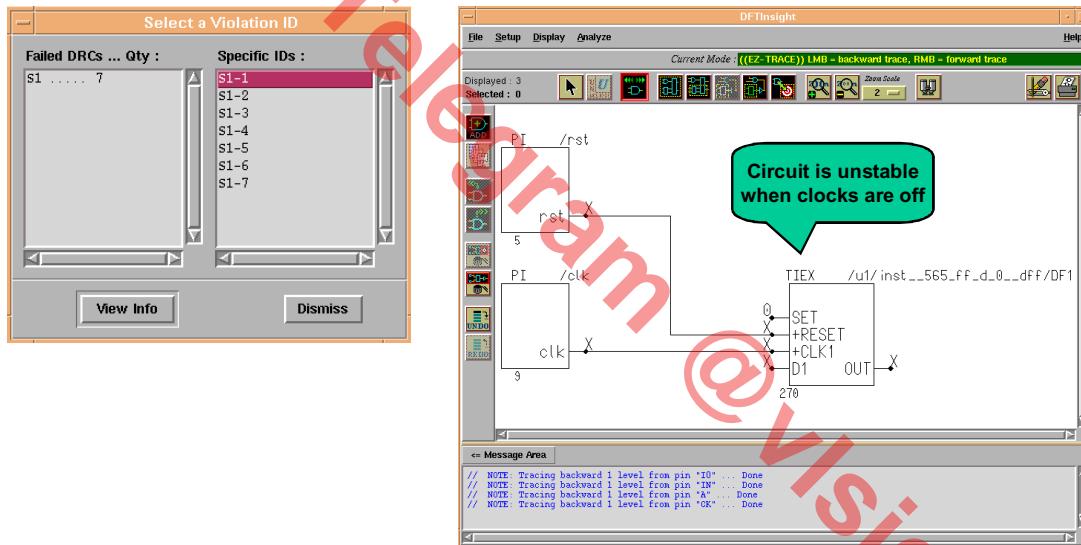
```
// WARNING: Unstable nonscan cell /u1/inst_565_ff_d_0_dff/DF1(270) when all clocks are off. (S1-1)  
// WARNING: Unstable nonscan cell /u1/inst_565_ff_d_1_13/DF1(271) when all clocks are off. (S1-2)  
// WARNING: Unstable nonscan cell /u1/inst_565_ff_d_2_13/DF1(272) when all clocks are off. (S1-3)  
...
```

## Notes:

# Viewing the Problem: Analyzing S1 Violations

## Viewing the Problem: Analyzing S1 Violations

- Use DFTInsight to analyze the S1 violations



2-32 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Troubleshooting DRC Violations: Adding Clocks

## Troubleshooting DRC Violations: Adding Clocks

### ♦ Solution: ADD Clocks

#### SETUP> ANALyze COntrOl Signals -Auto\_fix

```
// Begin control signals identification analysis.  
// -----  
// Identified 1 clock control primary inputs.  
// /clk (9) with off-state = 0.  
// Identified 0 set control primary inputs.  
// Identified 1 reset control primary inputs.  
// /rst (5) with off-state = 0.  
// Identified 0 read control primary inputs.  
// Identified 0 write control primary inputs.  
// -----  
// Total number of internal lines is 21 (7 clocks, 7 sets , 7 resets, 0 reads, 0 writes).  
// Total number of controlled internal lines is 14 (7 clocks, 0 sets , 7 resets, 0 reads, 0 writes).  
// Total number of uncontrolled internal lines is 7 (0 clocks, 7 sets , 0 resets, 0 reads, 0 writes).  
// Total number of added primary input controls 2 (1 clocks, 0 sets , 1 resets, 0 reads, 0 writes).
```

#### SETUP> SET SYstem Mode Dft

```
// -----  
// Begin scan chain identification process, memory elements = 7.  
// -----  
// Begin scannable cell rules checking for 7 nonscan memory elements.  
// -----  
// 7 non-scan memory elements identified as scannable.  
// -----  
// Begin scan clock rules checking.  
// -----  
// 2 scan clock/set/reset lines have been identified.
```

// All scan clocks successfully passed off-state check.

// All scan clocks successfully passed capture ability check.

## Notes:

# Troubleshooting DRC Violations: Reporting S2 Fails

## Troubleshooting DRC Violations: Reporting S2 Fails

### ◆ Problem: DFTAdvisor does not pass DRC

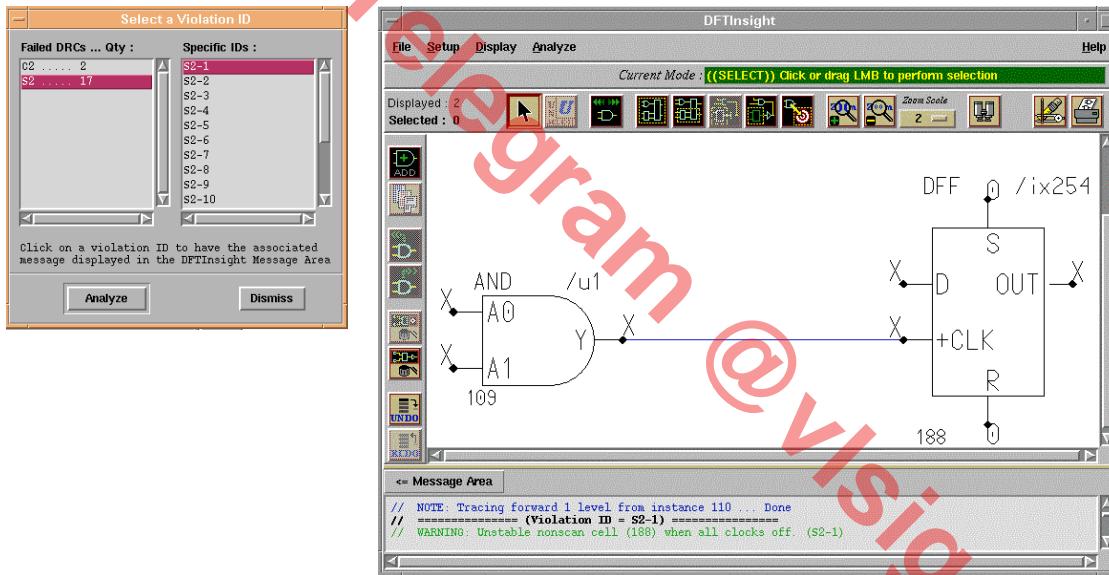
```
// command: SET SYstem Mode dft
// Flattening process completed, design_cells=93 leaf_cells=93 library_primitives=121 sim_gates=224 PIs=27 POs=20.
//
// -----
// Begin circuit learning analyses.
//
// -----
// Equivalent gates=0 classes=0 CPU time=0.00 sec.
// Learning completed, implications=0, tied_gates=28, CPU time=0.00 sec.
//
// -----
// Begin scan chain identification process, memory elements = 17.
//
// -----
// Begin scannable cell rules checking for 17 nonscan memory elements.
//
// -----
// WARNING: There were 17 scannability rule S2 fails (clock capture ability check).
// 17 non-scan memory elements identified as non-scannable.
// 0 non-scan memory elements identified as scannable.
//
// -----
SETUP> REPort DRC Rule S2
// WARNING: Clock capture ability check failed on /ix254(188). (S2-1)
```

## Notes:

## Viewing the Problem: Analyzing S2 Violations

### Viewing the Problem: Analyzing S2 Violations

- ♦ Use DFTInsight to analyze the S2 violations



2-35 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Troubleshooting DRC S2 Violation

## Troubleshooting DRC S2 Violation

### ◆ Solution: ADD Test Clock logic

```
// command: SET TEst Logic -Clock ON  
// command: set system mode dft  
// -----  
// Begin scan chain identification process, memory elements = 17.  
// -----  
// -----  
// Begin scannable cell rules checking for 17 nonscan memory elements.  
// -----  
// WARNING: There were 17 scannability rule S2 fails (clock capture ability check).  
// 17 non-scan memory elements identified as non-scannable.  
// 0 non-scan memory elements identified as scannable.  
// Checking test clocks  
// -----  
// Number of test clocks required = 1  
// command: run  
// Number of targeted sequential instances = 17  
// Performing scan identification ...  
// Total sequential instances identified = 17  
// command: INSert TEst Logic  
// WARNING: Flattened model has been freed  
// command: REPort TEst Logic -instance  
/u01_mux21_macro  
Number of mux21_macro inserted (instance based) = 1  
New pins added in top module: counter16  
/test_clk  
/scan_in1  
/scan_en  
Number of new pins inserted = 3
```

2-36 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

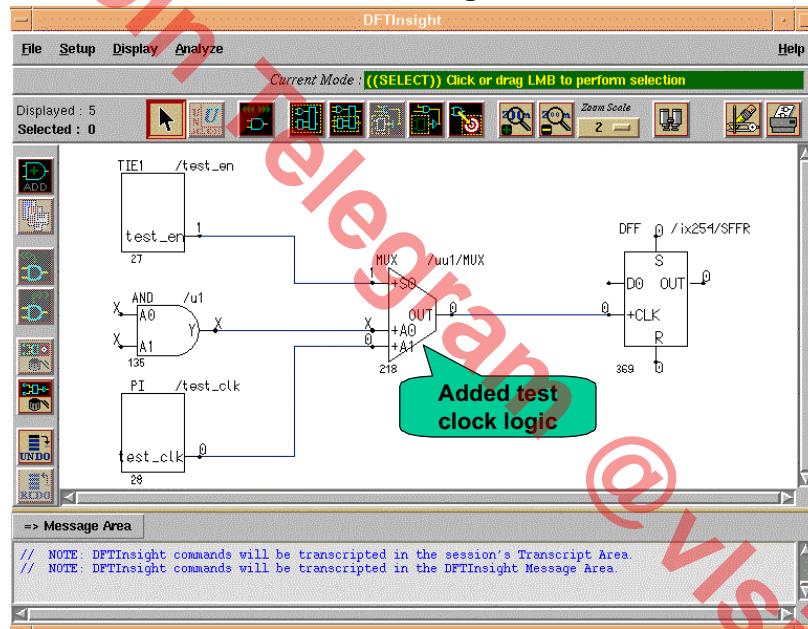
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Viewing the Added TestClock Logic

## Viewing the Added Test Clock Logic

- ◆ DFTAdvisor added test clock logic

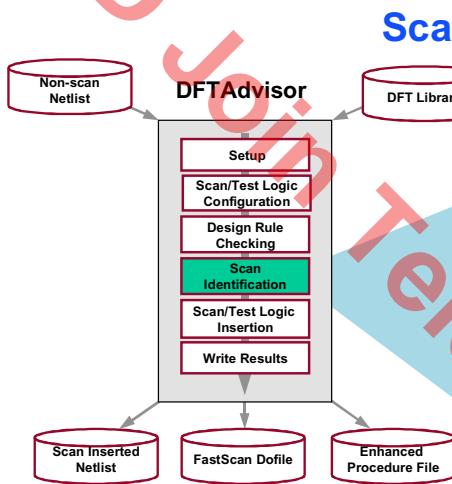


2-37 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Scan Identification

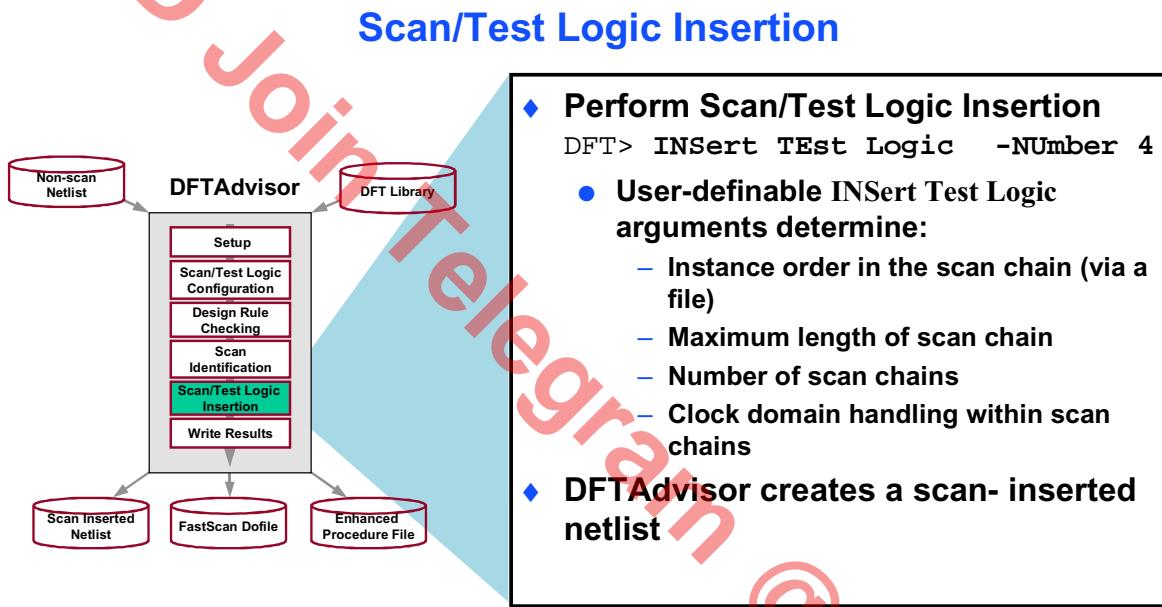


## Scan Identification

- ◆ **Perform Scan Identification**  
DFT> RUN
- ◆ **DFTAdvisor will do the following:**
  - Identify which instances to convert to scan
  - Determine netlist changes
    - Does not alter netlist
- ◆ **To display results:**  
DFT> REPORT STATISTICS
  - Sequential instances
  - Non-scan and scan instances
  - Instances scannable with test logic

## Notes:

# Scan/Test Logic Insertion



## Notes:

## Scan/Test Logic Insertion (Cont.)

### Scan/Test Logic Insertion (Cont.)

- ◆ To display results:

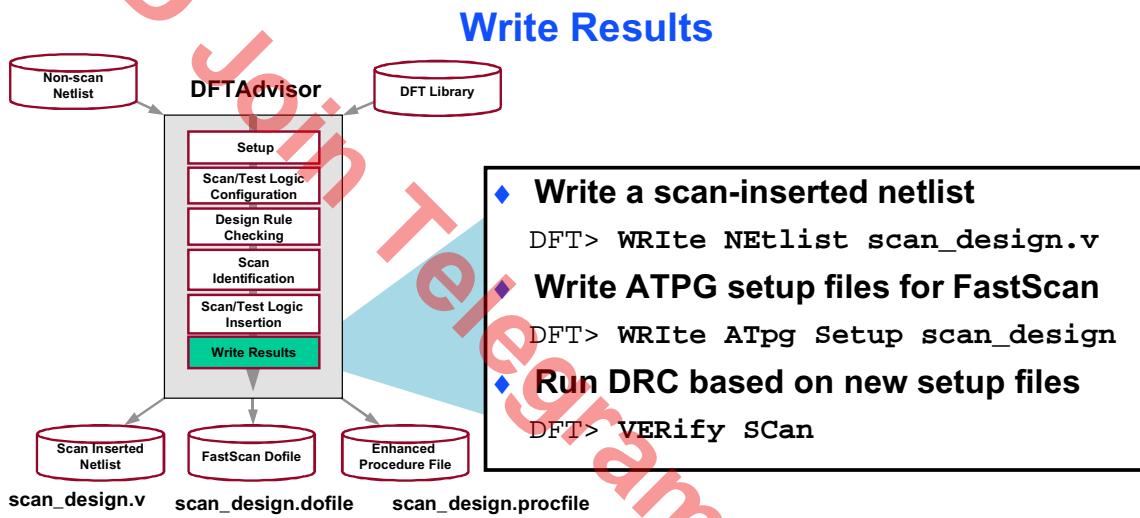
DFT> REPort SCan Chains

DFT> REPort TEst Logic

```
DFT> INSert TESt Logic -NUmber 2
// WARNING: Flattened model has been freed
DFT> REPort SCan Chains
chain = chain1 group = dummy input = /scan_in1 output = /scan_out1 length = 4
scan_enable = /scan_en clock = /clk
reset = /rst
chain = chain2 group = dummy input = /scan_in2 output = /scan_out2 length = 3
scan_enable = /scan_en clock = /clk
reset = /rst
DFT> REPort TESt Logic
New pins added in top module: example_ckt
/scan_in1
/scan_out1
/scan_in2
/scan_out2
/scan_en
Number of new pins inserted = 5
```

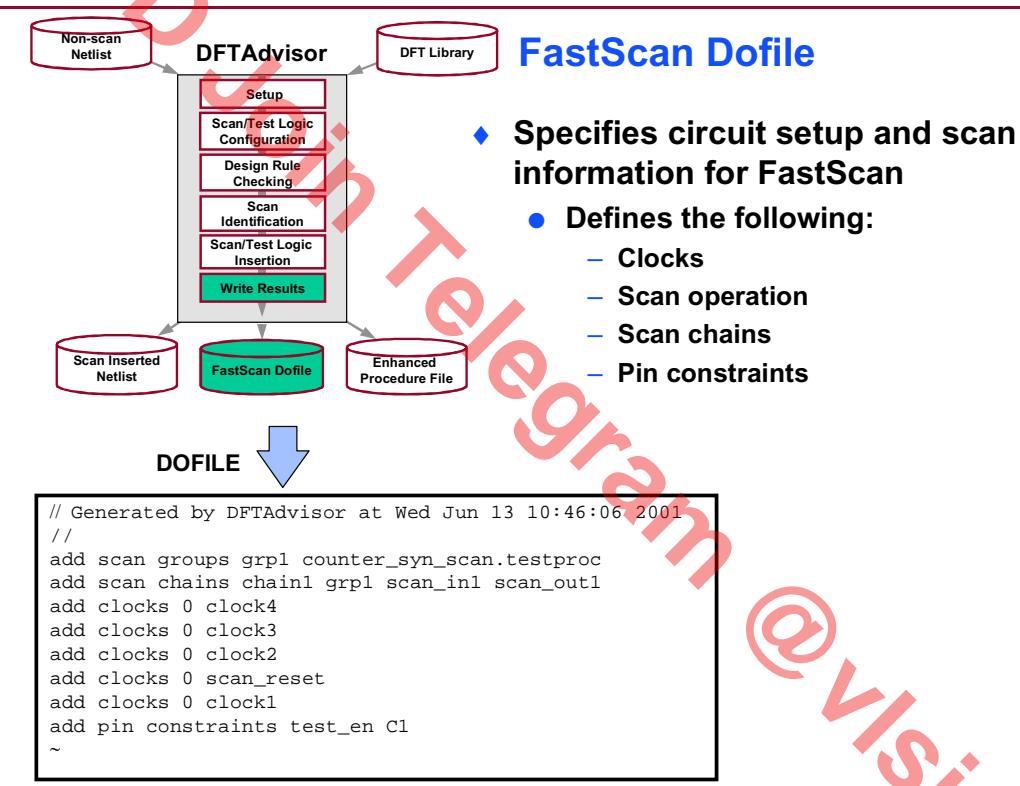
### Notes:

# Write Results



## Notes:

## FastScan Dofile

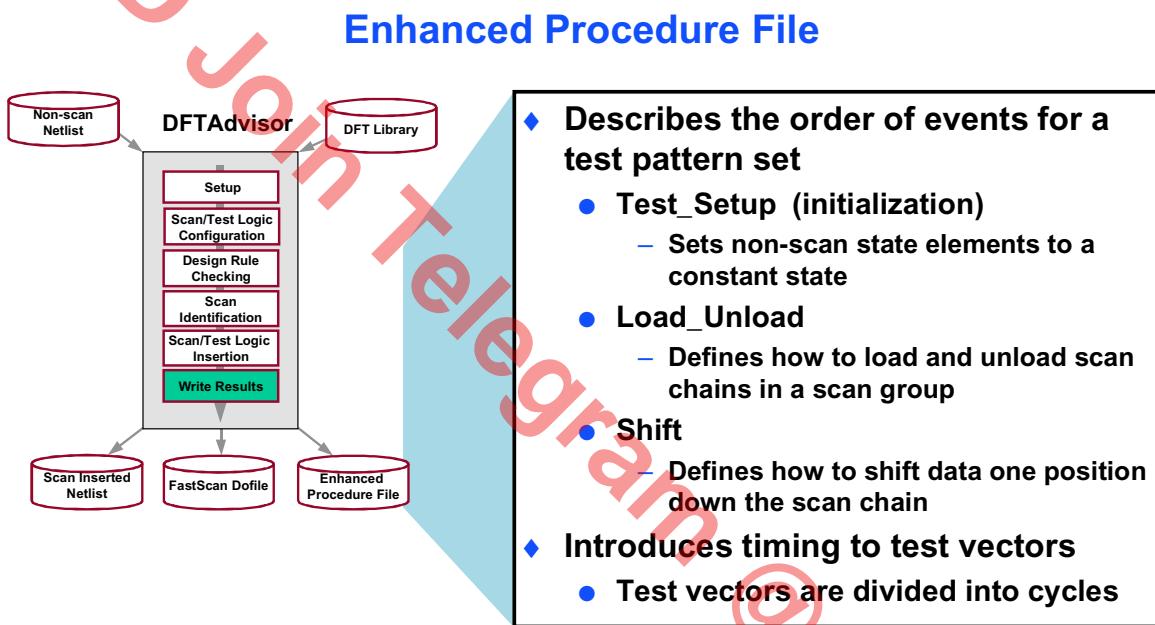


2-42 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

### Notes:

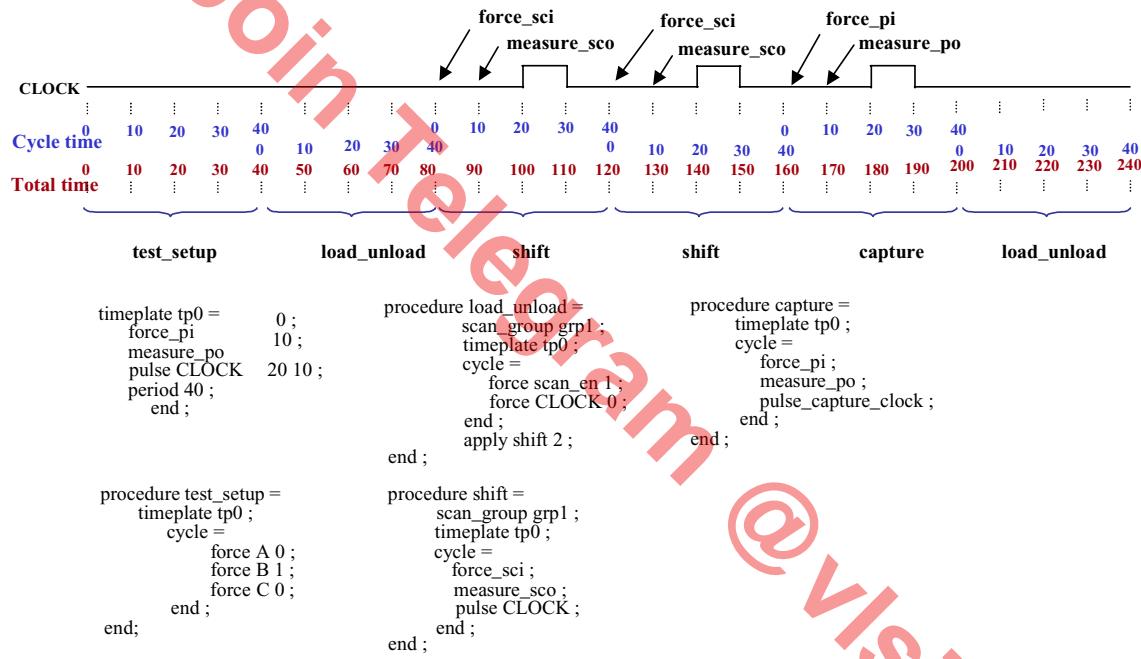
# Enhanced Procedure File



## Notes:

## Enhanced Procedure File (Cont.)

### Enhanced Procedure File (Cont.)



### Notes:

## Invoking FastScan

### Invoking FastScan

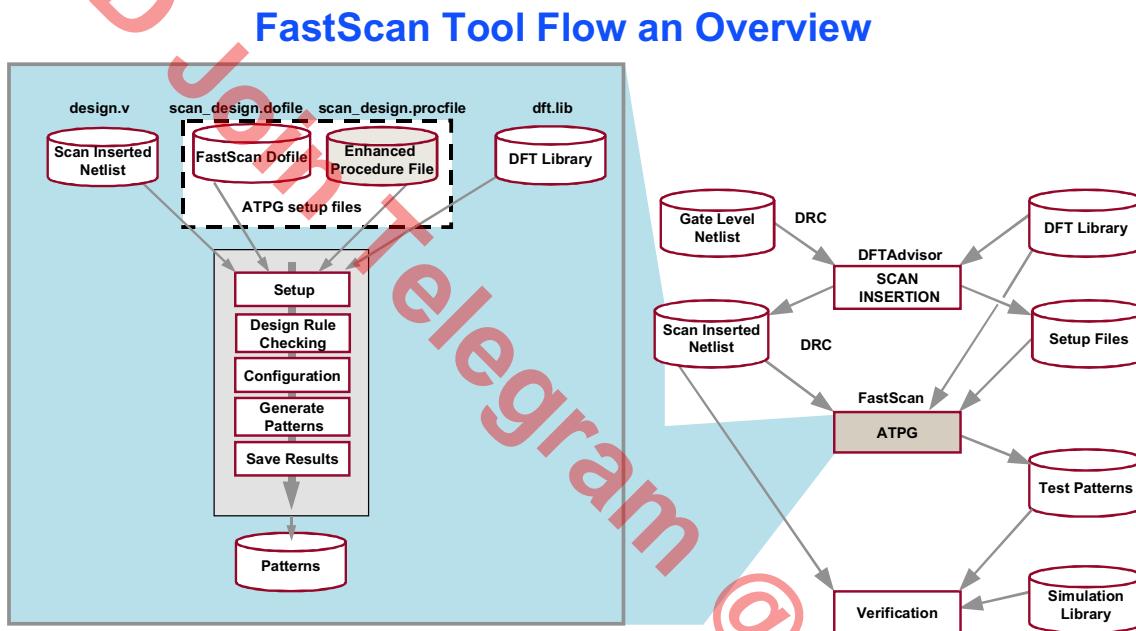
- ◆ The FastScan executable resides in the Mentor Graphics tree at:
  - \$MGC\_HOME/bin/fastscan
- ◆ Invocation requirements:
  - Scan-inserted netlist
  - DFT library

#### Invocation:

```
shell> fastscan design.v -verilog \
-lib dft.lib -log transcript.log -replace
```

### Notes:

# FastScan Tool Flow an Overview

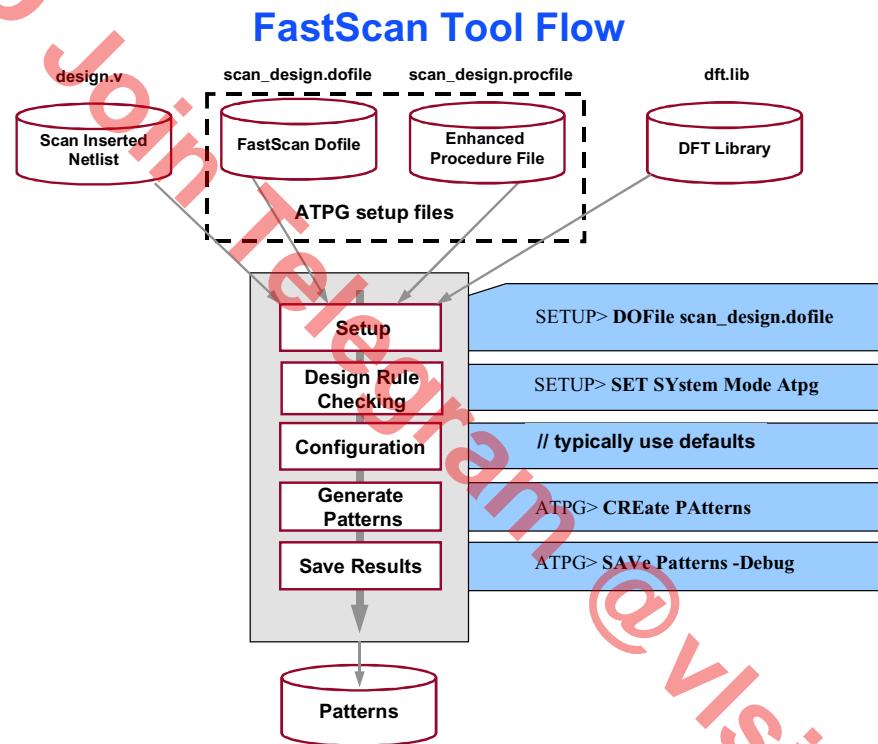


2-46 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

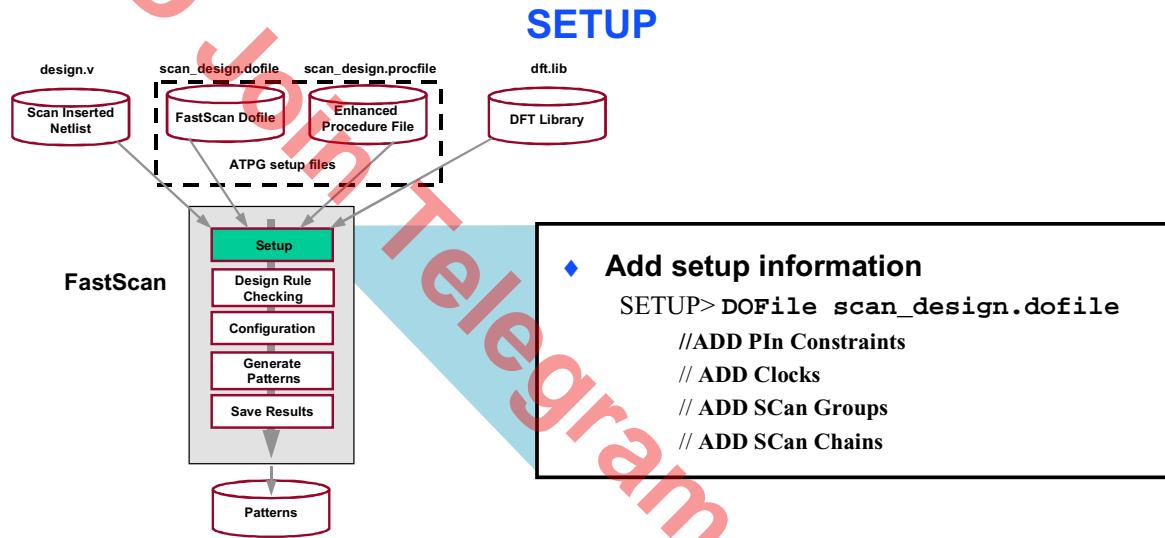
# FastScan Tool Flow



2-47 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

**SETUP****Notes:**

## SETUP (Cont.)

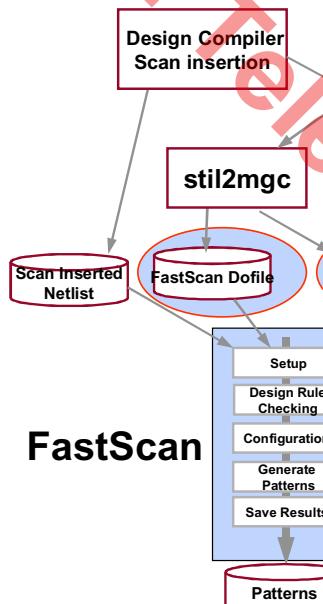
### SETUP (Cont.)

- ◆ Setting up the circuit includes several tasks
  - Constraining primary inputs
    - Holds specified pins at a constant value during ATPG
  - Adding clocks
    - Adds scan or non-scan clock pins to the clock list
  - Adding scan groups
    - Adds a scan chain group
  - Adding scan chains
    - Adds a scan chain to a scan group

### Notes:

# FastScan ATPG in a DC Scan Insertion Flow

## FastScan ATPG in a DC Scan Insertion Flow



2-50 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

# ATPG Setup Files

## ATPG Setup Files

- ◆ The following applies to ATPG setup files:
  - They are used to enter ATPG setup information  
>DOFile <name.dofiles>
  - They can be created manually or interactively
  - They can be created from STIL files

```
SHELL> stil2mgc -stil design.stp -tpf new_design.tpf -dofile
new_dofile.do
```

### DOFILE

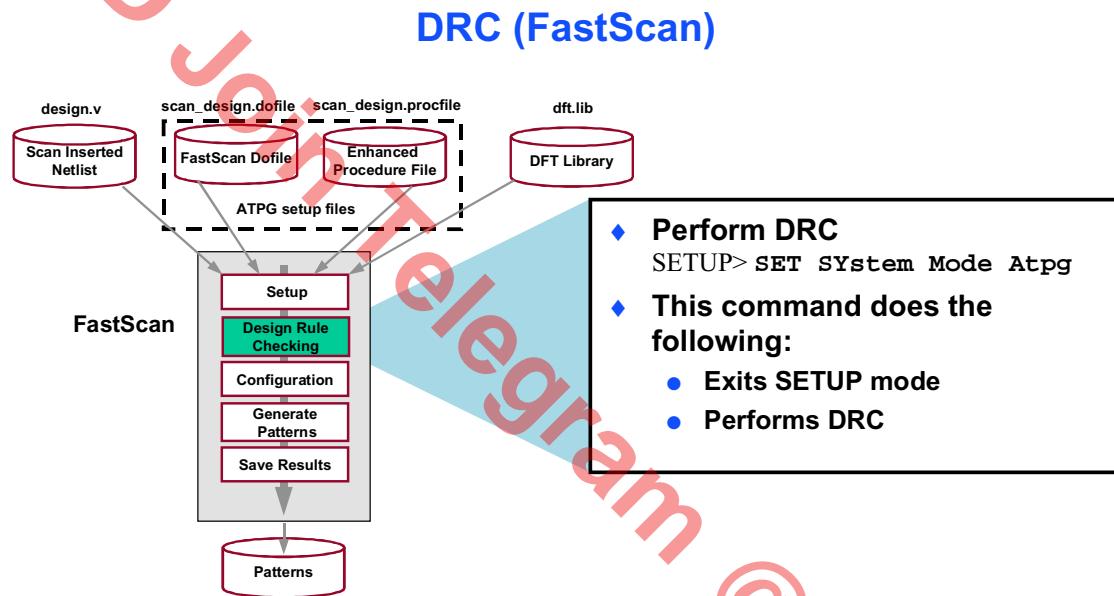
```
//  
// Generated by DFTAdvisor at Fri Jun 22 14:09:04 2001  
//  
add scan groups grp1 /scratch1/project_2/Pipe_test/l_pipe_arithoper_net_scan.v.testproc  
add scan chains chain1 grp1 scan_in1 d_out[0]  
add clocks 0 clk  
~
```

### TEST PROCEDURE FILE

```
// Generated by DFTAdvisor at Fri Jun 22 14:09:04 2001  
set time scale 1.000000 ns ; procedure shift =  
timeplate gen_tp1 = scan_group grp1 ;  
force_pi 0 ; timeplate gen_tp1 ;  
measure_po 10 ; cycle =  
pulse_elk 20 10; force_sci ;  
period 40 ; measure_sco ;  
end; pulse_clk ; end;
```

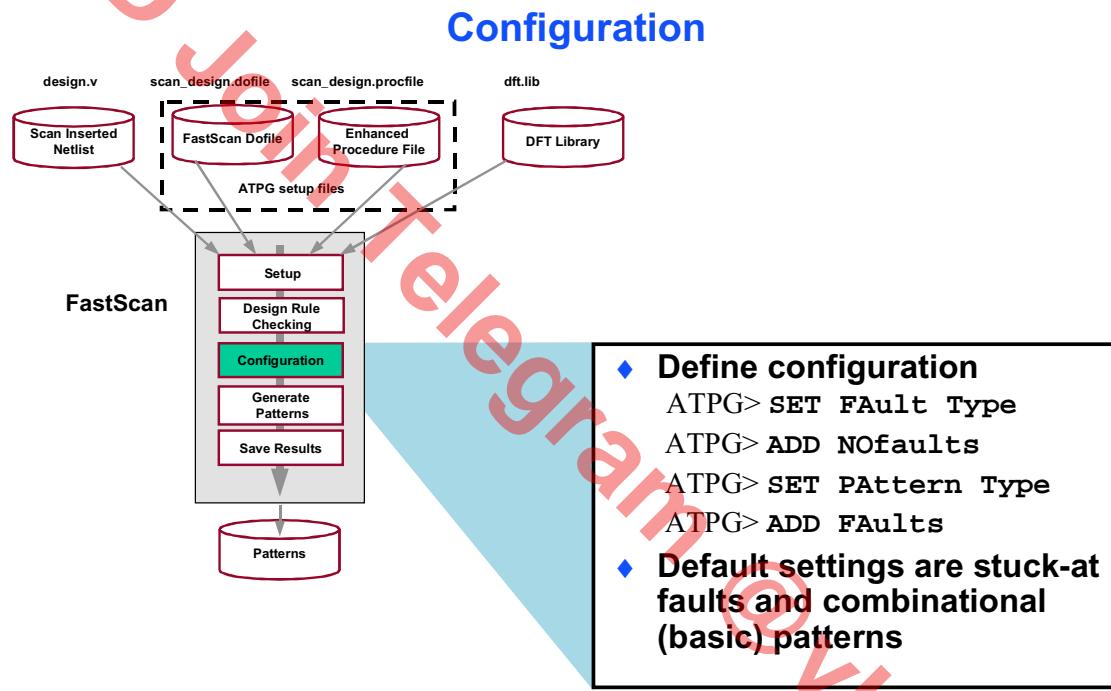
## Notes:

## DRC (FastScan)



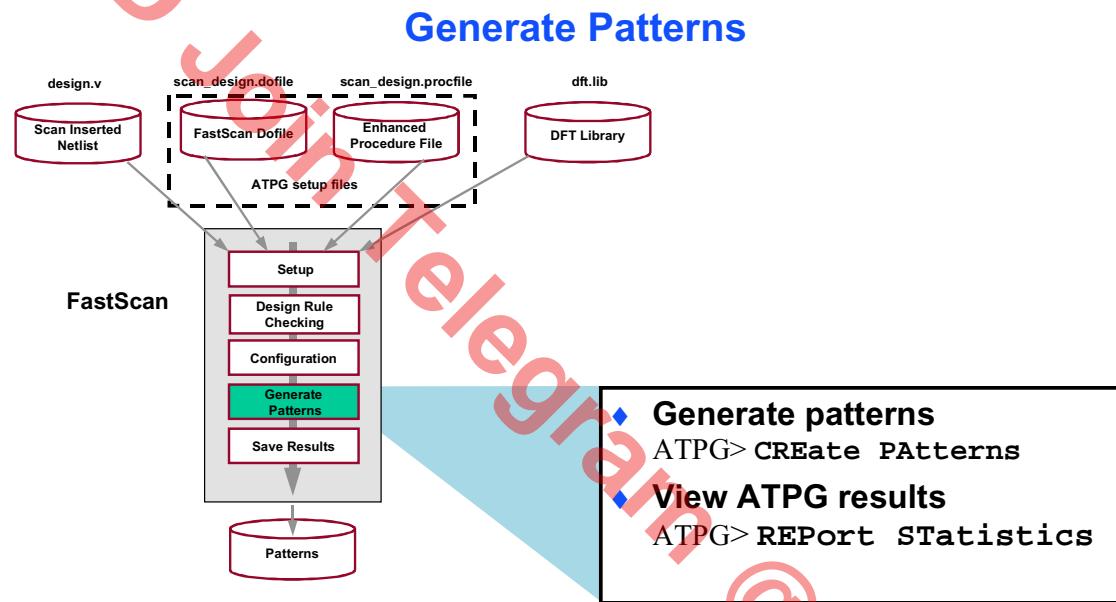
## Notes:

# Configuration



## Notes:

# Generate Patterns



2-54 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

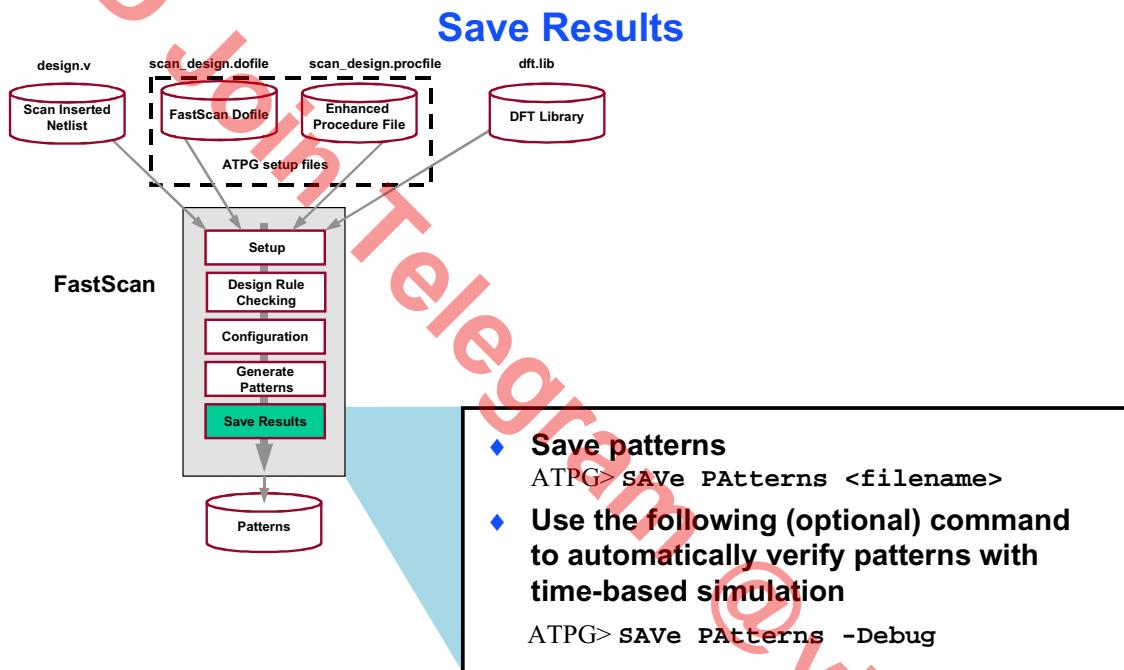
## Create Patterns

### Create Patterns

- ◆ CREAtE PAtterns is the recommended command to create a compressed high coverage pattern set
- ◆ Automatically performs the following operations:
  - Deletes existing patterns
  - Add Faults -All
  - Performs a reset state
  - Turns on ATPG compression, turns off random patterns, and executes Set Decision Order -Random
  - Performs ATPG

### Notes:

# Save Results



## Notes:

# Saving Test Patterns

## Saving Test Patterns

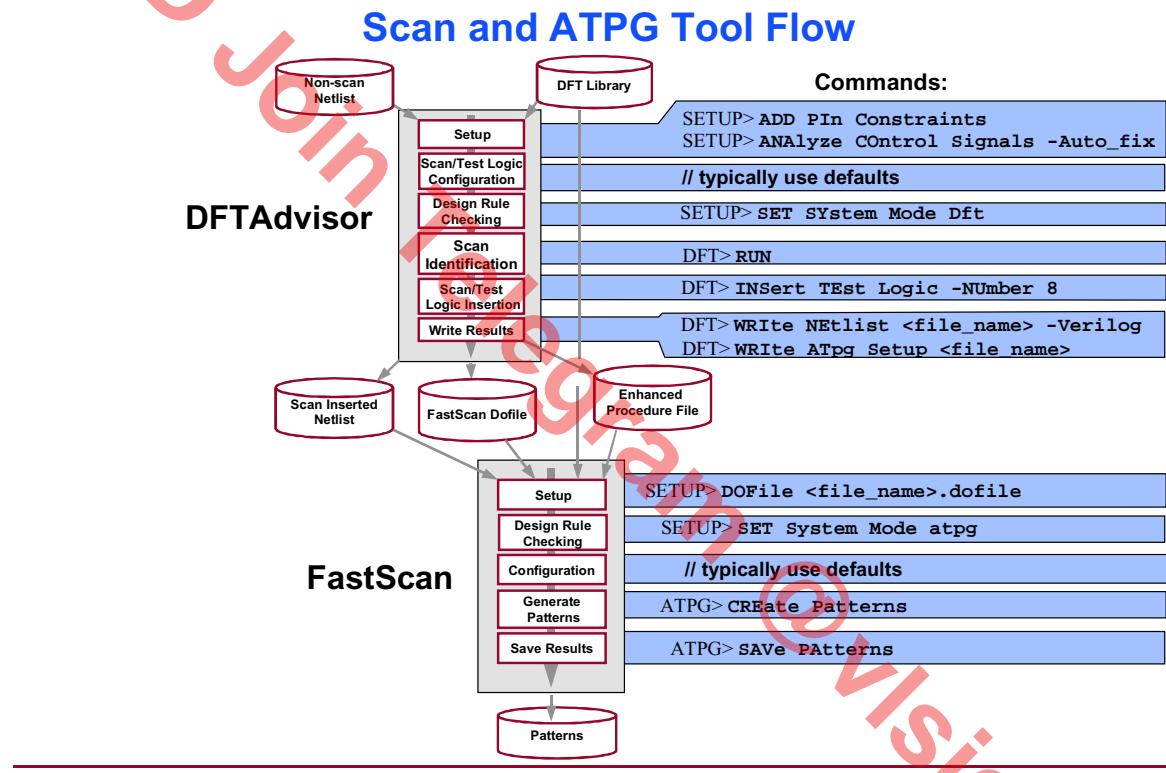
- ◆ You can save patterns in many ASIC vendor formats
  - For example:
    - WGL
    - STIL
    - TITDL
- ◆ You can save as Verilog or VHDL test benches
- ◆ Recommended saving practices
  - ASCII or binary
    - used later in FastScan
  - ASIC vendor format for tester
  - Verilog or VHDL test bench
    - used later for time-based simulation/verification

### Saving Patterns

```
:  
-----  
#test_patterns          48  
#simulated_patterns    416  
CPU_time (secs)        0.4  
-----  
// command: save patterns pat1_ser.v -procfile -serial -verilog -replace  
// command: save patterns pat1_par.v -procfile -parallel -verilog -replace  
// command: save patterns pat1_ascii -ascii -replace  
// command: exit -d
```

## Notes:

# Scan and ATPG Tool Flow



2-58 • Design-for-Test: Scan and ATPG:  
Full Scan DFT Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Lab: Full Scan DFT Flow

### Objectives

- Invoke DFTAdvisor to insert full scan in a design.
- Write a scan-inserted netlist file.
- Write ATPG setup files.
- Invoke FastScan to create, compress, and save patterns.
- Invoke DFTInsight to troubleshoot minor Design Rules Checking (DRC) violations.

### List of Exercises

- Exercise 3: Scan and ATPG Tool Flow
- Exercise 4: Scan and ATPG Tool Flow (With a Design Rule Violation)

### Getting Started

1. Log in to your workstation if you are not already logged in.
2. Change to the \$ATPGNW/lab2/exercise\_3 directory.

```
shell> cd $ATPGNW/lab2/exercise_3
```

### Exercise 3: Scan and ATPG Tool Flow

In this exercise, you invoke DFTAdvisor on a simple design to insert full scan. You use DFTAdvisor to write a scan-based netlist file and ATPG setup files. Then you invoke FastScan using the ATPG setup file information to create, compress, and save patterns.

This lab gives you experience using the tools in a typical scan and ATPG tool flow utilizing default configurations.

The Verilog design contains no errors, so you can use it as a future reference to help you navigate through a scan and ATPG tool flow.

Understanding and applying these concepts assists you when you begin to make customized configurations to enhance tool performance.

1. Invoke DFTAdvisor.

```
shell> dftadvisor
```

This invokes the DFTAdvisor Invocation Arguments dialogue box. You must enter a design, a design format, and an ATPG library in order to invoke the Graphical User Interface (GUI).

- a. Enter the following in the appropriate dialogue boxes. Use the Browse button to quickly find the design and design library, found in the exercise\_3 and libraries\_1\_to\_4 directories.

Design: pipe\_net\_noscan.v

Design Format: Verilog

Library: libraries\_1\_to\_4/adk.atpg

- b. Click on the Invoke DFTAdvisor button. DFTAdvisor should now be up and running with the Main and Control Panel Windows open.

DFTAdvisor and FastScan have various modes of operation called system modes. Each system mode facilitates a specific task. The tool is now in SETUP mode, which is the default system mode. In SETUP mode, you can specify or set up circuit, scan, and tool behavior.

In this lab you use default configurations to minimize complexity in the tool setup. The primary goal for this exercise is to familiarize you with both the DFTAdvisor and FastScan tool flows.

2. Define control signals.

- a. In this exercise there are no requirements to hold pins at constant values. However, setting pin constraints is something you often do when setting up control signals.

What command do you use to hold a pin to a constant value?

---

- b. Define clocks automatically via a dialogue box or via a command.

i. What command do you use to automatically define clocks?

---

- ii. Where do you find the dialogue box to automatically add and identify clocks? \_\_\_\_\_

What is the name of the dialogue box? \_\_\_\_\_

---

What button do you click on? \_\_\_\_\_

- iii. Use one of the above methods to automatically add and identify system clocks.

Default Scan configurations are used for this exercise, so we are finished defining the design and are ready to go to the next stage, DFT mode, where scan/test circuitry is identified and inserted.

3. Go to DFT mode.

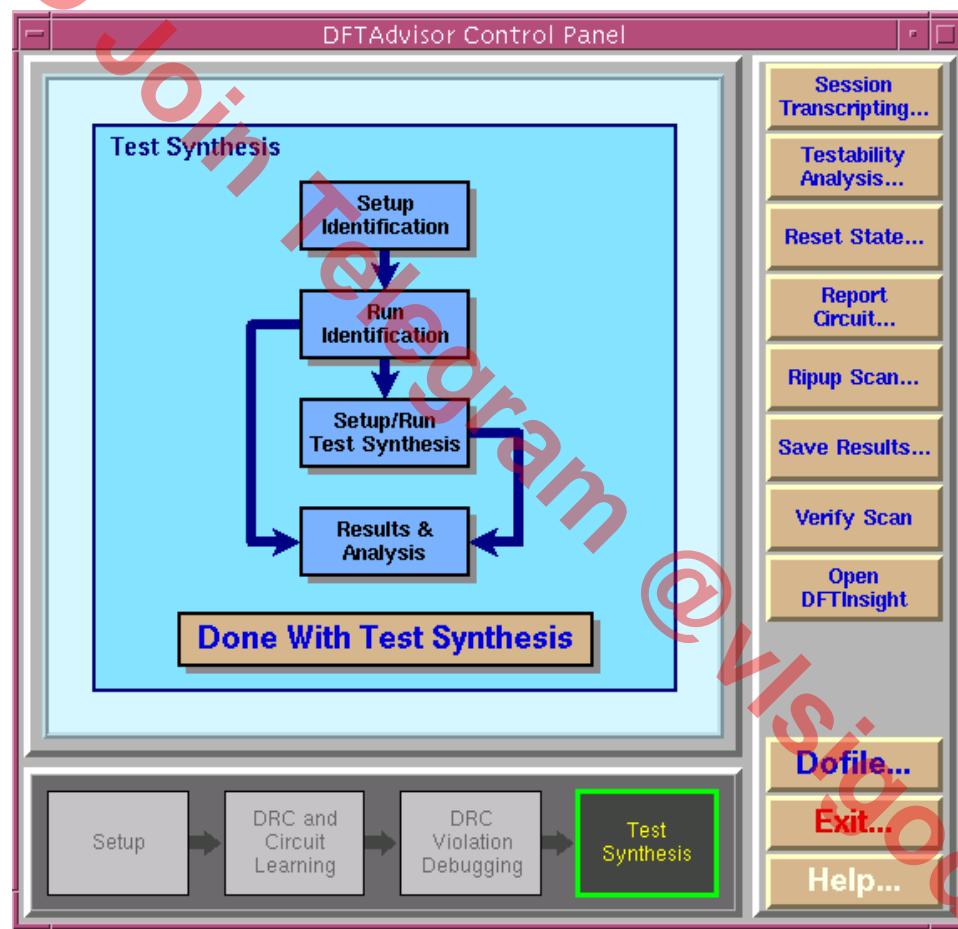
You can go into DFT mode using the command line, or by selecting Done With Setup in the Control Panel window.

Clicking on any button in the tool echoes the command to the session transcript pane in the Main window.

- i. What command do you use to go into DFT mode?
-

When you leave SETUP mode the tools automatically run DRC. You should not encounter any problems in this run, however.

**Figure 2-1. DFTAdvisor Control Panel**



The Control Panel now looks like the one shown here. You work through the process flow as indicated in the panel, starting with Setup Identification.

- Setup Identification:** Select Full Scan if not already selected.
- Run Identification:** Identify the items that will be altered in this step.

What is the default scan type? \_\_\_\_\_

Select Run with Existing Settings.

- i. The DFTAdvisor Identification Run Statistics window provides the answers to the following questions.

How many sequential elements are in the design? \_\_\_\_\_

How many scan cells have been identified? \_\_\_\_\_

What is the name of the top module for the design? (You need to click on View Details to find out). \_\_\_\_\_



**Note**

The View Details button displays the same information you saw after clicking on the Run Identification box.

Select Close when you are finished with the Scan Identification Settings window.

- ii. Generate a report by clicking on the Report... button. When the Results & Analysis window loads, click on the Generate Report button. (Identification Results tab)
- iii. Close and dismiss all windows until only the Main and Control Panel windows remain open.



**Note**

DFTAdvisor has identified scannable instances to convert to scan cells, but does not alter the netlist.

**c. Setup/Run Test Synthesis: Perform the test synthesis using the following (default) settings:**

- i. Setup Synthesize Scan Circuitry into Design:
  - a. Insert Scan Cells Based on: The Current Scan Identification List.
  - b. Insert 1 chain.
  - c. Allow Only One Enable Signal to control All Scan Chains.

d. Click Done.

ii. Setup Synthesize Identified Test Points:

a. Control Points: There should be no Scan Cells created at the Control Point.

b. Observe Points: Do Not Add a Scan Cell should be selected.

c. Click Done.

iii. Setup Test Logic to Control RAMs.

a. This is the default setting. You do not need to alter it.

DFTAdvisor can automatically insert a number of different test structures into the design. These structures include: scan circuitry, test points, test logic to RAMs, I/O buffers for added test pins, and buffer trees for test pins.

iv. Click **OK**. The **Use Existing Settings or Customize?** box opens.

v. Click **Run with Existing Settings**. You just inserted full scan into the design using the default settings.



**Note**

A scan-inserted netlist has now been created.

d. **Results and Analysis:** Clicking on this selection allows you to review and report on the results. (This brings up the same window you saw in step 3.b.ii above.) You may generate a report. When you are done, close the window.

4. Click on the Save Results... button in the Button pane to save a Verilog netlist and ATPG setup files.

a. Use the dialogue to save the netlist and the setup files for FastScan.

- i. Click the Browse button and save a new netlist in Verilog format called pipe\_scan.v to the ../results directory.
- ii. Save an enhanced procedure ATPG Setup file with a basename of ../results/pipe\_scan (no file extension).
- iii. Select the **Overwrite Existing File** button for both the netlist and ATPG Setup files procedure.
- iv. Select to view the files after generation.
- v. Finish by clicking OK.

DFTAdvisor has written the following three files:

- pipe\_scan.v, which is the Verilog netlist
- pipe\_scan.dofile, which is a dofile
- pipe\_scan.testproc, which is an Enhanced Procedure file

The File Viewer is now open.

- vi. Select the pipe\_scan.dofile if it is not already displayed in the File Viewer. This file inputs circuit setup and scan information needed by FastScan at invocation.
  - vii. Click pipe\_scan.testproc in the **Select File to View** display panel. This displays the shift and load\_unload procedures, as well as the timing plate (gen\_tp1), which is shared by both procedures.
- b. Close the Viewer.
5. Exit DFTAdvisor. (Click on the Exit... button in the Button pane.)

Next, you use FastScan to create, compress, and save patterns following the FastScan tool flow as seen in the lectures and shown in [Figure 2-2](#).

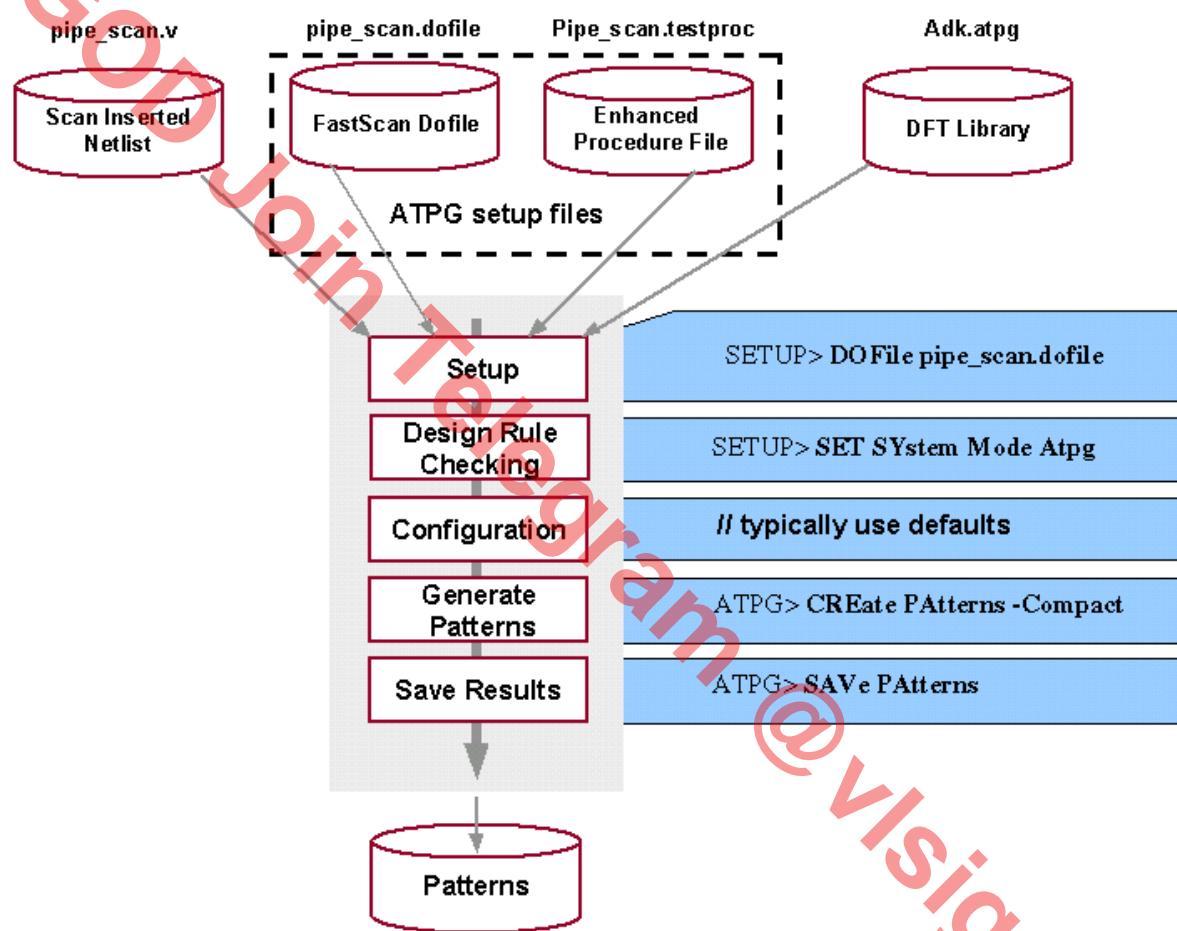


Figure 2-2. FastScan Tool Flow.

6. Run FastScan.

- a. For this exercise, invoke FastScan from the shell prompt.

```
shell> fastscan
```

- b. Enter the following in the appropriate dialogue boxes. Use the Browse button to quickly find the design and design library, found in the `exercise_3/results` and `..libraries` directories.

Design: `pipe_scan.v`

Design Format: Verilog

Library:                   libraries\_1\_to\_4/adk.atpg

- c. Click on the Invoke FastScan button.

FastScan is now up and running. You have successfully invoked FastScan on the pipe\_scan.v design. Both the Main and Control Panel windows are open. The **Circuit Setup** graphic pane opens, and **Setup** is highlighted in the Process Pane.

- d. Load the dofile pipe\_scan.dofile. (*File menu*)

- i. What command would you use to load the dofile at the **SETUP>** prompt?

- ii. The dofile contains a command that loads the enhanced procedure file. Write that command here:

---

---

---

For additional information on procedure files, refer to Chapter 9 of the *Design-for-Test Common Resources Manual*.

Let us review what FastScan performed in this step:

- Compiled the library
- Read in the Verilog Netlist (*pipe\_scan.v*)
- Read in the test procedure file (*pipe\_scan.testproc*)
- Read in the dofile (*pipe\_scan.dofile*)

All of this is displayed in the session transcript area, as previously discussed.

All the setup necessary for this design has been completed automatically.

7. Go to ATPG mode.

- a. Click the **Done with Setup** button in the Circuit Setup graphic pane.

The **FastScan — Session Purpose** dialogue box opens.

- b. Click the **Pattern Generation** button. The Test Pattern Generation dialogue box opens. Also, **Pattern Generation** is highlighted in the Process pane.
- c. Looking at the Session Transcript window, identify the processes that occurred when you clicked on this button.

---

---

---

---

- d. The tool exited SETUP mode and entered the **ATPG** mode. What does the prompt at the command line indicate now?

---

---

---

Working with the Process diagram shown in the Control Panel window, work through the Pattern Generation Process.

- e. **Fault Universe:** Select Typical settings. Look at the Session Transcript window. What commands does this process generate?

---

---

- f. **Test Generation:** Generate patterns by selecting the Run with Existing Settings button.

Fill in the first line of the following tables using the results.

**Table 2-1. Test Pattern Generation Results**

Run	No. of Effective Patterns	No. of Sim'd Patterns	No. of detected	No. of Aborted	No. of AU	No. Redun.

**Table 2-2. Test Pattern Generation Results (Cont.)**

Run	No. Undetect	Test Coverage %	Fault Coverage %	ATPG Effect %	CPU Runtime

- g. Statically compress patterns (Click on the Compress... button in the FastScan Pattern Compression Statistic dialogue box.). Use three compression passes. Click Compress again.

Fill in the second line in the table above. (Note: Not all columns apply).

**Compression Results:** How many patterns have been removed?

---

During static compression, faults are simulated first in reverse order, and then in random order. This reduces the pattern set because only those patterns that detect new faults are retained.

- h. Dynamically compress patterns. We do not compress the present set, but throw them away and start again, running compression as we generate patterns. Dynamic compression occurs during an ATPG run.

ATPG > **create patterns**

- i. Fill in the third line in the table above. (Note: Not all columns apply).

How many faults were detected? \_\_\_\_\_

- ii. Click Dismiss to exit the FastScan Pattern Compression Statistics dialogue window.

So far, you have created a test pattern set and you have compressed the set both statically and dynamically.

The next process in the ATPG tool flow is to save the patterns.

8. **Save Patterns:** Click on the Save Patterns... button in the Button pane. Save parallel Verilog patterns. When the dialogue opens save the pattern set to a file with the following:

- **File name:** lab2/exercise\_3/results/pat1\_par.v (overwrite the file if it already exists).
- Select Verilog Pattern Format for Parallel patterns.
- Save Chain and Scan tests.

Parallel Verilog test benches are used in time-based simulations to verify FastScan's expected values against simulated values (simulation mismatch).

9. Save serial Verilog patterns.

- **File name:** lab2/exercise\_3/results/pat1\_ser.v (overwrite the file if it already exists).
- Select Verilog Pattern Format for Serial patterns.

- Save Chain and Scan tests for the first 5 patterns. (Select Number of Patterns to Sample per Pattern Type. Make sure there is nothing in the Pattern Range to Save: beginning and ending boxes or you will generate an error.)
10. Exit FastScan.

## Exercise 4: Scan and ATPG Tool Flow (With a Design Rule Violation)

This exercise is similar to Exercise 3 in that you invoke DFTAdvisor on a simple design to insert full scan. A bug has been incorporated into the design that causes a minor DRC violation. You use DFTInsight to view the problem, and you correct the problem so DFTAdvisor will successfully pass DRC and enter the DFT system mode. You use DFTAdvisor to write a scan-based netlist file and ATPG setup files.

Then you invoke FastScan using the ATPG setup file information to create, compress, and save patterns.

This lab gives you experience using the tools in a typical scan and ATPG tool flow utilizing default configurations, with the added step of correcting a minor DRC violation.

1. Change to the \$ATPGNW/lab2/exercise\_4 directory.

```
shell> cd $ATPGNW/lab2/exercise_4
```

2. Invoke DFTAdvisor.

```
shell> dftadvisor
```

- a. Enter the following in the appropriate dialogue boxes. Use the Browse button to quickly find the design and design library, found in the exercise\_4 and libraries\_1\_to\_4 directories.

Design: pipe\_noscan.v

Design Format: Verilog

Library: libraries\_1\_to\_4/adk.atpg

Log file: results/dftadvisor.log

(Overwrite the existing log file)

- b. DFTAdvisor should be up and running with the Main and Control Panel windows both open.
3. Define the control signals as you did in section 2 of exercise 3.
4. Go to DFT mode.

You do this by typing a command, or by selecting Done with Setup in the Control Panel pane.

5. **Debug Errors/Warnings:** The DRC Warnings Occurred window opened. Proceed with the debugging process. This opens the DRC Graphic Pane.

- a. Look at the Session Transcript pane to find the warning message.

What type of DRC violation do you have?

---

What causes these? (Select Go to Manual for help.)

---

---

- b. Investigate them using DFTAdvisor:

- i. Click the **Select** button in the DRC graphic pane of the Control Panel window to open a dialogue box.

- ii. By clicking on the two ID numbers and View Info, find out specific information about the violations:

- iii. What instance is affected by S-1? \_\_\_\_\_

iv. What instance is affected by S-2? \_\_\_\_\_

The information is found in the Graphics pane. Dismiss the Select Violation ID window once you have obtained the information.

c. Investigate further using DFTInsight, opening an interactive graphical debugging environment:

i. Click on the Open DFTInsight button in the Button pane.

ii. Select **Analyze > DRC Violations** and investigate S1-1.

To what is the CLK input of the instance you identified above connected? \_\_\_\_\_

Why does this cause a problem in test?

---

iii. Select S1-2 in the Select a Violation ID dialogue to investigate the other violation.

To what is the CLK input of this instance connected?

---

Click on the EZ-trace button and the LMB to back trace the X on this instance two levels. What is the name of the instance you traced to?

---

Being able to trace backward and forward can be an invaluable aid in debugging.

iv. When you have finished, close DFTInsight.

6. We need to go back to setup mode to make corrections. Select Done with DRC Debugging in the Graphic pane to go back to SETUP mode. *Do not use the tool process flow buttons.*

The problem is fixed by enabling test logic insertion for clocks. Doing this does not alter the netlist that DRC checks, but enables the environment to deal with these errors.

7. Open the Setup for Test Synthesis dialogue box (Test Synthesis Setup... button) and select the Test Logic/Latch Scannability tab to turn on test logic insertion for clocks. (Enable Insertion of Test Logic for the Following Types of Uncontrollable Signals: Clocks)

What command does this generate?

---

8. Go to DFT mode.

You can issue a command on the command line, or select Done with Setup in the Control Panel pane.

The DRC warnings still appear, but you have dealt with them so there is no need to investigate further. Go to the next phase.

Observe the session transcript. How many sequential instances can be gated to be scannable? \_\_\_\_\_

Work through the process flow as you did in exercise 3, starting with Setup Identification.

- a. **Setup Identification:** Full Scan (default)
- b. **Run Identifications:** Identify the items to be altered in this step. Select Run with Existing Settings and then dismiss the DFTAdvisor Identification Run Statistics box.
- c. **Setup/Run Test Synthesis:** Use the following default settings:
  - i. Synthesize Scan Circuitry into the Design
  - ii. Synthesize Identified Test Points
  - iii. Synthesize Test Logic Circuitry

You get one more chance to alter things, but you don't need to; just run with existing settings.

9. Save a Verilog netlist and ATPG setup files.

- a. Click the **Save Results...** button in the button pane to bring up the **Save Results dialogue** box. Use the dialogue to save the netlist and the setup files for FastScan.
  - i. Save a new netlist in Verilog format to the file results/pipe\_scan.v.
  - ii. Save an enhanced procedure ATPG Setup file with the basename of results/pipe\_scan. (no file extension)
  - iii. Overwrite any files of the same names that may already be present for either the netlist or ATPG Setup files.

Finish by clicking OK.

You have written the following three files:

- pipe\_scan.v, which is the Verilog netlist
- pipe\_scan.dofile, which is a dofile file
- pipe\_scan.testproc, which is an Enhanced Procedure file

You now have created a scan inserted netlist. The next stage is to generate the test patterns to use with the netlist.

In this part of the exercise you invoke FastScan. The goal is to follow a typical ATPG flow.

10. Click the Done with Test synthesis button in the Graphic pane.

- i. Click Exit... in the DFTAdvisor — Test Synthesis Complete dialogue box. Confirm Exit when you are asked.

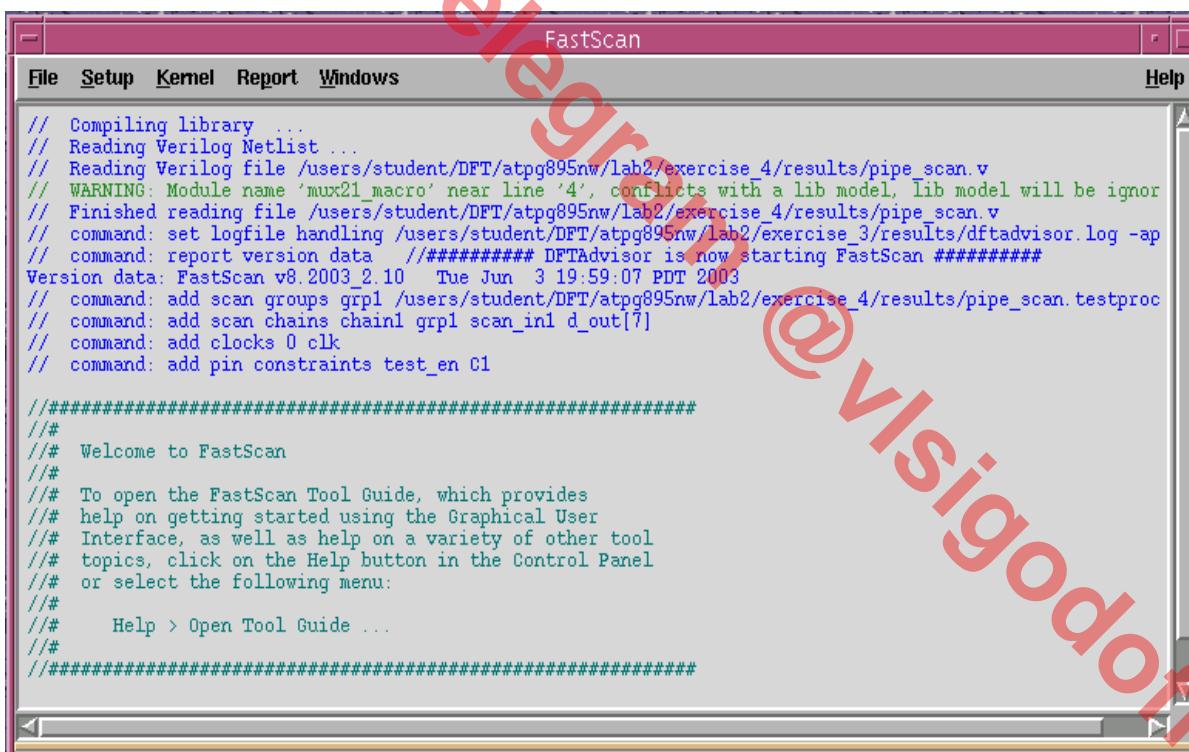
From the shell prompt, invoke Fastscan:

```
shell> fastscan
```

- a. Enter the following in the appropriate dialogue boxes. Use the Browse button to quickly find the design and design library, found in the exercise\_4/results and libraries\_1\_to\_4 directories.

Design: pipe\_scan.v  
 Design Format: Verilog  
 Library: libraries\_1\_to\_4/adk.atpg  
 Command File: pipe\_scan.dofile

This time we load the dofile from the invocation window.



The screenshot shows the FastScan application window with a menu bar (File, Setup, Kernel, Report, Windows, Help) and a toolbar. The main area displays a command-line interface with the following text:

```

// Compiling library ...
// Reading Verilog Netlist ...
// Reading Verilog file /users/student/DFT/atpg895nw/lab2/exercise_4/results/pipe_scan.v
// WARNING: Module name 'mux21_macro' near line '4', conflicts with a lib model. Lib model will be ignor
// Finished reading file /users/student/DFT/atpg895nw/lab2/exercise_4/results/pipe_scan.v
// command: set logfile handling /users/student/DFT/atpg895nw/lab2/exercise_3/results/dftadvisor.log -ap
// command: report version data //##### DFTAdvisor is now starting FastScan #####
Version data: FastScan v8.2003_2.10 Tue Jun 3 19:59:07 PDT 2003
// command: add scan groups grp1 /users/student/DFT/atpg895nw/lab2/exercise_4/results/pipe_scan.testproc
// command: add scan chains chain1 grp1 scan_in1 d_out[7]
// command: add clocks 0 clk
// command: add pin constraints test_en c1

#####
## Welcome to FastScan
##
## To open the FastScan Tool Guide, which provides
## help on getting started using the Graphical User
## Interface, as well as help on a variety of other tool
## topics, click on the Help button in the Control Panel
## or select the following menu:
##
## Help > Open Tool Guide ...
##
#####
  
```

- i. Click on the Invoke FastScan button.

FastScan is now up and running. You have successfully invoked FastScan on the *pipe\_scan.v* design. Both the Command Line and the Control Panel windows are open. The **Circuit Setup** graphic pane opens. Also, **Setup** is highlighted in the **Process Pane**.

All the setup for this design has been completed automatically. Note the addition of an extra command in the dofile, due to the fact that extra test logic was added in the form of a pin constraint.

What is the new command? \_\_\_\_\_

11. Go to ATPG mode.

- a. Click the **Done with Setup** button in the Circuit Setup graphic pane.

The **FastScan — Session Purpose** dialogue box opens.

- b. Click the **Pattern Generation** button. The Test Pattern Generation graphic pane opens. Also, **Pattern Generation** is highlighted in the Process pane.

Working with the Process diagram shown in the Control Panel window, work through the Pattern Generation Process.

- c. **Fault Universe:** Select Typical settings.
- d. **Test Generation:** Generate patterns by selecting the Run with Existing Settings button.

Fill in the first line of the following tables using the results.

**Table 2-3. Test Pattern Generation Results**

Run	No. of Effective Patterns	No. of Sim'd Patterns	No. of detected	No. of Aborted	No. of AU	No. Redun.

**Table 2-4. Test Pattern Generation Results (Cont.)****Table 2-5.**

Run	No. Undetect	Test Coverage %	Fault Coverage %	ATPG Effect %	CPU Runtime

- e. Statistically compress the existing patterns, using 3 compression passes.  
(Compression.. button in the Button pane)

Fill in the second line in the table above.

**Compression Results:** How many patterns have been removed?

---

During static compression faults are simulated first in reverse order, then in random order. This reduces the pattern set because only those patterns that detect new faults are retained.

- f. Dynamically compress patterns. We do not compress the present set, but throw them away and start again, running compression as we generate patterns. FastScan typically ends with a 1 pass static compression on the generated set. However, this pattern set is small and FastScan does not do perform static compression on this set.

Remember, dynamic compression occurs during an ATPG run.

ATPG > **create patterns**

Fill in the third line in the table above. (Note: Not all columns apply).

- i. Look at the session transcript window for information regarding information for the dynamic compression method.

How many faults were detected? \_\_\_\_\_

Click Dismiss to exit the FastScan Pattern Compression Statistics dialogue window.

So far, you have created a test pattern set and have compressed the set both statically and dynamically.

ii. The next step in the process is to save the ATPG patterns.

12. Save parallel Verilog patterns. Click on the Save Patterns... button in the Button pane. When the dialogue opens save the pattern set to a file with the following:

- **File name:** lab2/exercise\_4/results/pat1\_par.v (Overwrite the file if it already exists.)
- Select Verilog Pattern Format for Parallel patterns.
- Save Chain and Scan tests.

13. Save serial Verilog patterns:

- **File name:** lab2/exercise\_4/results/pat1\_ser.v (Overwrite the file if it already exists.)
- Select Verilog Pattern Format for Serial patterns.
- Save Chain and Scan tests for the first 5 patterns. (Select Number of Patterns to Sample per Pattern Type)

i. Exit FastScan.

Congratulations. You have used the DFT tools in a typical scan and ATPG tool flow.

## Test Your Knowledge

1. What can you specify in SETUP mode?  
\_\_\_\_\_
2. What are pin constraints?  
\_\_\_\_\_
3. What command automatically identifies clocks and control signals?  
\_\_\_\_\_
4. True or False. When scannable instances are identified, the netlist is altered.  
\_\_\_\_\_
5. What do you do if you see the message “Flattened model has been freed” in the session transcript area?  
\_\_\_\_\_
6. True or False. The DFT tools will not successfully exit the SETUP mode when a DRC violation occurs.  
\_\_\_\_\_
7. What is the recommended command to create a compressed high coverage pattern set.  
\_\_\_\_\_
8. Why should you use log files?  
\_\_\_\_\_
9. What is an Enhanced Procedure file?  
\_\_\_\_\_

10. Why create serial test patterns?

---

## Lab Summary

Now that you have completed the Full Scan DFT flow lab, you should know how to do the following:

- Invoke DFTAdvisor to insert full scan into a design following a typical scan tool flow process
- Write a scan-inserted netlist
- Write ATPG setup files
- Invoke FastScan to create, compress, and save patterns following a typical ATPG tool flow process
- Invoke DFTInsight to troubleshoot minor DRC violations

## Module 3

# Configuring Scan Chains/Test Logic and Full Scan Flow

### Objectives

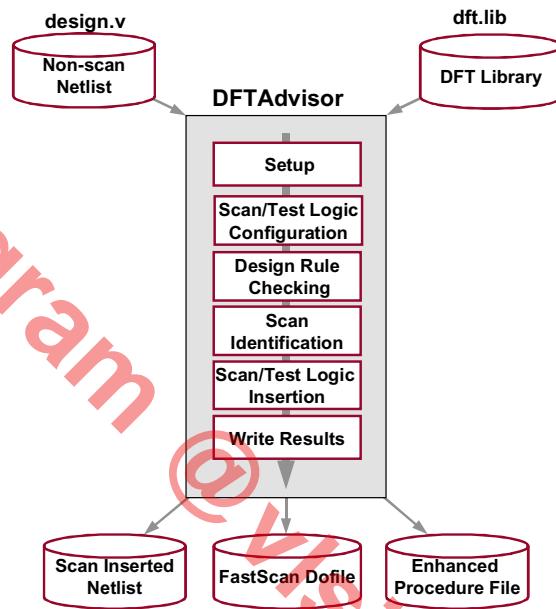
Upon completion of this module, you will be able to:

- Setup scan pins new and existing (internal or external).
- Insert test logic.
- Create, configure and balance scan chains.
- Insert scan cells without stitch and write out a scan chain order file.
- Edit a scan chain order file and change the order of the scan cells.
- Stitch the scan chain with the modified order file.

## Module Topics

### Module Topics

- ◆ This module addresses the following topics:
  - Scan methodology
  - Test logic
  - Defining pins
  - Multiple clocks
  - Stitching
  - Order files

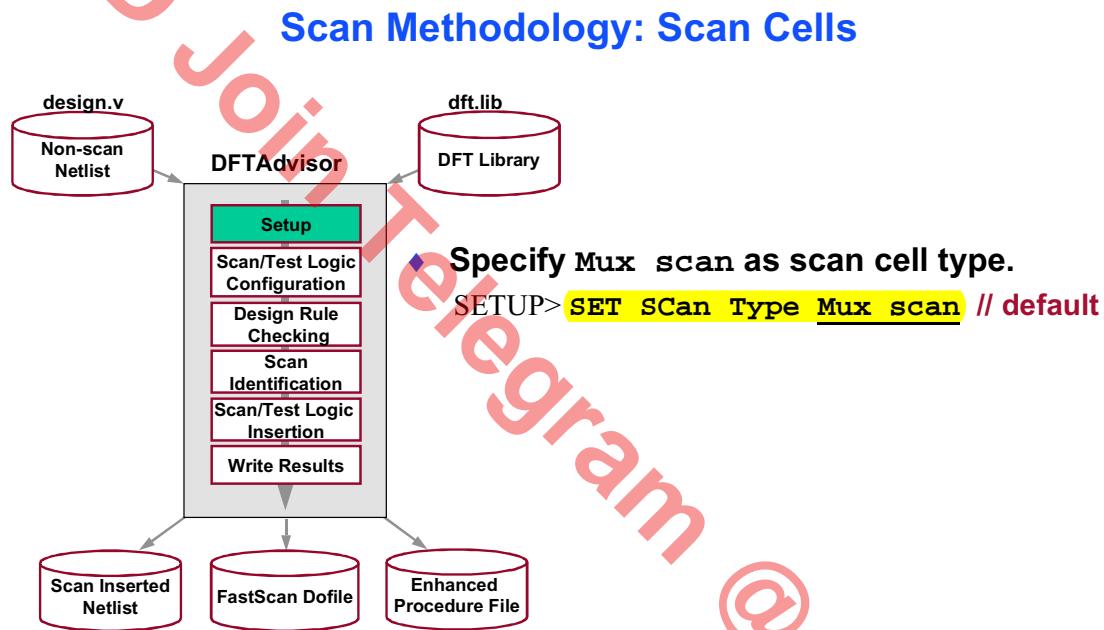


3-2 • Design-for-Test: Scan and ATPG: Configuring Scan Chains/Test Logic and Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

### Notes:

## Scan Methodology: Scan Cells



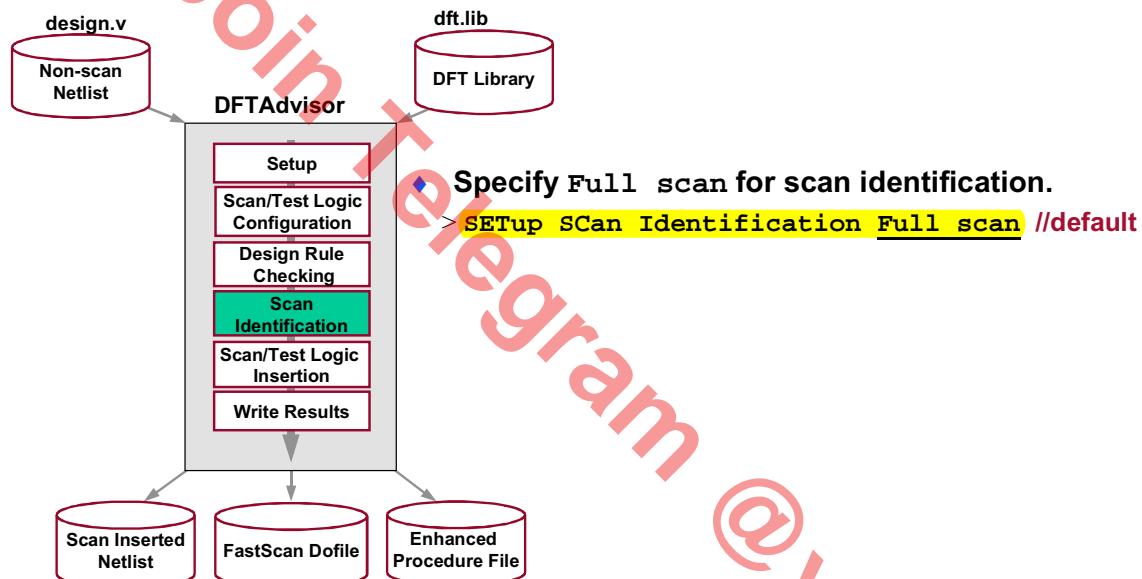
3-3 • Design-for-Test: Scan and ATPG: Configuring Scan Chains/Test Logic and Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

### Notes:

# Scan Methodology: Full Scan

## Scan Methodology: Full Scan



3-4 • Design-for-Test: Scan and ATPG: Configuring Scan Chains/Test Logic and Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Scan Methodology: Full Scan Versus Partial Scan

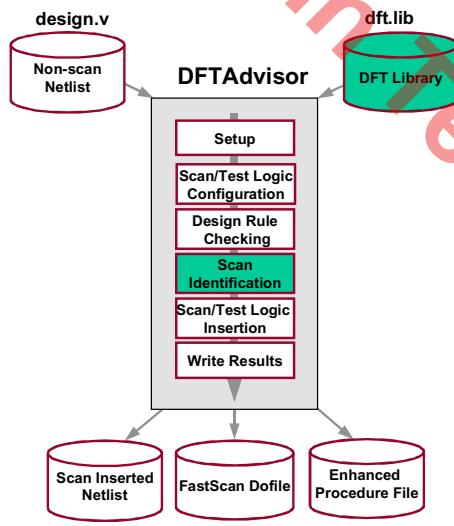
### Scan Methodology: Full Scan Versus Partial Scan

- ◆ **Full scan**
  - Converts all memory elements to scan
    - Sometimes a few non-scan restrictions
  - Proven methodology
    - Most common
  - Provides high test coverage/high quality
  - Uses a combinational ATPG tool which requires minimal test generation effort
- ◆ **Partial scan**
  - Converts a portion of memory elements to scan
    - Not a commonly used methodology
  - Increases test coverage time but requires less silicon area
  - Uses a sequential ATPG tool which requires maximum test generation effort

### Notes:

# Scan Methodology: DFT library and Scan Identification

## Scan Methodology: DFT Library and Scan Identification



♦ **The DFT library:**

- A model description defines a single cell in the technology library.
- A cell description (model or macro) describes a component in a specified design.
- A library is simply a set of models.

```
// =====
// Model: DFF
// =====
```

```
model DFF (PRE, CLR, CLK, D, Q, QB) (
  input (PRE, CLR, D)()
  input (CLK) (clock = rise_edge;
  output(Q, QB) (primitive = _dff (PRE, CLR, CLK, D, Q, QB) ;
```

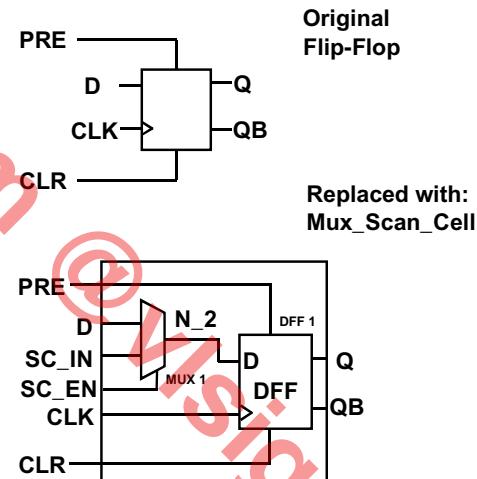
## Notes:

# Scan Methodology: DFT library and Scan Identification (Cont.)

## Scan Methodology: DFT library and Scan Identification (Cont.)

- ♦ If a library model is a scan cell, the model description contains a scan definition attribute:
  - Provides information for mapping non-scan sequential models (dffs and latches) to their associated scan cell models.

```
===== Model: DFF =====
=====
model DFF ( PRE,CLR,CLK,D, Q, QB )
  input ( PRE, CLR, D () )
  input(CLK) (clock = rise_edge;)
  output(Q QB) (primitive = _dff(PRE,CLR,CLK,D,QB);
  )
===== Model: MUX_SCAN _CELL =====
=====
model MUX_SCAN_CELL (PRE, CLR, SC_IN, SC_EN, CLK, D, Q, QB) (
  scan_definition (
    type = mux_scan
    scan_in = SC_IN;
    scan_enable = SC_EN;
    scan_out = Q, QB;
    non_scan_model = DFF ( PRE, CLR, CLK, D, Q, QB);
  )
  input (PRE, CLR, SC_IN, SC_EN, CLK ())
  intern(N_2) (primitive = _mux mux1 (D, SC_IN, SC_EN,N_2);)
  output(Q, QB) (primitive = _dff dff1(PRE, CLR, CLK, N_2, Q, QB);
  )
=====
```



3-7 • Design-for-Test: Scan and ATPG: Configuring Scan Chains/Test Logic and Full Scan Flow

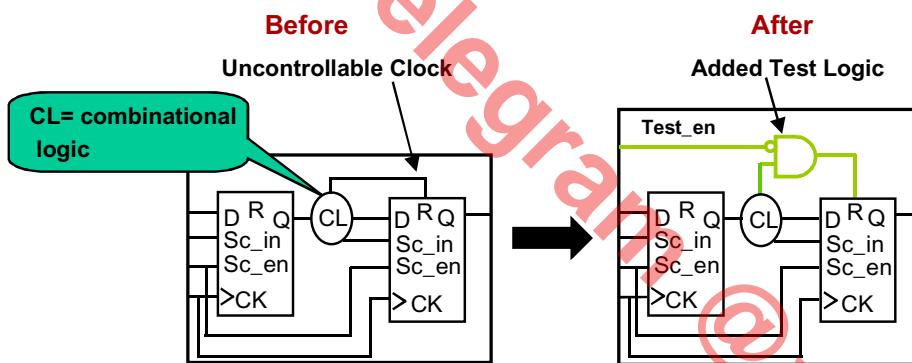
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Test Logic

## Test Logic

- ♦ Why do we add test logic?
  - Some designs contain uncontrollable clock circuitry.
  - Sequential devices must be controllable to be converted to scan.
  - RAM and three-state logic must be controllable to be testable.



3-8 • Design-for-Test: Scan and ATPG: Configuring Scan Chains/Test Logic and Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Test Logic (Cont.)

### Test Logic (Cont.)

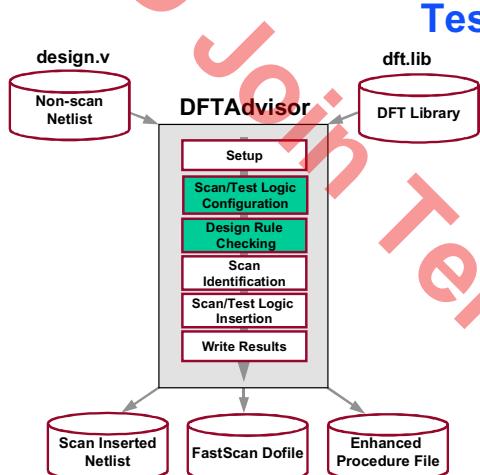
- ◆ To add test logic circuitry, DFTAdvisor uses a number of combinational gates from the ATPG library for example:
  - AND
  - OR
- ◆ A `cell_type` attribute defines valid test logic

```
// =====
// Model: AN2
// =====

model AN2 (A, B, Z) (
    cell_type = AND;
    input (A, B) ( )
    output (Z) (function = A * B;)
)
```

### Notes:

## Test Logic (Cont.)



### Test Logic (Cont.)

- ◆ Define library models used for test logic:
  - SETUP> ADD CELL Models dftlib\_model
  - Or automatically defined by DFT library if the model has a *cell\_type* attribute

- ◆ Generate scannability check results for non-scan instances:

DFT> REPort DFT Check

```
SETUP> ANALyze COntrol Signals -Autofix
SETUP> SET TESt Logic -set on
SETUP> SET SYstem Mode dft
DFT> REPort DFT Check
```

**Helpful Tip**  
Lists DFTAdvisor-identified pins  
that require test logic

### Notes:

## Test Logic: Defining Library Models

### Test Logic: Defining Library Models

```
// command: INSert TESt Logic  
// Warning: Flattened model has been freed  
// command: write atpg setup pipe_setup -procfile -rep  
// command: write netlist pipe_netlist.v -verilog -replace  
// Writing VERILOG netlist ...  
// command: REPort TESt Logic  
New pins added in top module: pipe  
/scan_in1  
/scan_en  
Number of new pins inserted = 2
```

Inserted test logic

- ◆ Define library models when inserting test logic for the following situations:
  - Set/reset clock access
  - Lockup latch between clock domains
  - RAM control
  - Three-state bus control
  - Control points
  - Observe points
- ◆ Display added test logic during scan insertion:  
DFT> REPort TESt Logic

### Notes:

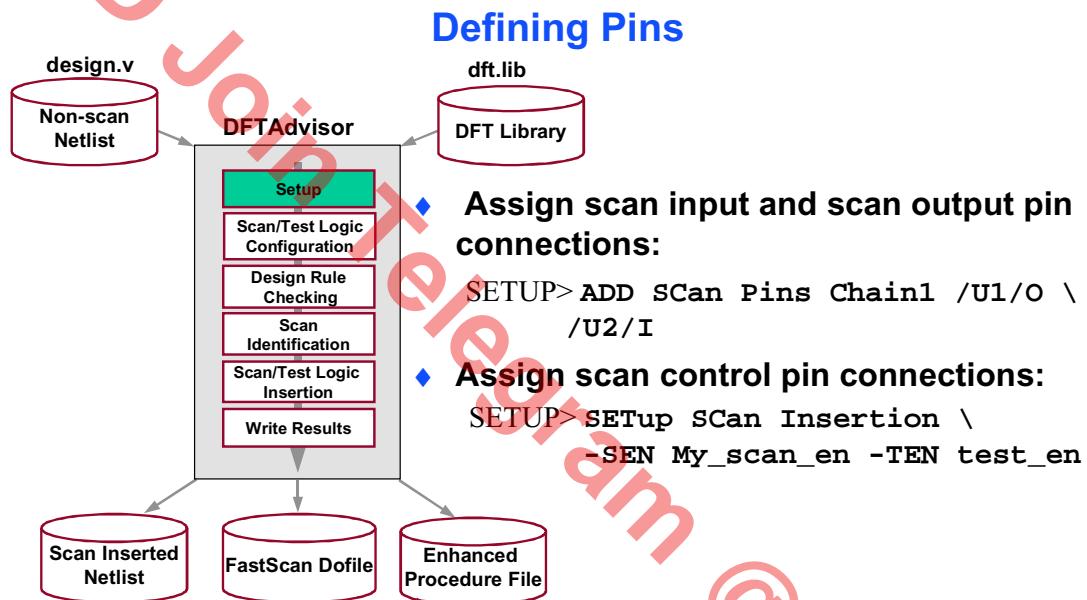
### Pins

#### Pins

- Fault sites include only:
  - Top-level module pins
  - Library model pins
  - Netlist primitive pins
- Pins are identified by unique pin names:
  - /I116/q
- Three types of pins:
  - Inputs (top-level, primary input)
  - Output (top-level, primary output)
  - Bidi

### Notes:

# Defining Pins



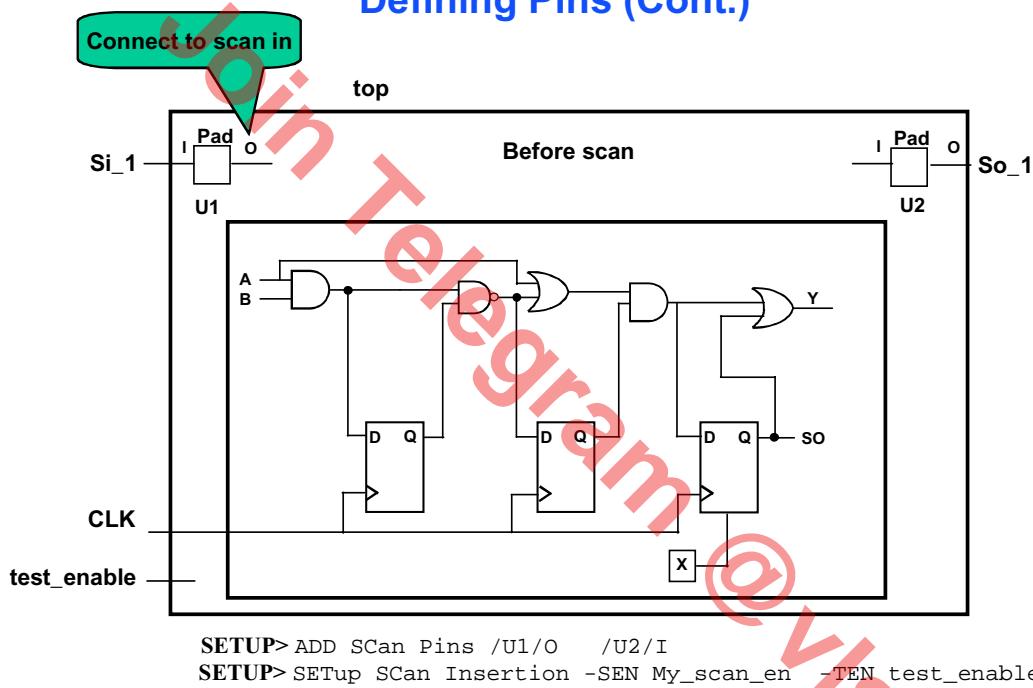
3-13 • Design-for-Test: Scan and ATPG: Configuring Scan Chains/Test Logic and Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Defining Pins (Cont.)

### Defining Pins (Cont.)



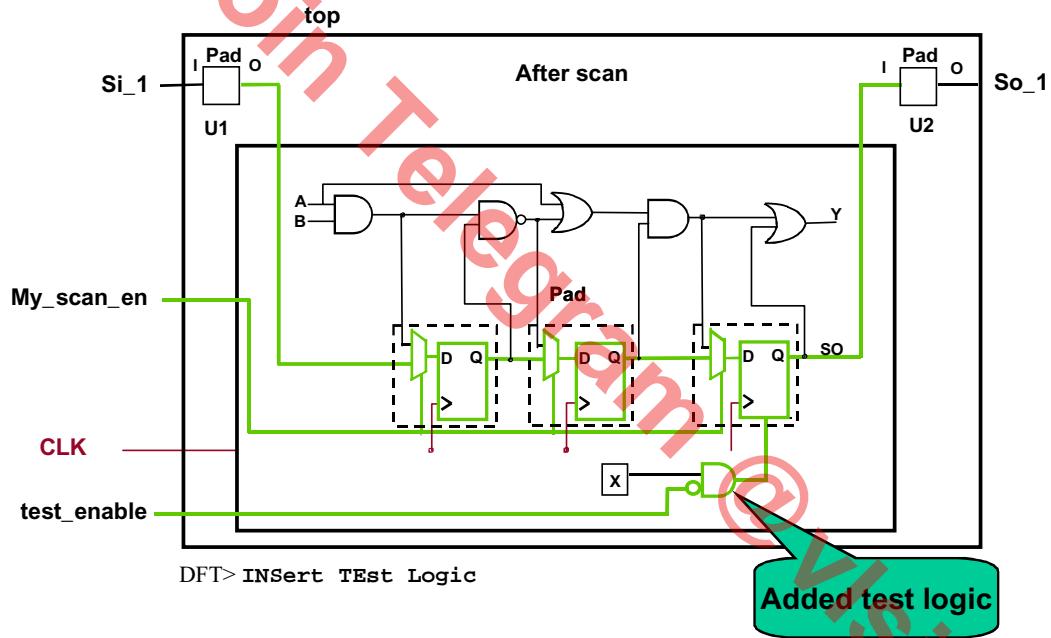
3-14 • Design-for-Test: Scan and ATPG: Configuring Scan  
Chains/Test Logic and Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

### Notes:

## Defining Pins (Cont.)

### Defining Pins (Cont.)



3-15 • Design-for-Test: Scan and ATPG: Configuring Scan Chains/Test Logic and Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Clocks

### Clocks

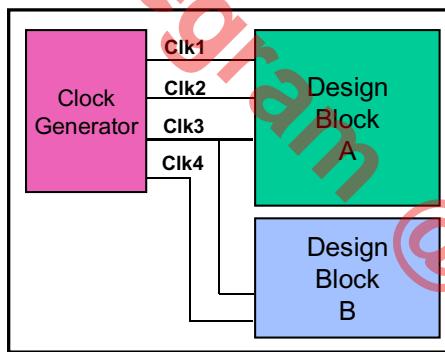
- ◆ The following applies to clocks:
  - Most designs have multiple clocks.
  - Clocks have a defined “off-state”
  - Clocks have two types of behavior:
    - Shift clocks shift data through the scan chain.
    - Capture clocks capture data into scan cells.

### Notes:

## Multiple Clock Issues

### Multiple Clock Issues

- ◆ Designs with multiple clock domains can produce clock skew during test.
- ◆ Different clock inputs and clock edges can cause the following skew problems:
  - Shifting data through scan chains.
  - Capturing data into scan chains.



### Notes:

## Multiple Clock Issues (Cont.)

### Multiple Clock Issues (Cont.)

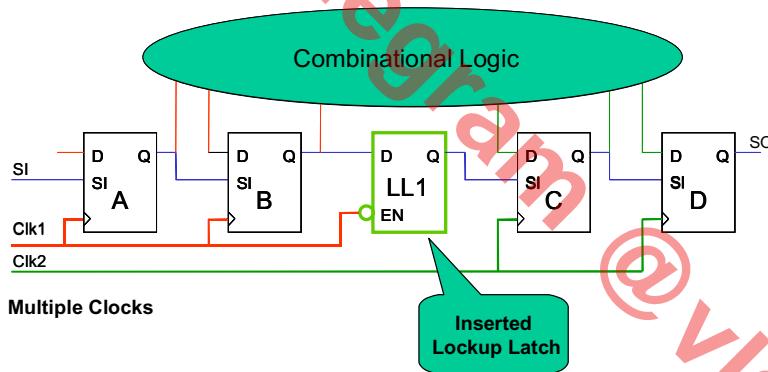
- ◆ During shift, all shift clocks are pulsed at the same frequency and time.
  - Clock skew results because of different clock domains.
- ◆ During capture, one or more clocks might be pulsed at the same time.
  - Clock skew results because of possible sequencing limitations of ATPG.

### Notes:

## Multiple Clocks: Minimizing Clock Skew

### Multiple Clocks: Minimizing Clock Skew

- ◆ Minimize clock skew during shift:
  - Order scan chains
    - Group flip-flops together into one clock domain.
    - Insert lockup latches where domains cross.



3-19 • Design-for-Test: Scan and ATPG: Configuring Scan Chains/Test Logic and Full Scan Flow

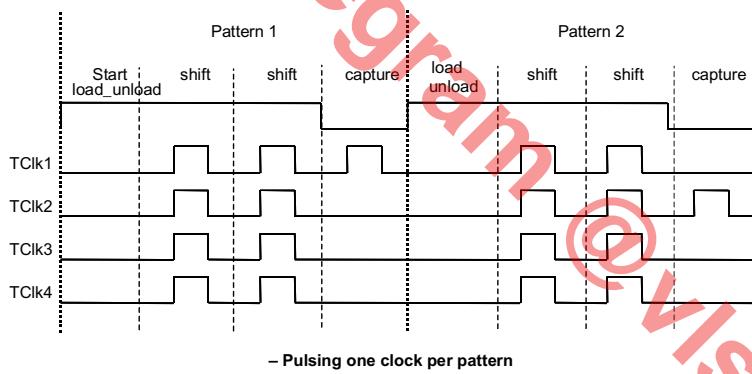
Copyright © 2003 Mentor Graphics Corporation

### Notes:

## Multiple Clocks: Minimizing Clock Skew (Cont.)

### Multiple Clocks: Minimizing Clock Skew (Cont.)

- ♦ Minimize clock skew during capture:
  - Control clock handling
    - Pulse one clock per pattern



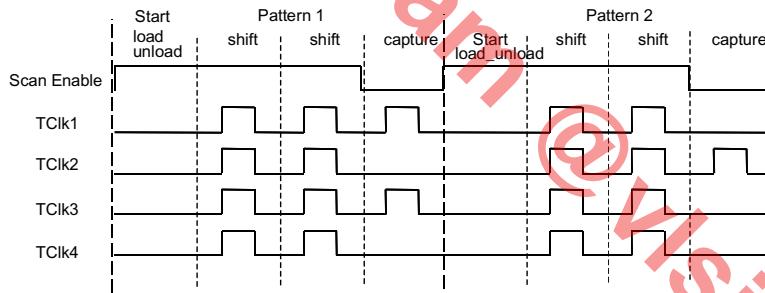
### Notes:

## Multiple Clocks: Minimizing Clock Skew (Cont.)

### Multiple Clocks: Minimizing Clock Skew (Cont.)

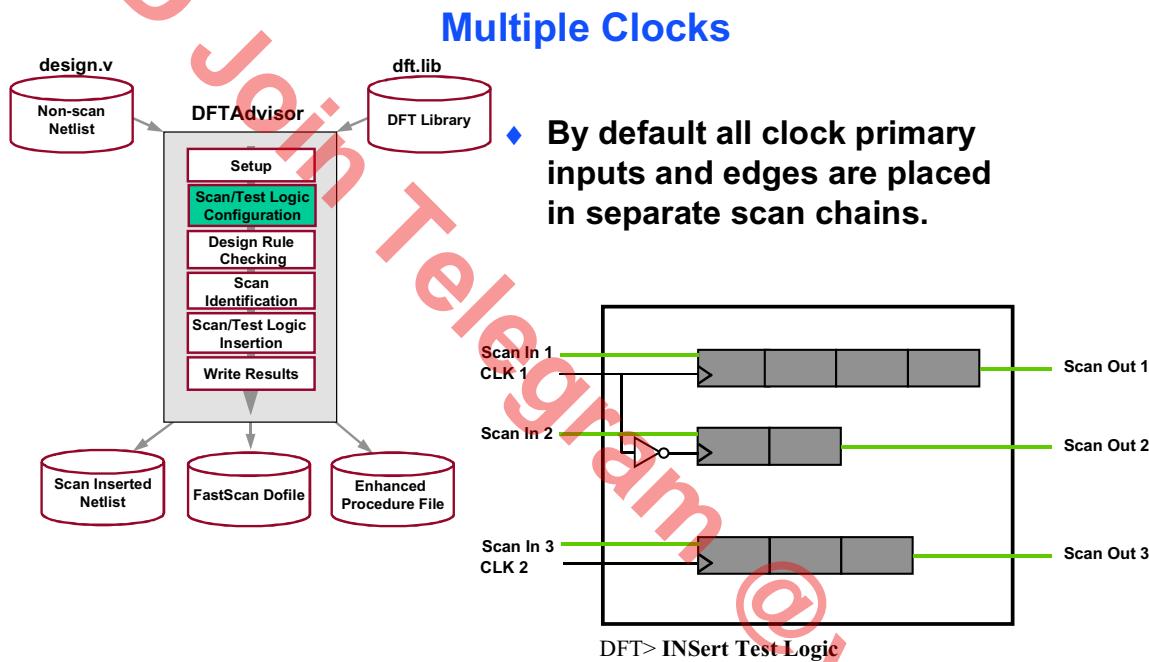
- Optimize clock handling during ATPG:

```
SETUP> SET Clock Restriction Domain_clock
SETUP> REPort Clock Domains
// clock Name : Clock_domain
-----
// /tck1 (8) : 1
// /tck3 (9) : 1
...
```



### Notes:

# Multiple Clocks



3-22 • Design-for-Test: Scan and ATPG: Configuring Scan  
Chains/Test Logic and Full Scan Flow

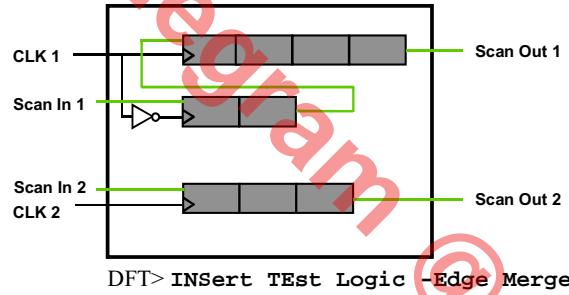
Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Multiple Clocks: Merging Clock Edges

### Multiple Clocks: Merging Clock Edges

- ♦ Leading and trailing edge clocks can be combined into the same scan chain.
- ♦ DFTAdvisor groups all trailing edge clock scan cells first.



### Notes:

# Multiple Clocks: Merging Different Clocks

## Multiple Clocks: Merging Different Clocks

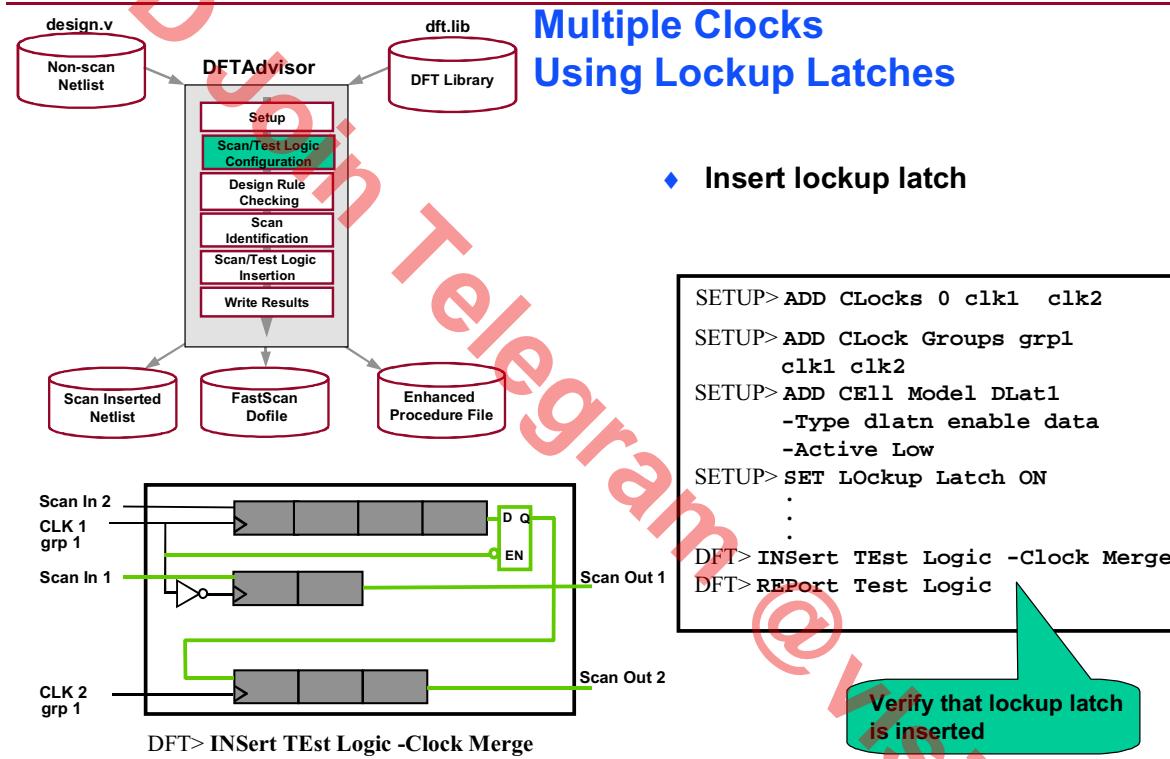
- ◆ Different clocks can be merged into the same chain.
  - DFTAdvisor selects scan cells to be merged.
  - DFTAdvisor places lockup latches between each clock domain.

```
DFT> SET Lockup Latch on  
DFT> INSERT Test Logic -clock merge
```
- ◆ Multiple clocks can be combined into 'clock groups'.
  - Explicitly defines which clocks can be placed into the same scan chain.

```
DFT> ADD Clocks 0 clk1 clk2 clk3  
DFT> ADD Clock Groups group1 clk1 clk2 clk3  
DFT> SET Lockup Latch on  
DFT> INSERT Test Logic -clock merge
```

## Notes:

# Multiple Clocks: Using Lockup Latches



3-25 • Design-for-Test: Scan and ATPG: Configuring Scan Chains/Test Logic and Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Balancing Scan Chains

### Balancing Scan Chains

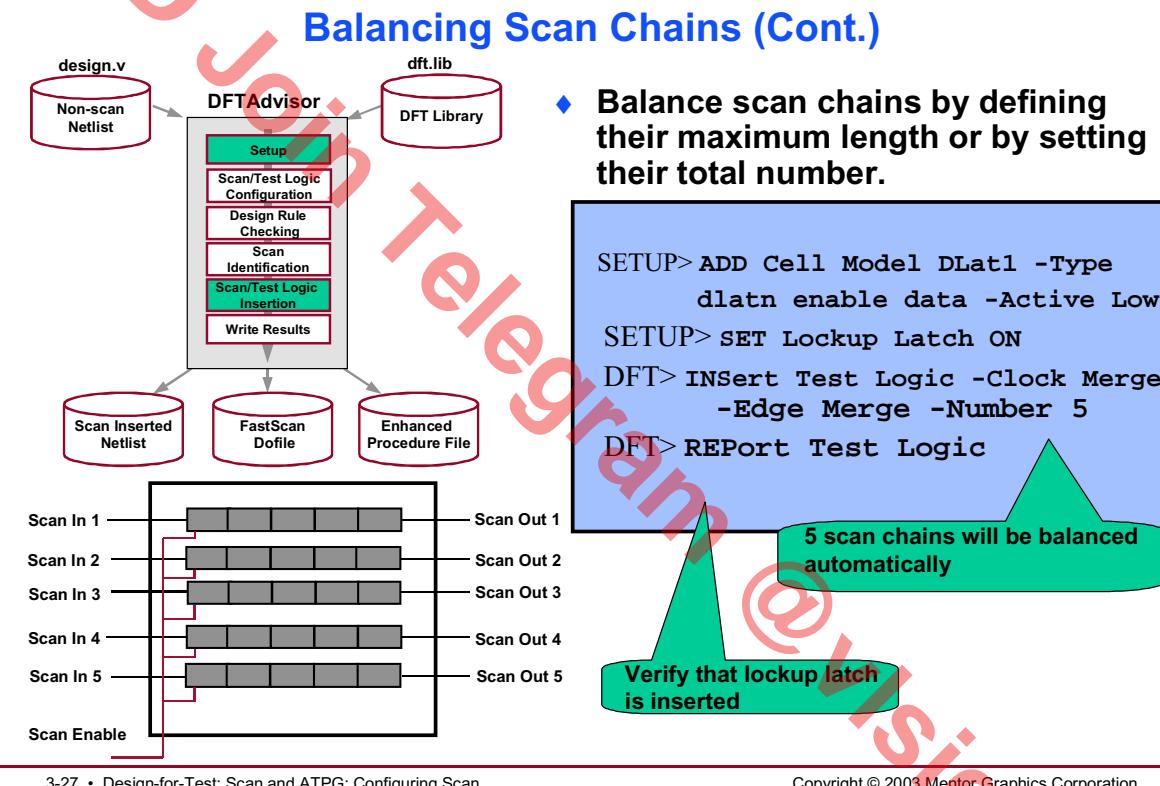


- ◆ Testers need deep serial memory for every scan input and output pin.
- ◆ Functional pins can be shared as scan pins in test mode.
- ◆ Test time and cost is reduced with more and shorter scan chains.

The number of scan chains is dependent upon tester capabilities.

### Notes:

## Balancing Scan Chains (Cont.)



3-27 • Design-for-Test: Scan and ATPG: Configuring Scan Chains/Test Logic and Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

### Notes:

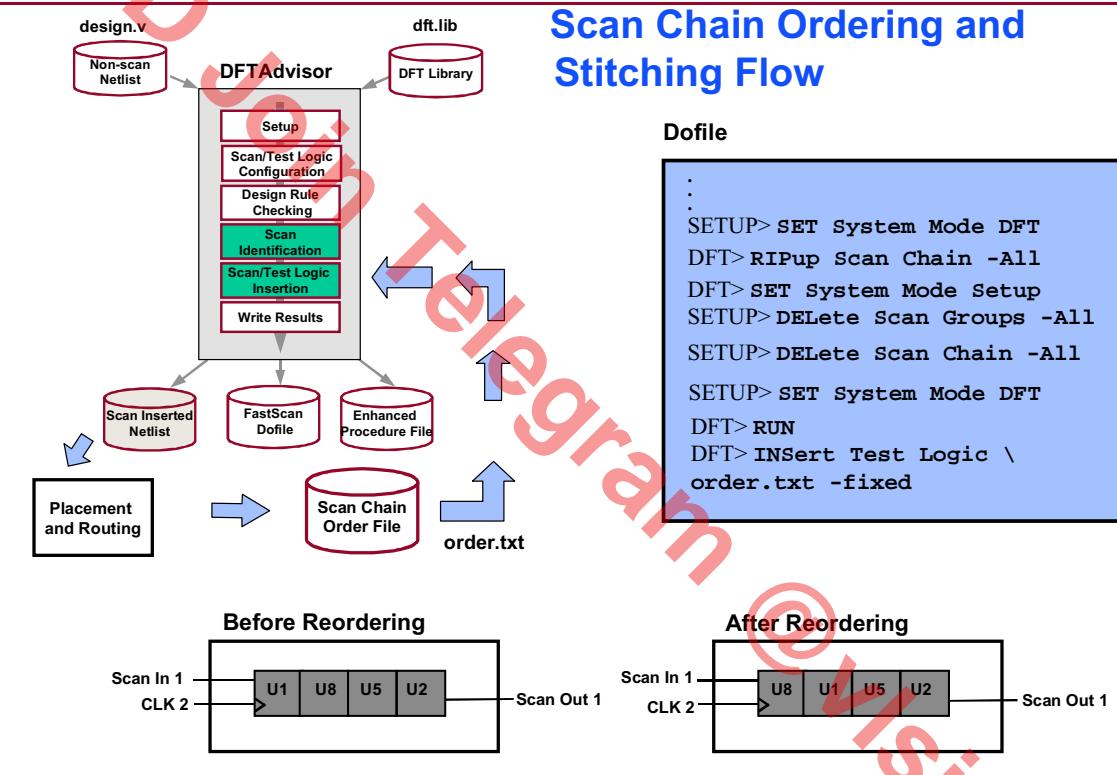
# Scan Chain Ordering and Stitching

## Scan Chain Ordering and Stitching

- ◆ Mux-scan designs:
  - Scan cells must be correctly ordered to prevent skew during shift.
  - Better placement and routing of scan cells results in better stitching.
- ◆ To optimize a scan design layout:
  - Remove all previous scan chains from the design.
  - Reorder the scan cells and write a scan cell order file.
  - Stitch scan cells into scan chains using the scan cell order file.

## Notes:

# Scan Chain Ordering and Stitching Flow



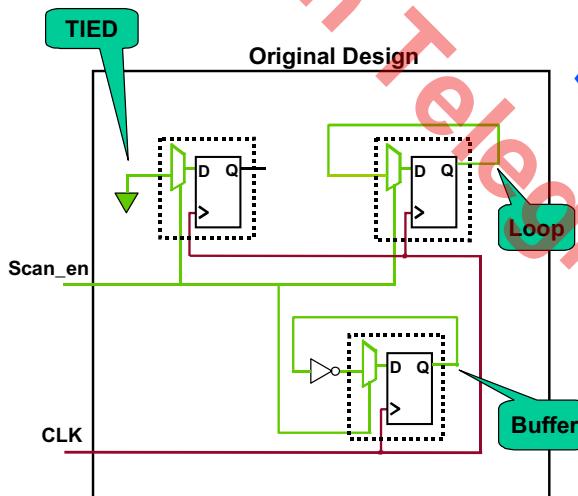
3-29 • Design-for-Test: Scan and ATPG: Configuring Scan Chains/Test Logic and Full Scan Flow

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Scan Chain Stitching: Unstitched Scan Cells

### Scan Chain Stitching: Unstitched Scan Cells



- Use the following command to avoid stitching scan cells into scan chains:

```
DFT> INSert TEst Logic -Connect  
Tied | Loop | Buffer
```

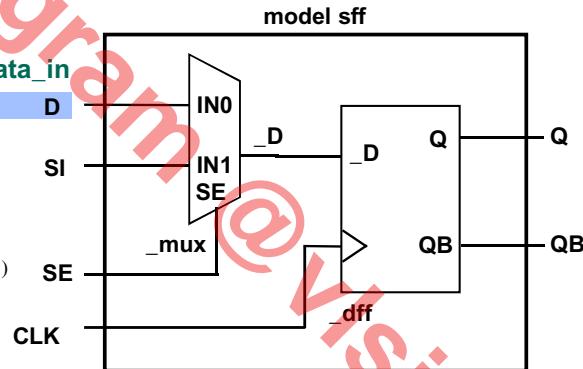
### Notes:

# Scan Chain Stitching: Stitching Existing Scan Cells

## Scan Chain Stitching: Stitching Existing Scan Cells

- ◆ When stitching existing scan, define the following :
  - Scan enable (If, previously connected)
   
SETUP> SETup SCAn Insertion -SEN scan\_en
  - The “data\_in” field of the DFT library model

```
model sff (D, SI, SE, CLK, Q, QB) (
  scan_definition(
    type = mux_scan;
    data_in = D;
    scan_in = SI;
    scan_enable = SE;
    scan_out = Q, QB;
    non_scan_model = dff (D, CLK, Q, QB);
  )
  input (D, SI, SE, CLK) ()
  intern(_D) (primitive = _mux (D, SI, SE, _D);)
  output(Q, QB) (primitive = _dff(, CLK, _D, Q, QB);)
```



## Notes:

# Lab: Configuring Scan Chains/Test Logic and Full Scan Flow

## Objectives

- Set up scan pins new and existing. (internal or external)
- Create, configure, and balance scan chains.
- Insert scan cells without stitching and write out a scan chain order file.
- Stitch the scan chain with the modified order file.

## List of Exercises

- Exercise 5: Setting Up New and Existing Scan Pins (Internal or External)
- Exercise 6: Balancing Scan Chains With Multiple Clock Domains
- Exercise 7: Writing and Editing Scan Chain Order Files and Stitching (Optional)

## Getting Started

1. Log in to your workstation if you are not already logged in.
2. Change to the `$ATPGNW/lab3/exercise_5` directory.

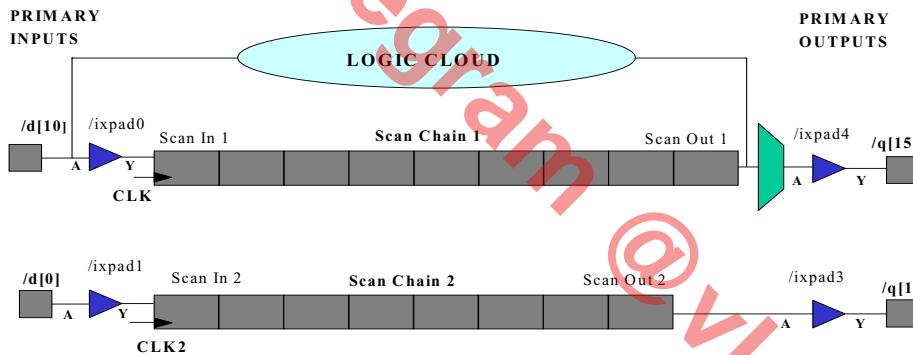
```
shell> cd $ATPGNW/lab3/exercise_5
```

## Exercise 5: Setting Up New and Existing Scan Pins (Internal or External)

In this exercise, you will:

- Invoke DFTAdvisor on a design to insert full scan.

- Use DFTAdvisor to connect existing internal pins as scan inputs and outputs for scan chains.
- Create an internal scan chain.
- Use DFTAdvisor to write a scan-inserted netlist file and ATPG setup files.
- Set up scan pins new and existing (internal and external) and insert test logic as shown in [Figure 3-1](#).



**Figure 3-1. Connecting Existing Pins/Pads as Scan Inputs and Outputs**

1. Invoke DFTAdvisor.

```
shell> dftadvisor
```

- a. Enter the following in the appropriate dialogue boxes. Use the Browse button to find the files quickly.

Design: 1\_design.v

Design Format: Verilog

Library: adk.atpg

Log file: results/dfta\_1\_design.log

- b. Click on Invoke DFTAdvisor. DFTAdvisor should be up and running with the Main and Control Panel windows both open.
2. Define the control signals.
  - a. No pins are required to be held at a constant value.
  - b. Define the clocks automatically via a dialogue box or a command.

- i. Which three control signals have been identified? What are their off states?

Signal: \_\_\_\_\_ Off State: \_\_\_\_\_

Signal: \_\_\_\_\_ Off State: \_\_\_\_\_

Signal: \_\_\_\_\_ Off State: \_\_\_\_\_

Default scan configurations will be used, so we are finished with defining the design and are ready to go to the next stage, DFT mode, where scan/test circuitry is identified and inserted.

3. Go to DFT mode.

Do this at the command line, or by clicking on Done with Setup in the Control Panel pane.

You now should be in the DFT system mode.

Work through the process flow as indicated in the panel, starting with Setup Identification.

- a. **Setup Identification:** Select Full Scan.
- b. **Run Identification:** Identify the items that need altering for scan/test in the next step. Select Run with Existing Settings.

Dismiss the window until only the Main and Control Panel windows remain open.



DFTAdvisor has identified scannable instances to convert to scan, but it does not alter the netlist.

**Note**

### c. Setup/Run Test Synthesis:

i. Run with the following settings; leave others with defaults.

- Synthesize Scan Circuitry into the Design

#### **Scan List and Chain Restrictions tab:**

- o Insert Scan Cells Based on: The Current Scan Identification List
- o Insert 2 chains
- o Allow Only One Scan Enable Signal to Control all Scan Chains

#### **Connection & Clocking Restrictions tab:**

- o Scan-Specific Pins on the Scan Cells Should be: Connected into a Chain
- o A Single Scan Chain Can Have: Scan Cells Controlled By Different Clocks

#### **Scan Chain I/O Naming tab:**

- |                      |           |
|----------------------|-----------|
| o Chain Name:        | Chain1    |
| o Scan Input:        | /ixpad0/Y |
| o Output:            | /ixpad4/A |
| o Controlling Clock: | /clk      |

- o Select — The Specified Scan Pins are Internal and Map to Top Level:

Scan In: d[10]

Out: q[15]

Add this chain definition to the list:

- o Click Add.
  - i. Compare the above definition with the figure 3-1 on page 3-33 to ensure that you understand the logic behind the choices made.
  - ii. Define Chain2 from the information in figure 3-33 and add it to the list following the steps outlined for Chain1.

### Other Test Pin Naming tab:

- o Scan Enable: Ensure the name is scan\_en (default)
- iii. Click Done.
  - No changes need to be made to the following:
    - o Synthesize Identified Test Points
    - o Synthesize Test Logic to Control RAMs

You mostly use the defaults for the test synthesis settings, with the exception of the insertion of the two scan chains defined as internal chains, and the mapping of these internal chains to the I/O of primary ports.

Automatic Test Equipment (ATE) needs deep serial memory for every scan input and output pin. Tester time and cost is reduced with more and shorter scan chains.

- d. You get one more chance to change things, but you don't need to; just run with existing settings.



A scan-inserted netlist has now been created that reflects the changes you specified,

- e. **Save Results:** Click on the Save Results... button in the Button pane to bring up the Save Results dialogue window. Use the dialogue to save the netlist and setup files for FastScan.
  - i. Save a new netlist in Verilog format to the file results/design\_scan.v
  - ii. Save an enhanced procedure ATPG setup file with a basename of results/design\_scan (no file extension)
  - iii. Overwrite any files of the same names that may already be present for either the netlist or ATPG setup files.
  - iv. Finish by clicking OK.

You have now written the following three files:

- design\_scan.v, which is the Verilog netlist
- design\_scan.dofile, which is a dofile
- designscan.testproc, which is an enhanced procedure file

You are finished with creating a scan inserted netlist. The next stage is to generate the test patterns that are used with the netlist.

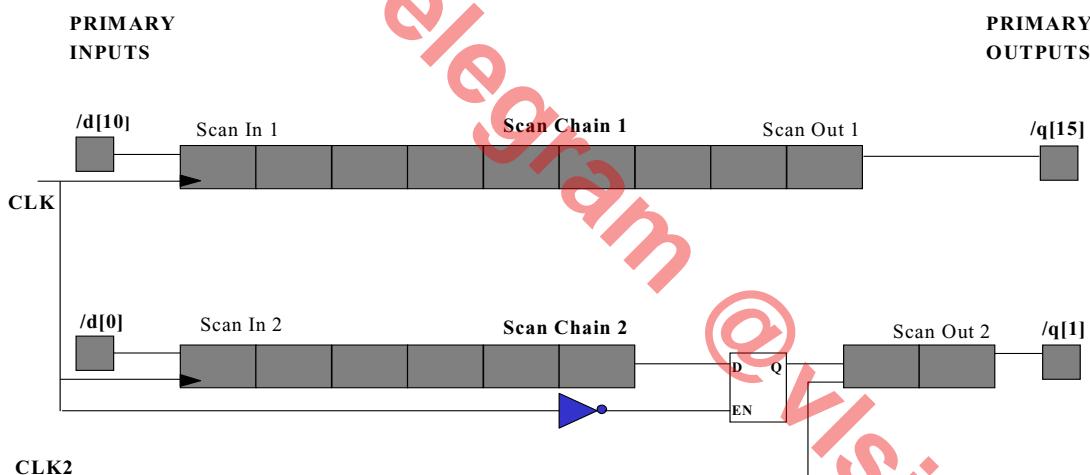
4. Exit DFTAdvisor.

### Exercise 6: Balancing Scan Chains With Multiple Clock Domains

In this exercise, you will:

- Invoke DFTAdvisor on a design to insert full scan.
- Use DFTAdvisor to create, configure, and balance scan chains using lockup latches.
- Insert test logic to select data from either the scan chains or the design.
- Write a scan-inserted netlist file and ATPG setup files.

You create two scan chains and insert a lockup latch as shown in [Figure 3-2](#).



**Figure 3-2. Balancing Scan Chains Using Lockup Latches**

1. Change to the `$ATPGNW/lab3/exercise_6` directory.

```
shell> cd $ATPGNW/lab3/exercise_6
```

2. Invoke DFTAdvisor.

```
shell> dftadvisor
```

- a. Enter the following in the appropriate dialogue boxes. Use the Browse button to find the files quickly.

Design: design\_noscan.v

Design Format: Verilog

Library: adk.atpg

Log file: results/design.log

- b. Click Invoke DFTAdvisor. DFTAdvisor should now be up and running with the Main and Control Panel windows open.
3. Define the control signals.
  - a. No pins are required to be held at a constant value.
  - b. Define the clocks automatically via a dialogue box or a command.
    - i. Which three control signals have been identified? What are their off states?

Signal: \_\_\_\_\_ Off State: \_\_\_\_\_

Signal: \_\_\_\_\_ Off State: \_\_\_\_\_

Signal: \_\_\_\_\_ Off State: \_\_\_\_\_

- c. Insert lockup latches and test circuitry for uncontrollable clocks.
  - i. Click on the Test synthesis Setup... button in the Button pane.
  - ii. After you click the **Test Synthesis Setup** button in the button pane the **Setup for Test Synthesis** dialogue box opens. Select the following settings:

### Automatic Scan Identification tab:

- o Full Scan (default)
- o Scan architecture: Mux\_Scan

### Lockup Latches tab:

DFTAdvisor lets you merge scan cells with different shift clocks into the same scan chain. However, to avoid synchronization problems lockup latches are inserted.

- d. Select Library Models...
  - o Specify the DFT Library Models That May be used: latch
  - o Enable pin: CLK
  - o Data Pin: D
  - o Output Pin: Q
- e. Click the Add button.

You have specified to the tool that you will insert a lockup latch. You selected the DLAT model from the DFT library.

**Test Logic/Latch Scannability tab:**

- f. Enable Insertion of Test Logic for the Following Types of Uncontrollable Signals:
  - o Sets.
- g. Click **OK** to finish.

You have enabled DFTAdvisor to insert test logic for uncontrollable clock circuitry (for example, the rstn control signal).

What command adds the latch cell model to the test logic? (*Session Transcript pane*)

---

4. Go to DFT mode.

Do this at the command line, or by clicking on Done With Setup in the Control Panel pane.

You now should be in the DFT system mode.

Work through the process flow as indicated in the panel, starting with Setup Identification.

- a. **Setup Identification:** Select Full Scan.
- b. **Run Identification:** Identify the items that need altering for scan/test in the next step. Select Run with Existing Settings.

Dismiss the window until only the Main and Control Panel windows remain open.



DFTAdvisor has identified scannable instances to convert to scan, but it does not alter the netlist.

### Note

- c. **Setup/Run Test Synthesis:** Perform the Test Synthesis, inserting scan chains and connecting scan circuitry, adding scan I/O and inserting a lockup latch.

- i. Run with the following settings; leave others as defaults.

- Synthesize Scan Circuitry into the Design

#### **Scan List and Chain Restrictions tab:**

- o Use the Current Scan Identification List
- o Insert 2 chains
- o Allow Only One Scan Enable Signal to Control all Scan Chains

#### **Connection & Clocking Restrictions tab:**

- o A Single Scan Chain Can Have: Scan Cells Controlled By Different Clocks

#### **Scan Chain I/O Naming tab:**

- o Chain Name: Chain1
- o Scan Input: /d[10]
- o Output: /q[15]
- o Controlling Clock: /clk

Compare the above definition with Figure 3-2 on page 3-38. and ensure that you understand the logic behind the diagram.

Define Chain2 from the information in the figure and add it to the list.

#### **Lockup Latch Insertion tab:**

- o Verify that the correct latch is being used:  
DLAT CLK D -noinvert 0
- o Select Automatically Insert Lockup-Latches Between Different Clock Domains.

#### **Grouping Cells by Clock tab:**

- o Type “group1” in the Specify a Name for this “Clock Group” field.
- o Click on the pulldown button in the Scan Cells in this Group are Controlled by these Clocks field. Highlight both /clk and /clk2.
- o Click the **Add** button.



**Note**

You must define clock group(s) to enable the insertion of lockup latches.

- No changes need to be made to the following:

- o Synthesize Identified Test Points
- o Synthesize Test Logic to Control RAMs

You have specified the scan chains, the lockup latch and the clock to be grouped on one chain.

- d. You get one more chance to change things, but you don't need to; just run with existing settings.

You inserted test logic and a lockup latch between the two clock domains, created two balanced scan chains, and mapped scan inputs and scan outputs to the top level external pins.

5. Check the results by issuing several report commands.

- a. DFT> **report scan chain**

This command displays a report on all the current scan chains and provides the following information in the session transcript area:

- Name of scan chain
- Name of the scan chain group
- Scan input and output pins
- Length of the scan chain

- b. DFT> **report scan cells**

This command displays a report or writes a file on the scan cells within specified scan chains and provides the following information in the session transcript area:

- Chain cell index number (where 0 is the scan cell closest to the scan-out pin)
- Scan chain in which the scan cell resides

- Scan group in which the scan cell resides (*dummy*) is the default group name
    - Instance name of the scan cell
    - Scan cell model
    - Global clock for each scan cell
    - Associated lockup latch for each scan cell
- c. DFT> **report test logic**
- Displays the test logic that DFTAdvisor added during the scan insertion process in the session transcript area.
- d. **Save Results:** Click the **Save Results...** button in the Button pane to open the **Save Results dialogue** box.
- i. Save a new netlist in Verilog format to the file results/IOPAD\_counter\_scan\_new.v.
  - ii. Save an enhanced procedure ATPG Setup file with a basename of results/IOPAD\_counter\_scan\_new. (no file extension)
  - iii. Overwrite any files of the same name that may be present for either the netlist or ATPG Setup files.
  - iv. Finish by clicking OK.

You have written the following three files:

- IOPAD\_counter\_scan\_new.v, which is the Verilog netlist
- IOPAD\_counter\_scan\_new.dofile, which is a dofile file
- IOPAD\_counter\_scan\_new.testproc, which is an Enhanced Procedure file

6. Exit DFTAdvisor.

## Exercise 7: Writing and Editing Scan Chain Order Files and Stitching (Optional)

In this exercise you invoke DFTAdvisor on a design to insert full scan. You insert scan cells, but they will not be stitched into a scan chain. Then you write out a scan cell order file.

Next, you edit the scan cell order file and then stitch scan chains using the modified scan cell order file.

1. Change to the \$ATPGNW/lab3/exercise\_7 directory.

```
shell> cd $ATPGNW/lab3/exercise_7
```

2. Invoke DFTAdvisor.

```
shell> dftadvisor
```

- a. Enter the following in the appropriate dialogue boxes. Use the Browse button to find the files quickly.

Design: IOPAD\_counter\_noscan\_new.v

Design Format: Verilog

Library: adk.atpg

Log file: results/dfta\_IOPAD.log

- b. Click Invoke DFTAdvisor. DFTAdvisor should now up and running with the Main and Control Panel windows open.

3. Define the control signals.

- a. No pins are required to be held at a constant value.

- b. Define the clocks automatically via a dialogue box or a command.

- i. Which three control signals have been identified? What are their off states?

Signal: \_\_\_\_\_ Off State: \_\_\_\_\_

Signal: \_\_\_\_\_ Off State: \_\_\_\_\_

Signal: \_\_\_\_\_ Off State: \_\_\_\_\_

#### 4. Go to DFT mode.

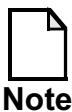
Click on Done with Setup in the DFTAdvisor control panel pane.

You should now be in DFT system mode.

Work through the process flow as indicated in the Control Panel, starting with Setup Identification.

- Setup Identification:** Select Full Scan.
- Run Identification:** Identify the items that need altering for scan/test in the next step. Select Run with Existing Settings.

Dismiss the window until only the Main and Control Panel windows remain open.



DFTAdvisor has identified scannable instances to convert to scan, but it does not alter the netlist.

#### Note

- Setup/Run Test Synthesis:** Perform the Test Synthesis, inserting scan chains and connecting scan circuitry, adding scan I/O and inserting a lockup latch.
  - Run with the following settings; leave others as defaults.
    - Synthesize Scan Circuitry into the Design

#### Connection & Clocking Restrictions tab:

- Scan-Specific Pins on Scan Cells Should be: Connect to a Self-Loop

- o A Single Scan Chain Can Have: Scan Cells Controlled by Different Clocks
- o Click Done to exit the Scan Synthesis Setup window.

DFT Advisor avoids stitching scan cells into scan chains by connecting the scan cells into self-loops.

Although the scan cells are not stitched into a scan chain at this time, DFT Advisor places scan cells using the same clock adjacent to each other. This step enables DFT Advisor to merge scan cells with different shift clocks into scan chains when they are stitched.

- No changes need to be made to the following:
  - o Synthesize Identified Test Points
  - o Synthesize Test Logic to Control RAMs

You have specified the scan chains to be connected in a self-loop and the clock that can be grouped on one chain.

- ii. You get one more chance to change things, but you don't need to; just run with existing settings.

5. Write out the scan chain information.

- a. **DFT>report scan cells -all -filename \ results\scan\_cell\_order\_file -replace**

You saved the order of the scan cells in a new file named: scan\_cell\_order\_file in the results directory.

6. View scan cell order file using the text viewer/editor by selecting **File > Open > Text File (Editable)**. This opens a **View Text File** dialogue box.

Instance pathname	Cell_id	Chain_id
Wix254	0	1
/reg_q_14	1	1
/reg_q_13	2	1
/reg_q_12	3	1
/ix304	4	1
/ix314	5	1
/ix324	6	1
/ix334	7	1
/ix274	8	1
/ix284	9	1
/ix294	10	1
/q_3_rename	11	1
/q_2_rename	12	1
/q_0_rename	13	1
/q_1_rename	14	1
/ix264	15	1
/reg_q_15	16	1

**Figure 3-3. Scan Cell Order File**

The file should look similar to the one in figure 3-3.

The following information is available from the scan\_cell\_order\_file file:

- **Instance.pathname**
  - A string that specifies the name of the scan cell to be placed in the scan chain.
- **Cell\_id**

- An integer that specifies the placement of the instance.pathname in relation to other instance.pathname s in the scan chain. All instances in the same chain must have unique cell\_ids.
- **Chain\_id**
  - An integer that specifies the scan chain in which you want DFTAdvisor to place the instance.pathname. DFTAdvisor places instances with the same chain\_id in the same chain.

As shown in Figure 3-3, the instances are positioned in the scan chain according to their cell\_id. Instance /ix254 has a cell\_id of 0 which indicates that it is positioned nearest to scan out.

Also note that it is located in scan chain 1.

Cell-ordering determines bit position in scan chains. Cell\_id 0 is the Least Significant Bit (LSB). FastScan loads patterns into the scan chains at bit 16 and unloads patterns out of the scan chains at bit 0.

DFTAdvisor views instances as being equal in precedence. They can be positioned in any order and also moved into different scan chains.



**Note**

Arbitrarily moving instances to different scan chains  
can result in clock domain issues.

Instances can be stitched into any order that you choose. However, better placement and routing of scan cells results in better stitching.



**Note**

In physical layout, the order of the scan cells is critical in routing other design criteria. Use a placement and routing (P&R) tool to reorder the scan cells in the scan chain and write a scan cell order file. Then use DFTAdvisor to stitch scan cells into scan chains using the scan cell order file that you created with the P&R tool.

If you create a design's scan on a block-by-block basis, then each instance or module has its own pre-existing scan chains or subchains. These

subchains need to be incorporated into the top-level scan chains. This is accomplished by DFTAdvisor during the stitch process.

Subchain instances always have a \$ sign in the instance\_pathname. For example, /I\$116 indicates a subchain of instance\_pathname.



Note

When writing a scan cell order file that includes subchains, take into account the number of scan cells within a subchain because this affects the overall cell\_id numbering. For example, the cell\_id number for instance /I\$116 is 10, the instance below it (/ix294) is 15; this indicates that /I\$116 contains 5 scan cells.

In the next section you are going to manually reorder the scan cells in the file: scan\_cell\_order\_file and stitch the reordered file into two scan chains.

7. Reorder the scan cells as shown in [Figure 3-4](#)

```
/ix254 4 2
/reg_q_14_ 3 2
/reg_q_13_ 2 2
/reg_q_12_ 1 2
/ix304 0 2
/ix314 8 1
/ix324 7 1
/ix334 2 1
/ix274 5 1
/ix284 0 1
/ix294 3 1
/q_3_rename_rename 6 1
/q_2_rename_rename 1 1
/q_0_rename_rename 4 1
/q_1_rename_rename 5 2
/ix264 6 2
/reg_q_15_ 7 2
```

**Figure 3-4. Scan Cell Reorder File**

- a. Save this file as results/scan\_cell\_reorder and close the viewer.
8. **Save Results:** Click the **Save Results...** button in the Button pane.
  - i. Use the dialogue to save the netlist in Verilog format to the file results/IOPAD1\_counter\_scan\_new.v.

- ii. Overwrite any files of the same name that may already be present for the netlist.
- iii. Finish by clicking OK.

You have written a new Verilog netlist file: IOPAD1\_counter\_scan.v

9. Exit DFTAdvisor.
10. Re-invoke DFTAdvisor.

shell> **dftadvisor**

- a. Enter the following in the appropriate dialogue boxes. Use the Browse button to find the files quickly.

Design: IOPAD\_counter\_noscan\_new.v

Design Format: Verilog

Library: adk.atpg

Log file: results/dfta\_IOPAD1.log

**Note:** This is the same Verilog file as before, NOT the one you have just generated.

- b. Click Invoke DFTAdvisor. DFTAdvisor should now up and running with the Main and Control Panel windows open.
11. Define the control signals.
  - a. No pins are required to be held at a constant value.
  - b. Define the clocks automatically via a dialogue box or a command.
    - i. Which three control signals have been identified? What are their off states?

Signal: \_\_\_\_\_ Off State: \_\_\_\_\_

Signal: \_\_\_\_\_ Off State: \_\_\_\_\_

Signal: \_\_\_\_\_ Off State: \_\_\_\_\_

12. Go to DFT mode.

Click on Done with Setup in the DFTAdvisor control panel pane.

You should now be in DFT system mode.

Work through the process flow as indicated in the Control Panel, starting with Setup Identification.

a. **Setup Identification:**

**Automatic Scan Identification** tab:

- o Full Scan (default)
- o Scan architecture: Mux\_Scan

**Lockup Latches** tab:

- i. Select Library Models...
  - o Specify the DFT Library Models: latch
  - o Enable pin: CLK
  - o Data Pin: D
  - o Output Pin: Q
- ii. Click the Add button and click OK.

- b. **Run Identification:** Identify the items that need altering for scan/test in the next step. Select Run with Existing Settings.

Dismiss the window until only the Main and Control Panel windows remain open.



DFTAdvisor has identified scannable instances to convert to scan, but it does not alter the netlist.

### Note

- c. **Setup/Run Test Synthesis:** Perform the Test Synthesis ordering and stitching scan cells into scan chains as specified in the scan cell reorder file, and insert a lockup latch.

Click Setup/Run Test Synthesis in the graphic pane. This opens the Setup/Run Test Synthesis box.

- i. Run with the following settings; leave others as defaults:

- Synthesize Scan Circuitry into the Design

#### **Scan List and Chain Restrictions tab:**

- o Select Insert Scan Cells Based On: Instances Specified in File: scan\_cell\_reorder
- o Select — Stitch Cells Using the Specified File Ordering
- o Insert 2 Chains

#### **Connection & Clocking Restrictions tab:**

- o Scan-Specific Pins on Scan Cells Should Be: Connected Into a Chain
- o A single Scan Chain Can Have: Scan Cells Controlled by Different Clocks

#### **Lockup Latch Insertion tab:**

- o Automatically Insert Lockup-Latches Between Different Clock Domains

### Grouping Cells by Clock tab:

- o Specify a Name for This “Clock Group”: Group1
- o Scan Cells in this Group are Controlled by These Clocks: clk & clk2

Select both. The pulldown menu allows you to select more than one clock.

Click Done.

- No changes need to be made to the following:

- o Synthesize Identified Test Points
- o Synthesize Test Logic to Control RAMs

You have specified the scan chains, the lockup latch and the clock that can be grouped on one chain.

- ii. You get one more chance to change things, but you don't need to.  
Run with existing settings.

13. Check the results using several **report** commands.

- a. DFT> **report scan cells**

Look in the Session Transcript area. What cell\_id has instance /ix284 been reordered to?

---

What cell\_id has instance q\_1\_rename\_rename been reordered to?

---

Note the other changes that you made to the scan cell ordering. Refer to [Figure 3-4](#).

- b. DFT> **report scan chain**

Observe in the session transcript area that you created two balanced scan chains.

c. DFT> **report test logic**

Observe in the session transcript area that you added two lockup latches, two inverters, and pins.

d. **Save Results:** Click the **Save Results...** button in the Button pane to open the **Save Results** dialogue box.

- i. Save a new netlist in Verilog format to the file results/IOPAD\_counter\_scan.v.
- ii. Save an enhanced procedure ATPG Setup file with a basename of results/IOPAD\_counter. (no file extension)
- iii. Overwrite any files of the same name that may be present for either the netlist or ATPG Setup files.
- iv. Finish by clicking OK.

You have written the following three files:

- IOPAD\_counter\_scan.v, which is the Verilog netlist
- IOPAD\_counter\_scan.dofile, which is a dofile file
- IOPAD\_counter\_scan.testproc, which is an Enhanced Procedure file

14. Exit DFTAdvisor.

## Test Your Knowledge

1. What command is used to assign scan control pin connections?

---

2. True or False. During Run Identification, the netlist is altered.

---

3. Why add more scan chains? Shorter scan chains?

---

4. What command is used to assign scan ports specific names?

---

5. Why are lockup latches used?

---

6. How do you balance scan chains?

---

7. To enable the insertion of lockup latches, what must be defined?

---

8. True or False. A cell\_id of 0 indicates that an instance is placed next to scan out.

---

## Lab Summary

Now that you have completed the Configuring Scan Chains/Test Logic and Full Scan Flow lab, you should know how to do the following:

- Setup scan pins new and existing (internal or external)
- Insert test logic
- Create, configure, and balance scan chains
- Insert scan cells without stitch and write out a scan chain order file
- Stitch the reordered scan cells into a scan chain
- Write a scan-inserted netlist
- Write ATPG setup files

VLSIGOD Join Telegram @vlsigodofficial

## Module 4

# Understanding ATPGMessaging

### Objectives

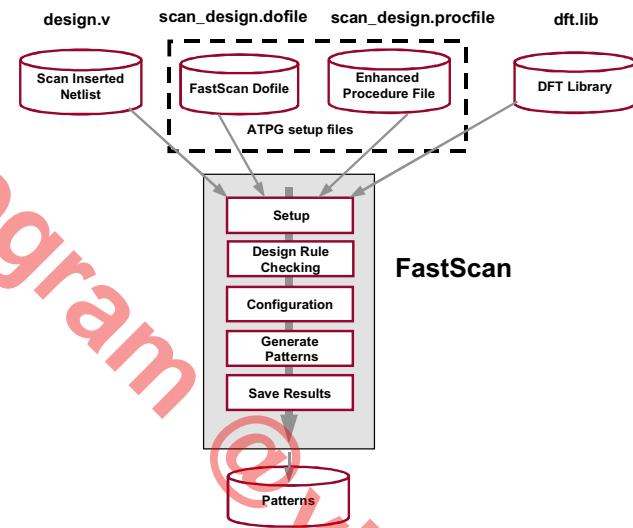
Upon completion of this module, you will be able to:

- Analyze messages at invocation.
- Analyze messages when exiting SETUP mode.
- Analyze ATPG reporting.
- Analyze test coverage reporting.
- Use common methodologies to attain a quick estimate of pattern coverage.

# Module Topics

## Module Topics

- ◆ This module addresses the following topics:
  - DRC reporting
  - ATPG reporting
  - Fault classifications
  - Test coverage reporting
  - Common methodologies



## Notes:

## Messages at Invocation

### Messages at Invocation

- ♦ FastScan displays the following messages at invocation:

The screenshot shows a terminal window with the following text:

```
// FastScan v8.2003_2.10 Wed Feb 28 22:59:12 PST 2003
// Copyright (c) Mentor Graphics Corporation, 1992-2003. All Rights Reserved.
//
// UNPUBLISHED, LICENSED SOFTWARE
// CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
// PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS.
//
// USE OF THIS SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS.
//
// Mentor Graphics software executing under Sun SPARC SunOS.
// 32 bit version
//
// Compiling library ...
// Reading Verilog Netlist ...
// Reading Verilog file test_scan.v
// Finished reading file test_scan.v
// command: add scan groups grp1 test_scan.testproc
// command: add scan chains chain1 grp1 scan_in1 scan_out1
// command: add scan chains chain2 grp1 scan_in2 scan_out2
// command: add clocks 0 CLOCK
...

```

Simple.log

Two blue arrows point from the text to the right:

- An arrow points to the first few lines of the log, labeled "Current version of FastScan".
- An arrow points to the command-line entries starting with "command:", labeled "Compiles DFT library then parses netlist".

### Notes:

# Messages at Invocation: Warnings

## Messages at Invocation: Warnings

- FastScan reports the following types of warnings at invocation:

```
// FastScan v8.2003_2.10  Wed Feb 28 22:59:12 PST 2003
// Copyright (c) Mentor Graphics Corporation, 1992-2003, All Rights Reserved.

// UNPUBLISHED, LICENSED SOFTWARE.
// CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
// PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS.
//
// USE OF THIS SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS.
//
//
// Mentor Graphics software executing under Sun SPARC SunOS,
// 32 bit version
//
// Compiling library ...
...
// Warning: Model name 'EN0_UDP' is duplicated; Last model definition is selected
...
// Warning: Net 'CADJL0' in model 'HSTL2OHV15C' is unused
// Reading Verilog Netlist ...
// Warning: Floating input 'fr_ct_sel' at instance 'design_core0' in module 'design'
// Warning: Net 'Q0' in module 'R64X8S' is not driven
...

```

Simple.log

- Major problems at invocation cause the tool to error and exit
- A file describing the problem will be created in the home directory  
~<toolname>.tx

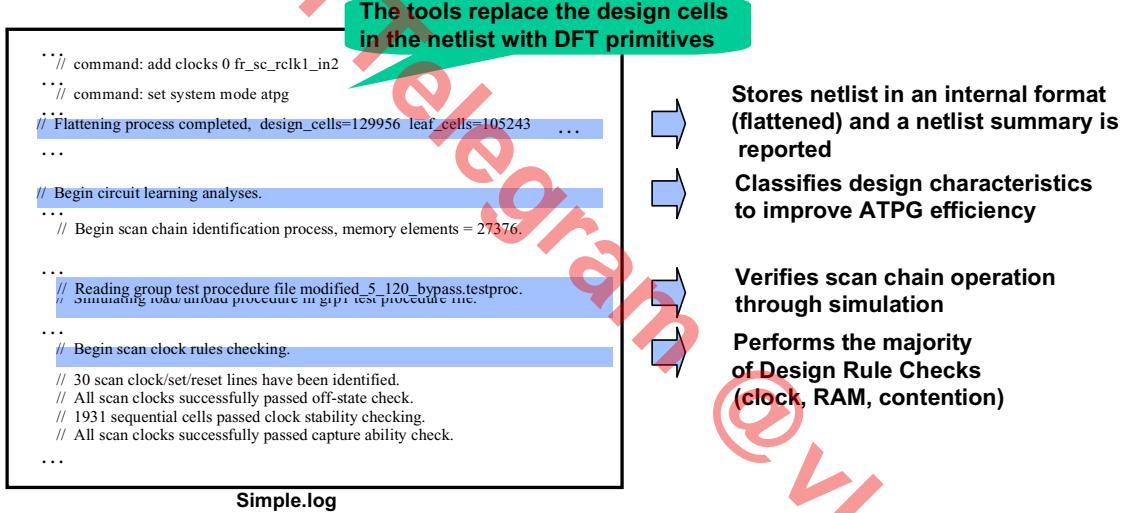
- If, a model is defined several times in the DFT library, the last defined (model) is used
- Warns if model has unused nets
- Warns if unused nets exist in netlist

## Notes:

# Messages when exiting Setup

## Messages When Exiting Setup

- FastScan reports the following messages when exiting Setup:



4-5 • Design-for-Test: Scan and ATPG:  
Understanding ATPG Messaging

Copyright © 2003 Mentor Graphics Corporation

## Notes:

# ATPG Reporting

## ATPG Reporting

- FastScan reports the following messages and status during ATPG

```

...
// command: create patterns
No faults in fault list. Adding all faults...
//
// Simulation performed for #gates = 133 #faults = 90
// system mode = ATPG pattern source = internal patterns
//
// #patterns test #faults #faults # eff. # test process
// simulated coverage in list detected patterns patterns CPU time
// begin random patterns: capture clock = /CLOCK, observe point = MASTER

// begin random patterns: capture clock = none, observe point = MASTER
// deterministic ATPG invoked with abort limit = 30
// --- ----- --- --- 0.00 sec 2/0/0
//
// Simulation performed for #gates = 133 #faults = 88
// system mode = ATPG pattern source = internal patterns
//
// #patterns test #faults #faults # eff. # test process
// simulated coverage in list detected patterns patterns CPU time
// deterministic ATPG invoked with abort limit = 30
// --- ----- --- --- 0.00 sec 2/0/0
// 32 100.00% 0 88 5 5 0.00 sec
...

```

Simple.log

- Use the command **CREATE PATTERNS** to start ATPG.
- By default, patterns are created “internally” rather than read in from an “external” file.
- By default, FastScan captures with one clock at a time. After the clock’s effectiveness is exhausted, FastScan moves to another clock.
- Deterministic patterns are created once all random patterns for all clocks are used up.
- Performs test generation on selected faults from current fault list.

## Notes:

# ATPG Reporting (Cont.)

## ATPG Reporting (Cont.)

- FastScan reports the following messages:

```
...
// command: add fau -all
...
// command: create patterns
// Simulation performed for #gates = 353875 #faults = 11683
// system mode = ATPG pattern source = internal patterns
// #patterns test #faults #faults # eff. # test process
// simulated coverage in list detected patterns patterns CPU time
// begin random patterns: capture clock = /clk1, observe point = MASTER
// deterministic ATPG invoked with abort limit = 30
...
// #patterns test #faults #faults # eff. # test
// simulated coverage in list detected patterns patterns
// 32 64.42% 4305 7107 31 31
// 64 80.27% 2367 1938 32 63
// 96 85.14% 1771 596 32 95
...

```

Simple.log

Command for pattern creation:  
CREATE PATterns

Compressed ATPG

## Notes:

# Special Messages in ATPG Reporting

## Special Messages in ATPG Reporting

- ♦ FastScan reports the following special messages:

```
// command: CREate PAtterns  
⋮  
// Warning: Contention on (205816), number patterns rejected = 32.  
⋮  
// Warning: Unsuccessful test for 2533 faults.
```

Simulated patterns that report bus contention are discarded

Summary of rejected patterns

## Notes:

# Test Coverage Reporting

## Test Coverage Reporting

Statistics report		
	#faults (coll.)	#faults (total)
FU (full)	562	896
DS (det_simulation)	440	746
DI (det_implication)	90	112
UU (unused)	10	10
TI (tied)	8	8
BL (blocked)	4	4
AU (atpg_untestable)	10	16
test_coverage	98.15%	98.17%
fault_coverage	94.31%	95.76%
atpg_effectiveness	100.00%	100.00%
#test_patterns	44	
#simulated_patterns	320	
CPU_time (secs)	0.3	

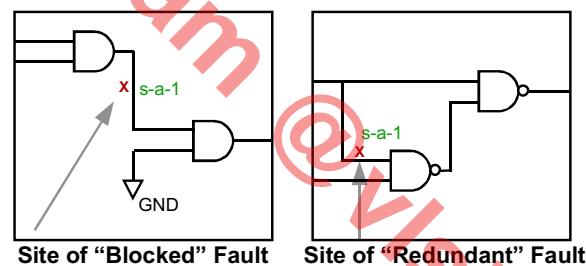
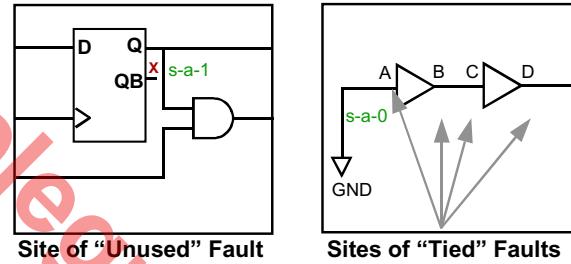
- ◆ FastScan reports faults as testable or untestable
- ◆ The following are testable (TE) faults:
  - Detected (DT):
    - Det\_Simulation (DS)
    - Det\_Implication (DI)
    - Det\_Robust (DR)
  - Posdet (PD)
    - Posdet\_Untestable (PU)
    - Posdet\_Testable (PT)
  - ATPG\_Untestable (AU)
  - Undetected
    - Uncontrolled (UC)
    - Unobservable (UO)

## Notes:

## Test Coverage Reporting (Cont.)

### Test Coverage Reporting (Cont.)

- ◆ The following are untestable faults:
  - Unused (UU)
  - Tied (TI)
  - Blocked (BL)
  - Redundant (RE)



### Notes:

# Test Coverage Reporting (Cont.)

## Test Coverage Reporting (Cont.)

Statistics report		
fault class	#faults (coll.)	#faults (total)
FU (full)	562	896
DS (det_simulation)	440	746
DI (det_implication)	90	112
UU (unused)	10	10
TI (tied)	8	8
BL (blocked)	4	4
AU (atpg_untestable)	10	16
test_coverage	98.15%	98.17%
fault_coverage	94.31%	95.76%
atpg_effectiveness	100.00%	100.00%
#test_patterns		
#simulated_patterns		3
CPW		

**ATPG Effectiveness:**  
A measure of the tool's ability to detect a fault or prove that a test cannot be created with current settings

Test coverage: percentage of all testable faults

$$\text{Test Coverage} = \frac{\#DT + (\#PD * \text{posdet\_credit})}{\#\text{Testable Faults}}$$

Fault coverage

Fault coverage: percentage of all faults both testable and untestable

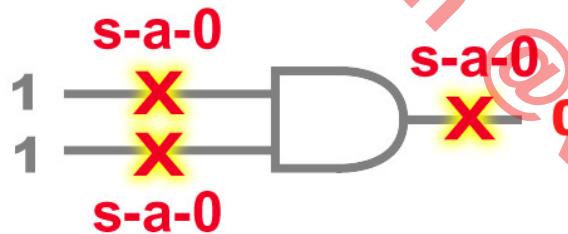
$$\text{Fault Coverage} = \frac{\#DT + (\#PD * \text{posdet\_credit})}{\#\text{Total Faults}}$$

## Notes:

# Test Coverage Reporting Fault Collapsing

## Test Coverage Reporting: Fault Collapsing

- ◆ After FastScan identifies faults, faults that behave the same are reduced, or collapsed, into one equivalent fault.
  - Multiple faults are targeted with one pattern.
  - Total number of generated patterns is reduced.
- ◆ EQ is the designator for equivalent faults.  
(Equal to the fault listed above it in the fault list.)
- ◆ A test for one of the faults above is equal to the test for the other two.



## Notes:

# Determining the Cause of Undetected Faults

## Determining the Cause of Undetected Faults

- ♦ Use the REPort TEstability Data command for the following:
  - To identify circuitry connections that cause test coverage problems
  - To analyze collapsed faults for a specific fault class
  - To display analysis

```
ATPG> REPort TEstability Data -class AU
// fault analysis summary of 7 faults
// number faults connected to tsd_enable = 7
// number faults unclassified = 0
```

## Notes:

# Determining the Cause of Undetected Faults (Cont.)

## Determining the Cause of Undetected Faults (Cont.)

- ♦ Use the REPort FAults command to display fault information from the current fault list

Fault value:  
Either 0 (for stuck-at-0)  
or 1 (for stuck-at-1)

Fault code

Fault site

```
ATPG> REPort FAults -class  
ATPG_UNTESTABLE  
0 AU /I$7/OUT  
1 EQ /I$7/IN  
0 EQ /I$1/en  
1 AU /I$7/OUT  
0 EQ /I$7/IN  
1 EQ /I$1/en  
0 AU /I$4/i1  
0 AU /I$20/en  
1 AU /I$20/en  
0 AU /I$2/en  
1 AU /I$2/en
```

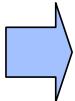
## Notes:

# Determining the Cause of Undetected Faults (Cont.)

## Determining the Cause of Undetected Faults (Cont.)

- ♦ Use the ANALyze FAult command to identify why a fault is not detected.

```
ATPG> REPort FAults -Class
ATPG_UNTESTABLE
 0  AU  /I$7/OUT
 1  EQ  /I$7/IN
 0  EQ  /I$1/en
 1  AU  /I$7/OUT
 0  EQ  /I$7/IN
 1  EQ  /I$1/en
 0  AU  /I$4/i1
 0  AU  /I$20/en
 1  AU  /I$20/en
 0  AU  /I$2/en
 1  AU  /I$2/en
```



```
ATPG> ANALyze FAult I$20/en -stuck_at 1
// -----
-- 
// Fault analysis for /I$20 (16) input en (0) stuck at 1
// -----
-- 
// Current fault classification = AU (atpg_untestable)
// Fault is blocked at /I$20 (16) due to tri-state enable.
// Controllability justification was successful (data
// accessible using parallel_pattern 0).
// Pattern type: Basic_scan.
// Test generation cannot be performed - no unblocked
// path to observe point.
```

## Notes:

# Lab: Understanding ATPG Messaging

## Objectives

- Read and analyze:
  - Messages at invocation.
  - Warnings at invocation.
  - Messages when exiting setup.
  - DRC reporting.
  - ATPG reporting.
  - Fault classifications.
  - Special messages.
  - Test coverage reporting.
- Determine the cause of undetected faults.
- Specify areas not to target for ATPG.
- Write an external fault list.
- Use common methodologies to attain a quick estimate of pattern coverage:
  - Assessing test pattern coverage through implication.
  - Using Fault sampling.
  - Limiting ATPG test coverage.
  - Loading in an external fault list.

## List of Exercises

- Exercise 8: Reading and Analyzing Messages
- Exercise 9: Determining the Cause of Undetected Faults and Adding NOfaults
- Exercise 10: Common Methodologies to Attain a Quick Estimate of Test Coverage

## Getting Started

1. Log in to your workstation if you are not already logged in.
2. Change to the \$ATPGNW/lab4/exercise\_8 directory.

```
shell> cd $ATPGNW/lab4/exercise_8
```

## Exercise 8: Reading and Analyzing Messages

- In this exercise, you will:
  - Invoke FastScan on a design and apply pattern types.
  - Read and analyze the following messages: invocation, warning, exiting setup mode, DRC, ATPG, fault classifications, special, and test coverage.
  - Create patterns using the **create patterns** command and observe ATPG messaging.

1. Invoke FastScan on the following circuit:

Design: module4\_8.v

Design Format: Verilog

Library: adk.atpg

Log file: results/ex\_8.log

2. Study the shell window to identify the current version of FastScan:

3. Study the Session Transcript window to identify the following:

- a. Information messages:

- i. \_\_\_\_\_  
ii. \_\_\_\_\_  
iii. \_\_\_\_\_  
iv. \_\_\_\_\_  
v. \_\_\_\_\_

- b. Warning messages:

- i. What problem is there with mux21\_macro?

\_\_\_\_\_

- ii. Which modules are undefined?

\_\_\_\_\_

- iii. What is the issue with the nets?

\_\_\_\_\_

4. Black box the undefined module. (The command is listed in the Session Transcript window as a warning.)

What command do you use? \_\_\_\_\_

Open the reference page for the **black\_box** command and explain what the **-auto** switch does (*Help menu*):

Black boxing is used for the following:

- Analysis of incomplete designs
- Isolating analog blocks
- Isolating proprietary Intellectual Property (IP)

### 5. Add clocks automatically.

Various messages and warnings appear in the Session Transcript window. Study these and answer the following question. Remember that documentation is available to you if you need help answering the questions.

a. How many loops were broken from adding the clocks?

---

b. How many, if any, of the loops were duplicated?

---

c. What is an FN1 violation?

---

d. How many floating nets are there?

---

e. FastScan replaces the design cells in the netlist with DFT primitives. The netlist is stored in an internal format (flattened).

The following information is available after flattening:

- i. No. of design cells: \_\_\_\_\_
  - ii. No. of leaf cells: \_\_\_\_\_
  - iii. No. of library primitives: \_\_\_\_\_
  - iv. No. of netlist primitives: \_\_\_\_\_
  - v. No. of simulation gates: \_\_\_\_\_
  - vi. No. of primary inputs: \_\_\_\_\_ primary outputs \_\_\_\_\_
- f. FastScan classifies the design's characteristics in order to enable it to improve ATPG efficiency. The process is called 'circuit learning'.
- Look at the results of the circuit learning analyses and identify the following:
- i. No. of equivalent gates: \_\_\_\_\_
  - ii. No. of learned relationships: \_\_\_\_\_
  - iii. No. of wired\_gates modeled as WIRE gates: \_\_\_\_\_
- g. The last step of adding control signals automatically is to actually identify them — control signals identification analysis.
- i. How many control signals were identified in all? \_\_\_\_\_
  - ii. How many clocks are there? \_\_\_\_\_
6. Add scan group, test procedure file, and scan circuitry.

Had we already inserted the scan circuitry using DFTAdvisor, the generated dofile would perform the setup for us. However, since we did not generate a dofile, we have to define the scan circuitry now.

- a. Click on Scan Circuitry in the Circuit Setup pane. A Setup Scan Circuitry dialogue box opens.

b. Scan Groups Tab:

Group Name: grp1

Test Procedure file: exercise\_8/gate\_afterdft\_100.testproc

c. Scan Chains tab:

Add the following scan definitions:

**Table 4-1. Scan Definitions**

Group	Chain	Scan_In	Scan_out
grp1	chain1	scan_in5	scan_out1
grp1	chain2	scan_in6	scan_out2
grp1	chain3	scan_in7	scan_out3
grp1	chain4	scan_in8	scan_out4



You may find it useful for future labs to create a dofile with the scan setup commands that are generated in this section. There are 5 command lines in the file. Remember to delete unwanted text, including the '// command:' at the beginning of each line.

7. The circuit in this lab has the potential for bus contention. For the initial run, force all buses to a non-contention state to ensure that the test generator does not create patterns that cause contentions.

- a. Set contention check by typing:

```
SETUP> set contention check on -atpg
```

If FastScan cannot satisfy this condition, the tool aborts the fault, excludes the pattern from the final test set, and displays a message indicating the number of these aborted faults for each simulation pass.

8. Go to ATPG mode.

- a. Click on the Done With Setup button in the graphics pane. A FastScan Session Purpose dialogue box opens.

- i. Click on the Pattern Generation button.

You should now be in ATPG system mode.

- ii. Upon exiting setup, FastScan performs the majority of Design Rule Checks (clock, RAM, contention) using the information in the test procedure file. Several messages are written to the Sessions Transcript window:

a. How many memory elements are there? \_\_\_\_\_

b. What is the length of each of the four scan chains?

chain1\_\_\_\_\_ chain2\_\_\_\_\_ chain3\_\_\_\_\_ chain4\_\_\_\_\_

c. What is a T18-1 violation? \_\_\_\_\_

\_\_\_\_\_

d. The warning refers to an ‘entered value’. Where is this value defined? \_\_\_\_\_

e. How many scan clock set/reset lines have been identified?

\_\_\_\_\_

f. How many gates have an E5 violation? \_\_\_\_\_ Define an E5 violation:

\_\_\_\_\_

9. Generate random and deterministic patterns for stuck-at-faults.

Observe in the Session Transcript window that FastScan reports messages and status during ATPG.

FastScan begins with random pattern generation, then deterministic pattern generation.

10. Note that FastScan reports messages and status during ATPG to the Sessions Transcript window. The flow goes from random pattern generation to deterministic pattern generation.
  - a. Enter the number of simulated and effective patterns for the various types:

**Table 4-2. ATPG Run Statistics**

Type	Simulated Patterns	Effective Patterns
Random		
Deterministic		
Totals		

- b. What is the difference between effective and simulated patterns?
- 
11. Close the FastScan ATPG Run Statistics dialogue box.
  12. Obtain a detailed report by issuing the following command:

```
ATPG> report statistics
```

The results are displayed in the Session Transcript window. This information provides data about the number of collapsed and total faults in each class and across all classes. Details of coverage, ATPG effectiveness, and simulated patterns are found here.

- a. Refer to the Session Transcript window to fill in the following:

**Table 4-3. Report Statistics ATPG**

Fault Class	# faults (coll.)	# faults (total)
FU (full)		
UO (unobserved)		

**Table 4-3. Report Statistics ATPG**

Fault Class	# faults (coll.)	# faults (total)
DS (det_sim)		
DI (det_imp)		
PU (posdet_untestable)		
PT (posdet_testable)		
UU (unused)		
TI (tied)		
RE (redundant)		
AU (atpg_untestable)		

**Note**

Refer to the ATPG Tools Reference Manual for clarification on the fault classes.

**Table 4-4. Report Coverage/Effectiveness ATPG**

Cov./Effect.	# faults (coll.)	# faults (total)
test_coverage		
fault_coverage		
atpg_effectiveness		
# test_patterns		
# simulated_patterns		
CPU_time (secs)		

13. Next, you are going to do a second run using the dynamic **create patterns** command.
  - a. To reset Fastscan ATPG go back to SETUP mode.

You can do this with a command or by clicking on the Done With Pattern Generation button in the Control Panel pane or by entering a command at the command line prompt.

What two things happen when you enter SETUP mode?

- i. \_\_\_\_\_
- ii. \_\_\_\_\_

14. Go to ATPG mode.

- a. Create dynamically compressed patterns.

ATPG > **create patterns**

Creating patterns using this command may take a little while. Look at the Session Transcript window to fill in the blanks.

What is the order of events as described in the Session Transcript window?

1. Adding \_\_\_\_\_
2. \_\_\_\_\_ performed for \_\_\_\_\_
3. System mode = \_\_\_\_\_
4. Pattern source = \_\_\_\_\_
5. Simulated coverage \_\_\_\_\_ detected \_\_\_\_\_
6. \_\_\_\_\_ ATPG invoked with abort limit = \_\_\_\_\_
  - i. What is the final pattern count? \_\_\_\_\_
  - ii. Obtain a detailed report using **report statistics** on the design to fill in the table:

**Table 4-5. Report Statistics ATPG****Table 4-6.**

Fault Class	# faults (coll.)	# faults (total)
FU (full)		
UO (unobserved)		
DS (det_sim)		
DI (det_imp)		
PU (posdet_untestable)		
PT (posdet_testable)		
UU (unused)		
TI (tied)		
RE (redundant)		
AU (atpg_untestable)		

**Table 4-7. Report Coverage/Effectiveness ATPG****Table 4-8.**

Cov./Effect.	# faults (coll.)	# faults (total)
test_coverage		
fault_coverage		
atpg_effectiveness		
# test_patterns		
# simulated_patterns		
CPU_time (secs)		

Compared to the previous run, how many patterns are there?

---

15. Exit FastScan.

## Exercise 9: Determining the Cause of Undetected Faults and Adding NOfaults

In this exercise you invoke FastScan on a design and apply pattern types. You read and analyze messages then determine the cause of undetected faults by using the following commands:

- **Report Testability Data**
  - Identify circuitry connections that cause problems.
  - Analyze collapsed faults for a specific fault class.
  - Display analysis.
- **Report Faults**
  - Display fault information from the current fault list.
- **Analyze Fault**
  - Identify why a fault is not detected.

Finally, you use the **NOfault** command to place nofault settings on specified instances, re-run FastScan, and observe test coverage results.

## Getting Started

1. Change to the \$ATPGNW/lab4/exercise\_9 directory.

```
shell> cd $ATPGNW/lab4/exercise_9
```

2. Invoke FastScan on the following circuit:

Design: module4.v

Library: atpglib

Log file: results/ex\_9.log

3. Study the Session Transcript window to identify the following:

a. Information messages:

- Compiling library
- Reading Verilog netlist
- Reading Verilog file

b. Warning messages:

What warning messages do you see?

---

---

4. Add clocks automatically.

Various messages and warnings appear in the transcript. What warning messages do you see?

---

---

5. The next step is to add the scan group, test procedure file, and scan circuitry, as you did in the previous exercise.

If you saved the scan setup file as a dofile in the last exercise you will find it useful here. You will have to edit it to alter the name of the test procedure (.testproc) file to make it applicable for this lab. If you did not save it, perform the following steps:

a. Set up the Scan Group with the following information:

i. **Scan Groups Tab:**

Group Name: grp1

Test Procedure file: exercise\_9/gate\_afterdft\_100.testproc

ii. Scan Chains tab:

- Add the following scan definitions:

**Table 4-9. Scan Definitions**

Group	Chain	Scan_In	Scan_out
grp1	chain1	scan_in5	scan_out1
grp1	chain2	scan_in6	scan_out2
grp1	chain3	scan_in7	scan_out3
grp1	chain4	scan_in8	scan_out4

- Normal full scan vectors are not sufficient for this circuit, so we are going to generate more complex vectors. These will be explained fully in the next module. For now, it is not necessary to understand the concept in order to use them.

Set simulation mode to RAM sequential and clock sequential.

```
SETUP > set simulation mode ram -depth 3
```

- Go to ATPG mode.

You should now be in ATPG system mode.

Upon exiting setup, FastScan performs the majority of Design Rule Checks (clock, RAM, and contention) using the information in the test procedure file. Several informational messages are output as it goes through the procedure.

- How many RAMs are there? \_\_\_\_\_
- Generate random and deterministic patterns for stuck-at faults.
- Note the number of simulated and effective patterns for the various types.

**Table 4-10. ATPG Run Statistics****Table 4-11.**

Type	Simulated Patterns	Effective Patterns
Random		
Deterministic		
Totals		

10. Close the FastScan ATPG Run Statistics dialogue box.
11. Obtain a detailed report of the design's statistics for the ATPG run and fill in the following statistics:

**Table 4-12. Report Statistics ATPG****Table 4-13.**

Fault Class	# faults (coll.)	# faults (total)
FU (full)		
UO (unobserved)		
DS (det_sim)		
DI (det_imp)		
PU (posdet_untestable)		
UU (unused)		
TI (tied)		
BL (blocked)		
RE (redundant)		
AU (atpg_untestable)		

**Table 4-14. Report Coverage/Effectiveness ATPG****Table 4-15.**

Cov./Effect.	# faults (coll.)	# faults (total)
test_coverage		
fault_coverage		
atpg_effectiveness		
# test_patterns		
# simulated_patterns		
CPU_time (secs)		

- a. What types of test patterns make up the complete set, and how many patterns of each type are there?
  - \_\_\_\_\_ No. of patterns \_\_\_\_\_
  - \_\_\_\_\_ No. of patterns \_\_\_\_\_
  - \_\_\_\_\_ No. of patterns \_\_\_\_\_
12. Notice that there are still undetected faults in several specific fault classifications that are affecting test coverage. You need to analyze these and determine their cause. First we look at the various fault classes and analyze them.



Refer to Chapter 2 of the *Scan and ATPG Process Guide* for more information on fault classes and their definitions.

**Note**

- a. Start with the ATPG untestable faults.

```
ATPG> report testability data -class au
```

- b. Repeat this for class ut.

- c. Repeat this for class ud.

Fill in the table below with the results from this command:

**Table 4-16. Report Testability Data**

Class	Total No. of faults	No. of faults tied by constr.	No. of faults connected to RAM	No. of faults unclassified
AU				
UT				
UD				

13. Next you look at individual faults, still on a class by class basis.

- a. First you get the fault information from the fault list.

```
ATPG> report fault -class atpg
```

This opens a Report Faults Output dialogue box, which shows the following information for every fault in the requested class:

**Type** (fault value)  
Stuck at 1 or 0

**Code** (fault code)  
AU (in this case)

**Pin Pathname** (fault site) e.g.  
/p6/pic1/regs/U43/S0

This gives you all the information needed to further investigate individual faults.

- a. Choose one of the faults in the list and investigate further.

```
ATPG> analyze fault /p1/pic1/regs/U37/A0 -stuck 0
```

Why did we use **-stuck 0**?

What does the resulting fault analysis (seen in the Sessions Transcript window) tell you about the fault?

- b. Now analyze another fault and investigate further.

ATPG> **analyze fault /p1/pic1/regs/U37/A1 -stuck 0**

What does the resulting fault analysis (seen in the Sessions Transcript window) tell you about the fault?

---

---

---

- c. Now look at the faults for the **Tied** fault class. What command do you use? \_\_\_\_\_

Type this command at the command prompt.

- d. Now analyze the following three faults and determine what they are tied to, using the same command as in part 13.b.

**Table 4-17. Analyze Faults**

Node	Stuck at value	Source of the tied gate	Tied to what value?
/p1/pic1/add_794/U39/Y	1		
/p1/pic1/add_794/U39/A0	0		
/p1/pic1/add_794/U39/A1	0		

What can you determine from analyzing these last three faults?

---

---

---

- e. Close the Report Faults Output dialogue box.

After FastScan identifies faults, faults that behave the same are reduced, or collapsed, into one equivalent fault. Multiple faults are targeted with one pattern, which reduces the number of generated patterns. EQ is the designator for equivalent faults. An equivalent fault is equal to the fault listed above it in the fault list.

14. Next we use the Hierarchy Browser to remove fault settings from specific instances, generate patterns and observe test coverage results. To do this, reset FastScan.

- a. Reset Fastscan ATPG by going back to Setup mode.
- b. Immediately go back to ATPG mode.

Using the Hierarchy Browser to delete faults enables FastScan to bypass specified pin pathnames and pin names from becoming fault sites.



You can issue the **Add Nofault** command before using the **Add Faults** command to achieve the same effect.

**Note**

Once you add nofault settings, the tool loses all information added after flattening such as ATPG functions and constraints.

15. Add all faults to the design.

- a. Click on Fault Universe then choose Customize. The Setup Fault Universe dialogue box opens.
  - i. **Fault Type>List** tab: Create a Fault List:

- a. Select — Add Faults to ALL DESIGN OBJECTS.
- ii. Leave the default settings for the rest.
- b. Click OK. In the Session Transcript window observe the command that has been executed.

What is the command?

---

- c. Click on Fault Universe then choose Customize again. The Setup Fault Universe dialogue box opens.
- i. **Delete Faults** tab:

Use the Hierarchy Browser to delete faults from the following instances:

/p6/pic1/regs/U41/A  
/p5/pic1/regs/U41/A  
/p4/pic1/regs/U41/A  
/p2/pic1/regs/U41/A  
/p1/pic1/regs/U41/A

After entering the first one, try other methods other than the Hierarchy Browser to enter the names. For example, selecting an entered fault puts it in the Design Object Pathname box. It then can be edited and the new path added to the list.

- ii. Leave the default settings for the rest.
- d. Click OK to exit the dialogue box.
- e. Look in the Session Transcript window for the command executed by this process.

The command will be:

```
Delete faults /p6/...../U41/A -stuck_at 01
```

- VL SIGO Join Telegram @visiodofficial
16. Generate random and deterministic stuck-at faults.
  17. Note the number of simulated and effective patterns for the various types.

**Table 4-18. ATPG Run Statistics****Table 4-19.**

Type	Simulated Patterns	Effective Patterns
Random		
Deterministic		
Totals		

18. Obtain a detailed report of the design's statistics for the ATPG run.

Fill in the following statistics:

**Table 4-20. Report Statistics ATPG****Table 4-21.**

Fault Class	# faults (coll.)	# faults (total)
FU (full)		
UO (unobserved)		
DS (det_sim)		
DI (det_imp)		
PU (posdet_untestable)		
UU (unused)		
TI (tied)		
BL (blocked)		
RE (redundant)		

**Table 4-21.**

Fault Class	# faults (coll.)	# faults (total)
AU (atpg_untestable)		

**Table 4-22. Report Coverage/Effectiveness ATPG****Table 4-23.**

Cov./Effect.	# faults (coll.)	# faults (total)
test_coverage		
fault_coverage		
atpg_effectiveness		
# test_patterns		
# simulated_patterns		
CPU_time (secs)		

a. What types of test patterns make up the complete set, and how many patterns of each type are there?

- \_\_\_\_\_ No. of patterns \_\_\_\_\_
- \_\_\_\_\_ No. of patterns \_\_\_\_\_
- \_\_\_\_\_ No. of patterns \_\_\_\_\_

By using the **NOfault** command to place nofault settings on specified instances, Fastscan generated patterns that produced better fault coverage, although only slightly better in this lab.

In the next exercise, you use common methodologies to attain a quick estimate of pattern coverage.

19. Exit FastScan.

## Exercise 10: Common Methodologies to Attain a Quick Estimate of Test Coverage

In this exercise, you invoke FastScan on a design and apply pattern types to determine initial pattern coverage. Then you use the following common methodologies to attain a quick estimate of pattern coverage:

- Assessing through implication
- Using fault sampling
- Limiting ATPG test coverage
- Loading in an external fault list

### Getting Started

1. Change to the \$ATPGNW/lab4/exercise\_10 directory.

```
shell> cd $ATPGNW/lab4/exercise_10
```

2. Invoke FastScan on the following circuit:

Design: module4.v

Library: atpglib

Log file: results/ex\_10.log

3. Add clocks automatically.

Various messages and warnings appear in the transcript. What warning messages do you see?

4. The next step is to add the scan group, test procedure file, and scan circuitry, as you did in the previous exercise.

If you saved the scan setup file as a dofile in exercise 8 you will find it useful here. You will have to edit it to alter the name of the test procedure (.testproc) file to make it applicable for this lab.

- a. Set up the Scan Group with the following information:
- b. **Scan Groups Tab:** Add scan group, test procedure file, and scan circuitry.

Group Name: grp1

Test Procedure file: exercise\_10/gate\_afterdft\_100.testproc

- i. **Scan Chains tab:**

- a. Add the following scan definitions:

**Table 4-24. ATPG Run Statistics**

Group	Chain	Scan_In	Scan_out
grp1	chain1	scan_in5	scan_out1
grp1	chain2	scan_in6	scan_out2
grp1	chain3	scan_in7	scan_out3
grp1	chain4	scan_in8	scan_out4

5. As in exercise 9, normal full scan vectors are not sufficient for this circuit so we are going to generate more complex vectors. These will be explained fully in the next module. For now, it is not necessary to understand the concept in order to use them.

Set simulation mode to RAM sequential and clock sequential.

```
SETUP > set simulation mode ram -depth 3
```

6. Go to ATPG mode.

You should now be in ATPG system mode.

7. Generate random and deterministic patterns for stuck-at-faults.
  8. Observe and record the CPU time from the Fastscan ATPG Run Statistics dialogue box:
- 



**Note**

The FastScan ATPG Run Statistics dialogue box show the run time for this particular run.

9. Close the FastScan ATPG Run Statistics dialogue box.

Obtain a detailed report of the design's statistics for the ATPG run and fill in the following statistics:

**Table 4-25. Report Statistics ATPG**

**Table 4-26.**

Fault Class	# faults (coll.)	# faults (total)
FU (full)		
UO (unobserved)		
DS (det_sim)		
DI (det_imp)		
PU (posdet_untestable)		
UU (unused)		
TI (tied)		
BL (blocked)		
RE (redundant)		

**Table 4-26.**

Fault Class	# faults (coll.)	# faults (total)
AU (atpg_untestable)		

**Table 4-27. Report Coverage/Effectiveness ATPG****Table 4-28.**

Cov./Effect.	# faults (coll.)	# faults (total)
test_coverage		
fault_coverage		
atpg_effectiveness		
# test_patterns		
# simulated_patterns		
CPU_time (secs)		

In simulation statistics, CPU time represents cumulative time not the run time.

**Note**

What are the number of patterns?

- Basic: No. of patterns \_\_\_\_\_
- RAM sequential: No. of patterns \_\_\_\_\_
- Clock Sequential: No. of patterns \_\_\_\_\_

Fault sampling is another method to attain a quick estimate of pattern coverage for large circuits. Fault sampling creates a smaller pattern set for simulation and detects problems earlier. Use the Set Fault Sampling command to specify a percentage (between 0 and 100) of the total number of faults you want to process.

This is the next step of the exercise.

10. Specify the fault sampling to 10%.

What command do you use? \_\_\_\_\_

11. Reset circuit status, and delete internal patterns.

ATPG > **reset state**

12. Generate random and deterministic patterns.

13. Study the results and compare them with the results from the last run.

- a. Observe and record the CPU time from the Fastscan ATPG Run Statistics dialogue box:

---

How does this compare with the first run?

---

- b. Close the FastScan ATPG Run Statistics dialogue box.
- c. Obtain a detailed report of the design's simulation statistics and fill in the following statistics:

**Table 4-29. Report Statistics ATPG**

**Table 4-30.**

Fault Class	# faults (coll.)	# faults (total)
FU (full)		
UO (unobserved)		
DS (det_sim)		
DI (det_imp)		
PU (posdet_untestable)		
UU (unused)		

**Table 4-30.**

Fault Class	# faults (coll.)	# faults (total)
TI (tied)		
BL (blocked)		
RE (redundant)		
AU (atpg_untestable)		

**Table 4-31. Report Coverage/Effectiveness ATPG****Table 4-32.**

Cov./Effect.	# faults (coll.)	# faults (total)
test_coverage		
fault_coverage		
atpg_effectiveness		
# test_patterns		
# simulated_patterns		
CPU_time (secs)		

What are the number of patterns?

- Basic: No. of patterns \_\_\_\_\_
- RAM sequential: No. of patterns \_\_\_\_\_
- Clock Sequential: No. of patterns \_\_\_\_\_

i. How does the number of faults compare to the first run?

---

ii. How do the test & fault coverage figures compare to the first run?

- 
- iii. How do the number of test patterns compare to the first run?
- 

FastScan reported the preceding analysis using fault sampling.

Limiting ATPG test coverage is used to attain a quick assessment of pattern coverage. Using the Set ATPG Limits command specifies the ATPG process limits at which the FastScan terminates ATPG. The **-Test** switch and argument pair specifies the maximum percentage of test coverage.

You will now set ATPG limits to 50% coverage, and then determine test coverage.

14. Reset Fastscan.

15. Set fault sampling percentage back to 100%.

```
SETUP> set fault sampling 100
```

16. Set ATPG test coverage percentage to 50%.

```
SETUP > set atpg limit -test 50
```

17. Go to ATPG mode.

18. Generate random and deterministic patterns for stuck-at-faults. (*Typical Settings*)

19. Study the results and compare them with the first run.

- a. Observe and record the CPU time from the Fastscan ATPG Run Statistics dialogue box:
- 

How does this compare with the first run? \_\_\_\_\_

- b. Close the FastScan ATPG Run Statistics dialogue box.
- c. Obtain a detailed report of the design's simulation statistics and fill in the following statistics:

**Table 4-33. Report Statistics ATPG**

**Table 4-34.**

Fault Class	# faults (coll.)	# faults (total)
FU (full)		
UC (uncontrolled)		
UO (unobserved)		
DS (det_simulation)		
DI (det_imp)		
UU (unused)		
TI (tied)		
BL (blocked)		

**Table 4-35. Report Coverage/Effectiveness ATPG**

**Table 4-36.**

Cov./Effect.	# faults (coll.)	# faults (total)
test_coverage		
fault_coverage		
atpg_effectiveness		
# test_patterns		
# simulated_patterns		
CPU_time (secs)		

- i. How does the number of faults compare to the first run?

ii. How do the test & fault coverage figures compare to the first run?

iii. How do the number of test patterns compare to the first run?

FastScan reported the preceding analysis from setting ATPG test coverage limits.

In the final part of this exercise, you write a fault list from the previous run, and load that external fault list into the current internal fault list. (Remember the previous run was limited in coverage and finished short. This acts as a representation of a fault list generated from a set of existing patterns, enabling ‘top up’ to occur.)

You use the **Reset AU Faults** command to analyze and reclassify previously untestable faults from the external fault list. Then you generate patterns and observe the results.

The **Load Faults** command affects the current fault population by either adding or removing faults from an external fault file. The **-Retain** switch specifies for FastScan to retain the fault class of each fault that is in the fault list. This switch ensures that no DS faults are reclassified as AU faults.

20. First write an external fault list.

```
ATPG > write faults -all results/ external_flt.list \
-replace
```

(Remember that the **-replace** switch allows previous files of the same name to be overwritten.)

21. Reset circuit status, and delete internal patterns.

ATPG> **reset state**

22. Load external faults and analyze the coverage given by these faults.

a. Load the external fault list.

ATPG > **load faults results/external\_flt.list -retain**

b. Use the **report statistics** command to fill in the following:

**Table 4-37.**

**Table 4-38.**

Fault Class	# faults (coll.)	# faults (total)
FU (full)		
UO (unobserved)		
DS (det_sim)		
DI (det_imp)		
PU (posdet_untestable)		
UU (unused)		
TI (tied)		
BL (blocked)		
RE (redundant)		
AU (atpg_untestable)		

**Table 4-39. Report Coverage/Effectiveness ATPG**

**Table 4-40.**

Cov./Effect.	# faults (coll.)	# faults (total)
test_coverage		
fault_coverage		

**Table 4-40.**

Cov./Effect.	# faults (coll.)	# faults (total)
atpg_effectiveness		
# test_patterns		
# simulated_patterns		
CPU_time (secs)		

- c. What types of test patterns make up the complete set, and how many patterns of each type are there?
- \_\_\_\_\_ No. of patterns \_\_\_\_\_
  - \_\_\_\_\_ No. of patterns \_\_\_\_\_
  - \_\_\_\_\_ No. of patterns \_\_\_\_\_
23. Reset any faults classified as atpg untestable to allow them to be investigated and set the ATPG test coverage back to 100%. Analyze and reclassify previously untestable faults.
- ```
ATPG> reset au faults
```
- ```
ATPG> set atpg limit -test off
```
- Note that the second command does more than just reset the faults to 100%.
24. Generate patterns, this time using the command line. You only need one word as all has been set up by previous commands.
- ```
ATPG> run
```
25. Study the results and compare them with the first run.

- a. Observe and record the CPU time from the Fastscan ATPG Run Statistics dialogue box:

How does this compare with the first run? \_\_\_\_\_

- b. Close the FastScan ATPG Run Statistics dialogue box.
- c. Obtain a detailed report of the design's simulation statistics and fill in the following statistics:

**Table 4-41.**

**Table 4-42.**

| Fault Class            | # faults (coll.) | # faults (total) |
|------------------------|------------------|------------------|
| FU (full)              |                  |                  |
| UO (unobserved)        |                  |                  |
| DS (det_sim)           |                  |                  |
| DI (det_imp)           |                  |                  |
| PU (posdet_untestable) |                  |                  |
| UU (unused)            |                  |                  |
| TI (tied)              |                  |                  |
| BL (blocked)           |                  |                  |
| RE (redundant)         |                  |                  |
| AU (atpg_untestable)   |                  |                  |

**Table 4-43. Report Coverage/Effectiveness ATPG**

**Table 4-44.**

| Cov./Effect.       | # faults (coll.) | # faults (total) |
|--------------------|------------------|------------------|
| test_coverage      |                  |                  |
| fault_coverage     |                  |                  |
| atpg_effectiveness |                  |                  |
| # test_patterns    |                  |                  |

**Table 4-44.**

| Cov./Effect.         | # faults (coll.) | # faults (total) |
|----------------------|------------------|------------------|
| # simulated_patterns |                  |                  |
| CPU_time (secs)      |                  |                  |

- i. How does the number of FU faults compare to the first run?

---
- ii. How do the test & fault coverage figures compare to the first run?

---

These results have been calculated from loading faults from an external fault list into the current internal fault list.

26. Exit FastScan.

## Test Your Knowledge

1. What command do you use to ‘black box’ a module?

---

2. What are black boxes used for?

---

3. What command is used to set a contention check on a bus?

---

4. What pattern generation does FastScan begin with? Proceed to?

---

5. What command do you use when you want to report run statistics?

---

6. What command do you use to set simulation mode to RAM and clock sequential?

---

7. What command do you use to report on the testability of various fault classes?

---

8. What command do you use to analyze a fault in a list?

---

## Lab Summary

Now that you completed the Understanding ATPG Messaging lab, you should know how to do the following:

- Read and analyze:
  - Messages and warnings.
  - DRC reporting.
  - ATPG and test coverage reporting.
  - Fault classifications.
- Determine the cause of undetected faults.
- Add no fault settings.
- Write an external fault list.
- Use common methodologies to attain a quick estimate of pattern coverage.

## Module 5

# Achieving High Test Coverage

### Objectives

Upon completion of this module, you will be able to:

- Initiate an ATPG Run.
- Apply FastScan pattern types to relevant circuits.
- Apply pattern sequencing to achieve high test coverage.
- Use ModelSim to simulate and verify the following testbenches:
- Verilog.
- Serial and parallel.
- Chaintest.
- Pattern sample.
- Save patterns in three formats: ASCII, Verilog, and WGL.
- Read an ASCII pattern back into FastScan.

## Module Topics

### Module Topics

- ◆ This module addresses the following topics:
  - Initial ATPG Run
  - Pattern types to test through complex circuitry
    - Non-scan logic
    - RAM/ROM
    - Small embedded memories
  - Pattern outputs and inputs in FastScan flow
  - Pattern verification

### Notes:

## Methodologies: Initial Run (Fault Sampling)

### Methodologies: Initial Run (Fault Sampling)

- ◆ Fault sampling provides the following:
  - Quick estimate of coverage for a large circuit.
  - Creates a smaller pattern set for simulation.
    - Detects problems early.
- ◆ Use The SET FAult Sampling command to specify a percentage (between 0 and 100) of the total faults you want processed.

```
ATPG> SET FAult Sampling 1
ATPG> CREAtE PAterns
ATPG> REPort STatistics
ATPG> SAve PAterns <FILENAME1.v> -Verilog
ATPG> SAve PAterns <FILENAME2.v> -Verilog -Serial -Sample 2
```

### Notes:

# Methodologies: External Fault List

## Methodologies: External Fault List

- ♦ Use the **LOAD FAults** command to place faults from a previous run (from an external file) into the internal fault list
  - FastScan creates additional patterns

Enables FastScan to analyze and reclassify previously untestable faults

Retains original faults in the fault list

```
ATPG> LOAD FAults <-FILENAME> -Retain  
ATPG> REPORT STratistics //verify initial coverage  
ATPG> RESet AU Faults  
ATPG> CREAtE PAtterns  
ATPG> REPORT STratistics //shows total coverage
```

## Notes:

## Adding NOfaults

### Adding NOfaults

- ◆ Use the **ADD NOfault command** to place nofault settings on the following:
  - Pin pathnames
  - Pin names of
    - specified instances
    - modules
- ◆ Issue the **ADD NOfaults command before using the ADD FAults command**
  - Specified pin pathnames and pins names will not become fault sites
  - If design was previously flattened, using the add nofault command will delete the flattened model

The tool loses all information added after flattening,  
such as ATPG functions and constraints

### Notes:

# FastScan's Test Pattern Types

## FastScan's Test Pattern Types

- ◆ FastScan generates the following test pattern types:
  - Basic Scan
    - Used on full-scan design circuitry
  - Clock Sequential
    - Used to propagate values through non-scan latches and DFFs with limited sequential depth
  - Clock PO
    - Used on circuitry where a clock signal passes through combinational logic to a primary output
  - RAM Sequential
    - Used to propagate values through RAM
  - Multi Load
    - Used on RAM/ROM designs that contain non-scan cells
  - MacroTest
    - Used to test the cell array of small embedded memories

## Notes:

## Basic Scan Patterns

### Basic Scan Patterns

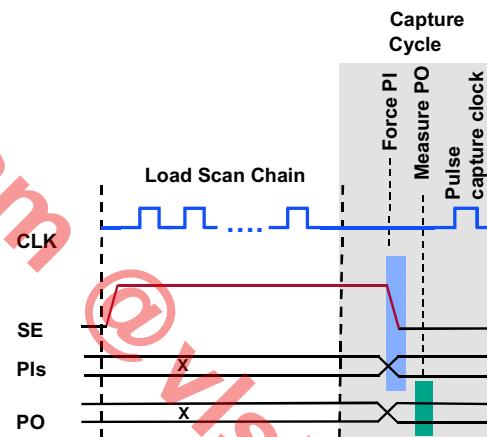
- ◆ The following apply to basic scan patterns:
  - Generated by default.
  - Use appropriate test procedures to define control and observation of scan cells.
  - Independent from each other.

### Notes:

## Basic Scan Patterns (Cont.)

### Basic Scan Patterns (Cont.)

- ◆ Basic scan patterns contain the following events:
  1. Load scan chain
  2. Force primary inputs
  3. Measure primary outputs
  4. Pulse capture clock
  5. Unload values from scan cells
    - Load next pattern



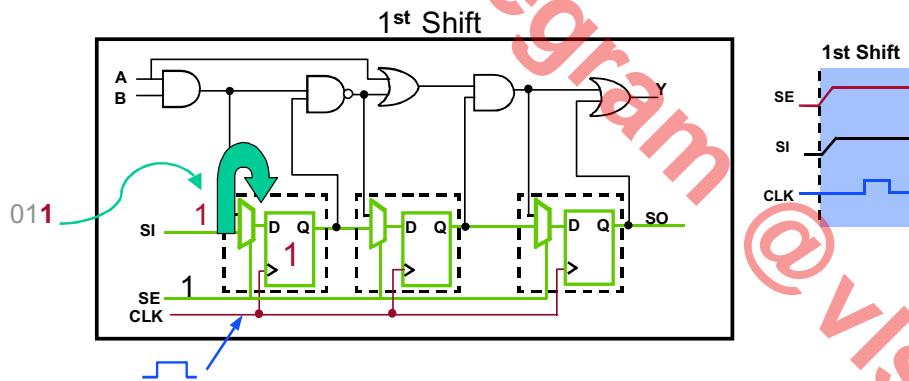
### Notes:

# Basic Scan Pattern Operation

|                    |
|--------------------|
| LOAD               |
| FORCE PI           |
| MEASURE PO         |
| PULSE CAPTURE CLK. |

## Basic Scan Pattern Operation

- ◆ Load Initialization procedure first
- ◆ Load values into the scan cells
  1. Force SE to “1”
  2. Force SI (scan chain input pin)
  3. Pulse shift clock
  4. Repeat steps 2 and 3 until all scan cells are loaded



5-9 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

Copyright © 2003 Mentor Graphics Corporation

## Notes:

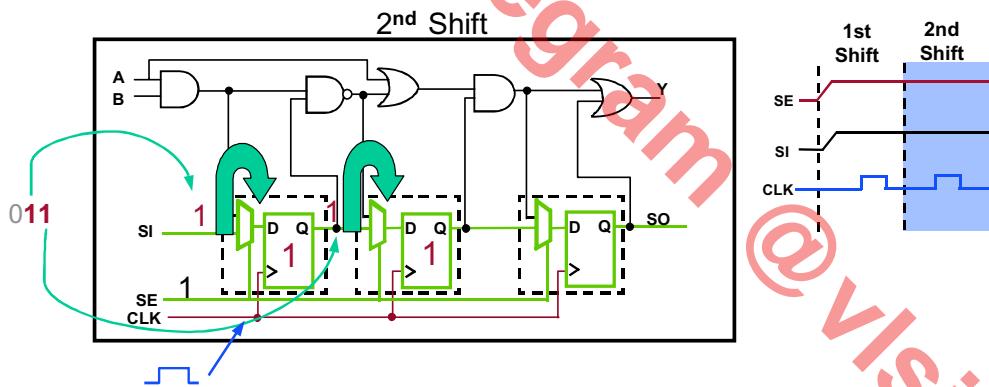
## Basic Scan Pattern Operation (Cont.)

### Basic Scan Pattern Operation (Cont.)

#### Load values into the scan cells

|                    |
|--------------------|
| LOAD               |
| FORCE PI           |
| MEASURE PO         |
| PULSE CAPTURE CLK. |

- 1. Force SE to “1”.
- 2. Force SI.
- 3. Pulse shift clock.
- 4. Repeat steps 2 and 3 until all scan cells are loaded.



5-10 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

Copyright © 2003 Mentor Graphics Corporation

### Notes:

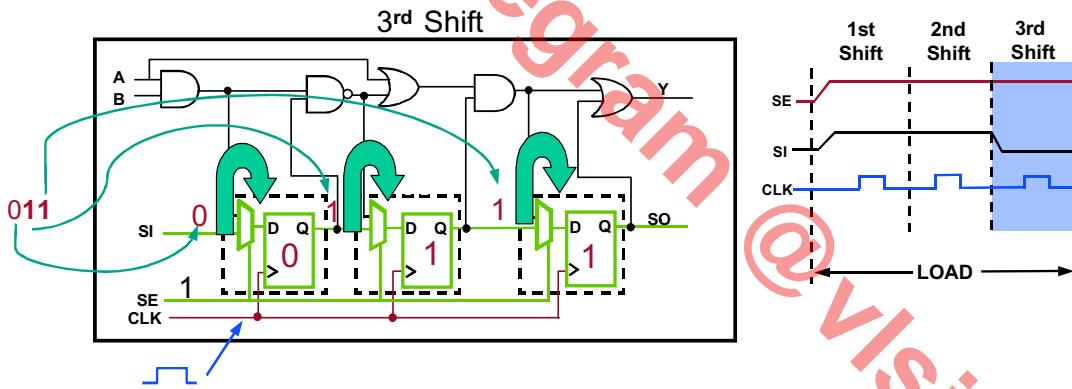
## Basic Scan Pattern Operation (Cont.)

### Basic Scan Pattern Operation (Cont.)

#### Load values into the scan cells

|                    |
|--------------------|
| LOAD               |
| FORCE PI           |
| MEASURE PO         |
| PULSE CAPTURE CLK. |

1. Force SE to “1”.
2. Force SI.
3. Pulse shift clock.
4. Repeat steps 2 and 3 until all scan cells are loaded.



5-11 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

Copyright © 2003 Mentor Graphics Corporation

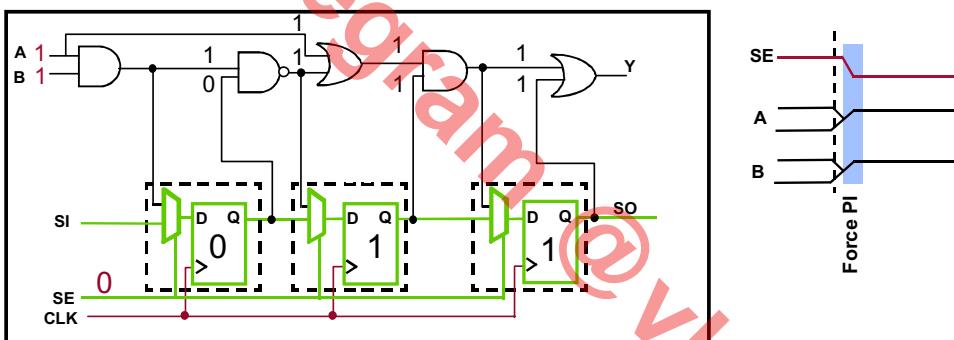
### Notes:

## Basic Scan Pattern Operation (Cont.)

### Basic Scan Pattern Operation (Cont.)

|                    |
|--------------------|
| LOAD               |
| FORCE PI           |
| MEASURE PO         |
| PULSE CAPTURE CLK. |

- ◆ Force primary inputs
  - ◆ Force normal primary inputs.
  - ◆ Force SE to “0” (exits shift mode).
    - Now all internal values can be predicted.



### Notes:

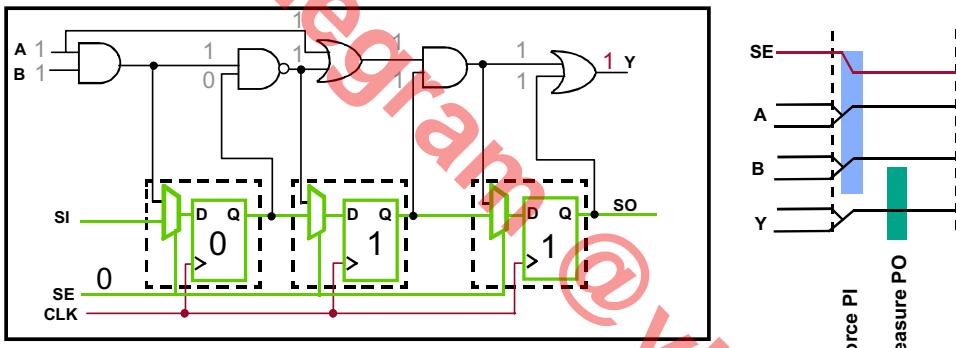
## Basic Scan Pattern Operation (Cont.)

### Basic Scan Pattern Operation (Cont.)

|                    |
|--------------------|
| LOAD               |
| FORCE PI           |
| MEASURE PO         |
| PULSE CAPTURE CLK. |

Measure primary outputs

Measure "1" on Y.



### Notes:

## Basic Scan Pattern Operation (Cont.)

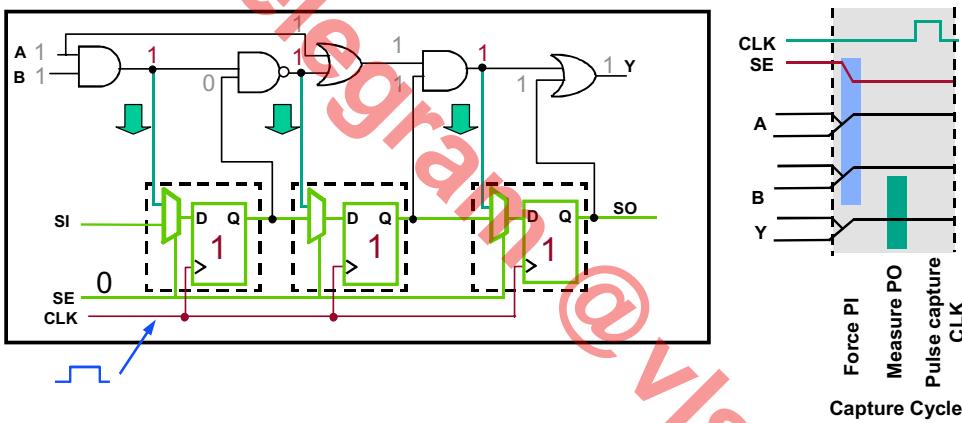
### Basic Scan Pattern Operation (Cont.)

|                    |
|--------------------|
| LOAD               |
| FORCE PI           |
| MEASURE PO         |
| PULSE CAPTURE CLK. |

Capture Cycle

#### Pulse Capture Clock

- Loads scan cells with functional inputs to observe circuit status.



### Notes:

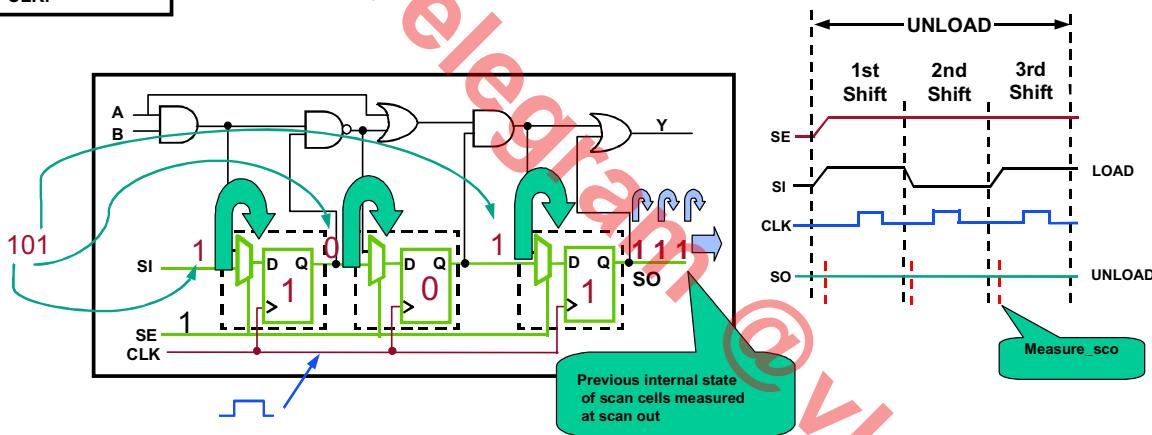
## Basic Scan Pattern Operation (Cont.)

### Basic Scan Pattern Operation (Cont.)

| LOAD/UNLOAD        |
|--------------------|
| FORCE PI           |
| MEASURE PO         |
| PULSE CAPTURE CLK. |

#### Unload the scan chain

- As new data is being shifted into the scan chain during load, the previous internal circuit state is being shifted out and measured at scan out (SO).



### Notes:

# Clock Primary Output Patterns

## Clock Primary Output Patterns

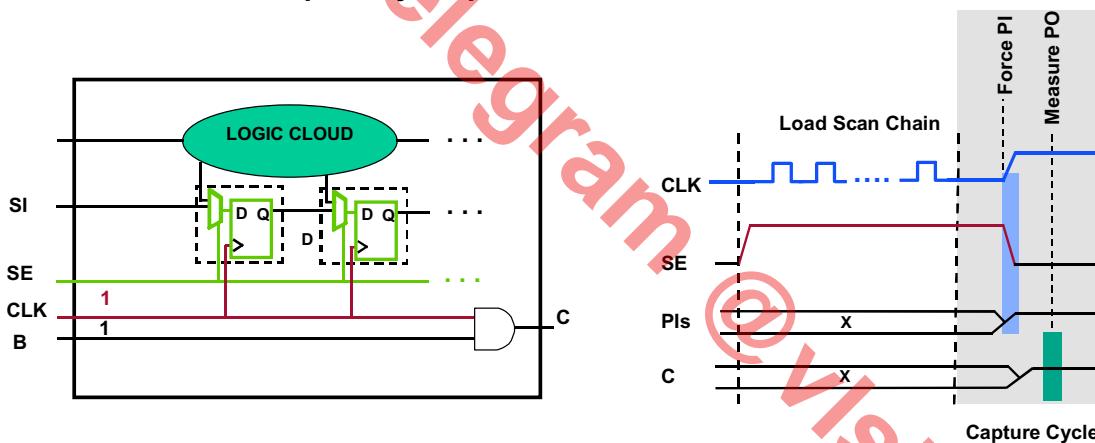
- ◆ The following apply to Clock Primary Output (PO) patterns:
  - FastScan generates clock PO patterns whenever it learns that a clock connects to a primary output.
  - Generated if there is a C8 violation.
  - Allows clocks to be active during force and measure events.
  - Some testers cannot control clocks using clock PO patterns.
  - Use the following command to prevent clock PO generation.
    - `SETUP>SET PAttern Type -Clockpo OFF`

## Notes:

# Clock Primary Output Patterns (Cont.)

## Clock Primary Output Patterns (Cont.)

- ♦ Clock PO patterns contain the following events:
  1. Load values into the scan chain.
  2. Force values on all primary inputs, including clocks connected to primary outputs.
  3. Measure all primary outputs that are connected to scan clocks.



5-17 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Clock Sequential Patterns

## Clock Sequential Patterns

- ◆ The following apply to Clock Sequential patterns:
  - Test through scan-based designs that contain limited non-scan sequential logic or non-scan latches.
  - FastScan reports non-scan logic as tie-x by default.
  - To enable, set the sequential depth to a number greater than 1.
    - `SETUP>SET PAttern Type -SEquential 2`
  - (Clock sequential depth -1) defines the number of non-scan cells connected in series that FastScan can test through.
    - Do not set the depth greater than 5

## Notes:

## Clock Sequential Patterns (Cont.)

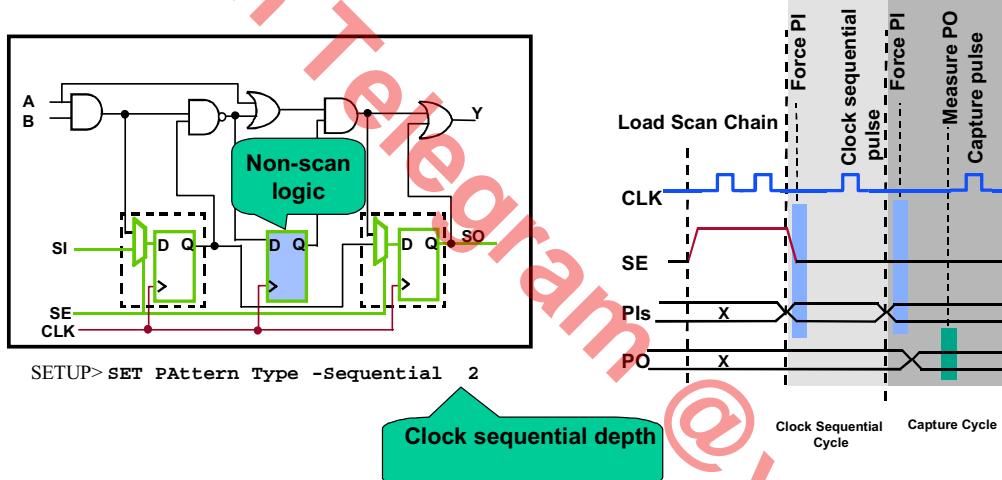
### Clock Sequential Patterns (Cont.)

- ◆ Clock sequential patterns contain the following events:
  1. Load scan chains
  2. Apply clock sequential cycle
    - a. Force PIs
    - b. Pulse clock
    - c. Repeat a and b up to "N" times, where N is the sequential depth -1
  3. Apply capture cycle
    - a. Force PIs
    - b. Measure PO
    - c. Pulse capture clock
  4. Unload values from scan cells

### Notes:

## Clock Sequential Patterns (Cont.)

### Clock Sequential Patterns (Cont.)

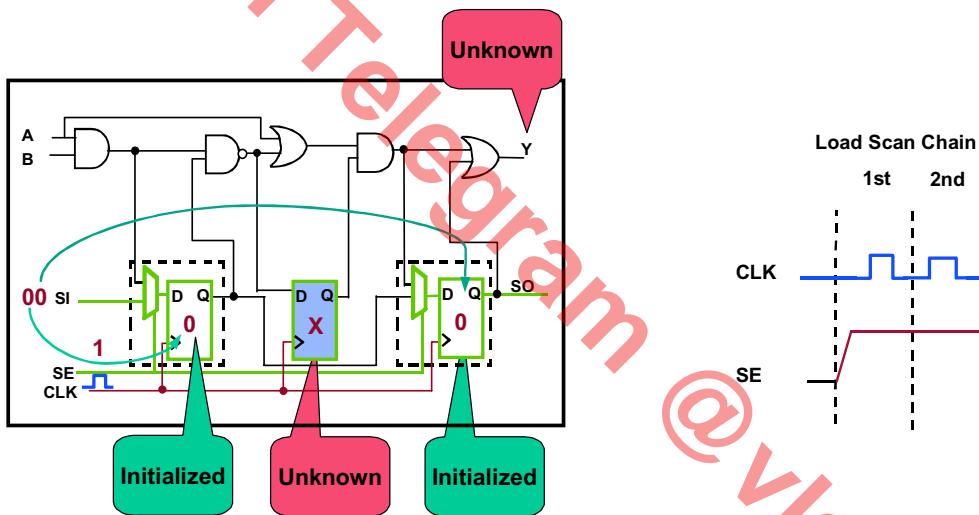


### Notes:

# Clock Sequential Pattern Operation

## Clock Sequential Pattern Operation

- Load values into the scan cells



5-21 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

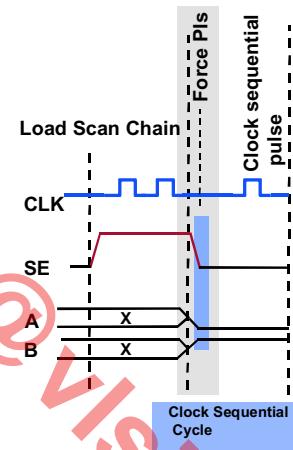
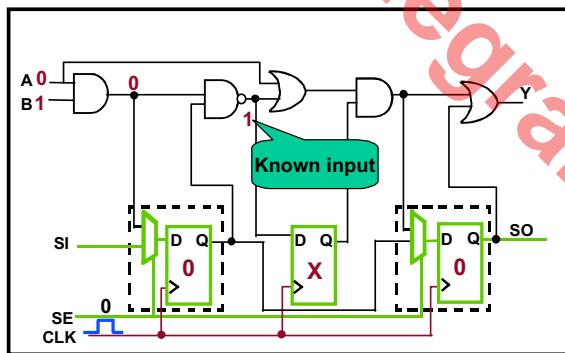
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Clock Sequential Pattern Operation (Cont.)

## Clock Sequential Pattern Operation (Cont.)

- ♦ Apply clock sequential cycle
  - a. Force primary inputs.



5-22 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

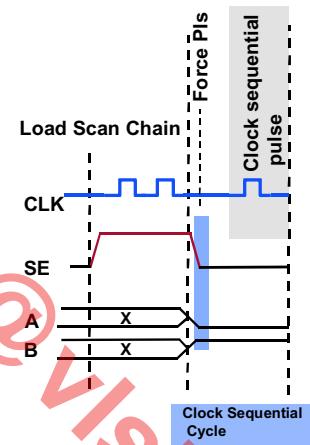
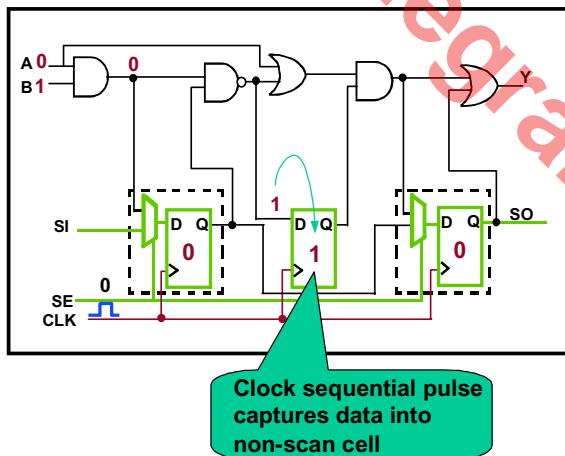
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Clock Sequential Pattern Operation (Cont.)

## Clock Sequential Pattern Operation (Cont.)

- ◆ Apply clock sequential cycle
  - b. Pulse clock.
  - c. Repeat a and b up to “N” times, where N is the sequential depth –1.



5-23 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

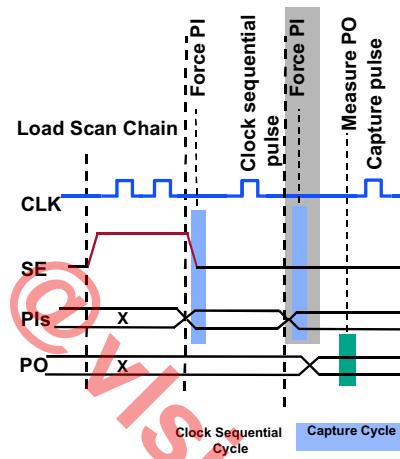
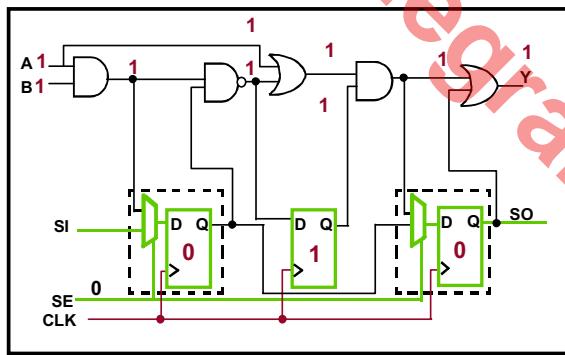
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Clock Sequential Pattern Operation (Cont.)

## Clock Sequential Pattern Operation (Cont.)

- ◆ Apply capture cycle
  - a. Force PI.



5-24 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

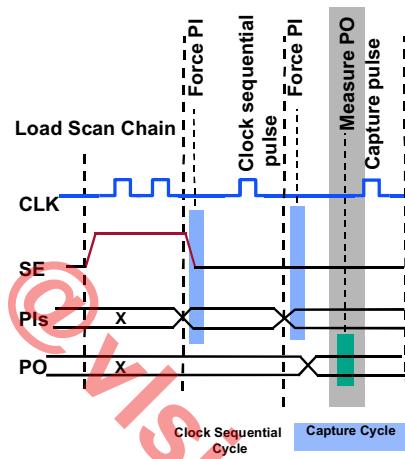
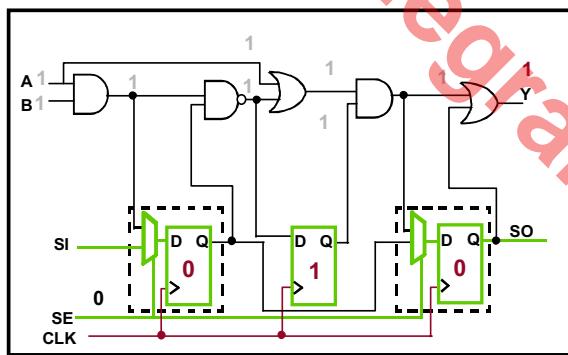
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Clock Sequential Pattern Operation (Cont.)

## Clock Sequential Pattern Operation (Cont.)

- ◆ Apply capture cycle
  - a. Force PI.
  - b. Measure PO.



5-25 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

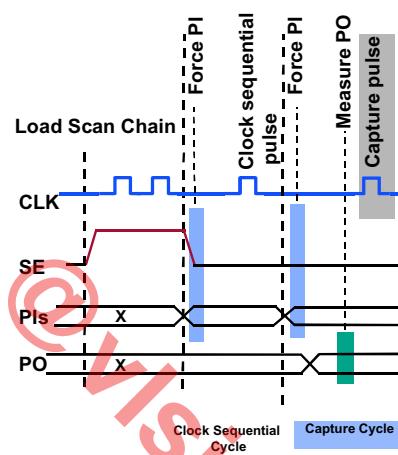
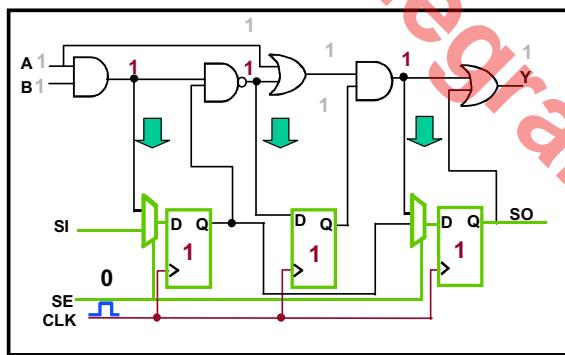
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Clock Sequential Pattern Operation (Cont.)

## Clock Sequential Pattern Operation (Cont.)

- ◆ Apply capture cycle
  - a. Force PI.
  - b. Measure PO.
  - c. Pulse capture clock.

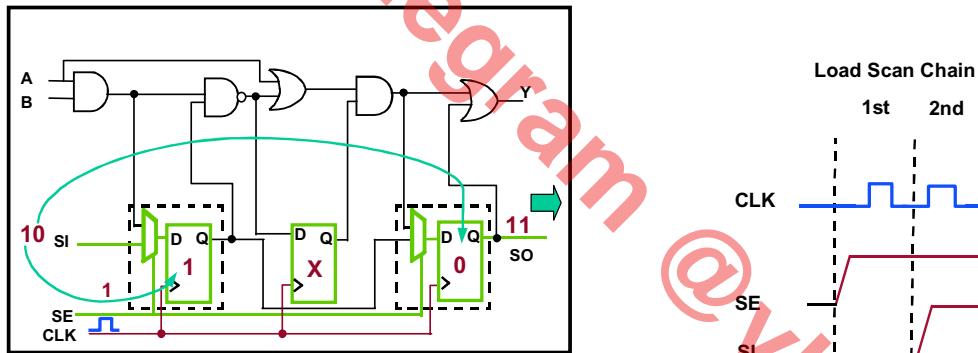


## Notes:

# Clock Sequential Pattern Operation (Cont.)

## Clock Sequential Pattern Operation (Cont.)

- ♦ Unload values from scan cells
  - As new data is being shifted into the scan chain during load, the previous internal circuit state is being shifted out and measured at scan out (SO).



## Notes:

## RAM Sequential Patterns

### RAM Sequential Patterns

- ◆ The following applies to RAM Sequential patterns:
  - Target faults associated with address and data lines not detected by default patterns (pass-through)
  - Automatically determine writes and reads to test logic around memories
  - Are single patterns with multiple loads
    - Load events include: two address writes and a read

### Notes:

## RAM Sequential Patterns (Cont.)

### RAM Sequential Patterns (Cont.)

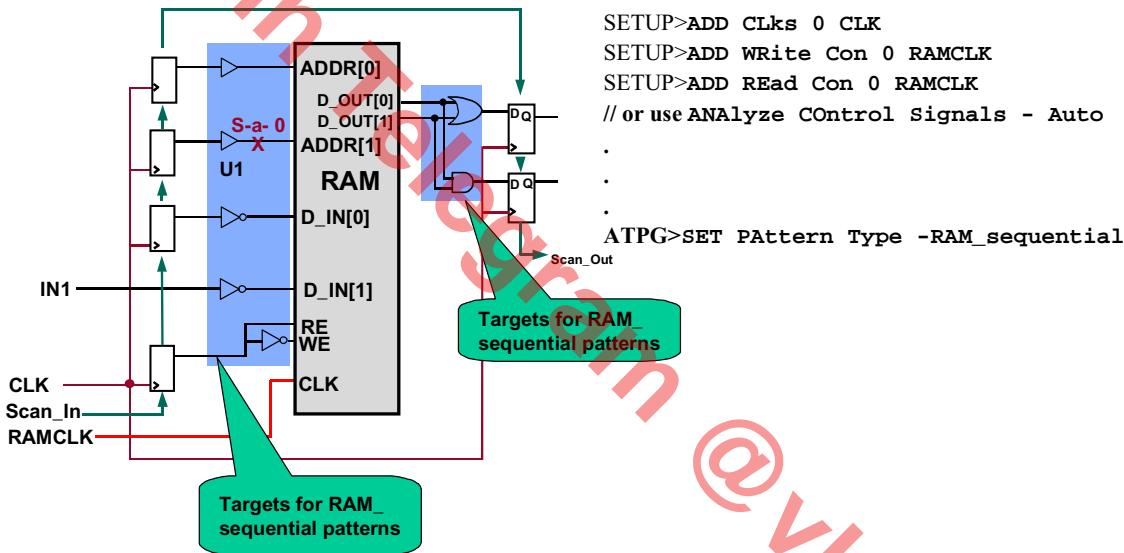
- ♦ Clock sequential patterns contain the following events:
  1. Write to 1<sup>st</sup> address
    - a. Load scan cells
    - b. Force primary inputs
    - c. Pulse write line(s)
  2. Write to 2<sup>nd</sup> address
    - Repeat steps a through c for a different address
  3. Read 1<sup>st</sup> address
    - Load scan cells
    - Force primary inputs
    - Pulse read lines
  4. Capture read values
    - Load scan cells
    - Force primary inputs
    - Measure primary outputs
    - Pulse capture clock
  5. Unload values from scan cells

### Notes:

## RAM Sequential Patterns (Cont.)

### RAM Sequential Patterns (Cont.)

- Do the following to generate RAM sequential patterns

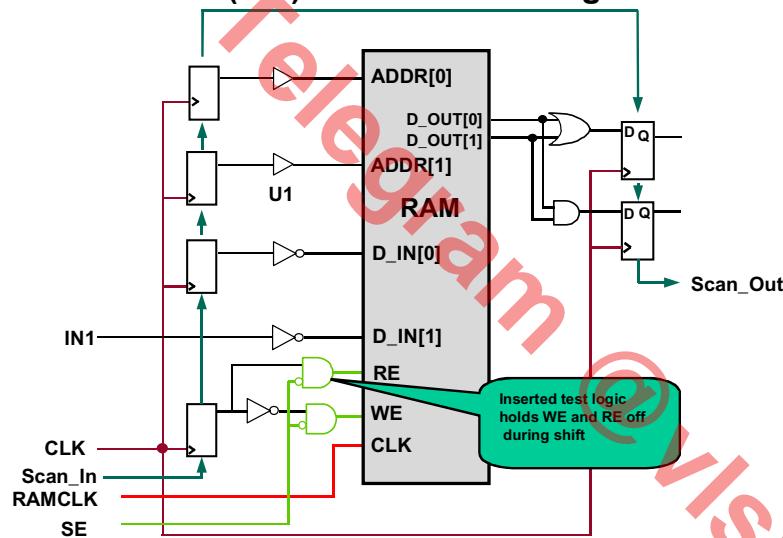


### Notes:

## RAM Sequential Patterns (Cont.)

### RAM Sequential Patterns (Cont.)

- RAM must be stable during LOAD/UNLOAD events.
- If the scan clock is used for RAM, read enable (RE) and write enable (WE) must be off during shift.



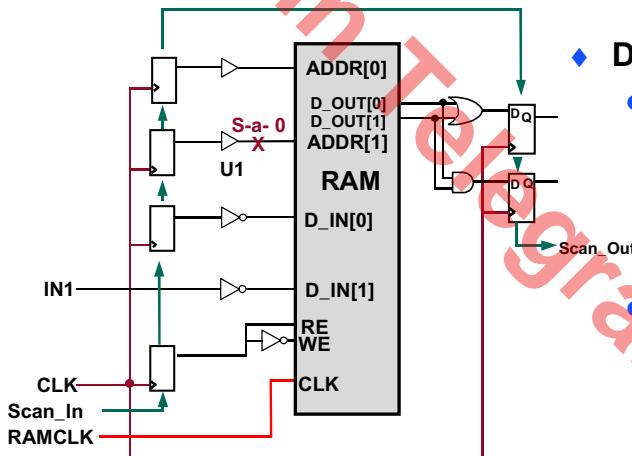
5-31 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

Copyright © 2003 Mentor Graphics Corporation

### Notes:

# RAM Sequential Patterns Example: To Test For Stuck-At-0 at the Output of U1

## RAM Sequential Patterns Example: To Test For Stuck-At-0 at the Output of U1



- ◆ Do the following to test for S-a-0
  - Write data to 1<sup>st</sup> address.
  - ADDR [0] = 1
  - ADDR [1] = 1
    - Data In = 00
    - Address = 11
  - Write different data to 2<sup>nd</sup> address.
  - ADDR [0] = 1
  - ADDR [1] = 0 (assume S-a-0)
    - Data In = 11
    - Address = 01
  - Read 1<sup>st</sup> address.
  - Capture read values.
    - Correct captured data will read 00
    - If U1 is S-a-0, captured data will read 11

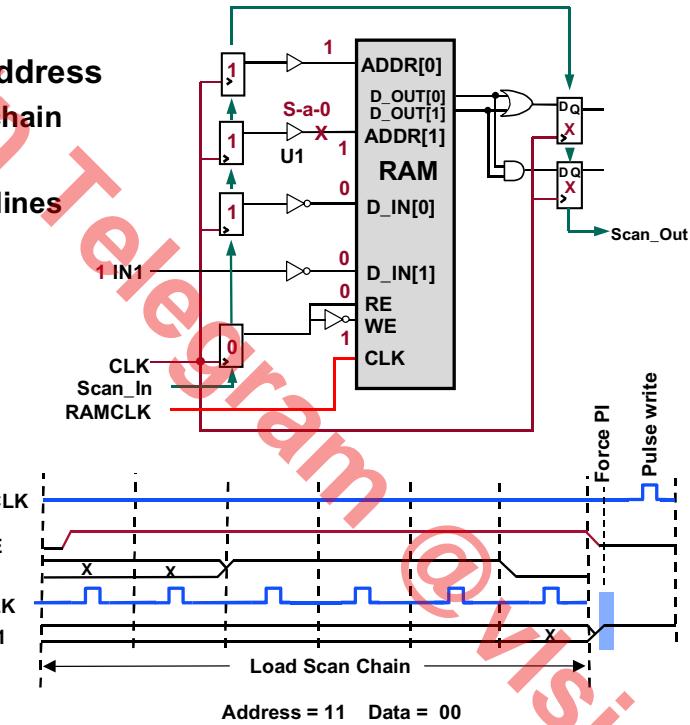
5-32 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

Copyright © 2003 Mentor Graphics Corporation

## Notes:

# RAM Sequential Pattern Operation

- ◆ Write to 1st Address
  - Load scan chain
  - Force PI
  - Pulse write lines



5-33 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

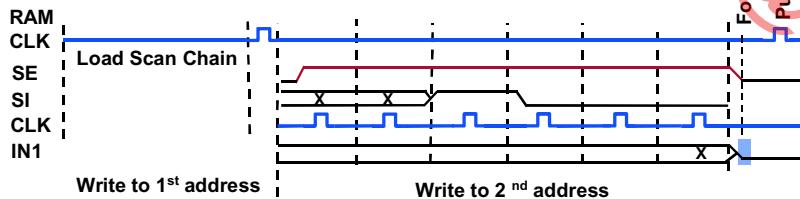
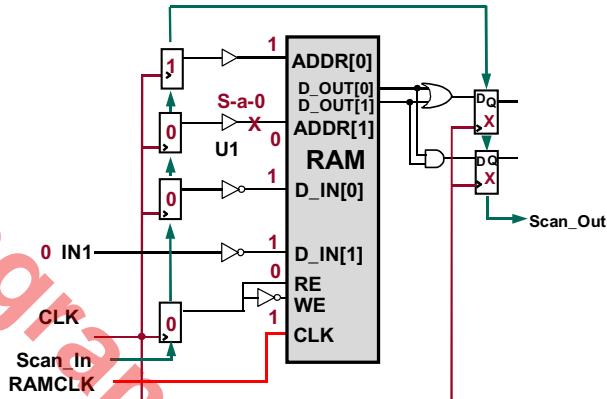
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# RAM Sequential Pattern Operation (Cont.)

## RAM Sequential Pattern Operation (Cont.)

- ♦ Write to 2<sup>nd</sup> address
  - Load scan chain
  - Force PI
  - Pulse write lines



5-34 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

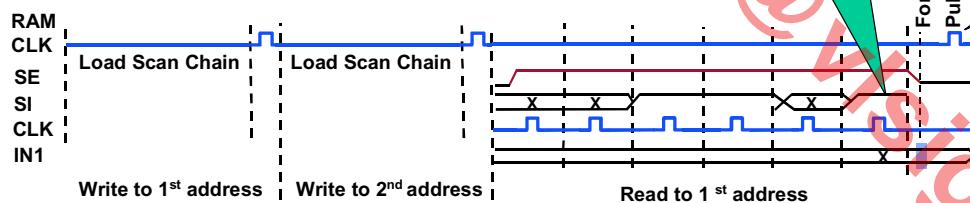
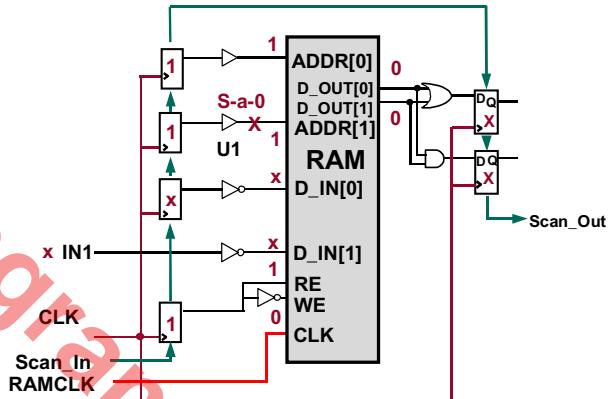
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# RAM Sequential Pattern Operation (Cont.)

## RAM Sequential Pattern Operation (Cont.)

- ♦ Read 1st address
  - Load scan chain
  - Force PI
  - Pulse read lines



5-35 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

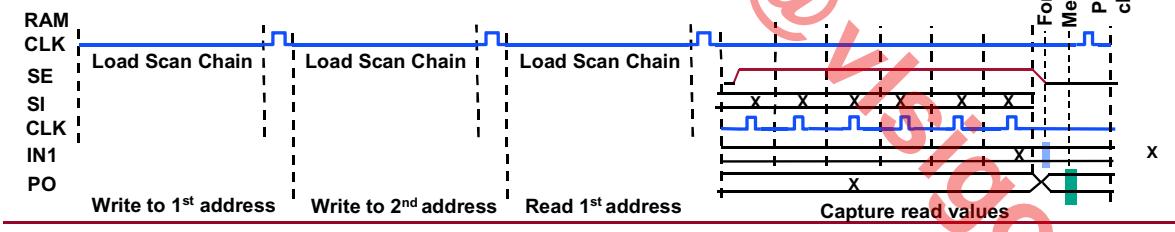
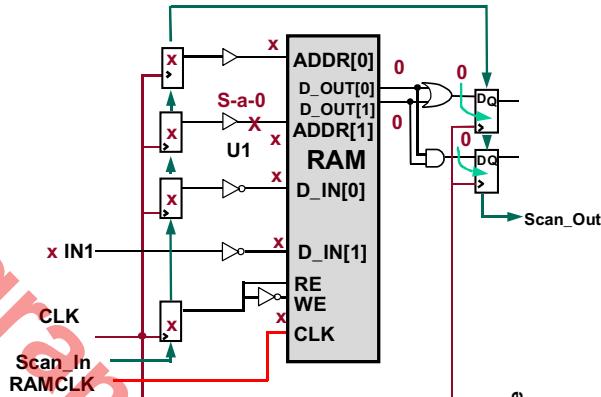
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# RAM Sequential Pattern Operation (Cont.)

## RAM Sequential Pattern Operation (Cont.)

- ◆ Capture read values
  - Load scan cells
  - Force primary inputs
  - Measure primary outputs
  - Pulse capture clock



5-36 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Multi Load Patterns

### Multi Load Patterns

- ◆ The following apply to Multi load patterns:
  - Tests through RAM/ROM and sequential logic that contain non-scan cells.
  - Clock sequential patterns are generated to load scan chains and non-scan cells without disturbing the logic state of the non-scan cells.
  - Optionally, writes and reads to test logic around memories.
  - Use the following commands to generate multi load patterns.

```
SETUP> SET PAttern Type -MULtiple_load ON -Sequential 2
```

### Notes:

## Multi Load Patterns (Cont.)

### Multi Load Patterns (Cont.)

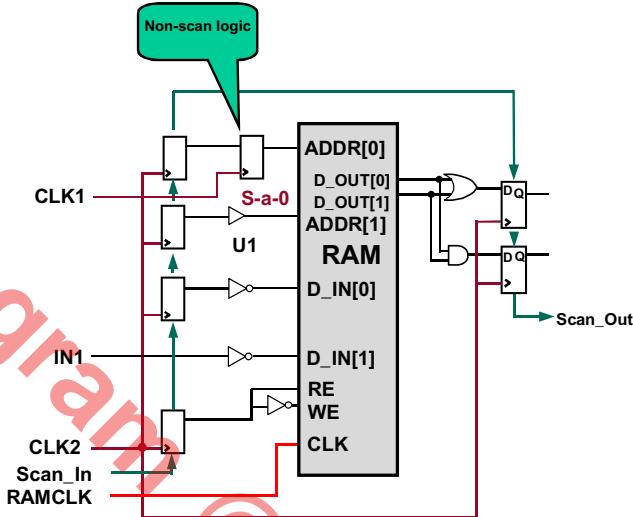
- ◆ Multi load patterns contain the following events:
  1. Load scan chains (optional after first time)
  2. Force PIs
  3. Pulse sequential clock/write
  4. Repeat steps 1-3 and/or 2-3 up to “1 . . . N” times
  5. Apply capture cycle
    - a. Force PI
    - b. Measure PO
    - c. Pulse clock
  6. Unload scan chains

### Notes:

## Multi Load Patterns Example

### Multi Load Patterns Example

- ◆ This design requires the following multi load events:
  1. Load scan chain
  2. Load non-scan cell
    - a. Pulse non-scan clock (CLK1)
  3. Load scan chain
  4. Perform a write
    - a. Pulse RAMCLK
  5. Load scan chain
  6. Load non-scan cell with next value and perform a write
    - a. Pulse CLK2 and RAMCLK
  7. Load scan chain
  8. Perform a read
    - a. Pulse RAMCLK
  9. Capture data into scan cells
    - a. Pulse CLK1
  10. Unload scan chains

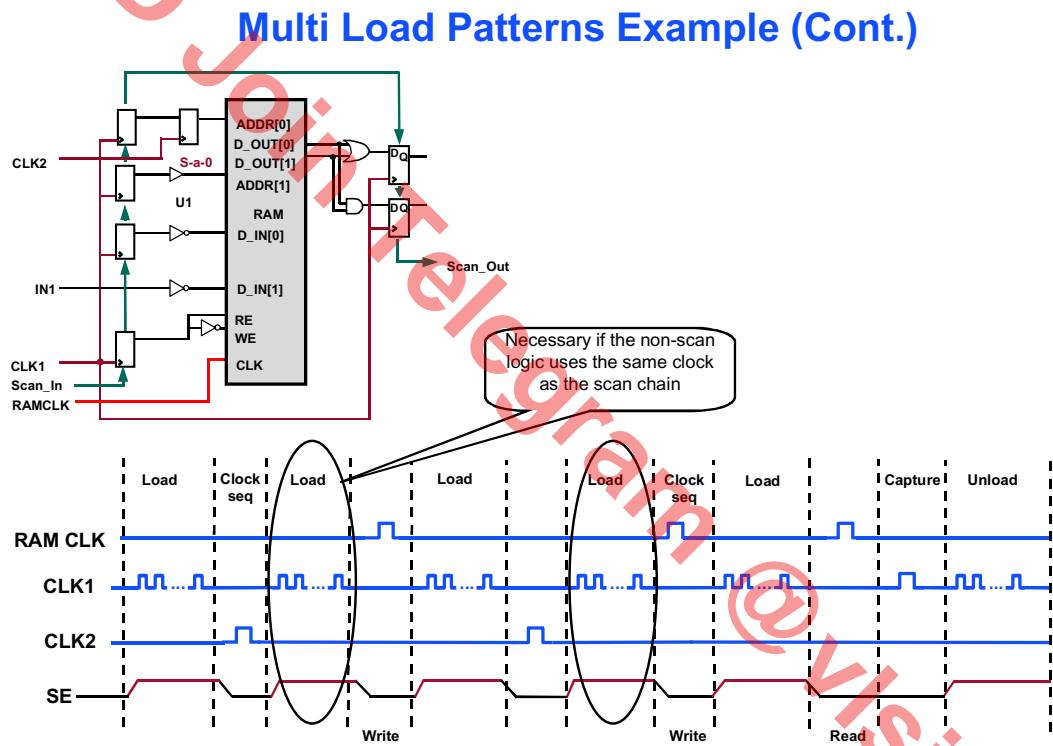


5-39 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

Copyright © 2003 Mentor Graphics Corporation

### Notes:

## Multi Load Patterns Example (Cont.)



5-40 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

Copyright © 2003 Mentor Graphics Corporation

### Notes:

## MacroTest Patterns

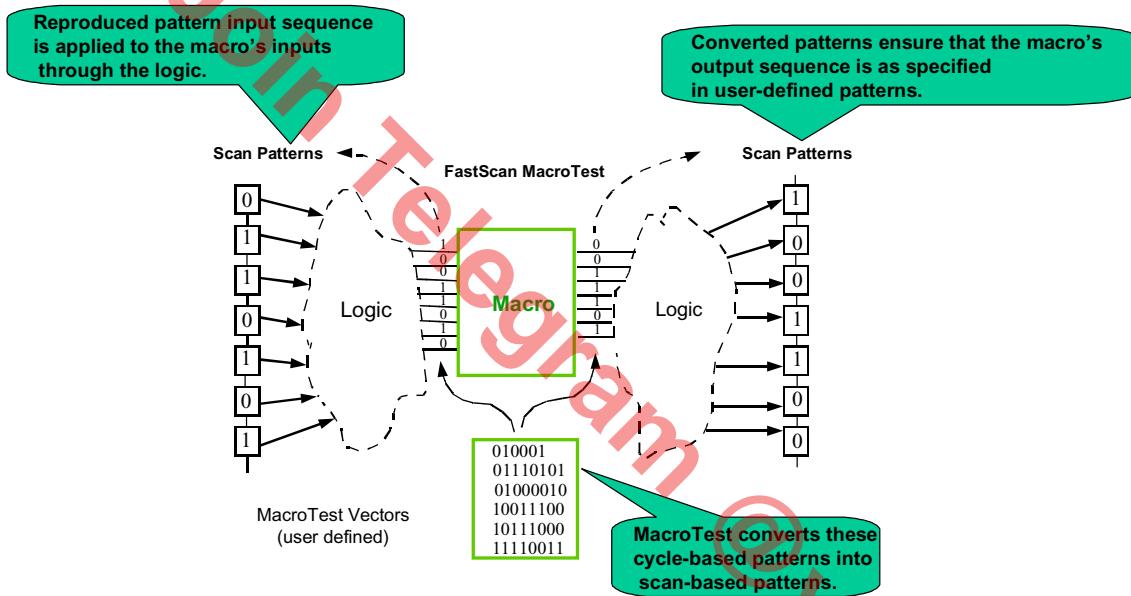
### MacroTest Patterns

- Is a utility that helps automate the testing of embedded logic and memories (macros) by automatically translating user-defined patterns for the macros into scan patterns
- User defines the pattern file and the instance to apply them to
- No extra logic required
- No performance impact

### Notes:

# MacroTest

## MacroTest



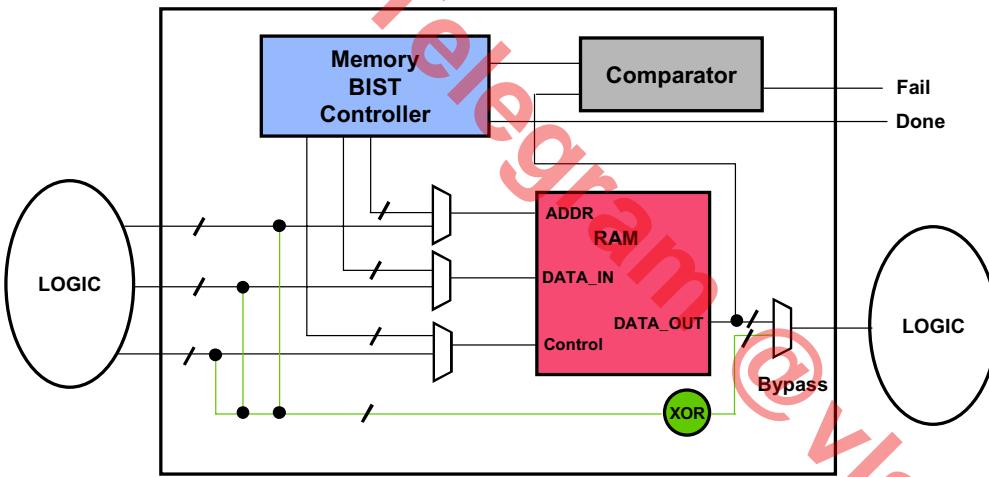
ATPG>macrotest <instance> <pattern-file>

## Notes:

# Memory BIST

## Memory BIST

- ◆ The following applies to Memory BIST patterns:
  - Places pattern generation and analysis in the chip
  - Often includes memory bypass logic to simplify ATPG



5-43 • Design-for-Test: Scan and ATPG:  
Achieving High Test Coverage

Copyright © 2003 Mentor Graphics Corporation

## Notes:

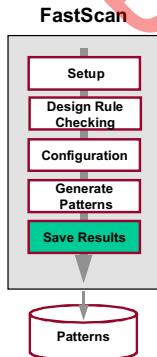
## Test Pattern Type Summary

**Test Pattern Type Summary**

| CIRCUITRY                 | PATTERN TYPES |           |            |         |          |           |       |
|---------------------------|---------------|-----------|------------|---------|----------|-----------|-------|
|                           | BASIC         | CLOCK SEQ | MULTI LOAD | RAM SEQ | CLOCK PO | MacroTest | MBIST |
| FULL SCAN                 | ■             |           |            |         |          |           |       |
| RAMs/ROMs                 |               | ■         |            |         |          | ■         | ■     |
| RAM/ROM<br>SHADOW LOGIC   | ■             | ■         | ■          | ■       |          |           |       |
| BLACK BOXES               |               |           |            |         |          | ■         |       |
| BLACK BOX<br>SHADOW LOGIC |               |           |            |         |          | ■         |       |
| NONSCAN<br>CIRCUITRY      |               | ■         | ■          | ■       |          |           |       |
| NONSCAN<br>LATCHES        | ■             | ■         | ■          |         | ■        |           |       |
| SPECIAL (C8/C9)           |               |           |            |         | ■        |           |       |

### Notes:

# Saving Patterns



## Saving Patterns

- ◆ Save ATPG patterns in the following formats:
  - Reuse, debugging, and diagnostics
    - ASCII            ATPG >SAVe PAtterns <Filename> -Ascii
    - Binary          ATPG >SAVe PAtterns <Filename> -BInary
    - These are the only pattern formats that can be read back into FastScan
  - For time-based verification
    - Verilog         ATPG >SAVe PAtterns <Filename> -Verilog
    - VHDL           ATPG >SAVe PAtterns <Filename> -VHdl
    - Allows user to independently verify test patterns with circuit timing
  - Manufacturing test (ATE)
    - WGL            ATPG >SAVe PAtterns <Filename> -Wgl
    - STIL           ATPG >SAVe PAtterns <Filename> -STil
    - TI-TDL        ATPG >SAVe PAtterns <Filename> -TItdl
    - ...
- ◆ Use .gz or .Z filename extension to save compressed files

## Notes:

## Reuse, Debugging, and Diagnostics

### Reuse, Debugging, and Diagnostics

- ◆ FastScan can write out a subset of patterns in any format.
- ◆ External patterns can be read back into FastScan...
  - for debugging (timing checks).
  - and reuse (format translation).
- ◆ Failing pattern data from the tester can be read back into FastScan to determine which set of faults match actual failures.

### Notes:

# Reuse, Debugging, and Diagnostics: ASCII and Binary Formats

## Reuse, Debugging, and Diagnostics: ASCII and Binary Formats

- ◆ ASCII is the default pattern format.
  - Only format other than binary that can be read back into FastScan.
  - Does not contain timing information.
  - Fully commented and readable.
    - Test procedures
    - Scan test procedures
    - Scan memory elements
    - Test coverage statistics
- ◆ Binary format contains the same information as ASCII but is in a condensed form.
  - Used for archival purposes for large designs.

## Notes:

# Reuse, Debugging, and Diagnostics: Reading ASCII Files Back into FastScan

---

## Reuse, Debugging, and Diagnostics: Reading ASCII Files Back into FastScan

- ◆ Do the following to read ASCII files back into FastScan

```
ATPG>SAVe PAtterns <testpat.ascii>
```

```
.
```

```
.
```

```
.
```

```
// Read existing patterns
```

```
ATPG> SET PAttern Source External <testpat.ascii>
```

```
ATPG> DIAgnoSe FAilure <failurefile.txt>
```

- ◆ Use .gz or .Z filename extension to automatically read compressed file formats

## Notes:

## Time-Based Verification

### Time-Based Verification

- ◆ ATPG is event-based.
- ◆ Simulation using a time-based simulator validates patterns with actual timing.
- ◆ By default, FastScan writes events with 10 ns separation.
- ◆ User enters procedure file timing information based on circuit timing requirements and tester specifications.

### Notes:

## Time-Based Verification (Cont.)

### Time-Based Verification (Cont.)

- ♦ Do the following to enter procedure file timing information:

```
SETUP>ADD SCan Groups grpl master.testprog  
:  
//timing in master.testproc  
ATPG> SAVe PAtterns <testpat1.v> -Verilog -Parallel  
//timing in new.testproc  
ATPG> SAVe PAtterns <testpat2.v> -Verilog -Parallel \  
-Procfile <new.testproc>
```

### Notes:

# Verification of Pattern Formats

## Verification of Pattern Formats

- ◆ Scan patterns and chain test
  - By default, test patterns contain a chain integrity test in addition to the ATPG patterns.
  - Chain test shifts a repeated data pattern through the scan chains.
  - Chain test detects problems in the shift path.
- ◆ Parallel and serial patterns
  - Serial patterns behave as if applied by the tester (one clock cycle for each shift).
  - Parallel pattern values are directly applied to scan cell inputs to speed up simulation (one clock cycle for all shifts).
  - Verify all patterns in parallel format and a few in serial.

```
ATPG> SAVE PAtterns <testpat_p.v> -Verilog -Parallel
```

```
ATPG> SAVE PAtterns <testpat_s.v> -Verilog -Serial -Sample 2
```

## Notes:

## Manufacturing Test

### Manufacturing Test

- ◆ Many testers have special pattern formats.
- ◆ FastScan can save patterns for ATE in specific formats and industry standard formats.
  - WGL
  - STIL
  - TI-TDL
  - ⋮
- ◆ User defined pattern timing in the procedure file is based on circuit timing and tester specifications.

### Notes:

# Lab: Achieving High Test Coverage

## Objectives

- Apply FastScan pattern types to relevant circuits.
- Apply pattern sequencing to achieve high test coverage.
- Save patterns in three formats: ASCII, Verilog, and WGL.
- Save parallel patterns.
- Save serial sample patterns.
- Save chaintest patterns.
- Use ModelSim to simulate and verify the following testbenches:
  - Verilog.
    - Parallel patterns.
    - Serial sample patterns.
  - Chaintest patterns.
- Read an ASCII pattern file into FastScan, create new patterns and save the new patterns in TI-TDL format.

## List of Exercises

- Exercise 11: Creating and Saving Patterns in Different Formats
- Exercise 12: Verifying Patterns Using ModelSim
- Exercise 13: Reading ASCII files into FastScan (Optional)

## Getting Started

1. Change to the \$ATPGNW/lab 5/exercise\_11 directory.

```
shell> cd $ATPGNW/lab5/exercise_11
```



Remember that for the exercises in this lab you use the libraries found in the **libraries\_5\_to\_6** directory.

Note

## Exercise 11: Creating and Saving Patterns in Different Formats

In this exercise, you invoke FastScan on a design and apply FastScan pattern types to relevant circuits. Then you apply pattern sequencing to achieve higher test coverage. Finally, you create and save test patterns in the following formats:

- For reuse, debugging, and diagnostics
  - ASCII
- For time-based verification
  - Verilog
    - Parallel
    - Serial
    - Chaintest
- For manufacturing test
  - WGL

1. Invoke FastScan on the following circuit:

Design: gate\_2001\_scan.v

Library: adk.atpg

Log file: results/ex\_11.log

2. The initial setup commands for this design are in a dofile. Run the file gate\_2001\_scan.dofile.
3. Study the Session Transcript window to see what commands have been executed by the dofile:

What actions do the commands perform?

---

---

---

---

This circuit design contains the following:

- Non-scan elements
- Full scan elements (scan chains)
- RAMs

By default, FastScan generates basic scan patterns, which produce high test coverage on full-scan circuitry. Typically, more complex designs require the application of specific pattern types to achieve higher overall test coverage.

You achieve higher test coverage by using patterns designed to simulate faults for specific circuitry. Also, higher test coverage is achieved by applying the appropriate pattern type in the proper sequence.

Use the following pattern sequence to achieve higher test coverage:

- **Basic**— used on full-scan design circuitry.

- **Clock PO**— used on circuitry where a clock signal passes through combinational logic to a primary output.
  - **Clock sequential**— propagates values through non-scan latches.
  - **Multi-load**— used on RAM/ROM designs that contain non-scan cells.
  - **RAM sequential**— used to propagate values through RAM.
4. Enable FastScan to generate clock sequential patterns, with up to two unscanned d-types between scanned ones.

```
SETUP> set simulation mode combinational -depth 3
```



Remember, the recommendation is that you should not have a depth greater than 5. (4 unscanned d-types)

**Note**

This enables FastScan to test through scan-based designs that contain limited non-scan sequential logic or non-scan latches.

(Clock sequential depth - 1) defines the number of non-scan cells connected in series that FastScan can test through (two in this case).

5. Enable FastScan to generate multi-load patterns.

```
SETUP> set multiple load on
```

Clock sequential patterns are generated to load scan chains and set desired values in non-scan cells without disturbing any scan or pertinent non-scan logic.

6. Go to ATPG mode.
- a. Click Done with Setup in the FastScan Control Panel pane.
  - b. Click on the Pattern Generation button. When the DRC Warning Occurred dialogue box opens, click No. Go straight into ATPG mode.

- c. Observe the warning messages in the session transcript area.

What types of DRC warning messages were identified, and what do they mean?

Type: \_\_\_\_\_ error: \_\_\_\_\_

Type: \_\_\_\_\_ error: \_\_\_\_\_

Type: \_\_\_\_\_ error: \_\_\_\_\_

Type: \_\_\_\_\_ error: \_\_\_\_\_

7. Select Typical settings for the ATPG run (stuck-at-fault model for all circuit fault locations).
8. Generate random and deterministic patterns.

- a. Notice in the Session Transcript window that FastScan begins with random pattern generation. FastScan identifies and stores only those patterns that detect faults. Random pattern generation never identifies redundant faults, nor faults with low probability of detection. Capture clocks are tested until exhausted.



It is possible to use the Set Random Patterns command to change the number of random patterns generated. This changes the test coverage percentage using random patterns. The default integer value is 1024.

```
SETUP> set random patterns integer
```

You also can turn random pattern generation off.

```
SETUP> set random atpg off
```

We will not be using these commands in this exercise.

- b. Observe in the Session Transcript window that FastScan uses deterministic test pattern generation when it creates a test pattern intended to detect a given fault. The procedure is to pick a fault from

the fault list, create a pattern to detect the fault, fault simulate the pattern, and check to make sure the pattern detects the fault.

- Fill in the following statistics from the ATPG Run Statistics dialogue box and the **report statistics** command:

**Table 5-1. Test Pattern Generation Results**

**Table 5-2.**

| Type of Pattern | No. of Effective Patterns | No. of Sim'd Patterns | No. of detected | No. of Aborted | No. of AU | No. Redun. |
|-----------------|---------------------------|-----------------------|-----------------|----------------|-----------|------------|
|                 |                           |                       |                 |                |           |            |
|                 |                           |                       |                 |                |           |            |
|                 |                           |                       |                 |                |           |            |

**Table 5-3. Report Statistics ATPG**

**Table 5-4.**

| Fault Class            | # faults (coll.) | # faults (total) |
|------------------------|------------------|------------------|
| FU (full)              |                  |                  |
| UC (uncontrolled)      |                  |                  |
| UO (unobserved)        |                  |                  |
| DS (det_sim)           |                  |                  |
| DI (det_imp)           |                  |                  |
| PU (posdet_untestable) |                  |                  |
| PT (posdet_testable)   |                  |                  |
| UU (unused)            |                  |                  |
| TI (tied)              |                  |                  |
| RE (redundant)         |                  |                  |

**Table 5-4.**

| Fault Class          | # faults (coll.) | # faults (total) |
|----------------------|------------------|------------------|
| AU (atpg_untestable) |                  |                  |

**Table 5-5. Report Coverage/Effectiveness ATPG****Table 5-6.**

| Cov./Effect.         | # faults (coll.) | # faults (total) |
|----------------------|------------------|------------------|
| test_coverage        |                  |                  |
| fault_coverage       |                  |                  |
| atpg_effectiveness   |                  |                  |
| # test_patterns      |                  |                  |
| # basic_patterns     |                  |                  |
| # clock_po_patterns  |                  |                  |
| # clock_seq_patterns |                  |                  |
| # mult_load_patterns |                  |                  |
| # simulated_patterns |                  |                  |
| CPU_time (secs)      |                  |                  |

- a. How many faults were undetected? \_\_\_\_\_
- b. How many faults were tested unsuccessfully? \_\_\_\_\_
10. Display a fault analysis summary using the **report testability data** command.

How many faults are connected to RAMs? \_\_\_\_\_

How many faults are connected from RAMs? \_\_\_\_\_

The Report Testability Data command identifies and displays any circuitry connections that may cause test coverage problems. FastScan reports a

large number of faults connected to RAM. This indicates that you need to generate RAM sequential patterns to propagate values through the RAM circuitry.

11. Enable FastScan to generate RAM sequential patterns, with up to two unscanned d-types between scanned ones.

What command do you use?

---

12. Generate ATPG patterns for RAM sequential. This adds to the patterns already generated; it does not start from scratch.

ATPG> **run**

13. Fill in the following statistics from the ATPG Run Statistics dialogue box and the **report statistics** command:

**Table 5-7. Test Pattern Generation Results**

**Table 5-8.**

| Type of Pattern | No. of Effective Patterns | No. of Sim'd Patterns | No. of detected | No. of Aborted | No. of AU | No. Redun. |
|-----------------|---------------------------|-----------------------|-----------------|----------------|-----------|------------|
|                 |                           |                       |                 |                |           |            |
|                 |                           |                       |                 |                |           |            |
|                 |                           |                       |                 |                |           |            |

**Table 5-9. Report Statistics ATPG**

**Table 5-10.**

| Fault Class       | # faults (coll.) | # faults (total) |
|-------------------|------------------|------------------|
| FU (full)         |                  |                  |
| UC (uncontrolled) |                  |                  |

**Table 5-10.**

| Fault Class            | # faults (coll.) | # faults (total) |
|------------------------|------------------|------------------|
| UO (unobserved)        |                  |                  |
| DS (det_sim)           |                  |                  |
| DI (det_imp)           |                  |                  |
| PU (posdet_untestable) |                  |                  |
| PT (posdet_testable)   |                  |                  |
| UU (unused)            |                  |                  |
| TI (tied)              |                  |                  |
| RE (redundant)         |                  |                  |
| AU (atpg_untestable)   |                  |                  |

**Table 5-11. Report Coverage/Effectiveness ATPG****Table 5-12.**

| Cov./Effect.         | # faults (coll.) | # faults (total) |
|----------------------|------------------|------------------|
| test_coverage        |                  |                  |
| fault_coverage       |                  |                  |
| atpg_effectiveness   |                  |                  |
| # test_patterns      |                  |                  |
| # basic_patterns     |                  |                  |
| # clock_po_patterns  |                  |                  |
| #ram_seq_patterns    |                  |                  |
| # clock_seq_patterns |                  |                  |
| # mult_load_patterns |                  |                  |
| # simulated_patterns |                  |                  |
| CPU_time (secs)      |                  |                  |

- a. How many faults were undetected? \_\_\_\_\_
- b. How many faults were tested unsuccessfully? \_\_\_\_\_
14. Close the FastScan ATPG Run Statistics dialogue box.

You have created and increased patterns that achieve higher test coverage. In the following steps, you will save these patterns in three formats: ASCII, Verilog, and WGL.

15. Use the Save Patterns... button in the Button pane to save the patterns as an ASCII file with the following characteristics:

File name: results/testpat.ascii

Pattern format: ASCII, with default ASCII options

Save Scan Cell Data in: Parallel

- a. Overwrite any existing files of the same name.  
b. Leave the default settings for the rest.

The ASCII patterns can be utilized for reuse, debugging, and diagnostics. This is the default format that FastScan generates when you save patterns. This format contains test pattern data in a text-based parallel format. This format is the one of only two formats that can be read back into FastScan.

- c. Select **File > Open > Text File (Read-Only)** to view the file you have written.

16. Use the Save Patterns... button in the Button pane to save the patterns in parallel Verilog format with the following characteristics:

File name: results/testpat\_par.v

Pattern format: Verilog

Save Scan Cell Data in: Parallel

Test to be Saved: Scan & Chain

- a. Overwrite any existing files of the same name.
- b. Leave the default settings for the rest.

The Verilog pattern file contains procedures to apply test patterns, compare expected output with simulated output, and print out a report containing information about failing comparisons. FastScan writes all patterns and comparison functions into one main file (*filename*), while writing the primary output names in another file (*filename.po.name*). Choosing parallel loading of scan chains, enables FastScan to write the names of the scan output pins of each scan subchain of each scan chain into separate files (for example, *filename.chain1.name*). This allows time-based simulators to report output pins that have discrepancies between expected and simulated outputs.

- c. Select **File > Open > Text File (Read-Only)** to view the file you have written.
- 17. Use the Save Patterns... button in the Button pane to save the patterns in serial Verilog format with the following characteristics:

File name: results/testpat\_ser.v

Pattern format: Verilog

Save Scan Cell Data in: Serial

Test to be Saved: Scan & Chain

Number of Patterns to Sample Per Pattern Type: 5

- a. Overwrite any existing files of the same name.
- b. Leave the default settings for the rest.

By setting the number of patterns to sample per pattern type to five you have reduced the CPU processing time.

- c. How many patterns were actually sampled? \_\_\_\_\_
- d. Select **File > Open > Text File (Read-Only)** to view the file you have written.
18. Use the Save Patterns... button in the Button pane to save the patterns to effect a chaintest only, in serial Verilog format with the following characteristics:
- |                         |                         |
|-------------------------|-------------------------|
| File name:              | results/chaintest_ser.v |
| Pattern format:         | Verilog                 |
| Save Scan Cell Data in: | Serial                  |
| Tests to be Saved:      | Chain Test              |
- Overwrite any existing files of the same name.
  - Leave the default settings for the rest.
- Chain test patterns check scan chain integrity. The chain test applies the test\_setup procedure, followed by the load\_unload procedure for loading scan chains, and the load\_unload procedure (again) for unloading scan chains. Each load\_unload procedure in turn calls the shift procedure. This operation typically loads a repeating pattern of “0011” into the chains.
- c. Select **File > Open > Text File (Read-Only)** to view the file you have written.
19. Use the Save Patterns... button in the Button pane to save the patterns to serial WGL format with the following characteristics:

|                         |                         |
|-------------------------|-------------------------|
| File name:              | results/testpat_par.WGL |
| Pattern format:         | WGL                     |
| Save Scan Cell Data in: | Parallel                |

Tests to be Saved: Chain & Scan Test

- a. Overwrite any existing files of the same name.
- b. Leave the default settings for the rest.

The WGL format supports parallel loading of the scan cells.

- c. Select File > Open > Text File (Read-Only) to view the file you have written.
20. Exit FastScan.

## Exercise 12: Verifying Patterns Using ModelSim

In this exercise, you verify the patterns you created in Exercise 11. You use ModelSim to simulate and verify the following testbenches for time-based verification using Verilog:

- Chaintest patterns
- Parallel patterns
- Serial sample patterns

### Getting Started

1. You should be in the \$ATPG/lab5/exercise\_11/results directory.

```
shell> cd $ATPGN/1ab5/exercise_11/results
```

2. Compile the following files:  
*<testbench> <netlist> <verilog library>*

```
shell> vlib work
```

```
shell> vlog testpat_ser.v ../gate_2001_scan.v -V adk.v
```

Use the “V” switch to improve the runtime.



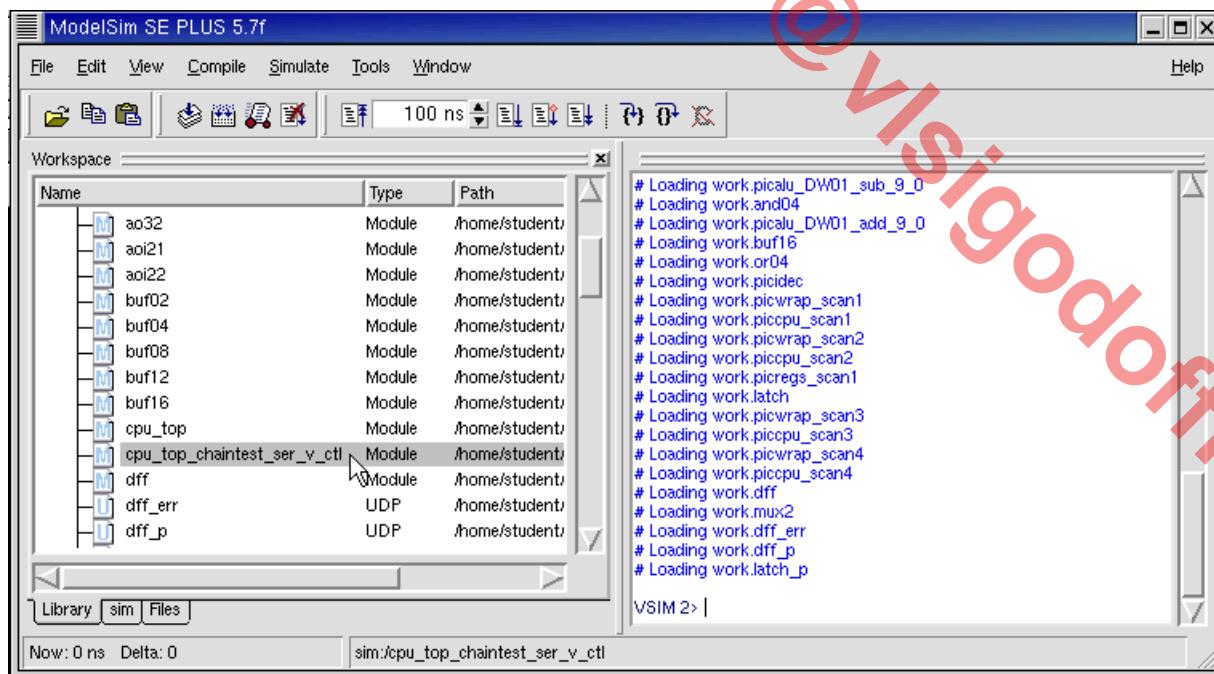
### 3. Open ModelSim.

```
shell> vsim
```

A Welcome to ModelSim window opens. Close it and proceed directly to ModelSim.

### 4. Load the design and run the simulation:

- In the Library tab, expand the work directory and double click on the cpu\_top\_chaintest\_ser\_v\_ctl block as shown:



Double clicking on the block loads it into the simulator.

- b. The testbench generated by FastScan contains all the stimulus required, so you only need to run the simulation. Use the pulldown menu **Simulate> Run> Run -All**, the Run All button  or type **run -all** at the VSIM> command line prompt to run the simulation from start to finish.

ModelSim is checking for errors between the simulated and expected pattern values.

This takes about 2 minutes to run.

- c. Look at the Transcript window in ModelSim to determine the outcome of simulation.
- i. How many nanoseconds does the simulation run for? \_\_\_\_\_
  - ii. Are there any errors? \_\_\_\_\_
  - iii. How do you know this? \_\_\_\_\_  
\_\_\_\_\_
- d. After checking all the messages in the ModelSim transcript, click Yes in the Finish Vsim dialogue box.



**Note**

If you would like to see what has been simulated, click No in the finish Vsim dialogue box and select **Simulate > Run > Restart** to restart the simulation. Open a Wave window and add the top level signals to this window (**add wave \*** at the command line prompt), then do a **run -all** again.

Adding the Wave window is not necessary for this simulation as the testbench does all the checking and the test out is sufficient enough to say that it has passed.

ModelSim simulated and verified that your serial Verilog patterns do not have simulation mismatch.

5. Next you simulate and verify the parallel Verilog patterns. Remember that this uses all the patterns, but does not shift them in and out of the scan chain.

- a. Compile the parallel testbench (remember that the netlist and library are already compiled from the last run.)

What command do you use? \_\_\_\_\_

- b. Open ModelSim.

What command do you use? \_\_\_\_\_

- c. In the Library tab, expand the work library and double click on cpu\_top\_testpat\_par\_v\_ctl to load the parallel testbench into the simulator.
- d. The testbench generated by FastScan contains all the stimulus required, so you only need to run the simulation. Use the pulldown menu **Simulate > Run > Run -All**, the Run All button, or the command to run the simulation from start to finish.

ModelSim is checking for errors between the simulated and expected pattern values.

This may take 10-13 minutes, depending upon the operating system you use.

- e. Look at the Transcript window in ModelSim to determine the outcome of simulation.
  - i. What test is done first? \_\_\_\_\_
  - ii. How many nanoseconds does the simulation run for? \_\_\_\_\_
  - iii. Are there any errors? \_\_\_\_\_
  - iv. How do you know this? \_\_\_\_\_

- f. After checking all the messages in the ModelSim transcript, click Yes in the Finish Vsim dialogue box.

ModelSim simulated and verified that your sampled parallel Verilog patterns do not have simulation mismatch.



**Note** The next part takes around a half an hour to simulate, so if you are running out of time, you can leave the lab at Exercise 12. The process is the same as the two previous simulations.

6. Now you simulate and verify the serial Verilog patterns. Recall that the number of patterns to sample per pattern was set to five. This gives a good estimate of coverage, plus simulation and verification processing time is greatly reduced (but it is still pretty high).

- a. Compile the serial testbench (remember that the netlist and library are already compiled from the last run.)

What command do you use? \_\_\_\_\_

- b. Open ModelSim.
- c. In the Library tab, expand the work library and double click on cpu\_top\_testpat\_ser\_v\_ctl to load the parallel testbench into the simulator.
- d. Run the simulation.
- e. ModelSim is checking for errors between the simulated and expected pattern values.

This takes around 30 minutes to run.

- f. Look at the Transcript window in ModelSim to determine the outcome of simulation.
  - i. What test is done first? \_\_\_\_\_
  - ii. How many nanoseconds does the simulation run for? \_\_\_\_\_

- iii. Are there any errors? \_\_\_\_\_
- iv. How do you know this? \_\_\_\_\_  
\_\_\_\_\_

- g. After checking all the messages in the ModelSim transcript, click Yes in the Finish Vsim dialogue box.
- h. ModelSim simulated and verified that your serial Verilog patterns do not have simulation mismatch.

### Exercise 13: Reading ASCII files into FastScan (Optional)

In this exercise, you invoke FastScan on a design and apply FastScan pattern types. Then you read in an external ASCII pattern file into FastScan. Next you generate patterns and observe the results. Finally, you save these patterns in a TI-TDL format.

#### Getting Started

1. Change to the \$ATPGNW/lab 5/exercise\_13 directory.

```
shell> cd $ATPGNW/lab5/exercise_13
```

2. Invoke Fastscan.

Design: gate\_2001\_scan.v

Library: /libraries\_5\_to\_6/adk.atpg

Log file: results/ex\_13.log

You could invoke FastScan on the design straight from the shell prompt and not use the Invocation dialogue box.

What command do you use? \_\_\_\_\_

3. Run a dofile.
  - a. Execute the file gate\_2001\_scan.dofile. (*Use the Dofile... button in the Button pane*).
4. Study the Session Transcript window to see what commands have been executed by the dofile:

What actions do the commands perform?

---

---

---

---

---

This design is same as in Exercise 11. It contains the following:

- Non-scan elements
- Full scan elements (scan chains)
- RAMs

You have already learned that in order to achieve higher test coverage, FastScan must be enabled to generate specific pattern types to test specific circuitry and apply them in the proper sequence. If you have questions about FastScan patterns, refer to Exercise 11.

5. Enable FastScan to generate multi-load patterns and RAM sequential patterns.

What two commands do you use to do this?

- a. \_\_\_\_\_
  - b. \_\_\_\_\_
6. Go to ATPG mode, ignoring any DRC warning messages. You should now be in ATPG system mode.
  7. Read in an external ASCII file.
    - a. Click on the Pattern Source button in the graphics pane. When the Use Typical Settings or Customize? dialogue box opens, select the Customize button.
    - b. In the Setup Pattern Source dialogue window, click on External Patterns From: and browse to the file testpat.ascii.
    - c. Patterns Are in the Following Format: Click on ASCII.
    - d. Verify that Sequential is selected in the Pattern Set Creates/Contains RAM Test Patterns That Are section.
    - e. Click OK.

You have read in an external ASCII file. This option enables you to use test patterns created previously or perhaps use patterns created by another ATPG test group.

8. Select fault model and add faults. Use the default values (fault type stuck, and add all faults).

What commands are executed? (*Session Transcript pane.*)

---

---

9. Generate random and deterministic patterns for stuck-at faults.
  - a. Click on the Run button in the Button pane. The FastScan ATPG Run Statistics dialogue box opens.

10. Obtain a detailed report of the design's simulation statistics and fill in the following. Use all of the information and command used in previous exercises to generate reports and statistics:

**Table 5-13. Report Statistics ATPG****Table 5-14.**

| Fault Class          | # faults (coll.) | # faults (total) |
|----------------------|------------------|------------------|
| FU (full)            |                  |                  |
| UC (uncontrolled)    |                  |                  |
| UO (unobserved)      |                  |                  |
| DS (det_sim)         |                  |                  |
| DI (det_imp)         |                  |                  |
| PT (posdet_testable) |                  |                  |
| UU (unused)          |                  |                  |
| TI (tied)            |                  |                  |

**Table 5-15. Report Coverage/Effectiveness ATPG****Table 5-16.**

| Cov./Effect.         | # faults (coll.) | # faults (total) |
|----------------------|------------------|------------------|
| test_coverage        |                  |                  |
| fault_coverage       |                  |                  |
| atpg_effectiveness   |                  |                  |
| # test_patterns      |                  |                  |
| # basic_patterns     |                  |                  |
| # ram_seq_patterns   |                  |                  |
| # clock_seq_patterns |                  |                  |
| # mult_load_patterns |                  |                  |

**Table 5-16.**

| Cov./Effect.         | # faults (coll.) | # faults (total) |
|----------------------|------------------|------------------|
| # simulated_patterns |                  |                  |
| CPU_time (secs)      |                  |                  |

- a. How many faults were undetected? \_\_\_\_\_
11. Use the Save Patterns... button in the Button pane to save parallel TI-TDL patterns with the following characteristics:
- File name: results/testpat\_par.ti.tdl
- Pattern format: TITDL
- Save Scan Cell Data in: Parallel
- Tests to be Saved: Chain & Scan Test
- a. Overwrite any existing files of the same name.
- b. Leave the default settings for the rest.
- You have created parallel TI-TDL manufacturing test patterns. Data is loaded into the scan cells in parallel. The TI-TDL ASIC vendor data format is text based.
- c. use **File > Open > Text File (Read-Only)** to look at the file you have written.
12. Exit FastScan.

## Test Your Knowledge

1. What pattern type tests through non-sequential logic or non-scan latches?

---

2. What command do you use to change the number of random patterns generated?

---

3. What command is used to display a fault analysis summary?

---

4. What pattern format can be read back into FastScan?

---

5. What happens when you choose parallel loading of scan chains when creating Verilog pattern files?

---

6. Why create chaintest patterns?

---

7. Why would you sample patterns?

---

8. What pattern formats are used by ATE?

---

## Lab Summary

Now that you completed the achieving High Test Coverage lab, you should know how to do the following:

- Apply FastScan pattern types to relevant circuits in the proper sequence.
- Save patterns in the following formats: ASCII, Verilog, WGL, and TI-TDL.
- Save parallel, serial, and chaintest patterns.
- Use ModelSim to simulate and verify testbenches.
- Read an ASCII pattern file back into FastScan.

## **Module 6**

# **Creating High Quality Patterns at Low Cost**

### **Objectives**

Upon completion of this module, you will be able to:

- Optimize patterns for quality and cost.
- Create patterns for At-speed ATPG.
- Create transition fault patterns.
- Create path delay patterns.
- Create IDDQ patterns.

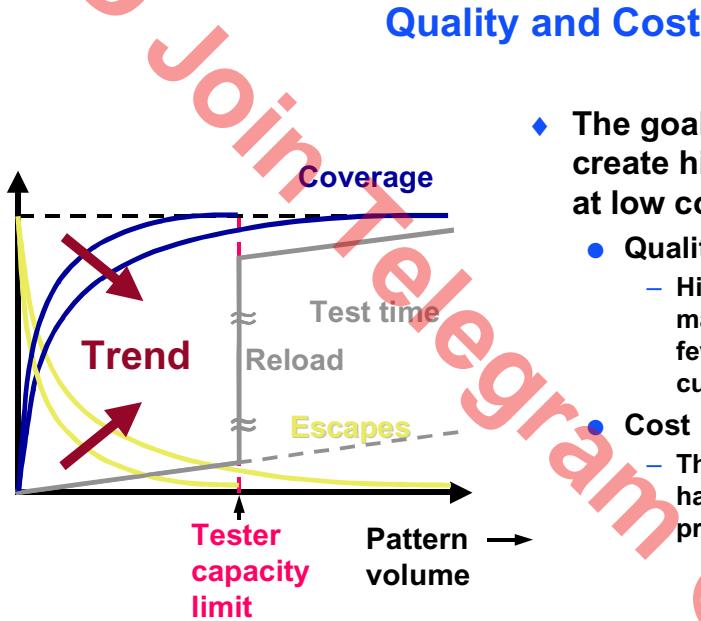
## Module Topics

### Module Topics

- ◆ This module addresses the following topics:
  - Quality
  - Cost
  - Fault models
  - At-speed ATPG
  - Creating transition, path delay, and IDDQ patterns
  - ATE characteristics

### Notes:

## Quality and Cost

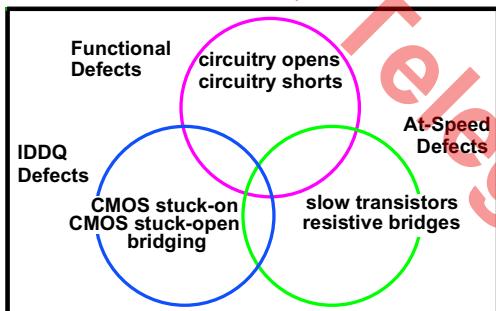


- ◆ The goal of test generation is to create high quality test patterns at low cost.
  - Quality
    - High quality test detects manufacturing defects; therefore, fewer defective parts shipped to customers.
  - Cost
    - The time it takes to test one chip has a major impact on production costs.

### Notes:

# Quality

## Quality



- ◆ To adequately test ICs for various types of possible defects, use the following fault models:
  - Stuck-at.
  - Path delay.
  - Transition.
  - IDDQ.
- ◆ Using a combination of fault models ensures high test quality.
- ◆ Poor test quality can adversely affect production schedules and costs.

## Notes:

Cost

## Cost

- ◆ ATE (tester) time is a recurring cost for each chip.
  - Scan test application time is equal to the pattern count times the length of the longest scan chain.
  - Reload is extremely slow but may be necessary if pattern count exceeds tester memory.
  - Patterns are often truncated to avoid reloads but pattern quality is often impacted.
- ◆ The following factors affect tester costs:
  - Number of scan channels.
  - Amount of required memory.
  - Number of clocks.
  - Required test frequency.

Notes:

## At-Speed ATPG

### At-Speed ATPG

- ◆ Checks the amount of time it takes for a device to change logic states
- ◆ Detects timing failures that occur when a circuit operates correctly at a slow clock rate, but then fails when run at clock speed

### Notes:

## At-Speed ATPG (Cont.)

### At-Speed ATPG (Cont.)

- ◆ Traditional functional testing for At-speed defects
  - Long development time.
  - Required detailed circuit knowledge
  - Difficult to fault grade.
- ◆ At-speed ATPG
  - Patterns are automatically generated.
  - Simpler.
  - Easier to evaluate.
    - Apply At-speed test with slow shift.

### Notes:

# At-Speed ATPG and the Transition Fault Model

## At-Speed ATPG and the Transition Fault Model

- ◆ The transition fault model:
  - Detects gross pin-to-pin delay effects
  - Requires no timing information
  - Paths are randomly chosen
  - Behaves as a stuck-at fault for a brief period of time
- ◆ Includes two fault models:
  - Slow-to-rise
    - Models a pin slow to change from 0 to 1
  - Slow-to-fall
    - Models a pin that is slow to change from 1 to 0
- ◆ Requires two test patterns for detection:
  - Initialization vector
    - Places initial transition value at the point of the fault
  - Transition vector
    - Places final transition value at the point of the fault

## Notes:

# At-Speed ATPG and the Path Delay Fault Model

---

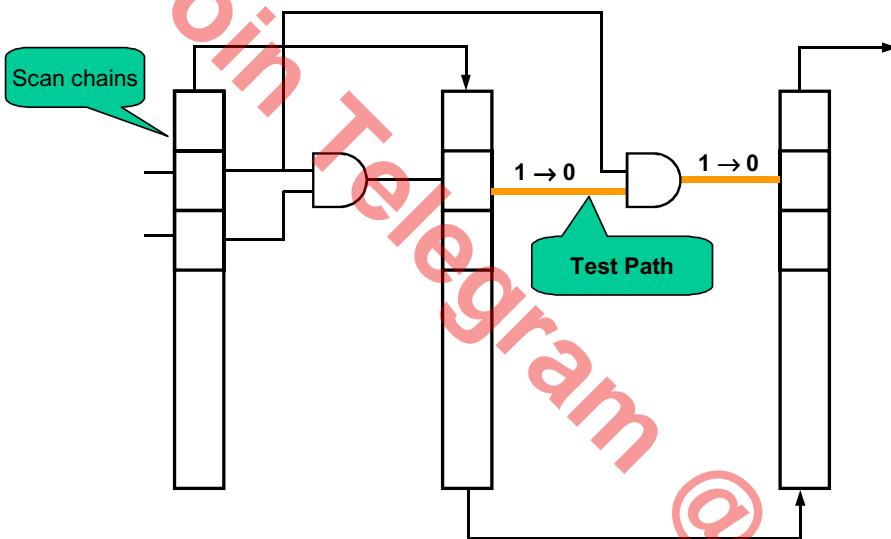
## At-Speed ATPG and the Path Delay Fault Model

- ◆ Models defects in circuit paths
- ◆ Associated with testing AC performance of critical paths
- ◆ Path is defined by a timing analysis tool
- ◆ Identified by path topology
  - Launch point: PI or state element
  - Capture point: PO or state element

## Notes:

# The Path Delay Model

## The Path Delay Model



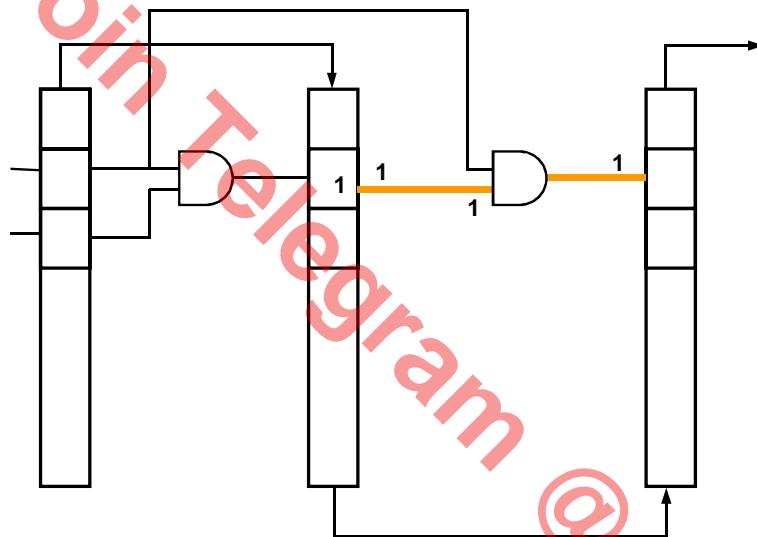
**Goal = test the path for a 1 to 0 transition**

FastScan will automatically determine appropriate values  
to load scan chain and perform test.

## Notes:

## The Path Delay Model (Cont.)

### The Path Delay Model (Cont.)

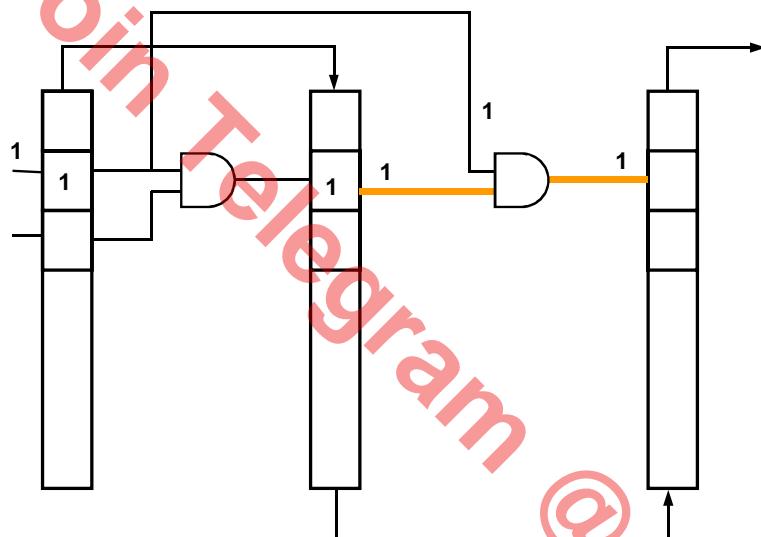


Define the initial path values

### Notes:

## The Path Delay Model (Cont.)

### The Path Delay Model (Cont.)

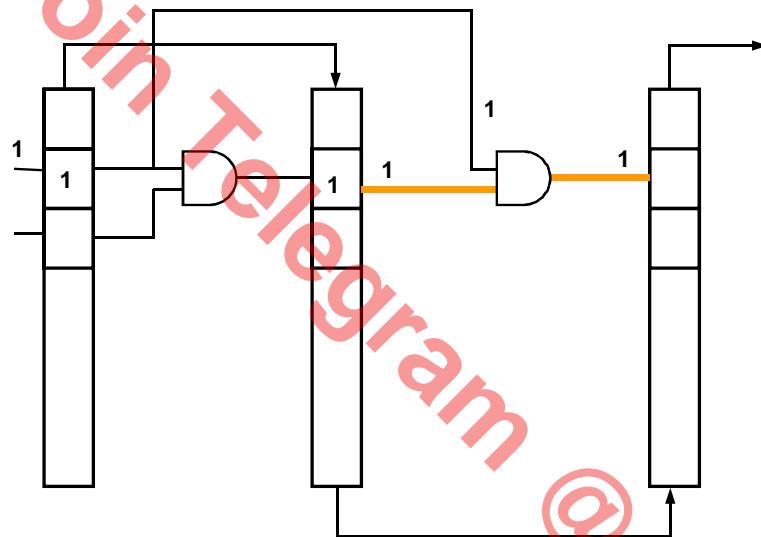


Define other values to sensitize path

### Notes:

## The Path Delay Model (Cont.)

### The Path Delay Model (Cont.)

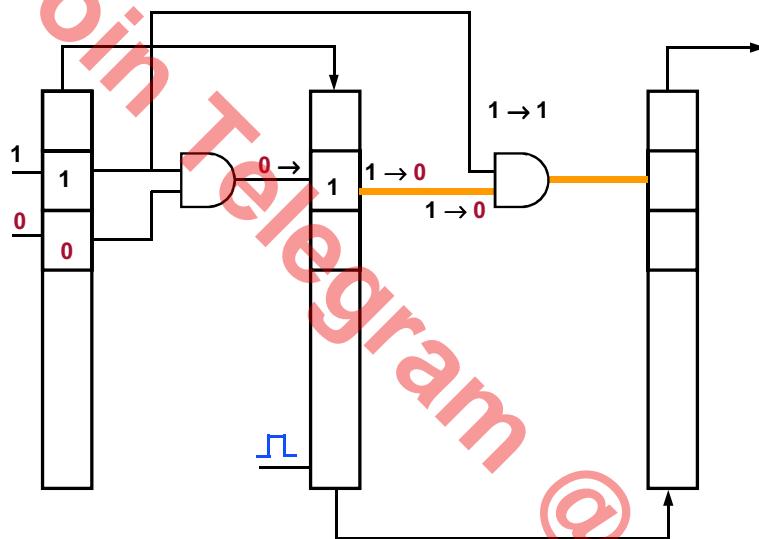


Define other values to sensitize path

### Notes:

## The Path Delay Model (Cont.)

### The Path Delay Model (Cont.)

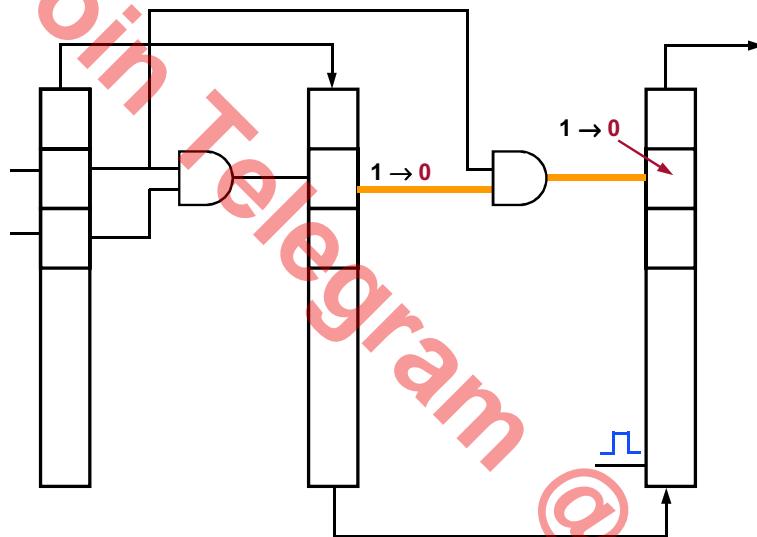


Pulse clock in functional mode to cause launch  
Causes the values to propagate through the path

### Notes:

## The Path Delay Model (Cont.)

### The Path Delay Model (Cont.)



Pulse clock in functional mode to capture value  
if transition propagated in time

### Notes:

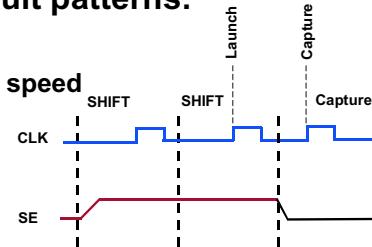
# Transition Fault Patterns

## Transition Fault Patterns

- FastScan uses the following transition fault patterns:

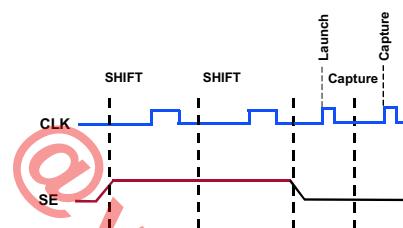
- Launch-off shift

- Allows FastScan to switch scan enable At-speed (requires additional clock routing of SE)
- Applies combinational ATPG



- Broadside

- Scan enable timing is not critical
- Applies clock sequential ATPG
- Activated when clock-sequential depth is >2



## Notes:

## Creating Transition Fault Patterns: Launch-Off Shift

### Creating Transition Fault Patterns: Launch-Off Shift

- ♦ Do the following to create launch-off shift transition fault patterns:

```
ATPG> SET FAult Type Transition  
ATPG> SET PAttern Type -Sequential 0 // default  
ATPG> CREAtE PAtterns
```

*It is recommended to use Broadside transition fault patterns.*

### Notes:

## Creating Transition Fault Patterns: Broadside

### Creating Transition Fault Patterns: Broadside

- ♦ Do the following to create broadside transition fault patterns:

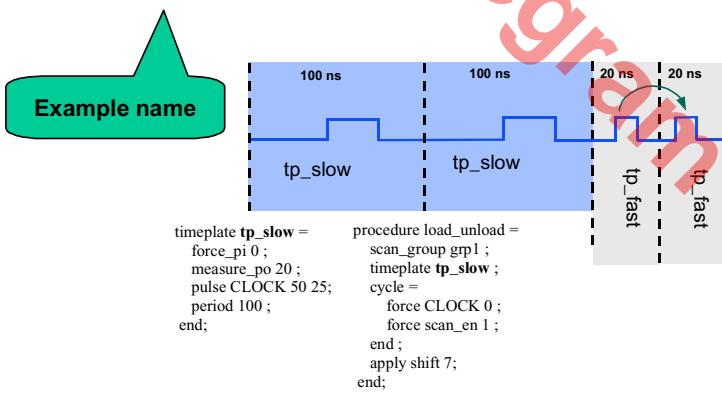
```
ATPG> SET FAult Type Transition -no_shift_launch  
ATPG> SET PAttern Type -Sequential 2  
ATPG> CREAtE PAtterns
```

### Notes:

# Timing for At-Speed Test

## Timing for At-Speed Test

- ♦ Timing is added to transition fault patterns by timeplates in test procedure files.
- ♦ FastScan uses the following types of timeplates for broadside transition fault patterns:
  - tp\_slow
    - Used for load/unload and shift events.
  - tp\_fast
    - Used for clock sequential and capture events.



```

timeplate tp_fast =
  force_pi 0 ;
  measure_po 5 ;
  pulse CLOCK 10 5;
  period 20 ;
end;

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

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

6-19 • Design-for-Test: Scan and ATPG :  
Creating High Quality Patterns at Low Cost

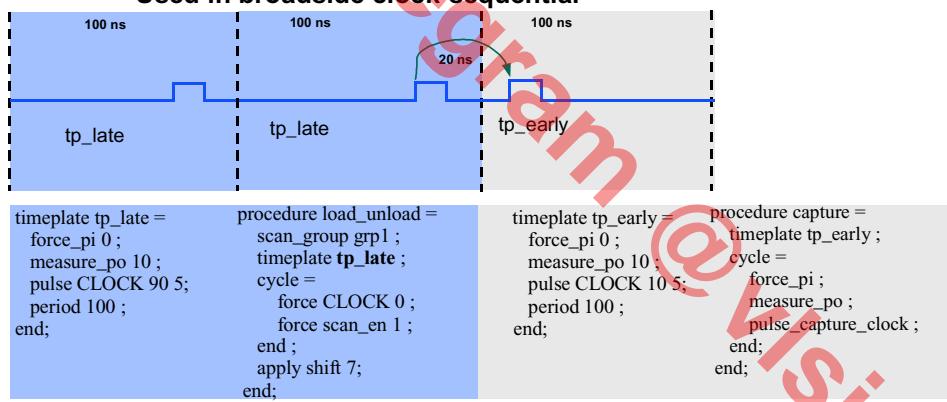
Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Timing for At-Speed Test (Cont.)

### Timing for At-Speed Test (Cont.)

- ◆ FastScan uses the following timeplates for skewed clocks
  - tp\_early
    - Used in launch-shift capture
    - Used in broadside capture
  - tp\_late
    - Used in launch-shift load/unload and shift
    - Used in broadside clock-sequential



6-20 • Design-for-Test: Scan and ATPG :  
Creating High Quality Patterns at Low Cost

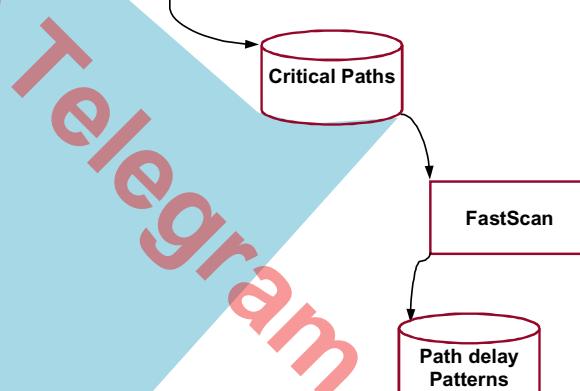
Copyright © 2003 Mentor Graphics Corporation

### Notes:

# Path Delay Pattern Flow

```
PATH "path0"
PIN /p5/pic1/inst_reg_8/DFF1/Q +;
PIN /p5/pic1/inst_reg_8/BUF1/IN +;
PIN /p5/pic1/inst_reg_8/BUF1/OUT +
PIN /p5/pic1/inst_reg_8/Q +;
PIN /p5/pic1/U888/B +;
PIN /p5/pic1/U888/UP1/IN1 +;
PIN /p5/pic1/U888/UP1/OUT +;
PIN /p5/pic1/U888/Z +;
PIN /p5/pic1/U953/B +;
PIN /p5/pic1/U953/UP1/IN1 +;
PIN /p5/pic1/U953/UP1/OUT -;
PIN /p5/pic1/U953/Z -;
PIN /p5/pic1/U966/P -;
PIN /p5/pic1/U966/UP3/IN1 -;
PIN /p5/pic1/U966/UP3/OUT -;
PIN /p5/pic1/U966/UP4/IN2 -;
PIN /p5/pic1/U966/UP4/OUT +;
PIN /p5/pic1/U966/UP5/IN +;
PIN /p5/pic1/U966/UP5/OUT +;
PIN /p5/pic1/U966/Z +;
PIN /p5/pic1/U1054/D +;
PIN /p5/pic1/U1054/UP1/IN3 +;
PIN /p5/pic1/U1054/UP1/OUT -;
PIN /p5/pic1/U1054/Z -;
PIN /p5/pic1/pe_reg_2/D -;
PIN /p5/pic1/pe_reg_2/INV1/IN -;
PIN /p5/pic1/pe_reg_2/INV1/OUT +;
PIN /p5/pic1/pe_reg_2/INV2/IN +;
PIN /p5/pic1/pe_reg_2/INV2/OUT -;
PIN /p5/pic1/pe_reg_2/MUX1/IN1 -;
PIN /p5/pic1/pe_reg_2/MUX1/OUT -;
PIN /p5/pic1/pe_reg_2/MUX2/IN0 -;
PIN /p5/pic1/pe_reg_2/MUX2/OUT -;
PIN /p5/pic1/pe_reg_2/DFF1/D1 -;
END ;
:
```

## Path Delay Pattern Flow



6-21 • Design-for-Test: Scan and ATPG :  
Creating High Quality Patterns at Low Cost

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Path Delay Patterns (Cont.)

### Path Delay Patterns

- ◆ Path delay patterns contain the following events:
  - Load scan chains
  - Force PIs
  - Pulse clock
  - Force PIs
  - Measure POs
  - Pulse clock
  - Unload scan chains

### Notes:

## Path Definition Files

### Path Definition Files

- ◆ In an external ASCII file, you must define all paths that you want tested in the test set.
- ◆ For each path, specify the following:
  - Path\_name
    - A unique name that defines the path.
  - Path\_definition
    - The topology of the path from launch to capture point as defined by an ordered list of pin pathnames.

### Notes:

## Creating Path Delay Patterns

### Creating Path Delay Patterns

- ♦ Do the following to create path delay patterns:

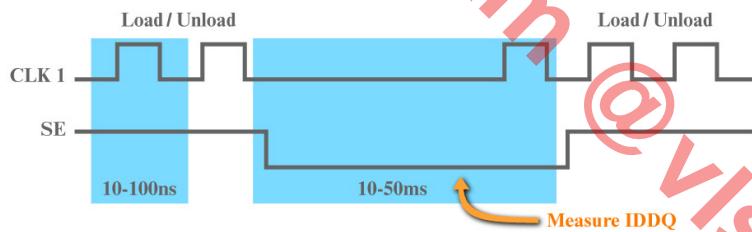
```
ATPG> SET PAttern Type -Sequential 2  
ATPG> SET FAult Type path_delay  
ATPG> LOAd PAths <path filename>  
ATPG> CREAtE PAtterns
```

### Notes:

## IDDQ Patterns

### IDDQ Patterns

- ◆ FastScan supports the pseudo stuck-at fault model for IDDQ testing.
- ◆ IDDQ patterns check for excessive leakage current.
- ◆ FastScan supports:
  - Selective IDDQ testing.
    - Selects IDDQ patterns from pre-existing generated patterns.
  - Supplemental IDDQ testing.
    - Creates supplemental IDDQ patterns.



6-25 • Design-for-Test: Scan and ATPG :  
Creating High Quality Patterns at Low Cost

Copyright © 2003 Mentor Graphics Corporation

### Notes:

# Creating IDDQ Patterns

## Creating IDDQ Patterns

- ♦ Do the following to create IDDQ patterns from an Existing Pattern Set:

```
ATPG> SET FAult Type IDDQ
ATPG> SET IDDQ Checks -All
ATPG> SET IDDQ Strobe -All
ATPG> SET PAttern Source External <stuckatpat.Ascii>
ATPG> SElect IDDQ Patterns -Max 10
ATPG> REPort STatistics
```

Use the "Noeliminate" switch for the SELECT IDDQ Patterns command; this will keep the stuck-at faults and 10 patterns in one set.

## Notes:

## Creating IDDQ Patterns (Cont.)

### Creating IDDQ Patterns (Cont.)

- ♦ Do the following to create IDDQ patterns from scratch:

```
SETUP> SET SYstem Mode ATPG  
ATPG> SET FAult Type IDDQ  
ATPG> SET IDDQ Checks -All  
ATPG> SET ATpg Compression ON  
ATPG> RUN  
ATPG> SElect IDDQ Patterns -Max 100  
ATPG> SAve PAtterns <pattern_name>
```

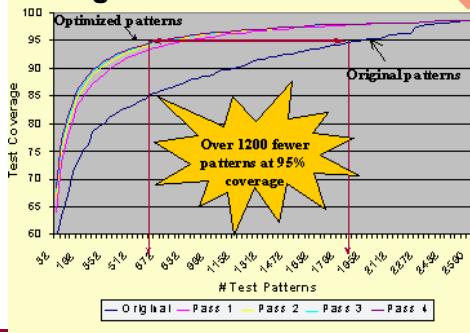
### Notes:

# Optimizing Quality and Cost

## Optimizing Quality and Cost

- ◆ Pattern optimization:
  - Patterns must fit within tester memory
  - Compression can reduce pattern count by a factor of 10
    - Dynamic
    - Static
    - Multiclock // activated when sequential depth>2
  - If patterns do not fit into tester memory, truncation is necessary
  - The ORDER PAttern command reorders the pattern set from highest fault detection to least fault detection

Included when using  
CREATE PAtterns



```
SETUP>SET SImulation Mode Comb -Depth 2
//Multiclock
.
.
ATPG>CREATE PAtterns -Compact
ATPG>ORDER PAttern 3
```

6-28 • Design-for-Test: Scan and ATPG :  
Creating High Quality Patterns at Low Cost

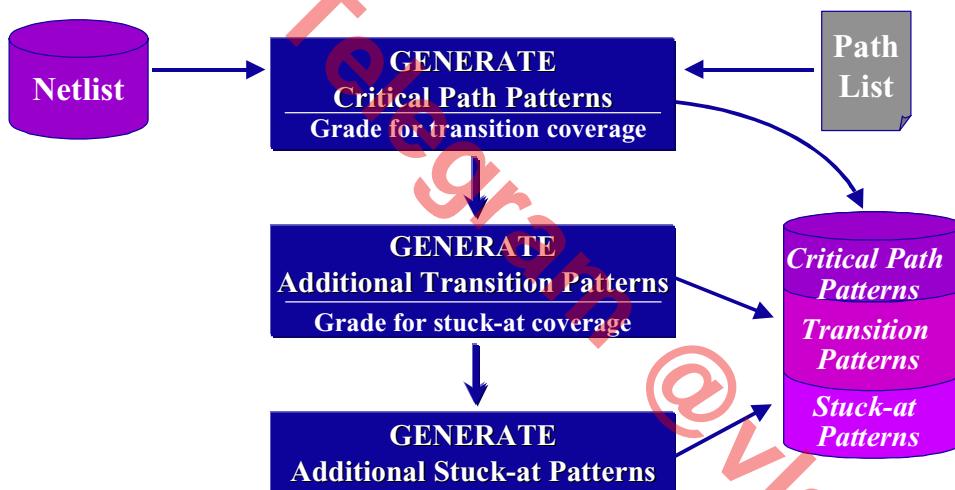
Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Optimizing Quality and Cost (Cont.)

### Optimizing Quality and Cost (Cont.)

- Always utilize compression and fault hierarchy when optimizing your test pattern set.



### Notes:

## ATE Characteristics

### ATE Characteristics

- ♦ Know the following ATE characteristics before you work on your design:
  - When using DFTAdvisor, consider the following:
    - The number of scan channels available to the tester
    - The number of clocks that the tester can support
  - When using FastScan, consider the following:
    - The frequency of the tester
    - The maximum number of timeplates that the tester can support (restrictions might effect pattern types: At-speed and clock PO)
    - The memory depth of the tester



6-30 • Design-for-Test: Scan and ATPG :  
Creating High Quality Patterns at Low Cost

Copyright © 2003 Mentor Graphics Corporation

### Notes:

## Lab: Creating High Quality Patterns at Low Cost

### Objectives

- Optimize patterns for quality and cost.
- Write a new enhanced procedure file.
- Modify timeplates:
  - Edit an existing timeplate.
  - Create a new timeplate.
- Apply new timing to patterns.
- Save patterns with a new enhanced procedure file:
  - Parallel.
  - Serial.
  - WGL.
- Create patterns and compress using multiple processes.
- Reorder patterns.
- Create fault models and fault grade:
  - Path delay.
  - Transition.
  - Stuck-at.
  - IDDQ.

## List of Exercises

- Exercise 14: Modifying Timeplates
- Exercise 15: Compression Techniques
- Exercise 16: Creating Fault Models and Fault Grading



Remember that for the exercises in this lab you use the libraries found in the **libraries\_5\_to\_6** directory.

Note

## Getting Started

1. Log in to your workstation if you are not already logged in.
2. Change to the `$ATPGNW/lab 6/exercise_14` directory.

```
shell> cd $ATPGNW/lab6/exercise_14
```

## Exercise 14: Modifying Timeplates

In this exercise, you optimize a test pattern set to have a low vector count, and then write a new enhanced procedure file. You then modify the timing of the enhanced procedure file by creating and editing a new timeplate. Next, you apply the new timing to ATPG and save those pattern sets with the newly modified enhanced procedure file.

1. Invoke FastScan on the following circuit:

Design: gate\_2001\_scan.v

Library: adk.atpg

Log file: results/ex\_14.log

2. The initial setup commands for the circuit set up and scan information for this design are in a dofile.

- a. Run the file gate\_2001\_scan.dofile.
3. The commands to generate clock sequential and RAM sequential test patterns with a low vector count are also in a dofile.
  - a. Run the file fs.do.
  - b. What commands does this file execute?

---

---

---

---

---

---

---

4. Fill in the following statistics using the information in the FastScan ATPG Run Statistics dialogue box and the **report statistics** command:

**Table 6-1. Test Pattern Generation Results**

**Table 6-2.**

| Type of Pattern | No. of Effective Patterns | No. of Sim'd Patterns | No. of detected | No. of Aborted | No. of AU | No. Redun. |
|-----------------|---------------------------|-----------------------|-----------------|----------------|-----------|------------|
|                 |                           |                       |                 |                |           |            |
|                 |                           |                       |                 |                |           |            |
|                 |                           |                       |                 |                |           |            |

**Table 6-3. Report Statistics ATPG****Table 6-4.**

| Fault Class          | # faults (coll.) | # faults (total) |
|----------------------|------------------|------------------|
| FU (full)            |                  |                  |
| DS (det_sim)         |                  |                  |
| DI (det_imp)         |                  |                  |
| TI (tied)            |                  |                  |
| RE (redundant)       |                  |                  |
| AU (atpg_untestable) |                  |                  |

**Table 6-5. Report Coverage/Effectiveness ATPG****Table 6-6.**

| Cov./Effect.         | # faults (coll.) | # faults (total) |
|----------------------|------------------|------------------|
| test_coverage        |                  |                  |
| fault_coverage       |                  |                  |
| atpg_effectiveness   |                  |                  |
| # test_patterns      |                  |                  |
| # basic_patterns     |                  |                  |
| # ram_seq_patterns   |                  |                  |
| # clock_seq_patterns |                  |                  |
| # simulated_patterns |                  |                  |
| CPU_time (secs)      |                  |                  |

5. Write a new test procedure file.

```
SETUP> write procfile results/ NEW.proc -full -replace
```

6. Open the NEW.proc test procedure file. Select **File > Open > Text File (Editable)** menu item.

Test procedure files define scan operations and their timing. Non-scan-related events include the remaining test pattern events not defined in the test procedure files. While ATPG does not require real timing information, Automatic Test Equipment (ATE) does require this information. Therefore, you must modify the test procedures files that you use for ATPG to include real timing information.

Thus, the patterns that you apply to ATE must specify the waveforms of each input, output, or bidirectional pin for each cycle.

7. In the following activity, you create a new timeplate and edit the clock times of this timeplate, and apply this new timing to the clock sequential procedure.

To modify the timing of the NEW.proc test procedure file, you must create a new timeplate. First, make a copy of timeplate gen\_tp1 as shown in [Figure 6-1](#).

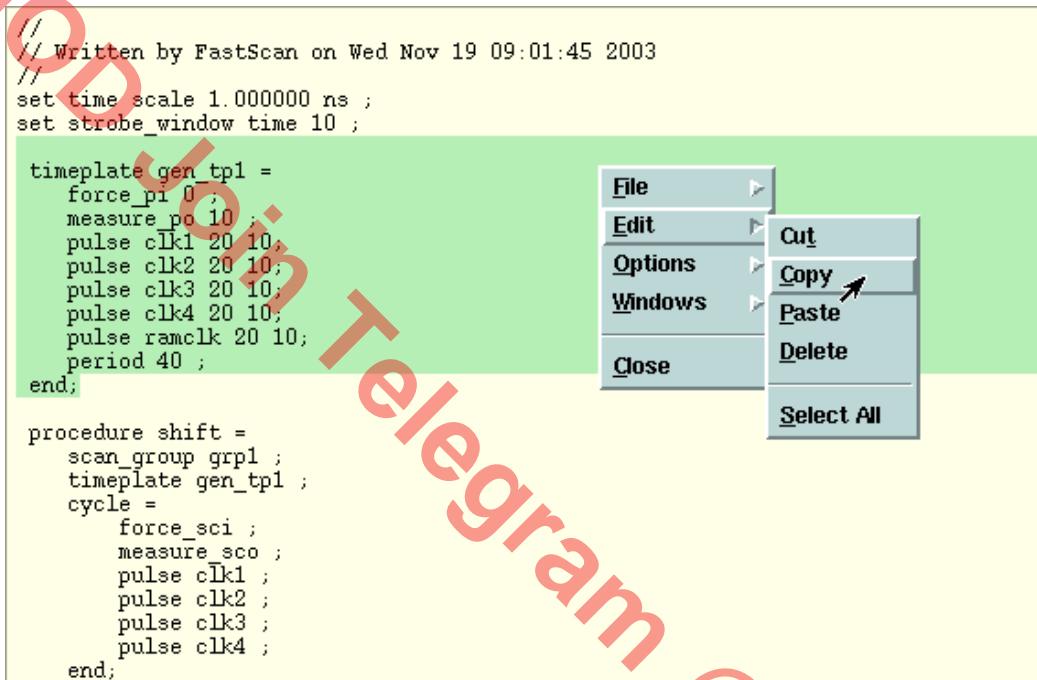


Figure 6-1. Copying the Timeplate

- Using the RMB select Copy, then Paste within the File Viewer.



A quicker way to copy and paste is to highlight the text you want to copy, then click on the middle mouse button.

Note

- Edit the new timeplate as shown in [Figure 6-2](#).

- Rename tp1 to tp2.

- b. Change all clock times for tp2 as shown.

**Figure 6-2. Renaming the Timeplate**

```

// Written by FastScan on Wed Nov 19 09:01:45 2003
//
set time scale 1.000000 ns ;
set strobe_window_time 10 ;

timeplate gen_tp1 =
  force_pi 0 ;
  measure_po 10 ;
  pulse clk1 20 10;
  pulse clk2 20 10;
  pulse clk3 20 10;
  pulse clk4 20 10;
  pulse ramclk 20 10;
  period 40 ;
end;

timeplate gen_tp2 =
  force_pi 0 ;
  measure_po 10 ;
  pulse clk1 15 5;
  pulse clk2 15 5;
  pulse clk3 15 5;
  pulse clk4 15 5;
  pulse ramclk 15 5;
  period 30 ;
end;

```

The name of the template is gen\_tp2 and its period is 30 ns. The various clocks also have different waveform timing information.

- c. Further down in the file you find a clock\_sequential procedure. As it is written, it uses the first timeplate.

Alter it to reference the second timeplate as shown in [Figure 6-3](#).

```

procedure clock_sequential =
  timeplate gen_tp2 ;
  cycle =
    force_pi ;
    pulse_capture_clock ;
    pulse_read_clock ;
    pulse_write_clock ;
  end;
end;

```

**Figure 6-3. Editing Timeplate**

- i. Rename timeplate gen\_tp1 to gen\_tp2 as shown in [Figure 6-3](#).

- d. Save the edited file as results/NEW2.proc. Make sure to overwrite any existing files.
- e. Close the File Viewer.

The new timing information was added into the new test procedure file that you saved.

Next, you save patterns with the new timing information from the gen\_tp2 timeplate. This will effect patterns that use clock sequential procedures.

9. Using the Save Patterns... button in the Button pane, save the patterns to a file with the following characteristics:

File name: results/testpat\_par.v

Pattern format: Verilog

Save Scan Cell Data in: Parallel

Enhanced Procedure File: results/NEW2.proc

Tests to be Saved: Chain & Scan

- i. Overwrite any existing files of the same name.
- ii. Leave the defaults for everything else.

- Look at the Session Transcript window. What information is displayed as the file is being written?

- 
- 
- 
10. Using the Save Patterns... button in the Button pane, save the patterns to a file with the following characteristics:

File name: results/testpat\_ser.v  
Pattern format: Verilog  
Save Scan Cell Data in: Serial  
Enhanced Procedure File: results/NEW2.proc  
Tests to be Saved: Chain & Scan

Number of Patterns to Sample Per Pattern Type:3

- i. Overwrite any existing files of the same name.
- ii. Leave the defaults for everything else.

By setting the Number of Patterns to Sample per Pattern Type to three you have reduced CPU processing time.

How many patterns are actually sampled?

- Look at the Session Transcript window. What information is displayed as the file is being written?

---

---

---

11. Using the Save Patterns... button in the Button pane, save the patterns to a file with the following characteristics:

File name: results/testpat\_ser.WGL  
Pattern format: WGL  
Save Scan Cell Data in: Serial  
Enhanced Procedure File: results/NEW2.proc

Tests to be Saved:

Chain & Scan

- i. Overwrite any existing files of the same name.
  - ii. Leave the defaults for everything else.
  - Look at the Session Transcript window. What information is displayed as the file is being written?
- 
- 
- 

12. Using the Save Patterns... button in the Button pane, save the patterns to a file with the following characteristics:

File name: results/testpat\_par.WGL

Pattern format: WGL

Save Scan Cell Data in: Parallel

Enhanced Procedure File: results/NEW2.proc

Tests to be Saved: Chain & Scan

- i. Overwrite any existing files of the same name.
  - ii. Leave the defaults for everything else.
  - Look at the Session Transcript window. What information is displayed as the file is being written?
- 
- 
-

13. Select **File > Open > Text File (Read Only)** to look at the files you have written.
  - a. In each file verify that the gen\_tp2 timeplate is used, particularly in the clock\_sequential patterns.
  - b. When you have finished, close the File Viewer.
14. Exit FastScan.

## Exercise 15: Compression Techniques

In this exercise, you create patterns and compress them to achieve a lower vector count. You compress an initial pattern set using multiple compression techniques, observing the vector count and run time for each process such as:

- Static
- Dynamic
- Multiclock

You compress patterns using the **create patterns** command and use the **order patterns** command to reorder the test pattern set.

## Getting Started

1. Log in to your workstation if you are not already logged in.
2. Change to the **\$ATPGNW/lab 6/exercise\_15** directory.

```
shell> cd $ATPGNW/lab6/exercise_15
```

3. Invoke FastScan on the following circuit:

Design: gate\_2001\_scan.v

Library: adk.atpg

Log file: results/ex\_15.log

You could invoke the tool on the design from the command line and bypass the Invocation dialogue box.

What command do you use? \_\_\_\_\_

4. The initial setup commands for circuit set up and scan information for this design are in a dofile.

- a. Run the file gate\_2001\_scan.dofile.
- b. What do the commands in the file do?

---

---

---

---

5. Go to ATPG mode, ignoring any DRC warning messages. You should now be in ATPG system mode.

6. Generate random and deterministic patterns for stuck-at-faults.

7. Fill in the following statistics using the information in the FastScan ATPG Run Statistics dialogue box and the **report statistics** command. When you have finished, DO NOT close the FastScan ATPG Run Statistics window:

**Table 6-7. Test Pattern Generation Results****Table 6-8.**

| Type of Pattern | No. of Effective Patterns | No. of Sim'd Patterns | No. of detected | No. of Aborted | No. of AU | No. Redun. |
|-----------------|---------------------------|-----------------------|-----------------|----------------|-----------|------------|
|                 |                           |                       |                 |                |           |            |
|                 |                           |                       |                 |                |           |            |
|                 |                           |                       |                 |                |           |            |

**Table 6-9. Report Statistics ATPG****Table 6-10.**

| Fault Class            | # faults (coll.) | # faults (total) |
|------------------------|------------------|------------------|
| FU (full)              |                  |                  |
| UO (unobserved)        |                  |                  |
| DS (det_sim)           |                  |                  |
| DI (det_imp)           |                  |                  |
| PU (posdet_untestable) |                  |                  |
| UU (unused)            |                  |                  |
| TI (tied)              |                  |                  |
| RE (redundant)         |                  |                  |
| AU (atpg_untestable)   |                  |                  |

**Table 6-11. Report Coverage/Effectiveness ATPG****Table 6-12.**

| Cov./Effect.   | # faults (coll.) | # faults (total) |
|----------------|------------------|------------------|
| test_coverage  |                  |                  |
| fault_coverage |                  |                  |

**Table 6-12.**

| Cov./Effect.         | # faults (coll.) | # faults (total) |
|----------------------|------------------|------------------|
| atpg_effectiveness   |                  |                  |
| # test_patterns      |                  |                  |
| # simulated_patterns |                  |                  |
| CPU_time (secs)      |                  |                  |

So far you have generated random and deterministic test patterns that give you an initial vector count and run time. In the following lab activities, you compress the initial pattern set using standard compression techniques. Then, you observe the vector count and run time for each sequentially applied process.

Because a tester requires a relatively long time to apply each scan pattern, it is important to create as small a test pattern set as possible while still maintaining the same test coverage. Static compression minimizes the number of test patterns by rerunning fault simulation first in reverse order and then in random order.

You now statically compress the initial pattern set and observe the results.

8. Statically compress the existing patterns using the Compress... button at the bottom of the FastScan ATPG Run Statistics dialogue box.
  - a. Set the **Number of Compression Passes** to 3.
  - b. Click the **Compress** button. The **FastScan Pattern Compression Statistics** dialogue box opens.
9. Observe and record the following results:
  - a. What is the initial pattern count?

---

  - b. What is the final pattern count?

- c. How many patterns have been removed?
- 

Fill in the following:

**Table 6-13. Report Coverage/Effectiveness ATPG**

**Table 6-14.**

| Cov./Effect.          | # faults (coll.) | # faults (total) |
|-----------------------|------------------|------------------|
| Total No. Faults (FU) |                  |                  |
| test_coverage         |                  |                  |
| fault_coverage        |                  |                  |
| atpg_effectiveness    |                  |                  |
| # test_patterns       |                  |                  |
| # simulated_patterns  |                  |                  |
| CPU_time (secs)       |                  |                  |

- d. What is the actual CPU run time for the static compression? (*Hint: Subtract [Total\_CPU\_Time for step 7) from (Total\_CPU\_Time for step 9)]*) \_\_\_\_\_
- e. What do Pass #1 and Pass #3 have in common?  
 \_\_\_\_\_
- f. What type of compression is applied in Pass #2?  
 \_\_\_\_\_

During ATPG dynamic compression, FastScan generates single patterns that detect a multitude of faults. FastScan selects a target fault, determines

the pattern conditions necessary to detect the fault, then attempts to merge detection of other faults with the same pattern.

10. Next you dynamically compress the pattern set without multi\_clock\_capture compression.

Multi\_clock\_capture is an optional switch for compression that specifies that FastScan should use a compression algorithm capable of handling multiple clock designs. Multiple clock compression uses sequential ATPG. This is the default when there are multiple clocks in a design and the sequential depth is greater than one.

- a. Reset all detected faults to undetected, deleting the internally generated test set.

```
ATPG> reset state
```

This does not alter those faults classified as AU, but resets all others to undetected, thus improving the run's efficiency.

- b. Set ATPG compression.

```
ATPG> set atpg compression on -nomulti_clock_capture
```

- c. Generate random and deterministic patterns for stuck-at-faults using all defaults.

What command do you use to re-simulate existing patterns?  
Remember, the fault universe has already been set up.

- 
11. Display a detailed report of the design's simulation statistics.

**Table 6-15. est Pattern Generation Results****Table 6-16.**

| Run             | No. of Effective Patterns | No. of Sim'd Patterns | No. of detected | No. of Aborted | No. of AU | No. Redun. |
|-----------------|---------------------------|-----------------------|-----------------|----------------|-----------|------------|
| No multiclk     | _____                     | _____                 | _____           | _____          | _____     | _____      |
| W/ multiclk     | _____                     | _____                 | _____           | _____          | _____     | _____      |
| Create patterns | _____                     | _____                 | _____           | _____          | _____     | _____      |

| Run             | No. of Effective Patterns | No. of Sim'd Patterns | No. of detected | No. of Aborted | No. of AU | No. Redun. |
|-----------------|---------------------------|-----------------------|-----------------|----------------|-----------|------------|
| No multiclk     | _____                     | _____                 | _____           | _____          | _____     | _____      |
| W/ multiclk     | _____                     | _____                 | _____           | _____          | _____     | _____      |
| Create patterns | _____                     | _____                 | _____           | _____          | _____     | _____      |

| Run             | No. of Effective Patterns | No. of Sim'd Patterns | No. of detected | No. of Aborted | No. of AU | No. Redun. |
|-----------------|---------------------------|-----------------------|-----------------|----------------|-----------|------------|
| No multiclk     | _____                     | _____                 | _____           | _____          | _____     | _____      |
| W/ multiclk     | _____                     | _____                 | _____           | _____          | _____     | _____      |
| Create patterns | _____                     | _____                 | _____           | _____          | _____     | _____      |

**Table 6-17. Report Coverage/Effectiveness ATPG****Table 6-18.**

| Run                | Cov./Effect.     | # faults (coll.) | # faults (total) |
|--------------------|------------------|------------------|------------------|
| With no multiclock | Total No. Faults | _____            | _____            |
| With multiclock    | (FU)             | _____            | _____            |
| Create patterns    |                  | _____            | _____            |

**Table 6-18.**

| <b>Run</b>         | <b>Cov./Effect.</b>  | <b># faults<br/>(coll.)</b> | <b># faults<br/>(total)</b> |
|--------------------|----------------------|-----------------------------|-----------------------------|
| With no multiclock | test_coverage        | _____                       | _____                       |
| With multiclock    |                      | _____                       | _____                       |
| Create patterns    |                      | _____                       | _____                       |
| With no multiclock | fault_coverage       | _____                       | _____                       |
| With multiclock    |                      | _____                       | _____                       |
| Create patterns    |                      | _____                       | _____                       |
| With no multiclock | atpg_effectiveness   | _____                       | _____                       |
| With multiclock    |                      | _____                       | _____                       |
| Create patterns    |                      | _____                       | _____                       |
| With no multiclock | # test_patterns      | @vlsigodOfficial            | _____                       |
| With multiclock    |                      |                             | _____                       |
| Create patterns    |                      |                             | _____                       |
| With no multiclock | # simulated_patterns | @vlsigodOfficial            | _____                       |
| With multiclock    |                      |                             | _____                       |
| Create patterns    |                      |                             | _____                       |
| With no multiclock | CPU_time (secs)      | @vlsigodOfficial            | _____                       |
| With multiclock    |                      |                             | _____                       |
| Create patterns    |                      |                             | _____                       |

12. Next you dynamically compress the pattern set with multi\_clock\_capture.

- a. Reset all detected faults to undetected, deleting the internally generated test set.

What command do you use? \_\_\_\_\_

- b. Set ATPG compression.

What command do you use? \_\_\_\_\_

- c. Set simulation mode to a depth of 3.

ATPG> **set simulation mode combinational -depth 3**

- d. Generate random and deterministic patterns for stuck-at-faults using all defaults.

What command do you use to re-simulate existing patterns?

---



This could take around 15 minutes, depending upon your OS.

**Note**

13. Fill in the second line of statistics in the above two tables, using the **report statistics** command and the FastScan ATPG Run Statistics window.

What is the actual CPU run time for the static compression?

---

14. Next you dynamically compress the pattern set using the **create patterns** command.

- a. Reset all detected faults to undetected, deleting the internally generated test set.

What command do you use? \_\_\_\_\_

- b. Generate random and deterministic patterns for stuck-at-faults using all defaults.

ATPG> **create patterns**



This could take around 20 minutes, depending upon your OS.

**Note**

15. Fill in the third line of statistics in the above two tables, using the **report statistics** command and the FastScan ATPG Run Statistics window.

- a. How many basic\_patterns are there? \_\_\_\_\_
- b. How many clock\_sequential patterns are there? \_\_\_\_\_
- c. What is the actual CPU run time for the static compression?  
\_\_\_\_\_

So far you created a test pattern set and applied various compression techniques to achieve the lowest vector count with the highest test coverage. The best compression technique is achieved by using the **create patterns** command, and is the recommended method for creating and compressing patterns.

16. In the next activity, you use the **order patterns** command to reorder the current internal pattern set from highest fault detection to least fault detection.

The **order patterns** command rearranges the order of the internal set so that patterns that detect the most faults are first. This is useful when you need to:

- Truncate the test pattern set in order to fit it in the tester's memory.
- Detect failing chips earlier during test.

17. To order patterns, issue the command at the ATPG prompt:

ATPG> **order patterns 5**

The **order pattern** command supports stuck-at and transition fault models only.

In this case, the argument (5) indicates the number of passes FastScan takes through the data as it reorders the pattern set.

18. Consider the pattern runs and observe how the ordering has an effect on the faults by filling in the following table:

**Table 6-19. Order Patterns Results**

| Patterns Simulated | <u>Test Coverage</u> |        |        |        |        |
|--------------------|----------------------|--------|--------|--------|--------|
|                    | Pass 1               | Pass 2 | Pass 3 | Pass 4 | Pass 5 |
| 32                 |                      |        |        |        |        |
| 64                 |                      |        |        |        |        |
| 96                 |                      |        |        |        |        |
| 128                |                      |        |        |        |        |
| 160                |                      |        |        |        |        |
| 192                |                      |        |        |        |        |
| 224                |                      |        |        |        |        |
| 256                |                      |        |        |        |        |
| 288                |                      |        |        |        |        |
| 320                |                      |        |        |        |        |
| 352                |                      |        |        |        |        |
| 365                |                      |        |        |        |        |

19. At this point you have generated a compact pattern set in an optimal order. It is necessary to save the patterns as done in the previous exercise.

- a. What type of pattern would you save to archive the design, and/or to allow further work on the patterns at a later time?
- 
- b. What types of pattern would you save to allow simulation of scan tests, including the loading and unloading of patterns?
-

- c. What types of pattern would you save to simulate all the patterns?  
\_\_\_\_\_
- d. Give an example of one type of pattern that could be saved to be used on ATE:  
\_\_\_\_\_
- e. Save one or all of the pattern types as time permits.
20. Exit FastScan.

## Exercise 16: Creating Fault Models and Fault Grading

In this exercise, you create the following fault models:

- Path delay
- Transition
- Stuck-at
- IDDQ

As you create each fault model you save each pattern type and you fault grade that pattern into the next fault model. By fault grading your patterns, you create one high quality test pattern set.

## Getting Started

1. Log in to your workstation if you are not already logged in.
2. Change to the \$ATPGNW/lab 6/exercise\_16 directory.

```
shell> cd $ATPGNW/lab6/exercise_16
```

3. Invoke FastScan on the following circuit:

Design: gate\_2001\_scan.v

Library: adk.atpg

Log file: results/ex\_16.log

You could invoke the tool on the design from the command line and bypass the Invocation dialogue box.

What command do you use? \_\_\_\_\_

---

4. The initial setup commands for circuit set up and scan information for this design are in a dofile.

Run the file gate\_2001\_scan.dofile.

5. Go to ATPG mode, ignoring any DRC warning messages. You should now be in ATPG system mode.
6. Create path delay patterns:

In this activity, you create path delay patterns to detect timing related defects in circuit paths. The path delay test requires the at-speed application of two patterns: initialization and propagation, to detect delay faults in the critical path. These patterns test for lumped time delay—sum of time delays that stack up. Path delay faults do not have localized fault sites. Rather, they are associated with the AC testing of critical paths.

Path delay patterns detect partially conducting transistors and diffusion problems.

Path delay faults are identified by path topology.

Path topology is a user-defined path that describes the critical path from the launch point to the capture point as defined by an ordered list of pin pathnames.

The launch point is either a primary input or a state element. The capture point is either a primary output or a state element. State elements used for

launch or capture points are either scan elements or non-scan elements that qualify for clock-sequential handling.

A path definition file defines the paths for which you want patterns generated. FastScan can load in a path definition file and store it as an internal path list.

a. Set up the Fault Universe:

- i. Model: Path Delay
- ii. Setup for model: Load Path Definitions From a File:

pathfile\_1

Pattern Sequential depth = 2

You can set up the Fault Universe (Customize) using commands or dialogue boxes. After you finish, look at the Session Transcript window.

- iii. Click on the Run button in the Button pane. When the Fault List Required dialogue box opens, select Add all Faults.

What commands are executed?

---

---

---

- b. Fill in the following statistics using the information in the FastScan ATPG Run Statistics dialogue box and the **report statistics** command:

**Table 6-20. Test Pattern Generation Results****Table 6-21.**

| Run                              | No. of Effective Patterns | No. of Sim'd Patterns | No. of detected | No. of Aborted | No. of AU | No. Redun. |
|----------------------------------|---------------------------|-----------------------|-----------------|----------------|-----------|------------|
| Path_delay                       |                           |                       |                 |                |           |            |
| Path_delay graded for transition |                           |                       |                 |                |           |            |
| Transition Patterns              |                           |                       |                 |                |           |            |
| Transition graded for stuck-at   |                           |                       |                 |                |           |            |
| Stuck-at patterns                |                           |                       |                 |                |           |            |
| Stuck-at graded for IDDQ         |                           |                       |                 |                |           |            |

**Table 6-22. Report Coverage/Effectiveness****Table 6-23.**

| Run                              | Total No. of Full Faults (FU) | test cover. | fault cover. | atpg effective. | # test pat. | # clock seq. | # sim. pat. |
|----------------------------------|-------------------------------|-------------|--------------|-----------------|-------------|--------------|-------------|
| Path_delay                       |                               |             |              |                 |             |              |             |
| Path_delay graded for transition |                               |             |              |                 |             |              |             |

**Table 6-23.**

| Run                            | Total No. of Full Faults (FU) | test cover. | fault cover. | atpg effective. | # test pat. | # clock seq. | # sim. pat. |
|--------------------------------|-------------------------------|-------------|--------------|-----------------|-------------|--------------|-------------|
| Transition Patterns            |                               |             |              |                 |             |              |             |
| Transition graded for stuck-at |                               |             |              |                 |             |              |             |
| Stuck-at patterns              |                               |             |              |                 |             |              |             |
| Stuck-at graded for IDDQ       |                               |             |              |                 |             |              |             |

- c. Using the Save Patterns... button in the Button pane save the patterns to a file in ASCII format with the following characteristics:

File name: results/testpat\_path.ascii

Pattern format: ASCII

Save Scan Cell Data in: Parallel

- i. Overwrite any existing files of the same name.
- ii. Leave the defaults for everything else.

You now have created path delay patterns and saved them in ASCII format.

7. Fault grade the Path Delay patterns for transition faults.

Fault grading is the process of simulating a target vector against a good circuit description, and simultaneously a circuit description that contains a fault to detect a difference from the expected response.

In this activity, you grade the path delay patterns that detect transition faults for the transition fault model and create transition fault patterns.

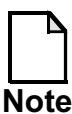
The transition faults model tests for gross pin-to-pin delay effects, which are the result of partially conducting transistors and interconnections. Because the transition fault model behaves as a stuck-at fault for a brief period of time, it requires the at-speed application of two test patterns for detection. The first vector is the initialization vector. It places the initial transition value at the fault site. The second vector is the transition vector. It is identical to the vector that would detect the associated stuck-at fault. It places the final transition value at the fault site. The transition fault model includes the slow-to rise and the slow-to fall fault models.

- a. Select the transition fault model.

```
ATPG> set fault type transition
```

- b. Set pattern source to external.

```
ATPG> set pattern source external \
./results/testpat_path.ascii -all_patterns
```



It is recommended to use the **-all\_patterns** switch to retain all pattern data.

**Note**

- c. Add faults to fault list.

What command do you use? \_\_\_\_\_

- d. Set the simulation mode to combinational, depth 2.

What command do you use? \_\_\_\_\_

- e. Fault grade the patterns then fill in the second line in the tables.

ATPG> **run**

8. Generate transition patterns for transition faults.

You fault graded the path delay patterns for the transition fault model. Next you add internally generated patterns into the test set in order to attain higher test coverage.

- a. Use the internally generated patterns.

ATPG> **set pattern source internal**

What warning message appears? \_\_\_\_\_

- b. Generate patterns, remembering that the fault universe was set up when we graded the faults.

What command do you use? \_\_\_\_\_

Note that even though the external patterns were removed, the fault universe still is aware of the coverage given by these external patterns, and the fault list does not contain the transition faults detected by the external patterns.

- c. Fill in the third line of the tables.
- d. Using the Save Patterns... button in the Button pane save the patterns to a file in ASCII format with the following characteristics:

File name: results/testpat\_transition.ascii

Pattern format: ASCII

Save Scan Cell Data in: Parallel

- i. Overwrite any existing files of the same name.
- ii. Leave the defaults for everything else.

You have now created transition patterns and saved them in ASCII format.

9. Fault grade transition patterns for stuck-at faults.

Now you are going to fault grade the transition patterns for the stuck-at fault model and create additional stuck-at test patterns to obtain better test coverage.

The stuck-at fault model models behavior that occurs at the gate level if the terminals of a gate are stuck at either high (stuck-at-1) or low (stuck-at-0) voltage levels. Most physical defects that occur exhibit behavior that makes a node appear to be stuck at power or ground.

- Select the stuck-at fault model.

ATPG> **set fault type stuck**

What warning messages appear?

---

---

- Set the pattern source to external since we use the patterns we just saved.

What command do you use? \_\_\_\_\_

- Add faults to the fault list.

- Set the simulation mode to combinational, depth 2

- Fault grade the patterns.

What command do you use? \_\_\_\_\_

- Fill in the fourth line in the tables.

10. Generate stuck-at patterns for stuck-at-faults.

You fault graded the transition patterns for the stuck-at-fault model. Next you add internally generated patterns into the test set to attain higher test coverage.

- a. Use the internally generated patterns.

What command do you use? \_\_\_\_\_

- b. Generate patterns, remembering that the fault universe was set up when we graded the patterns.

Note that even though the external patterns were removed, the fault universe still is aware of the coverage given by these external patterns, and the fault list does not contain the transition faults detected by the external patterns.

- c. Fill in the fifth line of the tables.
- d. Using the Save Patterns... button in the Button pane save the patterns to a file in ASCII format with the following characteristics:

File name: results/testpat\_stuck.ascii

Pattern format: ASCII

Save Scan Cell Data in: Parallel

- i. Overwrite any existing files of the same name.
- ii. Leave the defaults for everything else.

You have now created stuck-at patterns and saved them in ASCII format.

## 11. Fault grade stuck-at patterns for IDDQ faults and create IDDQ patterns.

IDDQ testing measures device current and rejects a part if the measured current is over a threshold level. Because IDDQ testing is costly and impractical to monitor current for every test vector in the set, a small efficient set of patterns can typically be selected or generated for IDDQ

testing. IDDQ testing provides a means of detecting CMOS transistor stuck-on conditions and bridge defects.

- a. Select the stuck-at fault model.

What command do you use? \_\_\_\_\_

- b. Set pattern source to external.

What command do you use? \_\_\_\_\_

- c. Add faults to the fault list.

- d. Set the simulation mode to combinational, depth 2.

- e. Set IDDQ checks.

```
ATPG> set Iddq checks -int_float -atpg
```

For CMOS circuits with pull-up or pull-down resistors or tri-state buffers, the good circuit should have a nearly zero IDDQ current. FastScan allows you to specify various IDDQ measurement checks to ensure that the good circuit does not raise IDDQ current during the measurement.

In this example, we create IDDQ patterns while checking that internal buses are not floating during an IDDQ measure, and requires deterministic ATPG to ensure that this is true.

Rarely do you have to check for all of them, so it is recommended that you select the appropriate checks.

- f. Set IDDQ strobe.

```
ATPG> set Iddq strobe -all
```

Specifies which fault grading patterns will perform IDDQ measures during simulation.

- g. Select IDDQ patterns -max 10.

```
ATPG> select iddq patterns -max 10
```

This command selects the patterns that most effectively detect IDDQ faults. The **-threshold** switch sets the number of IDDQ measures. This can take a long time.

The FastScan IDDQ Pattern Selection Statistics dialogue box opens.

- h. Fill in the sixth line of the tables.
- i. Using the Save Patterns... button in the Button pane save the patterns to a file in ASCII format with the following characteristics:

File name: results/testpat\_iddq.ascii

Pattern format: ASCII

Save Scan Cell Data in: Parallel

- i. Overwrite any existing files of the same name.
- ii. Leave the defaults for everything else.

You have now created iddq patterns and saved them in ASCII format.

12. Exit FastScan.

## Test Your Knowledge

1. Why must test procedure files include real timing information?

---

2. List the steps you would take to create a new timeplate.

---

3. What is the recommended command to use when creating and compressing patterns?

---

4. Why use the Order Patterns command?

---

5. What happens when you use the Reset State command?

---

6. What is a path definition file?

---

7. What do transition fault patterns detect?

---

8. Can you reorder IDDQ fault patterns?

---

## Lab Summary

Now that you completed the Creating High Test Coverage lab, you should know how to do the following:

- Modify and create timeplates.
- Save patterns with a new enhanced procedure file.
- Create patterns and compress using multiple processes.
- Reorder files.
- Create fault models and fault grade:
  - Path delay.
  - Transition.
  - Stuck-at.
  - IDDQ.

## Module 7

# Advanced ATPG

### Objectives

Upon completion of this module, you will be able to:

- Use FastScan to identify and black box missing modules.
- Use DFTAdvisor to write a netlist with a black box definition.
- Use MacroTest to test memories.
- Use BSDArchitect to write a test procedure file.

## Module Topics

### Module Topics

- ◆ This module addresses the following topics:
  - Black boxes
  - Testing embedded blocks
  - Boundary scan and other complex initialization issues
  - Top-up ATPG
  - Diagnostics

### Notes:

# Black Boxes

## Black Boxes

```

// FastScan v8.2003_3.10  Fri Jun 8 13:13:39 PDT 2003
// Copyright (c) Mentor Graphics Corporation, 1992-2003, All Rights Reserved.
// UNPUBLISHED, LICENSED SOFTWARE.
// CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
// PROPERTY OF MENTOR GRAPHICS CORP OR ITS LICENSORS.
//
// USE OF THIS SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS.
//
//
// Mentor Graphics software executing under Sun SPARC SunOS.
// 32 bit version
//
// Compiling library ...
// Reading Verilog Netlist ...
// Reading Verilog file top.v
// Finished reading file top.v
// Warning: Following modules are undefined:
// BU110
// TDN1J
// DPL61
// AN220
// Use "add black box -auto" to treat as black boxes
// command: set sys m atpg
// Error: Instance of undefined model found, check black
// box warnings and use Add Black Box -auto to
// make these default black box models
//
// command: report blac box -undefined
// Undefined Modules:
// BU110
// TDN1J
// DPL61
// AN220
//
// command: ADD BLack Box -Auto
...

```

- ◆ Allow analysis of incomplete designs
  - Isolates analog blocks
  - Isolates proprietary IP

- ◆ Warns if an undefined module is detected
  - The undefined module is not black boxed

**Use ADD Black Box -Auto command to black box all undefined modules**

## Notes:

# Black Box Examples

## Black Box Examples

- ◆ Example to create a black box for module *core* with tie value of 0

```
SETUP> ADD Black Box -module core 0
```

- ◆ Change I/O behavior of black boxes or to make an existing black box into a module:

```
SETUP> ADD Black Box -instance core1 -pin pin1 1
```

```
...
```

```
SETUP> DElete Black Box -module core 0
```

```
SETUP> REPort Black Box -all
```

- ◆ Write a netlist with black box definitions:

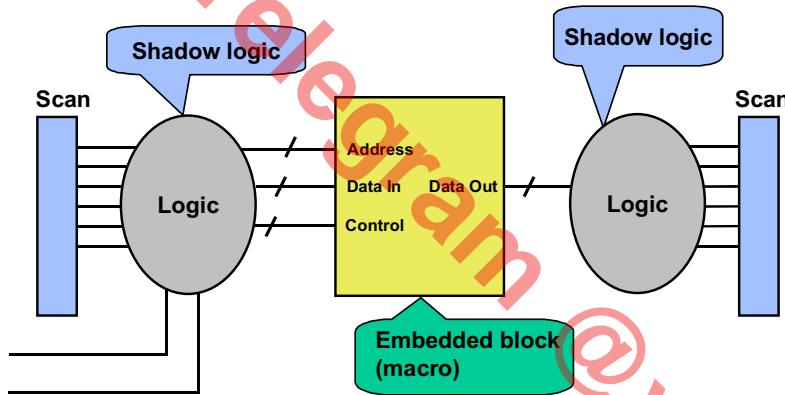
```
SETUP> WRite NETlist <file_name> -user_setup
```

## Notes:

# Testing Embedded Blocks

## Testing Embedded Blocks

- ◆ Testing embedded blocks introduces the following issues:
  - Testing challenges for blocks
  - Testing challenges of logic around the block (shadow logic)



## Notes:

## Testing Embedded Blocks (Cont.)

### Testing Embedded Blocks (Cont.)

- ◆ The following ATPG techniques are used to test embedded memories:
  - MacroTest is a FastScan feature that converts an existing pattern file into scan patterns.
  - Memory BIST adds built-in self-test logic.
  - Memory BIST bypass enables the testing of shadow logic.
  - RAM-sequential and multi-load patterns test through memory to target shadow logic.

### Notes:

# Testing Embedded Memories: MacroTest

## Testing Embedded Memories: MacroTest

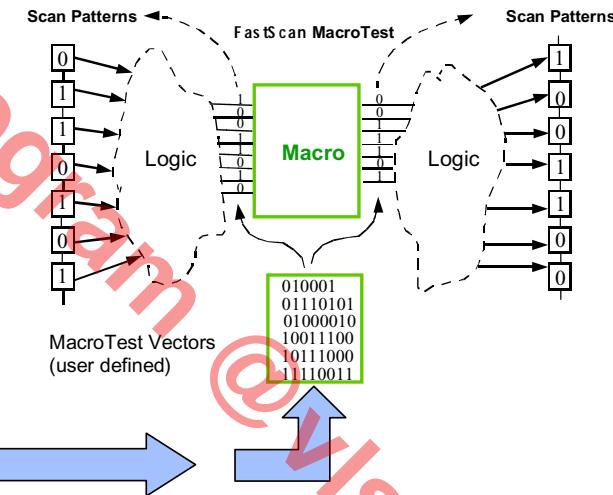
```

// Macrotest pattern file for ram
//
// model picdram (clk, address, we, din, dout) (
//   input(clk, we)
//   input(address) (array = 6:0)
//   input(din) (array = 7:0)
//   output(dout) (array = 7:0)

//      c a      d      d
//      l d      w i    o
//      k d      e n    t
P 0000000 1 0000000 XXXXXXXX
P 0000001 1 0000000 XXXXXXXX
P 0000010 1 0000000 XXXXXXXX
P 0000011 1 0000000 XXXXXXXX
P 0000100 1 0000000 XXXXXXXX
P 0000101 1 0000000 XXXXXXXX
P 0000110 1 0000000 XXXXXXXX
P 0000111 1 0000000 XXXXXXXX
P 0000000 0 XXXXXXXX LLLLLLLL
P 0000000 1 1111111 XXXXXXXX
P 0000000 0 XXXXXXXX HHHHHHHH
P 0000001 0 XXXXXXXX LLLLLLLL
:

```

- MacroTest converts each functional pattern into a scan pattern.



## Notes:

## Testing Embedded Memories: MacroTest (Cont.)

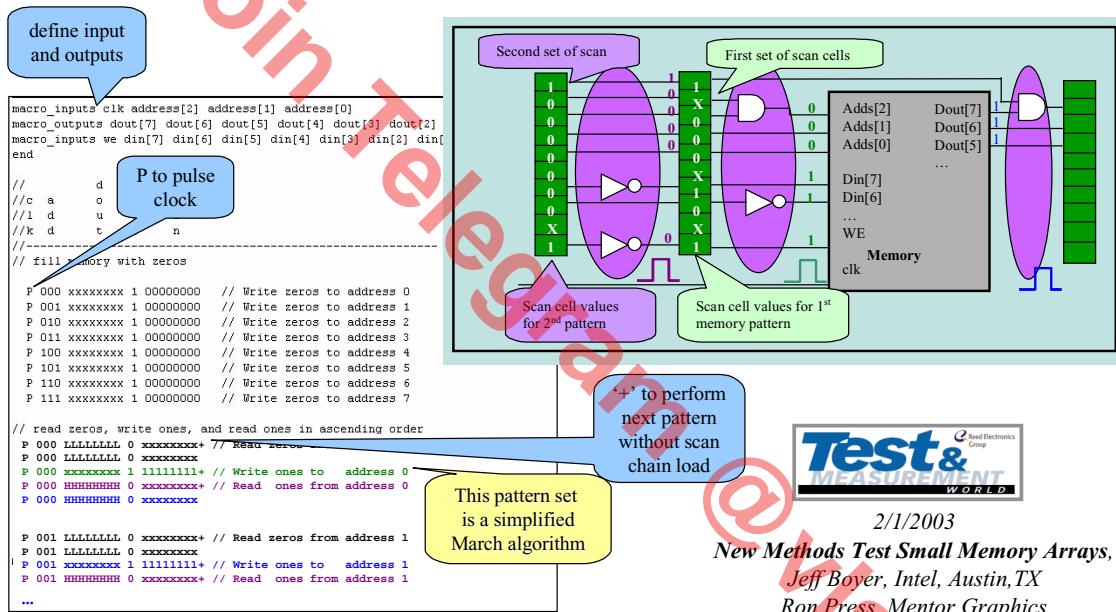
### Testing Embedded Memories: MacroTest (Cont.)

- ♦ The following apply to MacroTest:
  - Converts a pre-existing pattern set for the macro and delivers it at the IC level
    - No additional logic required
    - No extra routing
    - No area impact
  - Used with any embedded macro with digital I/Os
  - Patterns randomly filled and fault simulated to reduce pattern count
  - Has advanced debugging capabilities that report pattern translation problems
  - Not targeted for macros that contain internal scan

### Notes:

# At-Speed MacroTest

## At-Speed MacroTest



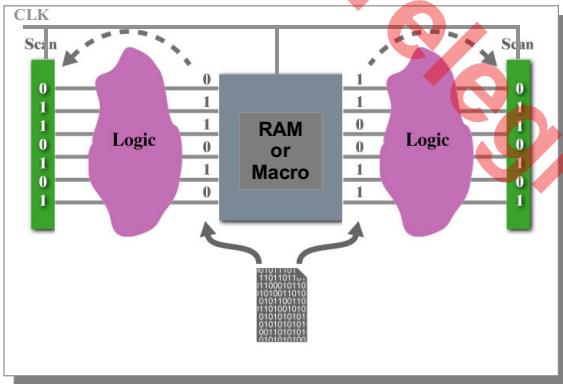
7-9 • Design-for-Test: Scan and ATPG:  
Advanced ATPG

Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Testing Embedded Memories: Synchronous MacroTest

## Testing Embedded Memories : Synchronous MacroTest



- ♦ The following apply to synchronous MacroTest ...
  - Allows for the testing of synchronous memories.
  - Uses the following circuit configuration:
    - A single clock is connected to the macro (read and write pin) and is shared with the scan chain.
    - A separate write enable.
  - Applies several macro patterns without reloading the scan chain (pipe line)

## Notes:

## Testing Embedded Memories: Synchronous MacroTest (Cont.)

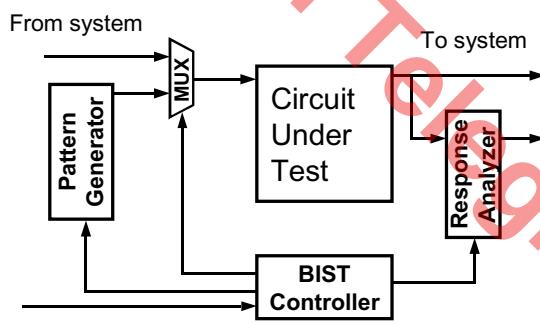
### Testing Embedded Memories : Synchronous MacroTest (Cont.)

- ◆ Writes using a one-cycle pattern.
  - Data is stable during shift because write enable is off during shift.
- ◆ Reads using a two-cycle read/observe pattern.
  - First clock pulses the RAM's read enable (data comes out of the RAM).
  - Second clock pulse captures data into the scan chain before shifting changes the RAM's output values (no independent read enable).
- ◆ Note: MacroTest patterns cannot be reordered.
  - Prior to generating additional patterns, save patterns and fault list.

### Notes:

# Built-In Self-Test Basics

## Built-In Self-Test Basics



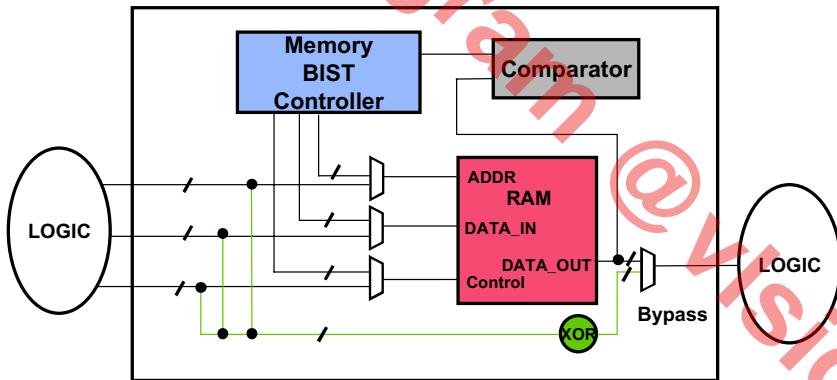
- ◆ BIST circuitry places the job of device testing inside the device itself.
- ◆ BIST circuitry generates its own stimulus and analyzes its own response.
- ◆ BIST has two modes of operation:
  - System mode - passes system data to the core, bypasses BIST circuitry.
  - Test mode - BIST circuitry runs self-test function.

## Notes:

# Testing Embedded Memories: Memory BIST

## Testing Embedded Memories: Memory BIST

- ◆ BIST provides a memory test solution for difficult-to-test RAM and ROM models in a design.
- ◆ Memory BIST circuitry generates patterns based on a variety of algorithms that focus on a particular type of circuitry or fault type.



7-13 • Design-for-Test: Scan and ATPG:  
Advanced ATPG

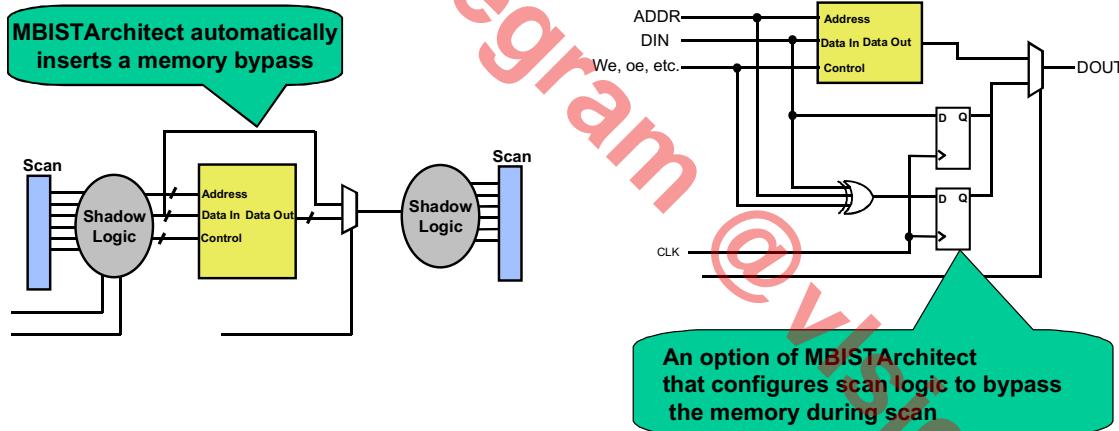
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Testing Embedded Memories: Memory BIST Bypass

## Testing Embedded Memories: Memory BIST Bypass

- MBISTArchitect uses the following bypass options to speed up ATPG:
  - Memory bypass
  - Memory bypass in scan mode



## Notes:

## Initialization Issues

---

### Initialization Issues

- ◆ Circuitry must be initialized prior to applying patterns.
  - Internally generated reset line.
  - Internally controlled test\_enable.
  - Gated clock control.
- ◆ This is accomplished by a test\_setup procedure.
- ◆ The test\_setup procedure is used to:
  - Set circuitry to a known initial state.
  - Initialize boundary scan: Joint Test Action Group (JTAG) circuitry.

### Notes:

## Initialization Issues (Cont.)

### Initialization Issues (Cont.)

- ◆ The test\_setup procedure does the following:
  - Sets non-scan state elements to known values
  - Sets pin constraints to maintain an initialized state
    - Use only *force* commands
- ◆ Applied only once - at the start of the pattern set
- ◆ Allowed only once for all scan groups

```
procedure test_setup =
    timeplate tp0;

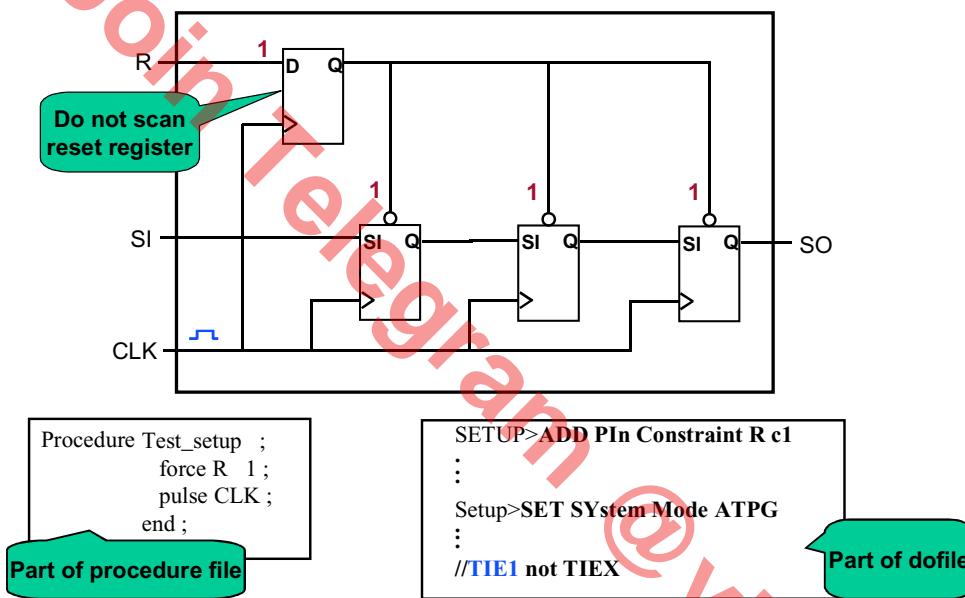
    // Apply reset procedure
    // Test cycle one

    cycle =
        force TMS 1;
        force TDI 0;
        force TRST 0;
        pulse TCK;
    end;
.
.
.
end;
```

### Notes:

# Initialization Example

## Initialization Example



## Notes:

# Auto Generate Test\_Setup

## Auto Generate Test\_Setup

- ◆ **FlexTest can automatically save out a simulation into a TEST\_SETUP procedure**  
ATPG> SAVe PATterns ... -test\_setup
- ◆ **Useful flow is to fault simulate functional initialization routines (boundary scan)**
  - Save fault list for FastScan ATPG
  - Save test\_setup for FastScan ATPG

## Notes:

## Boundary Scan Basics

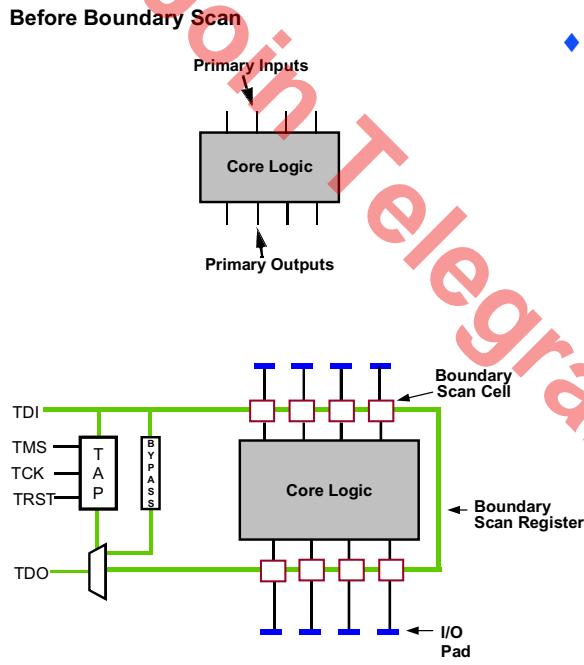
### Boundary Scan Basics

- ◆ Based on the IEEE 1149.1 standard
- ◆ Sometimes referred to as Joint Test Action Group (JTAG)
- ◆ Boundary scan is used to:
  - Facilitate testing of board interconnect circuitry
  - Access all chips on the board through a standard interface
  - Facilitate shifting of data to and from devices on the board
  - Provide access to internal chip test  
(controls BIST and custom functions)

### Notes:

# Boundary Scan Architecture

## Boundary Scan Architecture



- ◆ **Added after boundary scan insertion:**
- Primary inputs and outputs with boundary scan cells connected into a boundary scan register
- Boundary scan circuitry
- A test access port (TAP) controller that controls the operation of:
  - Boundary scan
  - Internal scan (optionally)

7-20 • Design-for-Test: Scan and ATPG:  
Advanced ATPG

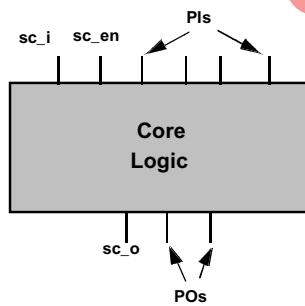
Copyright © 2003 Mentor Graphics Corporation

## Notes:

# Connecting Boundary Scan with Internal Scan

## Connecting Boundary Scan with Internal Scan

Before Boundary scan insertion



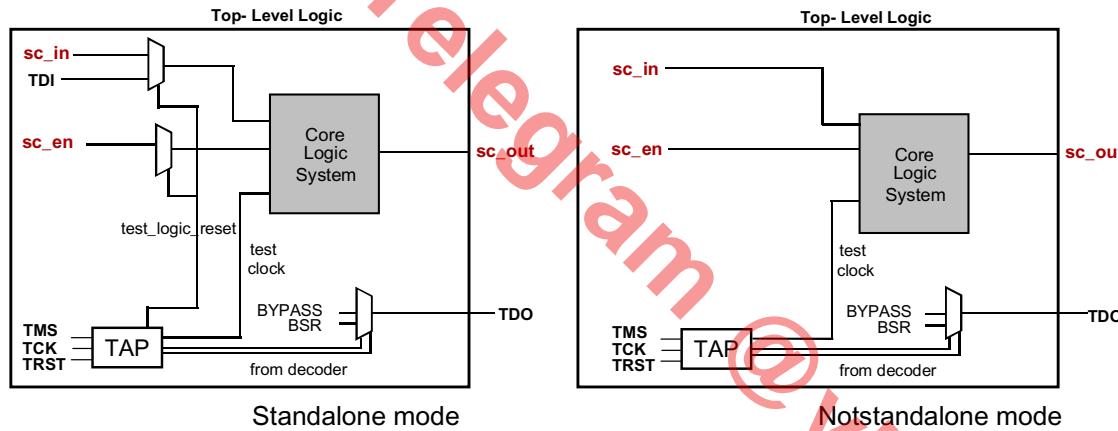
- ◆ BSDArchitect lets you choose several connection styles based on your preferred style for internal testing:
  - **Standalone mode**
    - Allows option to bypass the TAP controller which allows direct access to the scan pins at the top level
    - Adds extra circuitry
  - **Nostandalone mode**
    - Accesses internal scan only through the TAP

## Notes:

# Connecting Boundary Scan with Internal Scan (Cont.)

## Connecting Boundary Scan with Internal Scan (Cont.)

- After boundary scan insertion



## Notes:

# Accessing Internal Scan Instructions

## Accessing Internal Scan Instructions

- Internal scan circuitry is accessed through user-defined instructions.
  - INT\_SCAN
    - Connects Internal Scan Register between the TDI and TDO ports.



- MULT\_SCAN
  - Connects both the Boundary Scan Register (BSR) and the Internal Scan Register in series between TDI and TDO.



## Notes:

# Connecting Internal Scan to Boundary Scan Using BSDArchitect

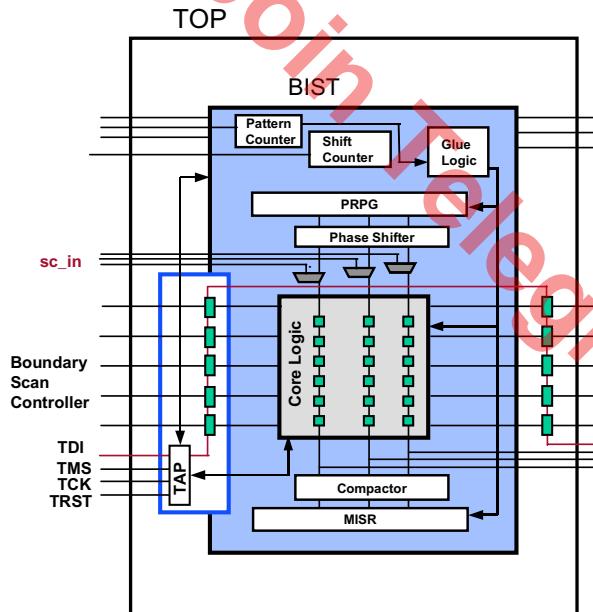
## Connecting Internal Scan to Boundary Scan Using BSDArchitect

- ◆ Set internal connection mode with **SET IScan Interface command.**
  - Standalone
  - Nostandalone
- ◆ Specify a name for the internal scan chain with **ADD Core Register command.**
- ◆ Define internal scan instruction with **ADD BScan Instruction command.**
  - INT\_SCAN
  - MULT\_SCAN
- ◆ Combine multiple scan chains into one scan chain with **CONNECT IScan Chains command.**
- ◆ Run boundary scan insertion.
- ◆ BSDArchitect automatically writes a test\_setup procedure used by FastScan.

## Notes:

# Top Up ATPG

## Top Up ATPG



- ◆ Used to supplement Logic BIST patterns
- ◆ An ATPG test strategy to improve ATPG efficiency
- ◆ Creates additional ATPG patterns that:
  - Test special circuitry: BIST, TAP controller, multiple cores, glue logic, and so on
  - Produce higher overall test coverage results

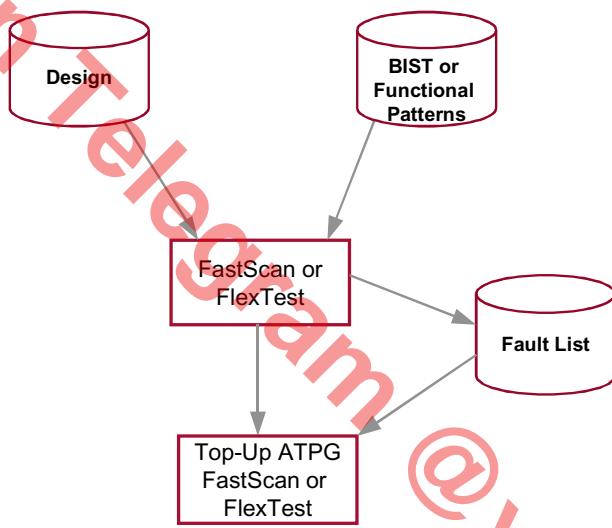
7-25 • Design-for-Test: Scan and ATPG:  
Advanced ATPG

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Top Up Patterns from BIST

### Top Up Patterns From BIST

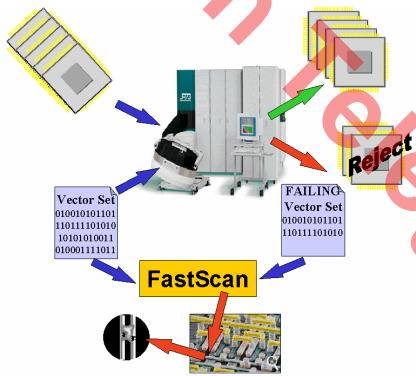


### Notes:

# Diagnostics

## Diagnostics

- ◆ Determine why a chip is faulty.
- ◆ Used to determine quality problems and to prevent their recurrence
- ◆ Involve both Automatic Test Equipment (ATE) and software-based diagnostics tools.
  - ATE records pattern number and observation points for all failing patterns (failure file).
  - FastScan diagnostics identifies the fault(s) associated with the defect.



7-27 • Design-for-Test: Scan and ATPG:  
Advanced ATPG

Copyright © 2003 Mentor Graphics Corporation

## Notes:

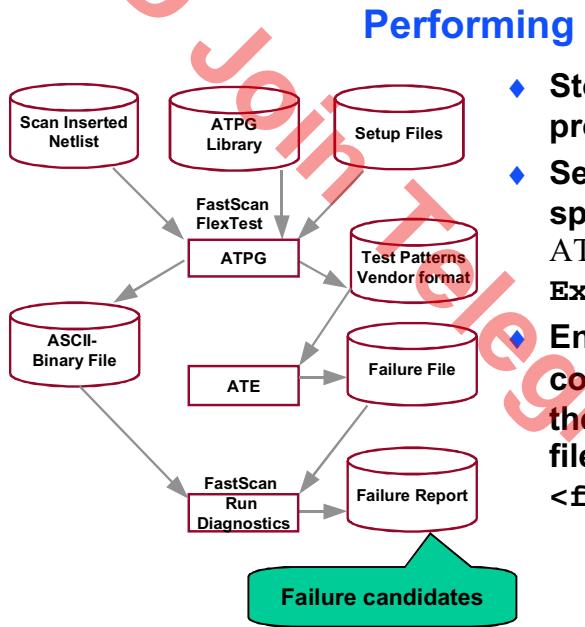
## DiagnosticsFastScan

### Diagnostics: FastScan

- ◆ Determines defect sites given:
  - Original stuck-at pattern set
  - Failing pattern set from ATE
- ◆ Better precision than standard fault dictionary approach
- ◆ Categorizes defects into:
  - Single fault sites
  - Multiple fault sites

### Notes:

## Performing a Diagnosis

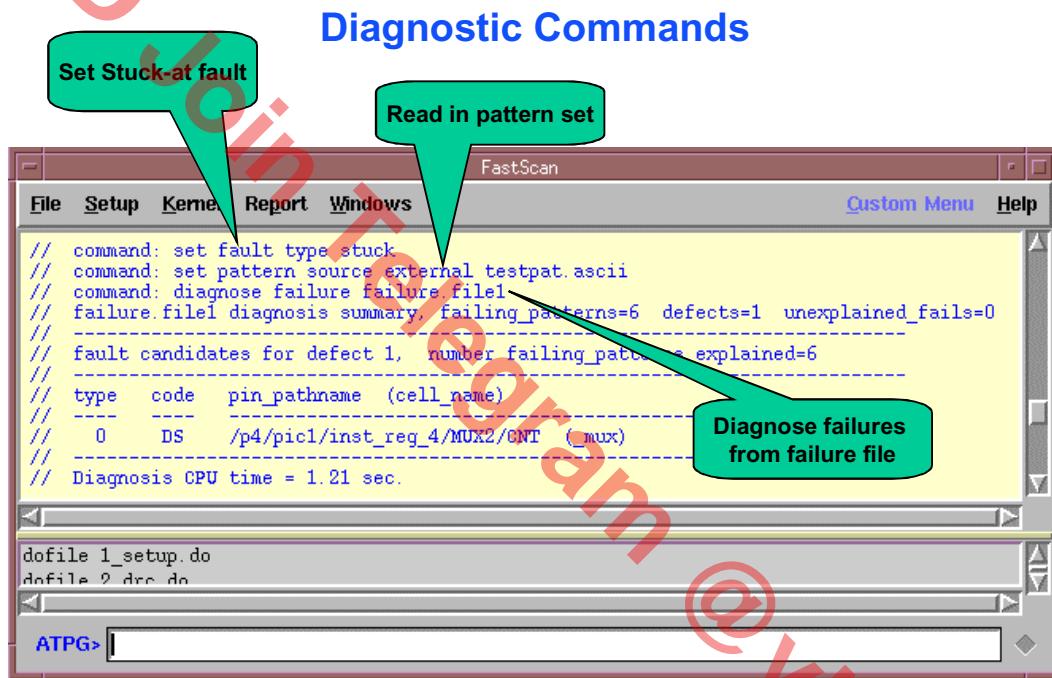


### Performing a Diagnosis

- ◆ Store the failing pattern data in the proper format.
- ◆ Set the pattern source to external and specify the test pattern file name:  
ATPG>SET PAttern Source External <pattern\_file>.
- ◆ Enter the DIAgnoSE FAilures command, identifies the failure file and the last pattern used from the pattern file: ATPG>DIAgnoSE FAilures <fails\_file> -Last 286.

### Notes:

# Diagnostic Commands



## Notes:

## Diagnostics Failure File

### Diagnostics: Failure File

```

// command: set fault type stuck
// command: set pattern source external testpat.ascii
// command: diagnose failure failure_file1
// failure_file1 diagnosis summary, failing_patterns=6 defects=1 unexplainedfails=0
// fault candidates for defect 1, number fail
// -----
// type code pin.pathname (cell.name)
// -----
// 0 DS /p4/pic1/inst_reg_4/MUX2/0
// Diagnosis CPU time = 1.21 sec.

dofile 1_setup.do
dofile 2_drc.do

ATPG>

```

Six Failing Patterns

Failing Pattern Number

Failing Scan Chain

Failing Scan Cell

7-31 • Design-for-Test: Scan and ATPG:  
Advanced ATPG

Copyright © 2003 Mentor Graphics Corporation

### Notes:

# Diagnostics Report

## Diagnostics Report

Defect Behavior is  
Stuck-at-0

The screenshot shows a window titled "FastScan" with a menu bar: Setup, Kernel, Report, Windows, Custom Menu, Help. The main text area displays the following command-line output:

```
command: set fault type stuck
command: set pattern source external testpat.ascii
command: diagnose failure failure_file1
failure_file1 diagnosis summary, failing_patterns=6 defects=1 unexplainedfails=0
// Fault candidates for defect 1, number failing_patterns_explained=6
// type code pin_pathname (cell_name)
// 0 DS /p4/pic1/inst_reg4/MUX2/0NT (.mux)
// Diagnosis CPU time = 1.21 sec.
```

Below the main text area, there is a smaller window titled "ATPG>" containing the text "dofile 1\_setup.do" and "dofile 2\_drc.do".

Hierarchical Path  
to Failing Part

Failing Part is a Mux

## Notes:

## Diagnostics Report (Cont.)

### Diagnostics Report (Cont.)

```
// command: set fault type stuck
// command: set pattern source external testpat.ascii
// command: diagnose failure failure_file1
// failure_file1 diagnosis summary, failing_patterns=6 defects=1 unexplainedfails=0
//
// fault candidates for defect 1, number failing_patterns_explained=6
//
// type code pin_pathname (cell_name)
// -----
// 0 DS /p4/pic1/inst_req_4/MUX2/CNT (.mux)
//
// Diagnosis CPU time = 1.21 sec.
```

dofile 1\_setup.do  
dofile 2\_drc.do

ATPG > [empty input field]

### Notes:

# Diagnostics Issues

## Diagnostics Issues

- ◆ Good diagnostics depend upon:
  - Completeness of the pattern set
    - More patterns the better
    - Save all failing patterns to a file
  - Accuracy of fault model(s)
  - Unsuitable for catastrophic fails (>5% flops failing) [Kinra, 1998]

## Notes:

## Lab: Advanced ATPG

### Objectives

- Identify and black box undefined modules.
- Use MacroTest to test undefined modules and RAM.
- Apply top-up ATPG to improve test coverage.
- Save patterns.
- Diagnose tester failures.
- Generate a boundary scan testbench in the FlexTest table format.
  - Use FlexTest to fault grade the boundary scan testbench.
  - Use FastScan to top-up ATPG.

### List of Exercises

- Exercise 17: MacroTest and Top-UP ATPG
- Exercise 18: Diagnosing Failure Files
- Exercise 19: Fault Grading Boundary Scan and Top-Up ATPG

### Getting Started

1. Log in to your workstation if you are not already logged in.
2. Change to the `$ATPGNW/lab 7/exercise_17` directory.

```
shell> cd $ATPGNW/lab7/exercise_17
```



Remember that for the exercises in this lab you use the libraries found in the **libraries\_7\_to\_9** directory.

## Exercise 17: MacroTest and Top-UP ATPG

In this exercise, you use FastScan to identify and black box undefined modules. Next, you use MacroTest to apply functional patterns to the undefined modules to generate scan patterns for that design circuitry. Then, you use FastScan to generate additional patterns to top-up ATPG.

1. Invoke FastScan on the following circuit:

Design: gate\_scan\_8.v

Library: atpglib

Log file: results/ex\_17.log

2. What three *types* of warnings appear in the Session Transcript window?

- a. \_\_\_\_\_
- b. \_\_\_\_\_
- c. \_\_\_\_\_

Which module(s) need to be defined as black boxes?

\_\_\_\_\_

3. The initial setup commands for circuit set up and scan information for the design are in the dofile atpg\_8\_scan.dofile.

Run the atpg\_8\_scan.dofile.

4. Black box undefined modules. For this exercise, use the following command:

```
SETUP> add black box -instance /p1/cordic1/AddY/Add2 \
-keep_boundary
```

5. Report black boxes to make sure the correct one(s) were defined.

```
SETUP> report black box -all
```

6. Set up the FastScan MacroTest utility.

```
SETUP> setup macrotest
```

The FastScan MacroTest utility is useful for applying specific patterns that test the internals of a macro. Although a macro is typically embedded logic or memory, it also can be a disjoint set of internal sites or a single block of hardware represented by an instance in HDL.

MacroTest reads test patterns from the functional test pattern file(s) you provide and converts them into scan-based manufacturing test patterns — thereby improving overall IC test quality.

7. Go to ATPG mode, ignoring any DRC warning messages.

8. Set up the Fault Universe:

Model: Single Stuck-At

Faults: Add Faults to ALL DESIGN OBJECTS

Turn On Fault Sampling: 10%

Leave the defaults for everything else.

Use dialogue boxes or commands to accomplish this. Whichever method you use, the commands executed should be the same. View them in the Session Transcript window.

What commands are executed?

i. \_\_\_\_\_

ii. \_\_\_\_\_

9. Convert functional patterns for the picdram block into scan patterns.



Note

You can use the hierarchy browser to find the *picdram* instance and automatically paste it into the Macrotest command.

```
ATPG> macrotest /p1/pic1/regs/picdram picdram.tbl
```

You can view the MacroTest patterns in the picdram.tbl file to observe their format. Observe the messages in the session transcript area.

How many patterns were converted and stored?

\_\_\_\_\_

10. Convert functional patterns for the Add2 black boxed instance (macro name CLA0) into scan patterns.

```
ATPG> macrotest /p1/cordic1/AddY/Add2 cla.tbl -NO_L_h
```

The *-NO\_L\_h* optional switch specifies that {0,1} will be used to specify {LO,HI} output values in the pattern files.

11. Observe the messages in the Session Transcript area.

How many patterns were converted and stored?

\_\_\_\_\_

Multiple macros can be tested in parallel. If you use the **-Multiple\_macro** switch, you must include in the macro\_filename all macros you want to be simultaneously tested. You should first test each macro individually in a separate run to ensure successful parallel testing.

12. Report statistics and fill in the lines (MacroTest) of the following tables:

**Table 7-1. Report Statistics ATPG****Table 7-2.**

| Fault Class            | # faults (coll.)                   | # faults (total)                   |
|------------------------|------------------------------------|------------------------------------|
| FU (full)              | MacroTest:<br>Top-Up:<br>Opt. run: |                                    |
| UC (uncontrolled)      | MacroTest:<br>Top-Up:<br>Opt. run: |                                    |
| UO (unobserved)        | MacroTest:<br>Top-Up:<br>Opt. run: |                                    |
| DS (det_sim)           | MacroTest:<br>Top-Up:<br>Opt. run: |                                    |
| DI (det_imp)           | MacroTest:<br>Top-Up:<br>Opt. run: |                                    |
| PU (posdet_untestable) | MacroTest:<br>Top-Up:<br>Opt. run: | MacroTest:<br>Top-Up:<br>Opt. run: |
| PT (posdet_testable)   | MacroTest:<br>Top-Up:<br>Opt. run: | MacroTest:<br>Top-Up:<br>Opt. run: |
| UU (unused)            | MacroTest:<br>Top-Up:<br>Opt. run: | MacroTest:<br>Top-Up:<br>Opt. run: |
| TI (tied)              | MacroTest:<br>Top-Up:<br>Opt. run: | MacroTest:<br>Top-Up:<br>Opt. run: |

**Table 7-2.**

| Fault Class          | # faults (coll.)                   | # faults (total)                   |
|----------------------|------------------------------------|------------------------------------|
| BL (blocked)         | MacroTest:<br>Top-Up:<br>Opt. run: | MacroTest:<br>Top-Up:<br>Opt. run: |
| RE (redundant)       | MacroTest:<br>Top-Up:<br>Opt. run: | MacroTest:<br>Top-Up:<br>Opt. run: |
| AU (atpg_untestable) | MacroTest:<br>Top-Up:<br>Opt. run: | MacroTest:<br>Top-Up:<br>Opt. run: |

**Table 7-3. Report Coverage/Effectiveness ATPG****Table 7-4.**

| Cov./Effect.       | # faults (coll.)                   | # faults (total)                   |
|--------------------|------------------------------------|------------------------------------|
| test_coverage      | MacroTest:<br>Top-Up:<br>Opt. run: | MacroTest:<br>Top-Up:<br>Opt. run: |
| fault_coverage     | MacroTest:<br>Top-Up:<br>Opt. run: | MacroTest:<br>Top-Up:<br>Opt. run: |
| atpg_effectiveness | MacroTest:<br>Top-Up:<br>Opt. run: | MacroTest:<br>Top-Up:<br>Opt. run: |
| # test_patterns    |                                    | MacroTest:<br>Top-Up:<br>Opt. run: |
| #basic_patterns    |                                    | MacroTest:<br>Top-Up:<br>Opt. run: |

**Table 7-4.**

| Cov./Effect.         | # faults (coll.) | # faults (total)                   |
|----------------------|------------------|------------------------------------|
| # macrotest_patterns |                  | MacroTest:<br>Top-Up:<br>Opt. run: |
| # simulated_patterns |                  | MacroTest:<br>Top-Up:<br>Opt. run: |
| CPU_time (secs)      |                  | MacroTest:<br>Top-Up:<br>Opt. run: |

13. Use the Save Patterns... button in the Button pane to save the patterns as an ASCII file with the following characteristics:

File name: results/macro.pat

Pattern format: ASCII, with default ASCII options

- i. Overwrite any existing files of the same name.
- ii. Leave the defaults for everything else.

14. Top-up ATPG patterns using existing options. Note that the Fault Universe is already set.

What command do you use? \_\_\_\_\_

Use the **report statistics** command and the ATPG Run Statistics window to fill in the second line of the tables above. (Top-Up)

15. Improve test coverage (optional activity).

- a. If you have time, experiment with improving test coverage and see how well you do. Suggest one approach to use:

- b. Use the **report statistics** command and the ATPG Run Statistics window to fill in the third line of the tables above. (Opt run)

16. Exit FastScan.

## Exercise 18: Diagnosing Failure Files

In this exercise, you read in two different failure files from a tester and use FastScan to diagnose the defect(s) resulting from these failures.

### Getting Started

1. Log in to your workstation if you are not already logged in.
2. Change to the *\$ATPGNW/lab 7/exercise\_18* directory.

```
shell> cd $ATPGNW/lab7/exercise_18
```

3. Invoke FastScan on the following circuit:

|           |                   |
|-----------|-------------------|
| Design:   | gate_scan_8.v     |
| Library:  | atpglib           |
| Log file: | results/ex_18.log |

4. The initial setup commands for circuit set up and scan information for the design are in the dofile atpg\_8\_scan.dofile.

Run the atpg\_8\_scan.dofile.

5. Black box undefined modules.

```
SETUP> add black box -auto
```

6. Report black boxes to make sure the correct one(s) were defined.

```
SETUP> report black box -all
```

What values are the ports tied to? \_\_\_\_\_

7. Go to Fault Simulation mode, ignoring any DRC warning messages. Do this by clicking on Done With Setup or by issuing a command.

What command is implemented? \_\_\_\_\_

8. Load the FastScan test pattern file by selecting the Pattern Source box in the Graphics pane. The Setup Pattern Source dialogue window opens.

Specify the Source of the Patterns: External Patterns

Source: fscan.pat

Pattern format: ASCII

Leave the defaults for the rest of the options.

What command does this set of options implement?

---

You use fault diagnostics on chips that fail during the application of the scan test patterns to identify the precise location of a fault(s).

You perform diagnosis by first collecting the full set of failing pattern data from the tester (failure file). FastScan uses this data during fault simulation to determine the set of faults whose simulated failures most closely match the actual failures.

Defects are categorized into single and multiple fault sites.

9. Load tester failure file.

FAULT> **diagnose failure fail1.list**

10. Observe the following messages in the session Transcript window and answer the following:

a. How many failing patterns are there? \_\_\_\_\_

b. How many defects are needed to explain these failing patterns?

- c. How many fault candidates are there? \_\_\_\_\_
- d. What are the pin\_pathnames and stuck-at values of the fault candidates?  
\_\_\_\_\_
- e. What are the names of the fault candidates?  
\_\_\_\_\_

11. There may be more than one tester failure file, as in this case. Load and diagnose the second tester failure file.

FAULT> **diagnose failure fail2.list**

12. Observe the following messages in the session Transcript window and answer the following:

- a. How many failing patterns are there? \_\_\_\_\_
- b. How many defects are needed to explain these failing patterns?  
\_\_\_\_\_
- c. How many fault candidates are there? \_\_\_\_\_
- d. What are the pin\_pathnames and stuck-at values of the fault candidates?  
\_\_\_\_\_
- e. What are the names of the fault candidates?  
\_\_\_\_\_

13. Sometimes you want to save a pattern file that has only a few patterns that are failing but passes the majority of them. If the fault sites are scan cells, you can mask the cells that fail and save the new pattern file.

In this step, we mask the three cells in the fault2.list file by adding cell constraints. You should open fault2.list and study it so you understand this next step.

- a. Go back to system mode SETUP.
- b. Type the following at the SETUP> prompt, or put them in a dofile that runs during setup:

```
SETUP> add cell cons chain6 169 OX  
SETUP> add cell cons chain6 171 OX  
SETUP> add cell cons chain6 172 OX
```

The argument OX tells FastScan to simulate the unloaded scan cell value as unknown (unobservable).

- c. Go to system mode ATPG.
  - i. Load the FastScan test pattern file by selecting the Pattern Source box in the Graphics pane. The Setup Pattern Source dialogue window opens.

Specify the Source of the Patterns: External Patterns

Source: fascan.pat

Pattern format: ASCII

Leave the defaults for the rest of the options.

- ii. Add all the faults.
  - iii. Run
  - iv. Save the new pattern set in ASCII format to a file called fscan2.pat.
14. Exit FastScan.

## Exercise 19: Fault Grading Boundary Scan and Top-Up ATPG

In this exercise, you use BSDArchitect to generate a default run of boundary scan logic generation and save the testbench in the FlexTest table format for fault grading. FlexTest will be used to fault grade the boundary scan logic testbench. Then you use FastScan to generate additional patterns to top-up ATPG.

### Getting Started

1. Log in to your workstation if you are not already logged in.
2. Change to the \$ATPGNW/lab 7/exercise\_19 directory.

```
shell> cd $ATPGNW/lab7/exercise_19
```

3. Invoke BSDArchitect.

```
shell> bsdarchitect gate_scan_8.v -verilog
```

4. Generate default boundary scan logic.
  - a. Click the Run button in the button pane.
  - b. If you are prompted to select an output format, select Verilog.

BSDArchitect is the Mentor Graphics boundary scan insertion tool. BSDArchitect creates and interconnects boundary scan logic at the RTL level, which is compliant with the IEEE 1149.1 and 1149.1a standards.

Boundary scan adds dedicated test circuitry to a chip. This circuitry provides board-level control of IC I/O pins as well as IC test structures for each chip with boundary scan. Boundary scan stitches control/observe cells at the input and output ports of chips into a boundary scan chain.

In this activity, BSDArchitect is used to generate a default run of boundary scan test logic for the Verilog design module.

5. Use the Save Results... button in the Button pane to save the patterns as a FlexTest file with the following characteristics:

|                 |                                                |
|-----------------|------------------------------------------------|
| a. Tab:         | Patterns                                       |
| Pattern format: | FlexTest                                       |
| Pattern File:   | flextest.pat                                   |
| b. Tab:         | Boundary Scan Data                             |
| Directory:      | /results                                       |
| i.              | Overwrite any existing files of the same name. |
| ii.             | Leave the defaults for everything else.        |

What commands are executed after these two saves?

---

---

---

---

---

---

---

- c. View the files in the lab7/exercise\_19/results directory:

Boundary scan-inserted netlist—bigchip\_bscan.v  
Testbench vectors in the FlexTest table format—flextest.pat

6. Exit BSDArchitect.

Normally at this time you would synthesize the boundary scan RTL logic created by BSDArchitect. Other sections of this lab utilize a synthesized netlist.

7. Invoke FlexTest on the following circuit:

|           |                   |
|-----------|-------------------|
| Design:   | incfile.v         |
| Library:  | atpglib           |
| Log file: | results/ex_19.log |

FlexTest is the Mentor Graphics all-purpose, sequential test generator. It is optimized for non-scan and partial-scan designs. FlexTest uses a general sequential ATPG algorithm called the BACK algorithm. The BACK algorithm uses the behavior of a target fault to predict which primary output (PO) to use as the fault effect observe point. Working from the selected PO, it sensitizes the path backward to the fault site. After creating a test sequence for the target fault, FlexTest uses a fault simulator to calculate all the faults for the test sequence.

8. Set up the circuit for FlexTest.

- Add clocks using the Clocks box in the Graphics pane.
  - Click the Manually Define button to define the following:

Primary Input = /tck; Off-state = 0

- Set redundant logic checking to Off.

```
SETUP> set redundancy identification off
```

This speeds up circuit analysis time for this evaluation.

9. Black box undefined modules.

This is the same command as the one used in FastScan. What is it?

10. Go to Fault Simulation mode, using either the command or by clicking Done With Setup in the Graphics pane. Ignore any DRC warning messages.

11. Set up the Fault Universe:

Model: Single Stuck-At

Faults: Add Faults to ALL DESIGN OBJECTS

Turn On Fault Sampling: 10%

Leave the defaults for everything else.

What commands are executed?

---

---



Fault sampling is used for analysis purposes only. It should never be used when you create production vectors.

**Note**

Next, FlexTest reads in the testbench that you saved in the FlexTest table format and fault grades the simulation vectors to produce an undetected faults list.

12. Load the test pattern file by selecting the Pattern Source box in the Graphics pane. The Setup Pattern Source dialogue window opens.

Specify the Source of the Patterns: External Patterns

Source: /results/flexttest.pat

Pattern format: table

Leave the defaults for the rest of the options.

Look at the Session Transcript window. What command is executed?

How many test cycles are there? \_\_\_\_\_ scan operations? \_\_\_\_\_  
iddq measurements? \_\_\_\_\_

13. Fault simulate the FlexTest table format file using all existing/default settings.
- Click on Fault Simulation in the graphics pane.
  - When the Use Existing Settings or Customize? dialogue box opens, click the Run With Existing Settings button.
    - When the FlexTest Run Options dialogue box opens, click Run.
    - The FlexTest Fault Simulation Run Statistics dialogue box opens. This simulation run takes several minutes.
    - Click the Dismiss button after the run.



**Note**

Several cycles simulate before fault detection occurs because fault detection does not occur until outputs are observed.

- Use the FlexTest Fault Simulation Run Statistics window and the **report statistics** command to fill in the following:

**Table 7-5. Circuit Statistics**

**Table 7-6.**

| Circuit Statistics        |  |
|---------------------------|--|
| # of primary inputs       |  |
| # of primary outputs      |  |
| # of library design cells |  |

**Table 7-6.**

| Circuit Statistics                    |  |
|---------------------------------------|--|
| # of library leaf cells               |  |
| # of netlist primitives               |  |
| # of combinational library primitives |  |
| # of sequential library primitives    |  |
| # of ram cells                        |  |
| # of strongly connected components    |  |
| # of simulation primitives            |  |

**Table 7-7. Fault List Statistics****Table 7-8.**

| Fault Class          | Uncollapsed | Collapsed |
|----------------------|-------------|-----------|
| Full (FU)            |             |           |
| Det_simulation (DS)  |             |           |
| Posdet_testable (PT) |             |           |
| Unused (UU)          |             |           |
| Tied (TI)            |             |           |
| Blocked (BL)         |             |           |
| Uninitializable (UI) |             |           |
| Atpg_untestable (AU) |             |           |
| Unobserved (UO)      |             |           |
| Uncontrolled (UC)    |             |           |

**Table 7-8.**

| Fault Class        | Uncollapsed | Collapsed |
|--------------------|-------------|-----------|
| Fault coverage     |             |           |
| Test coverage      |             |           |
| ATPG Effectiveness |             |           |

- d. How many test cycles were generated? \_\_\_\_\_
- e. How many test cycles were simulated? \_\_\_\_\_
- f. How much CPU time was used? \_\_\_\_\_
- g. How many faults were detected? \_\_\_\_\_ undetected? \_\_\_\_\_
14. Write a fault list.

```
FAULT> write faults results/flexttest.faults -replace
```

15. Optional Activity: Assessing why test coverage is low.
- a. If you have time, use the Hierarchy Browser to determine why the fault coverage is so low. Remember, the testbench exercises the boundary scan logic, not the test core.
- b. Why is the coverage so low? \_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_

16. Exit FlexTest.
17. Invoke FastScan on the following circuit:

Design: incfile.v

Library: atpglib  
 Log file: results/ex\_19\_fs.log

18. Complete the circuit setup using the following information. Use the command line or various dialogue boxes in the Graphics pane to add scan groups and chains.

- a. Scan Group:

Group Name: grp1

Test Procedure File: exercise\_19/atpg\_8.testproc

- b. Add the following Scan Chain definitions:

**Table 7-9.**

| Chain Name | Group | Scan_In   | Scan_out   |
|------------|-------|-----------|------------|
| chain1     | grp1  | /scan_in1 | /scan_out1 |
| chain2     | grp1  | /scan_in2 | /scan_out2 |
| chain3     | grp1  | /scan_in3 | /scan_out3 |
| chain4     | grp1  | /scan_in4 | /scan_out4 |
| chain5     | grp1  | /scan_in5 | /scan_out5 |
| chain6     | grp1  | /scan_in6 | /scan_out6 |
| chain7     | grp1  | /scan_in7 | /scan_out7 |
| chain8     | grp1  | /scan_in8 | /scan_out8 |

- c. Set up the RAM with the following information:

write controls = /ramclk Off-State = 0  
 read controls = /ramclk Off-State = 0

- d. Manually define the Clocks:

/clk1 Off-State = 0  
/clk2 Off-State = 0  
/clk3 Off-State = 0  
/clk4 Off-State = 0

- e. Constrain the Primary Inputs to constant 0:

/trst  
/tck

Why are these inputs held to 0?

---

- f. Black box undefined modules, using the **add black box -auto** command.
19. Go to ATPG mode, ignoring any DRC warning messages.

You now load the fault graded fault list from FlexTest into FastScan and run top-up ATPG.

20. Set up the Fault Universe.

- a. Click on Customize. The Setup Fault Universe dialogue box opens.

Fault Type/List tab: Load Existing Faults (Misc. Options area)

Load the Faults from the Following File: results/flexttest.faults

- b. Select Retain and Protect Faults in the Misc. Options dialogue window, then select Retain Each Fault's Current Classification.

What command is executed after you finish this step?

---

21. Dynamically compress patterns (remember this generates and compresses patterns at the same time) using the Compression... button in the Button pane or by typing the command at the command line prompt. In either case, the same command is executed.

What command is this? \_\_\_\_\_

This step takes about 15 minutes.

- Fill in the first line of the table. You may use the **report statistics** command to obtain the information you need.

**Table 7-10.**

| Design Block  | Total # of Faults | #test_patterns | test_cover-age | fault_coverage | atpg_effectiv. |
|---------------|-------------------|----------------|----------------|----------------|----------------|
| Entire Design |                   |                |                |                |                |
|               |                   |                |                |                |                |
|               |                   |                |                |                |                |
|               |                   |                |                |                |                |

- Click on the Report Statistics... button in the Button pane to open the Report Statistics dialogue box.
  - Select — Show Statistics For: Hierarchical Instance at Path and browse to the top level blocks in the design using the Design Hierarchy Browser.
  - Investigate the test coverage of the top level design blocks.
  - Fill in the next three lines of the table with the details reported on the top three hierarchical blocks.

- iv. Provide an explanation of the coverage percentages reported on these hierarchical blocks:
- 
- 
- 

22. Exit FastScan.

## Test Your Knowledge

1. What command is used to activate the FastScan MacroTest utility?

---

2. Does MacroTest create the functional test patterns?

---

3. When testing multiple macros, what is the recommended procedure?

---

4. What is a failure file?

---

5. Why did you save the boundary scan testbench in the FlexTest table format?

---

## Lab Summary

Now that you have completed the Advanced ATPG lab, you should know how to do the following:

- Use MacroTest to test undefined modules and RAM.
- Diagnose failure files.
- Use BSDArchitect to generate default boundary scan.
- Use BSDArchitect to save simulation vectors in the FlexTest table format.
- Use FlexTest to fault grade and save simulation vectors.
- Use FastScan to generate additional patterns to top-up ATPG.

## Module 8

# Troubleshooting DRC and Simulation Mismatch

### Objectives

Upon completion of this module, you will be able to:

- Debug design Rules Checking (DRC).
- Debug timing issues.
- debug simulation mismatches.

## Module Topics

### Module Topics

This module addresses the following topics:

- The hierarchy browser
- Design block coverage
- ATPG untestable (AU) faults
- Fault categories
- DFTInsight
- Fault analysis

### Notes:

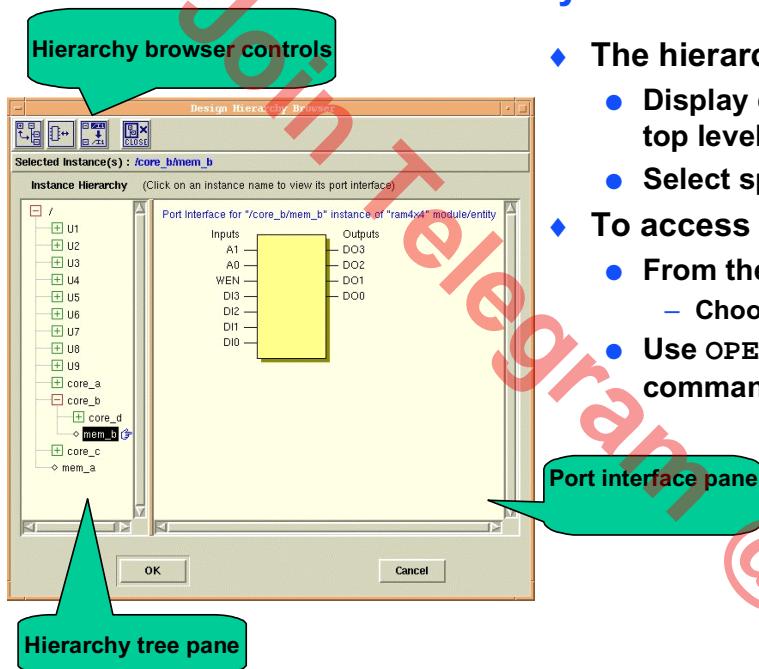
## Troubleshooting Areas of Low Coverage

### Troubleshooting Areas of Low Coverage

- ◆ Troubleshoot areas of low coverage:
  - Assess the problem
    - Determine the larger design blocks
    - Determine which blocks are reporting low coverage
    - Determine untestable fault categories
  - Determine why ATPG classified faults as untestable
    - Determine why faults are classified ATPG Untestable (AU)
    - Determine why faults are classified as Unobserved (UO) and Uncontrolled (UC)
  - Analyze the design fault by fault
    - Use DFTInsight or applicable commands and options
  - Debug the design

### Notes:

# Hierarchy Browser



- ◆ The hierarchy browser is used to:
  - Display design hierarchy from the top level to the lowest gate
  - Select specific instances and pins
  
- ◆ To access the browser:
  - From the GUI,
    - Choose design browser > show
  - Use OPEN Hierarchy Browser command

8-4 • Design-for-Test: Scan and ATPG:  
Troubleshooting Areas of Low Coverage

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Assessing the Problem

### Assessing the Problem

- ♦ Determine which blocks are reporting low coverage.
  - Use REPort SStatistics command.

```
SETUP>REPort SStatistics -Instance <instance.pathname>
```

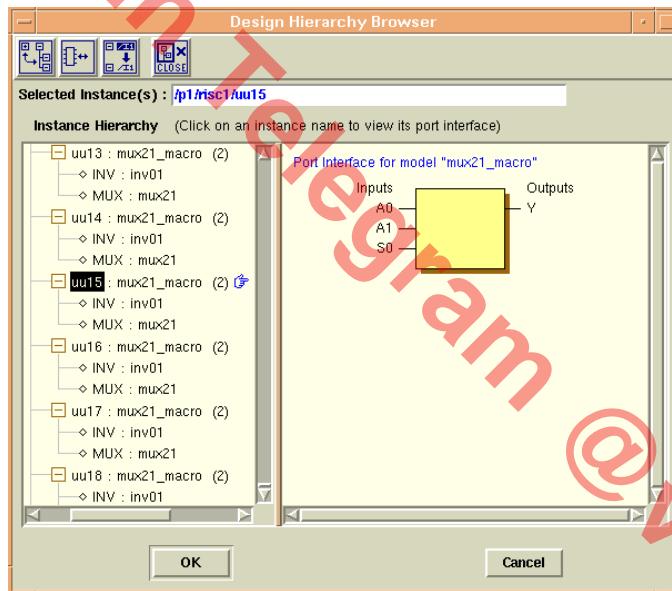


### Notes:

## Assessing the Problem (Cont.)

### Assessing the Problem (Cont.)

- ◆ Determine blocks with coverage issues.
  - Use the hierarchy browser to display instance and pathname.



8-6 • Design-for-Test: Scan and ATPG:  
Troubleshooting Areas of Low Coverage

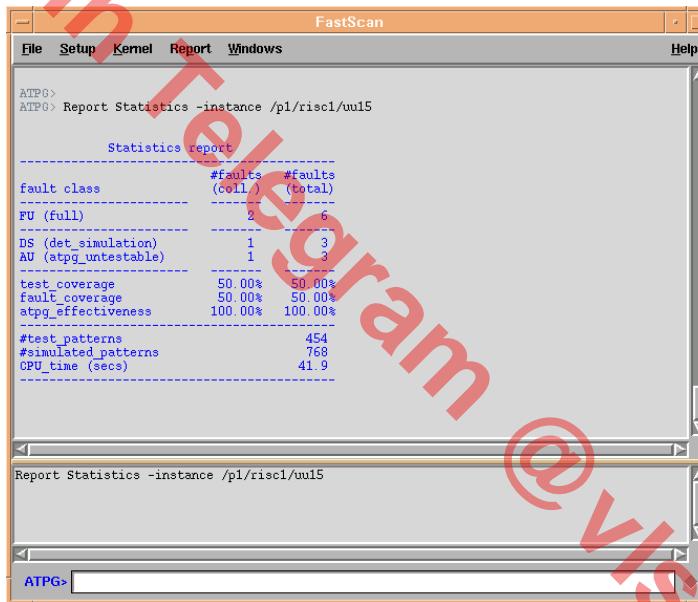
Copyright © 2003 Mentor Graphics Corporation

### Notes:

## Assessing the Problem (Cont.)

### Assessing the Problem (Cont.)

- ♦ Use the REPort SStatistics command to determine instance coverage and fault size.



The screenshot shows the FastScan software interface with a "Statistics report" window open. The report displays fault class distribution, test coverage, fault coverage, and ATPG effectiveness. The CPU time is also listed.

| fault class          | #faults (coll.) | #faults (total) |
|----------------------|-----------------|-----------------|
| FU (full)            | 2               | 6               |
| DS (det_simulation)  | 1               | 3               |
| AU (atpg_untestable) | 1               | 3               |

|  | test_coverage | fault_coverage | atpg_effectiveness |
|--|---------------|----------------|--------------------|
|  | 50.00%        | 50.00%         | 100.00%            |

|  | #test_patterns | #simulated_patterns | CPU_time (secs) |
|--|----------------|---------------------|-----------------|
|  | 454            | 768                 | 41.9            |

8-7 • Design-for-Test: Scan and ATPG:  
Troubleshooting Areas of Low Coverage

Copyright © 2003 Mentor Graphics Corporation

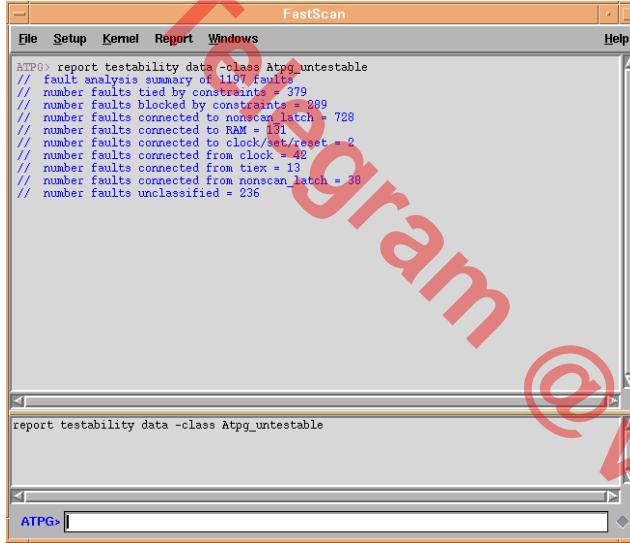
### Notes:

## Assessing the Problem (Cont.)

### Assessing the Problem (Cont.)

- ◆ Determine untestable fault categories:
  - Use REPort TEstability Data command

SETUP> REPort TEstability Data -Class AU



The screenshot shows the ATPG FastScan software interface. The main window title is "FastScan". The menu bar includes "File", "Setup", "Kernel", "Report", "Windows", and "Help". The "Report" menu is currently selected. The main pane displays a command-line report for "ATEPG report testability data -class Atpg\_untestable". The output shows fault analysis statistics:

```
ATEPG report testability data -class Atpg_untestable
// fault analysis summary of 1197 faults
// number faults tied by constraints = 379
// number faults blocked by constraints = 289
// number faults connected to nonscan latch = 728
// number faults connected to RAM = 131
// number faults connected to clock/set/reset = 2
// number faults connected from clock = 42
// number faults connected from tie = 13
// number faults connected from nonscan_latch = 38
// number faults unclassified = 236
```

8-8 • Design-for-Test: Scan and ATPG:  
Troubleshooting Areas of Low Coverage

Copyright © 2003 Mentor Graphics Corporation

### Notes:

## Faults Classified as ATPG Untestable

### Faults Classified as ATPG Untestable

- ◆ ATPG untestable faults (AU):
  - Typically, result of tied or blocked behavior, but not proven redundant.
    - Constraints prevent FastScan from generating a pattern to detect the fault.
  - An AU fault is testable because it has not been proven untestable, only undetected.

### Notes:

## Faults Classified as Undetectable

### Faults Classified as Undetectable

- ♦ Undetected faults (aborted faults) cannot be proven untestable or AU:
  - Uncontrollable (UC)
    - Faults that cannot be set to a known value at control point.
  - Unobservable (UO)
    - Faults that cannot be propagated to an observe point.
- ♦ Aborted faults are the result of exceeding an analysis limit.

### Notes:

## Addressing Aborted Faults

### Addressing Aborted Faults

- ◆ Addressing aborted faults:
  - Determine if many aborted faults exist

| Statistics report    |                    |                    |
|----------------------|--------------------|--------------------|
| fault class          | #faults<br>(coll.) | #faults<br>(total) |
| FU (full)            | 4280               | 10387              |
| UC (uncontrolled)    | 1                  | 1                  |
| UO (unobserved)      | 35                 | 77                 |
| DS (det_simulation)  | 2904               | 7395               |
| DI (det_implication) | 370                | 455                |
| PT (posdet_testable) | 1                  | 6                  |
| TI (tied)            | 30                 | 60                 |
| BL (blocked)         | 27                 | 56                 |
| RE (redundant)       | 59                 | 210                |
| AU (atpg_untestable) | 853                | 2127               |
| test_coverage        | 78.64%             | 78.05%             |
| fault_coverage       | 76.51%             | 75.60%             |
| atpg_effectiveness   | 99.14%             | 99.22%             |
| #test_patterns       |                    | 523                |
| #simulated_patterns  |                    | 864                |
| CPU_time (secs)      |                    | 25.5               |

### Notes:

## Addressing Aborted Faults (Cont.)

### Addressing Aborted Faults (Cont.)

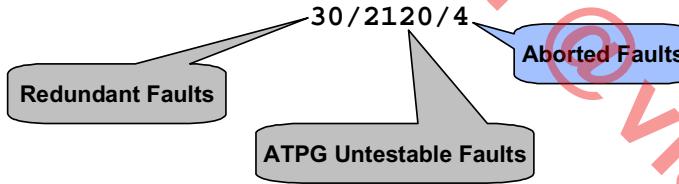
- Set a higher abort limit if there are many blocks

SETUP> SET Abort Limit 300

Use the number of reported aborted faults as the benchmark to determine abort limit

```
ATPG> run
// -----
// Simulation performed for #gates = 51199 #faults = 7696
// system mode = ATPG pattern source = internal patterns
//
// ----- #patterns test #faults #faults # eff. # test process
// simulated coverage in list detected patterns patterns CPU time
// begin random patterns: capture clock = /clk1, observe point = MASTER
// 32      10.84%    7039     657      14      14   1.15 sec
...
// begin random patterns: capture clock = none, observe point = MASTER
// deterministic ATPG invoked with abort limit = 30
//  ---  ---  ---  ---  --- 9.14 sec
18/2120/3
// 512      63.19%    1653     297      31      185   9.60 sec
//  ---  ---  ---  ---  --- 9.94 sec
30/2120/4
// 544      66.23%    1337     304      32      217   10.37 sec
```

The three additional fields are cumulative and broken down as follows:



### Notes:

## Bus Contention

### Bus Contention

- ◆ Bus contention causes Uncontrollable (UC) faults.
- ◆ FastScan generates patterns to detect UC faults.
  - If pattern simulates effectively, fault becomes Detected by Simulation (DS).
  - If pattern is rejected, fault remains UC and other patterns may detect it.
  - By default, patterns will be created without considering bus contention. If pattern simulation results in bus contention, then the pattern is rejected.

```
ATPG> report faults -class UNCONTROLLED
      1    UC    /p1/fpul/u4/U2711/Y
      0    EQ    /p1/fpul/u4/U2711/A1
      0    EQ    /p1/fpul/u4/U2711/A0
      0    EQ    /p1/fpul/u4/U3534/Y
      1    EQ    /p1/fpul/u4/U3534/A1
      1    EQ    /p1/fpul/u4/U3534/A0
      1    EQ    /p1/fpul/u4/U1302/Y
      ...
      ...
```

### Notes:

## Addressing Bus Contention

### Addressing Bus Contention

- ◆ **Avoid contention during pattern generation:**

SETUP> SET Contention Check ON -ATPG

- FastScan generates patterns that force buses to non-contention.
- If pattern does not detect fault, it becomes AU.

- ◆ **Analyze what caused the contention:**

SETUP> ANALyze Bus <bus\_gate> -Prevention -Exclusivity

- Prevention- ability of the tool to attain a state of non-contention.
- Exclusivity- allows only one driver to force a signal into a bus.

### Notes:

## Addressing Bus Contention: Types of Contention

### Addressing Bus Contention: Types of Contention

- ◆ Determine type of contention:
  - E4 violations:
    - Contention during procedures
    - Often due to unknown values at bidi pins
    - Usually fixed by force <bus> Z in test\_setup and load\_unload at event 0
  - E10 violations are contention after the scan chain is loaded
  - Use the SET DRC Handling E10 -ATPG command:
    - Determines buses that do not cause contention
    - Simplifies pattern generation to avoid bus contention

### Notes:

## Debugging Bus Contention

### Debugging Bus Contention

- ♦ The following transcript is an example of a dofile used for debugging bus contention:

```
ATPG> REPort DRC Rule E10
ATPG> ANALyze Bus <E10Violation_gate> (...arguments)
ATPG> SET GATE Report Constrain
ATPG> REPort Gate <E10Violation_gate>
```

### Notes:

## Fault-by-Fault AU Debugging: Report Testability Data Command

### Fault-by-Fault AU Debugging: Report Testability Data Command

- ◆ Use REPort TEStability Data command to check:
  - AU due to constraints or blockage
  - Connectivity to or from potential problems
  - TSD enables
  - Non-scan cells
  - Clock, set, and reset lines
  - Wire gates
  - RAMs

### Notes:

## Report Testability Data Command

### Report Testability Data Command

- ◆ REPort TEStability Data command reports the following types of data:
  - Tied
  - Blocked
  - Constrained Values
  - Forbidden Values
  - Constrained blockages

```
ATPG> report testability data -class Atpg_untestable
// fault analysis summary of 1197 faults
// number faults tied by constraints = 379
// number faults blocked by constraints = 289
// number faults connected to nonscan_latch = 728
// number faults connected to RAM = 131
// number faults connected to clock/set/reset = 2
// number faults connected from clock = 42
// number faults connected from tiex = 13
// number faults connected from nonscan_latch = 38
// number faults unclassified = 236
```

### Notes:

## TieX (D5)

---

### TieX (D5)

- ◆ A D5 violation occurs when memory elements are not identified as part of a scan chain.
- ◆ Non-scan elements are modeled as tie-X, unless set to a stable value.
- ◆ If too many tie-X gates, increase the clock sequential depth.

SETUP> SET PAttern Type -Sequential 2

### Notes:

# Fault-by-Fault AU Debugging: Set Gate Report Command

## Fault-by-Fault AU Debugging: Set Gate Report Command

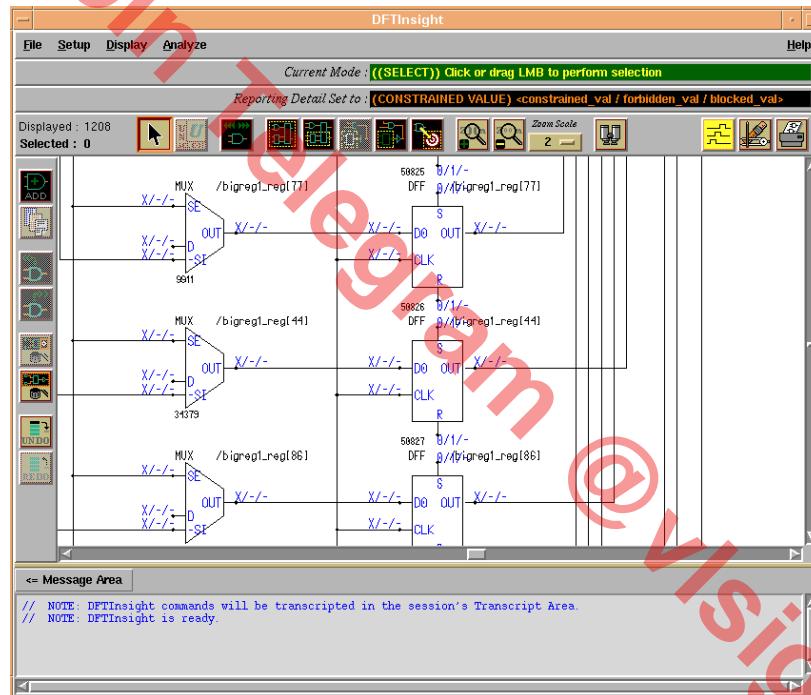
- ♦ Use SET GATE Report -COnstrain\_Value command to display constrained and forbidden values.
- ♦ The constrained value report provides three fields:
  - CV
  - FV
  - B



## Notes:

## Set Gate Report Command -Constrain\_Value

### Set Gate Report Command -Constrain\_Value



8-21 • Design-for-Test: Scan and ATPG:  
Troubleshooting Areas of Low Coverage

Copyright © 2003 Mentor Graphics Corporation

### Notes:

# Fault-by-Fault AU Debugging: Analyze Fault Command

## Fault-by-Fault AU Debugging: Analyze Fault Command

- ◆ Use ANALyze FAult command to identify why a fault is not detected.
- ◆ Performs fault analysis and displays circuitry in DFTInsight (-Display).
- ◆ Will generate and fault simulate a pattern trying to detect a specific fault.

```
ATPG> analyze fault /p1/pic1/U1839/B0 -stuck_at 0
// -----
// Fault analysis for /p1/pic1/U1839 (10267) input B0 (0) stuck at 0
//
// Current fault classification = AU (atpg_untestable)
// 1 potential observation points were identified:
//     Potential observe point: MASTER-data /p1/pic1/porta_reg[0] (49319).
//     Controllability justification was successful (data accessible using parallel_pattern 0).
//     Pattern type: Basic_scan.
// No potential detection path problems were identified.
// Test generation performed using all observable detection points.
//     Fault was detected at /p1/pic1/porta_reg[0]/ (49319) and fault effect
//     is propagated from fanin gate 13203 (data accessible using parallel_pattern 1)
//     Pattern type: Basic_scan.
```

## Notes:

# Fault-by-Fault AU Debugging: Report Test Stimulus Command

## Fault-by-Fault AU Debugging: Report Test Stimulus Command

- ◆ Use REPort TESt Stimulus command displays stimulus necessary for specified conditions:
  - Set
  - Write
  - Read
- ◆ Identifies how to sensitize scan chains blockage points

```
ATPG> repo gate 200
// /ix181 (200) AND
//      A1    I  (F:F)  192-/ix176/Y
//      A0    I  (F:F)  115-/modgen_and_6_ix5/Y
//      "OUT" O  (-:-) 202-
ATPG> report test stim -set /ix181/A0 1
// Time = 0
// Load 1 /ix334/SFFR (250), chain1 8
// Load 1 /ix324/SFFR (251), chain2 0
// Load 1 /ix314/SFFR (252), chain2 1
// Load 1 /ix304/SFFR (253), chain2 2
// Time = 1
```

## Notes:

# Lab: Troubleshooting Areas of Low Test Coverage

## Objectives

- Identify blocks of low test coverage.
- Determine untestable fault classifications.
- Use the hierarchy browser to locate instances within the design.
- Determine the cause of ATPG Untestable (AU) faults.
- Use DFTInsight to debug faults.
- Black box undefined design modules.
- Change the abort limit to improve ATPG effort.
- Select specific pattern types to improve test coverage.

## List of Exercises

- Exercise 20: Troubleshooting Areas of Low Test Coverage

## Getting Started

1. Log in to your workstation if you are not already logged in.
2. Change to the \$ATPGNW/lab 8/exercise\_20 directory.

```
shell> cd $ATPGNW/lab8/exercise_20
```



Note

Remember that for the exercises in this lab you use the libraries found in the **libraries\_7\_to\_9** directory.

## Exercise 20: Troubleshooting Areas of Low Test Coverage

In this exercise, you do the following in order to troubleshoot areas of low test coverage and to increase ATPG test coverage:

- Assess the problem:
    - Determine which blocks are reporting low coverage.
    - Determine untestable fault categories.
  - Determine why ATPG classified faults as untestable:
    - Determine why faults are classified ATPG Untestable (AU) using the hierarchy browser.
  - Analyze the design fault by fault:
    - Use DFTInsight.
  - Debug the design.
  - Black box undefined modules.
  - Improve ATPG effort and effectiveness:
    - Increase the abort limit.
    - Select specific pattern type(s) to improve test coverage.
1. Invoke FastScan on the following circuit:

|               |                   |
|---------------|-------------------|
| Design:       | gate_scan_8.v     |
| Library:      | atpglib           |
| Command File: | fs_8_sample.do    |
| Log file:     | results/ex_20.log |

Note that we run the dofile at invocation. You can set up the design using the Invocation dialogue box, or by executing shell commands.

What is the switch used for the dofile on the command line?

- 
2. Three *types* of warnings appear while invoking the dofile. What are they?
- 
- 
- 

Which module(s) need to be defined as black boxes?

---

3. Black box the undefined modules.
4. Go to ATPG mode, ignoring any DRC warning messages. You should be in ATPG system mode.

During DRC, a total of 237 warnings are detected and the warning messages are displayed in the session transcript area. You have two choices at this time — either debug the DRCs and then do pattern generation or proceed with pattern generation, assuming that the DRCs will not impact test coverage too much.

5. Set up the Fault Universe.
- a. Click on Customize. The Setup Fault Universe dialogue box opens.

Fault Type/List tab:                    Load Existing Faults (Misc. Options area)

Load the Faults from the Following File: fs\_sample\_faults.flt

- b. Select — Retain and Protect Faults in the Misc. Options dialogue window, then select Retain Each Fault's Current Classification.

Click Ok at the bottom of each dialogue window.

What command is executed after you finish this step?

6. Generate patterns using all current settings.

a. Click the \_\_\_\_\_ button or type \_\_\_\_\_ at the command line prompt.

b. Fill in the following table after the current run:

**Table 8-1. Report Statistics ATPG**

**Table 8-2.**

| Fault Class            | # faults (coll.) | # faults (total) |
|------------------------|------------------|------------------|
| FU (full)              |                  |                  |
| UC (uncontrolled)      |                  |                  |
| UO (unobserved)        |                  |                  |
| DS (det_sim)           |                  |                  |
| DI (det_imp)           |                  |                  |
| PU (posdet_untestable) |                  |                  |
| PT (posdet_testable)   |                  |                  |
| UU (unused)            |                  |                  |
| TI (tied)              |                  |                  |
| BL (blocked)           |                  |                  |
| RE (redundant)         |                  |                  |
| AU (atpg_untestable)   |                  |                  |

**Table 8-3. Report Coverage/Effectiveness ATPG****Table 8-4.**

| Total # of Faults | #test_pat. | test_cov. | fault_cov. | atpg_effect. |
|-------------------|------------|-----------|------------|--------------|
|                   |            |           |            |              |

7. Use the Report... button in the FastScan ATPG Run Statistics window to find out more about ATPG untestable (AU) faults.

- a. Select the Testability tab.
  - i. Click on the Select... button in the Fault Class (code or name) for Testability Analysis area. The Fault Class Choices dialogue box opens.
  - ii. Select AU. Click OK.
  - iii. Click on the Generate Report button in the Reported Data area.

 This report can be generated using the command **report testability data -class au**

**Note**

You have generated a testability analysis report of all the AU faults in the design. Note that a large number of faults are connected to non-scan latches.

What would you do to test these faults connected to non-scan latches?

- 
- b. Select the Faults tab.
    - i. Display the statistics for the fpu1 instance.
      - a. Click on the Show Statistics... button in the Reported Data area.

- b. Select the option Hierarchical Instance at Path and browse the bigchip hierarchy to find it. (*Hint: Look under p1*)
- c. Select fpu1 when you find it. The path should be displayed in the Selected Instance(s) field of the Design Hierarchy Browser.
- d. Click on the Report button.

What is the full pathname for the fpu1 instance?

---

The test coverage for the fpu1 instance is \_\_\_\_\_

The number of AU faults for instance fpu1 is \_\_\_\_\_

What command would you use to report on AU faults for the fpu1 instance? \_\_\_\_\_

So far in this lab activity you have located the instances in the design that are reporting low test coverage. Next, you are going to determine why AU classified faults are untestable.

- c. Determine why faults are classified as ATPG Untestable (AU).
  - i. Click on the Select... button in the Fault Class (Code or Name) area. When the Fault Class Choices dialogue box opens, select AU.
  - ii. Click on the Browse Hierarchy... button in the Options area. When the Design Hierarchy Browser opens, select the fpu1 instance.
  - iii. Click on the Report Faults button in the Reported Data area.

Now there should be a list of AU faults in the fpu1 module visible in the Reported Data area.

- iv. Somewhere in the list is the pathname fpu1/U1954/A1. Browse for it in the list.

Click the LMB on /p1/fpu1/U1954/A1—this is entered automatically into the Fault entry box (or type /p1/fpu1/U1954/A1 into the Fault entry box, especially if you have a difficult time finding it by browsing.)

- v. Click on the Analyze button.

What behavior is blocking the fault being tested?

---

- vi. Now analyze the fault at /p1/fpu1/opa\_r1\_reg[29]/D. (It is easier to type this node directly into the Fault field.)

What behavior is blocking the fault being tested?

---

- vii. Close the Results & Analysis dialogue box.

- viii. Close the FastScan ATPG Run Statistics dialogue box.

8. Using the results from the above analysis might be sufficient in order to understand how to fully test them, but it also might be useful to graphically analyze the faults. This is done through DFTInsight.

- a. Click on the Open DFTInsight button in the Button pane.

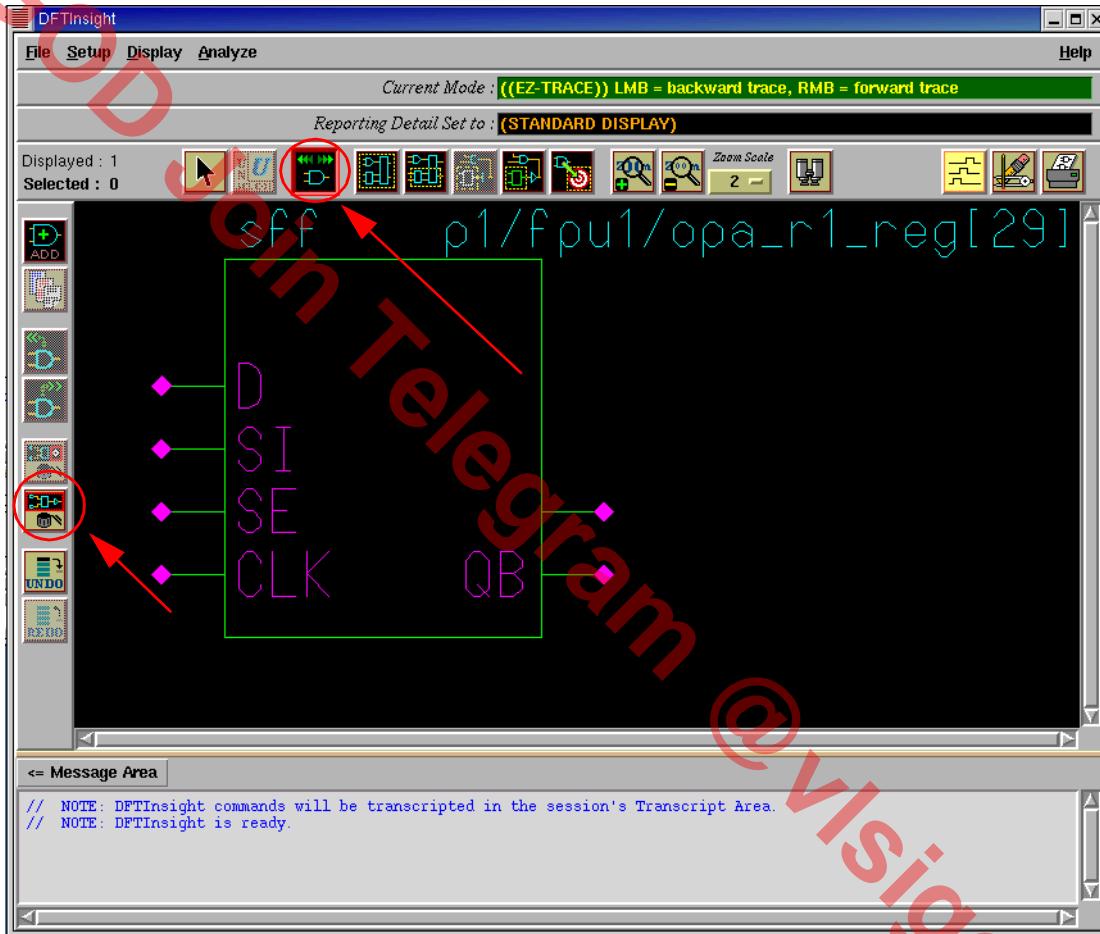
- b. Click on the **Display > Additions...** menu item. The Make Additions to the Display dialogue box opens.

- i. Type /p1/fpu1/opa\_r1\_reg[29] into the Instance Names, Gate IDs or Pin Pathname entry box and click the Add button.

- ii. Click OK.

- c. Click the LMB on the E-Z Trace button.

## Troubleshooting DRC and Simulation Mismatch



- i. Click the LMB on the flip-flop. This action traces back one level of inputs.

What inputs do you see? \_\_\_\_\_

- ii. Click the LMB on the scan input (SI) of opa\_r1\_reg[29].

What is the SI input connected to? \_\_\_\_\_

- iii. Click the RMB on the original scan flip-flop: opa\_r1\_reg[29]. This action traces forward to all outputs.

What does output Q connect to? \_\_\_\_\_

What does output QB connect to? \_\_\_\_\_

- iv. Click the RMB on the inverter /p1/fpu1/U1219.

What gate does the inverter connect to? \_\_\_\_\_

When you observe the Q output of opa\_r1\_reg[29], you notice that it is not connected to any other gate. However, this is not what is causing the fault to be undetected. In order for input D to report an AU fault, both Q and QB must not be connected anywhere. If either are connected, as QB is, then the fault is detectable. The cause of the AU fault on this register is due to tieX behavior. Refer to Chapter 4 "Understanding Testability Issues" in the Scan and ATPG Process Guide for understanding how and why tieX is inserted into circuitry.

- d. Clear the DFTInsight display using the Delete All button, or the menu option **Display > Deletions > All**.

9. Now examine how many non-scan cells there are in the circuit.

- a. Click on the Report Circuit... button in the FastScan Button pane.

What dialogue box opens? \_\_\_\_\_

- i. Click on the Report NonScan Cells tab and then click on the Generate Report button.

Note that there are quite a few flip-flops from the fpu1 module listed as non-scan cells.

- ii. Close the Report Circuit dialogue box.

10. Next we investigate the register that 'blocked' the fault on /p1/fpu1/U1954/A1 using DFTInsight. (Refer to section 7.c.iv)

- a. Add the register /p1/fpu1/fract\_denorm\_reg[46] into DFTInsight.
  - b. Investigate backwards from the register to find why the fault is blocked. If you see nothing, try setting the design level to primitive. (**Setup > Design Level > Primitive**)

What is the difference between the Design Levels? \_\_\_\_\_

\_\_\_\_\_

What caused the fault to be blocked? \_\_\_\_\_

\_\_\_\_\_

c. Exit DFTInsight.

11. You have assessed the problems with the design. You know where the areas of low test coverage are located and you know why faults have been classified as ATPG Untestable (AU).

You used DFTInsight, the graphical debugging tool, to analyze the design fault by fault.

The next step is to debug the design by editing the netlist. We fix a lot of the problems in the following sections.

- a. Edit the netlist `gate_scan_8.v` using an editor of your choice. You can run `vi` from the ATPG prompt if you want:

`ATPG> vi gate_scan_8.v`

- b. Correct the error. Look for `opa_r1_reg[29]` and see if you can isolate and fix the problem.

What did you do to fix the problem? \_\_\_\_\_

\_\_\_\_\_

- c. Remove the problem with the module that needed to be defined as a black box.

What did you do to fix the problem?

\_\_\_\_\_

Do not spend too much time trying to figure out how to edit the netlist. The edited netlist with the bugs removed —1\_netlist.v is in the directory.

You are encouraged to compare your editing efforts against the previously edited netlist and make corrections, if any, to your netlist.

- d. Save the newly edited Verilog netlist as new\_netlist.v in the results directory and exit FastScan.
12. Invoke FastScan on the following circuit:

|               |                    |
|---------------|--------------------|
| Design:       | new_netlist.v      |
| Library:      | atpglib            |
| Command File: | fs_8_sample.do     |
| Log file:     | results/ex_20a.log |
13. Go to ATPG mode. Ignore DRC warnings.
14. Set up the Fault Universe, adding faults from the external fault list, fs\_sample\_faults.flt, retaining each fault's classification.
15. Generate Patterns.
  - a. What test coverage did you achieve? \_\_\_\_\_
  - b. During ATPG FastScan reported that some faults were aborted.
    - i. How many? \_\_\_\_\_
    - ii. What is the current abort limit? \_\_\_\_\_

The test coverage percentage has improved, but your company requires the test coverage to be in the 90s.

During ATPG, FastScan reported that some faults were aborted. In the next activity, you raise the abort limit to improve ATPG effort.

16. Raise the abort limit to improve ATPG effort.
  - a. Click on Change Settings... in the FastScan ATPG Run Statistics dialogue box to open the Setup for Test Pattern Generation dialogue box.
    - i. Click on the ATPG Effort/Process tab. In the Fault Detection Effort area, type “500” into the Combinational ATPG Abort Limit field.
    - ii. Click OK.

What commands were written in the Session Transcript window?

a. \_\_\_\_\_

b. \_\_\_\_\_

- b. Click on the Run button to generate more patterns in addition to those already generated.

What test coverage was achieved in this run? \_\_\_\_\_

How many faults were aborted this time? \_\_\_\_\_

- c. Click the Run button again in the FastScan ATPG Run Statistics dialogue box.

Observe that there still are a number of faults aborted.

17. You should have noticed as the software moved from Setup mode to ATPG mode that Circuit Learning/DRC checking identified a RAM in the design. If you did not notice it, scroll up in the Session Transcript window and observe it.

Additionally, the dofile has read and write controls defined for RAM. The next logical step to improve coverage is to use RAM sequential testing.

- a. Click on Change Settings... in the FastScan ATPG Run Statistics dialogue box again. When the Setup for Test Pattern Generation dialogue box opens, turn on RAM Sequential Patterns in the ATPG Effort/Process tab.
- i. Click on the RAM Sequential Patterns button in the ATPG Algorithm area.
  - ii. Click OK.
  - iii. What commands were written in the Session Transcript window?
  - iv. \_\_\_\_\_
  - v. \_\_\_\_\_
- b. Click on the Run button in the FastScan ATPG Run Statistics dialogue box.
- c. Report Statistics and fill in the following tables:

**Table 8-5. Report Statistics ATPG****Table 8-6.**

| Fault Class          | # faults (coll.) | # faults (total) |
|----------------------|------------------|------------------|
| FU (full)            |                  |                  |
| UC (uncontrolled)    |                  |                  |
| UO (unobserved)      |                  |                  |
| DS (det_sim)         |                  |                  |
| DI (det_imp)         |                  |                  |
| TI (tied)            |                  |                  |
| BL (blocked)         |                  |                  |
| RE (redundant)       |                  |                  |
| AU (atpg_untestable) |                  |                  |

**Table 8-7. Report Coverage/Effectiveness ATPG.**

**Table 8-8.**

| Total # of Faults | #test_pat | #basic_patterns | #ram_seq_pat | test_cov. | fault_cov. | atpg_eff. |
|-------------------|-----------|-----------------|--------------|-----------|------------|-----------|
|                   |           |                 |              |           | .          |           |

19. Exit FastScan.

## Test Your Knowledge

1. List two steps you would take to assess problems with a design.

---

2. What tool is useful for locating fault locations?

---

3. What tool is useful for analyzing a design fault by fault?

---

4. What can you do to increase ATPG effort?

---

5. What can you do to improve ATPG effectiveness?

---

## Lab Summary

Now that you have completed the Advanced ATPG lab, you should know how to do the following:

- Identify blocks of low test coverage.
- Determine untestable fault classifications.
- Use the Hierarchy browser to locate instances.
- Determine the cause of ATPG Untestable faults.
- Black box undefined design modules.
- Change the abort limit.
- Select patterns.

VLSIGOD Join Telegram @vlsigodofficial

# **Module 9**

## **Troubleshooting DRC and Simulation Mismatch**

### **Objectives**

Upon completion of this module, you will be able to:

- Debug Design Rules Checking (DRC).
- Debug timing issues.
- Debug simulation mismatches.

## Module Topics

### Module Topics

This module addresses the following topics:

- ◆ Common DRC violations
  - E4
  - C3
  - C6
  - D5
  - D6
  - T3
- ◆ Test benches
- ◆ Clock skew issues
- ◆ Simulation mismatches

### Notes:

## Analyzing DRC Violations: Commands

### Analyzing DRC Violations: Commands

- ◆ Commands and options used in Setup mode to analyze DRC violations:
  - ATPG\_analysis option of SET DRC Handling command to fully analyze DRCs:
    - C1, C3, C4, C5, D6, E10, E11, and E12
  - SET GATE Level to specify the level of information reported/displayed
    - design cell, low design, or primitive
  - SET GATE Report to specify the type of information reported
    - Trace ( simulates shift of scan chain)
    - Error (reports values for error condition)
    - Drc\_pattern (access data during any procedure)
    - Normal (default)
    - Parallel\_pattern (capture cycle value for specified pattern)

### Notes:

# Analyzing DRC Violations: Report Gates Command

## Analyzing DRC Violations: Report Gates Command

- ◆ REPort Gates reports additional information for troubleshooting
  - Specific gate
    - SETUP> REPort GAtes <instance\_number>
  - All gates of a specific logic type
    - SETUP> REPort GAtes -Type *gate\_type* . . .
  - A histogram of all gate types
    - SETUP> REPort GAtes -Type Histogram
  - A path between two gates
    - SETUP> REPort GAtes -Path <gate1\_ID#> <gate2\_ID#>
  - The first input of a gate (back tracing)
    - Reports the gate connected to the first input listed of the previously reported gate
    - SETUP> B
  - Reporting on the first fanout (forward tracing)
    - SETUP> F

To use "B" or "F", the gate level must be set to primitive  
SETUP> SET Gate Level Primitive

## Notes:

## DRC Violations: E4 - Procedure (Bus Contention)

### DRC Violations: E4 - Procedure (Bus Contention)

- ◆ Bus contention occurs when tri-state drivers have conflicting values when driving the same net.
  - Any two inputs at values X & X, 0 & X, 1 & X, and 1 & 0
- ◆ E4 violations occur when bus contention occurs during scan procedure.
  - Example:
    - Shift
    - Load\_unload

### Notes:

# Debugging E4 Violations

## Debugging E4 Violations

### ♦ Use DFTInsight

- Make E4 violations occur as an error from within FastScan

SETUP> SET DRc Handling E4 Error

SETUP> SET System Mode ATPG // performs DRC checking

- Use 'analyze drc' from within DFTInsight

- Trace back through the design and correct the problem

### ♦ Optionally, use the REPort GAtes command

SETUP> SET DRc Handling E4 Error

SETUP> SET GAtes Report Error\_pattern

SETUP> SET SYstem Mode ATPG

SETUP> REPort GAtes <instance\_number>

SETUP> SET GAt Level Primitive

SETUP> B

Trace back to locate the problem

HINT: B 2 will back trace on the second  
input of the last gate reported

## Notes:

## E4 Contention on Bidirectionals

### E4 Contention on Bidirectionals

- ◆ Bidirectionals fail when bus contention occurs between the chip bidi outputs and the tester signal that drives the input of the chip bidi.
- ◆ By default the tester will drive an X into the bidi.
- ◆ Do the following to force the tester to drive a Z:
  - Edit test\_setup and load/unload procedures.

`force <signal_name> Z;`

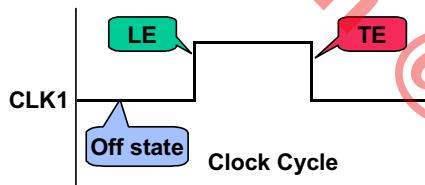
### Notes:

# Clocks

## Clocks

- ◆ **Clocks:**
  - A primary input (PI) that changes the state of a sequential element.
    - Includes set and reset inputs
  - The transition of the clock from OFF to ON is the leading edge (LE) of the clock.
  - The transition of clock from ON to OFF is the trailing edge (TE) of the clock.
- ◆ **Example:**

SETUP>ADD CLOCK 0 CLK1



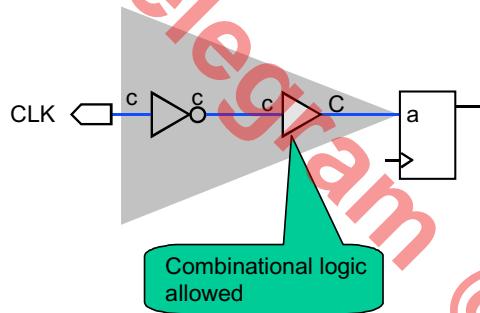
## Notes:

## Clock Cones

### Clock Cones

- ♦ **Clock cone:**

- A gate pin is defined as being in the **clock cone** if there is a path through combinational logic gates from the **clock pin** to the output pin.
- Pin "a" is in the "**clock cone**" of CLK.



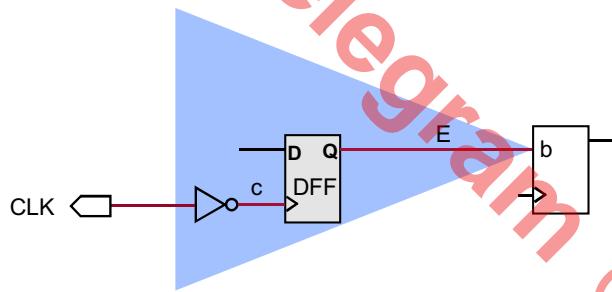
- **Display clock cone with SET GATE Report -Clock <clock>.**

### Notes:

## Effect Cones

### Effect Cones

- ◆ Effect cone:
  - A signal is defined as being in the effect cone if there is a sequential element between the output pin and the clock pin.
  - Pin "b" is in the "effect cone" of CLK.



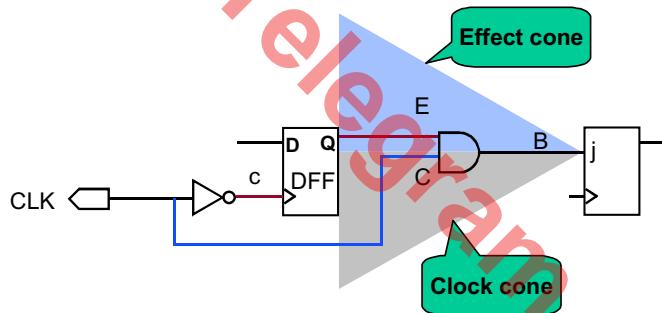
### Notes:

## Both Cones

### Both Cones

- ♦ Both cones:

- An output pin that is in both the clock cone and the effect cone.
- Pin "j" is in both the "clock" and "effect cone" of CLK.

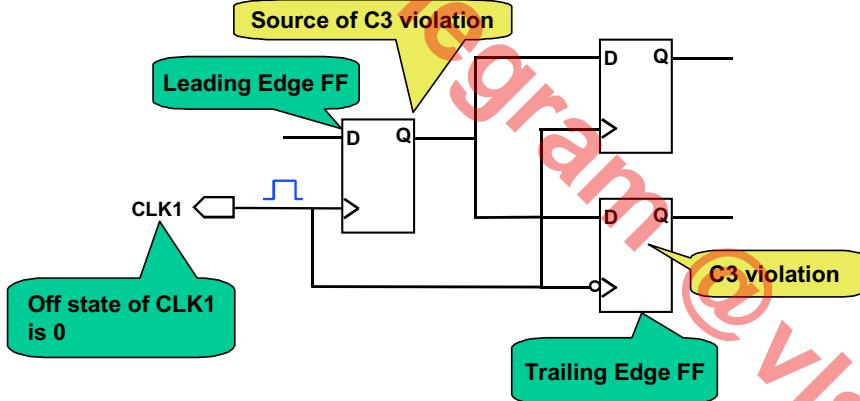


## Notes:

## Clock Rules: C3

### Clock Rules: C3

- ◆ C3 clock rules:
  - Designs that contain both leading edge and trailing edge flops have the potential for C3 DRC violations.
    - Data must not be captured into a FF on a clock's trailing edge.



### Notes:

## DRC Violations: C3

### DRC Violations: C3

- ◆ A C3 violation occurs if one of the following is true:
  - The output of a leading edge triggered FF is connected to the D input of a trailing edge triggered FF
  - RAM Write input is in the clock cone and a data-in or address input of the Write port is in the effect cone
  - RAM Read is in the clock cone and an address input of the associated Read port is in the effect cone

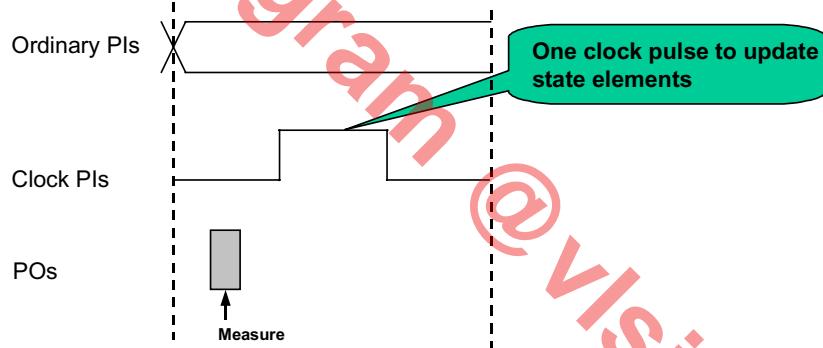


### Notes:

# FastScan Event Simulation

## FastScan Event Simulation

- ◆ By default, FastScan simulates a single event per test cycle:
  - Clocks have pulsed
  - Combinational logic is updated based on state element values from previous clock pulse
  - State elements have not changed
  - Data is captured on one clock edge
  - Simulation data is not updated between clock edges



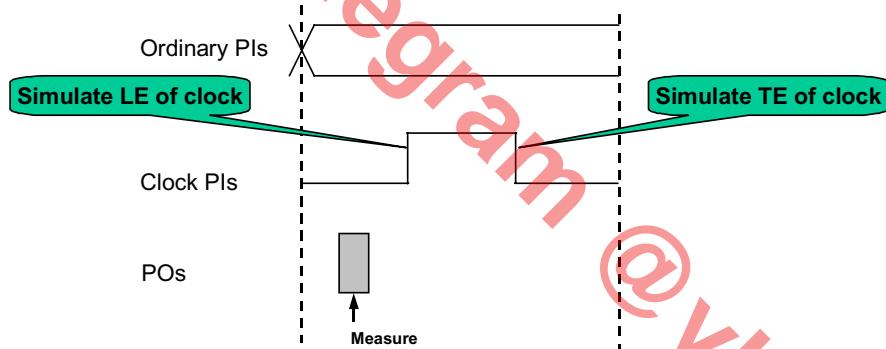
## Notes:

## Setting Event Simulation

### Setting Event Simulation

- Default event simulation is not appropriate if there are C3/C4/C6 DRC violations.
- Do the following to change event simulation:

```
SETUP> SET SPLIT Capture_cycle ON // simulates 2 events in clock pulse  
// Used to properly simulate and avoid  
C3 and C4 problems
```



### Notes:

## Handling C3 Violations

### Handling C3 Violations

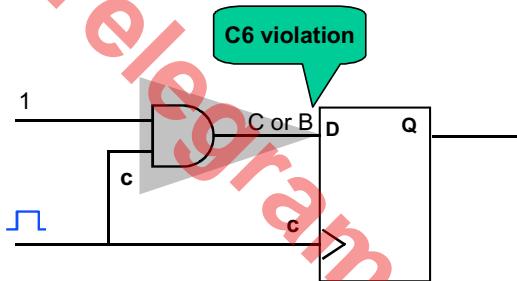
- ◆ Capture data for LE and TE flip-flops between clock edges  
SETUP> SET SPLIT Capture\_cycle ON
- ◆ The SET SPLIT Capture\_cycle command cannot be used with RAM\_sequential patterns
  - Use multi-load patterns  
SETUP> SET Multiple Load on  
SETUP> SET Pattern Type -Sequential 4

### Notes:

## Clock Rules: C6

### Clock Rules: C6

- ◆ C6 clock rules:
  - A clock may not affect data that it is capturing.
  - A rule violation occurs when a clock input of a scannable element and its data line are in the same cone.



- Default is to simulate the D input with clock value at the ON state.
  - Default simulation will capture a 1 in this case.

### Notes:

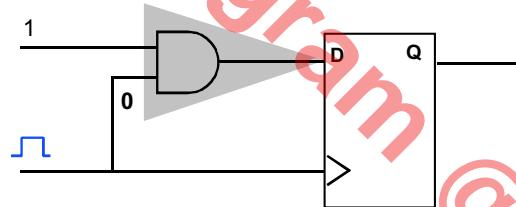
# Handling C6 Violations

## Handling C6 Violations

- ◆ Handling a C6 violation:

SETUP>SET CLOCK\_off SIMULATION ON

- Enables the simulation of the event where all clock primary inputs are at their “off” value
- Forces PIs
- Maintains previous state element values



- Simulation will capture a 0 in this case
- Not compatible with RAM sequential

## Notes:

## Data Rules: D5

### Data Rules: D5

- ◆ All memory elements must be scannable.
  - A D5 violation occurs when memory elements are identified as not belonging to a scan chain.
  - The default handling is a warning.
    - Failure to satisfy this rule results in loss of test coverage.
  - Applies to both latches and FFs.
    - For latches, the D6 rule will also apply.

### Notes:

## Data Rules: D6

### Data Rules: D6

- ◆ All non-scan latches must behave as transparent latches.
- ◆ A D6 violation occurs if a latch fails one of the following conditions:
  - If a latch creates a feedback path, that path must be broken.
  - Latches must have a propagable path to an observe point.
  - Latches must pass a value when all clocks are off.
  - All clock, set, and reset inputs must be at determinate state when all clocks are off.
  - Latches must have only one set/reset/clock input when all defined clocks are off.

### Notes:

## Handling D5 and D6 Violations

### Handling D5 and D6 Violations

- ♦ Handling D5 and D6:

- Enable clock sequential ATPG

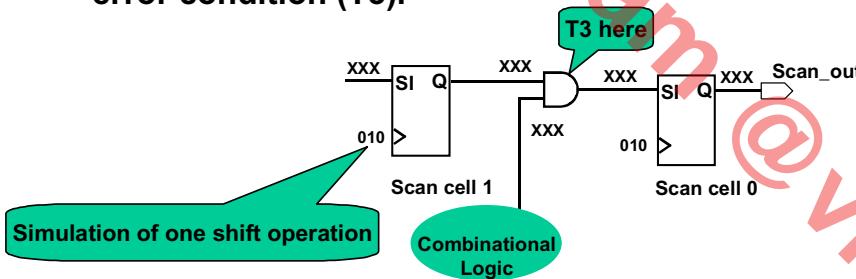
SETUP>SET PATTERN Type -Sequential 2 (or more)

### Notes:

## Scan Chain Trace Rules: T3

### Scan Chain Trace Rules: T3

- ◆ Tool traces from scan output pin to scan input pin using load-unload and shift procedures
- ◆ An unknown (x) is shifted backwards from scan-out.
  - Some circuit values are learned “tied” from test\_setup, pin constraint, and other procedures.
  - Follows sensitized path backward.
  - Simulates shift to back trace through sequential gates.
- ◆ An improperly sensitized gate in the scan path will cause an error condition (T3).



### Notes:

## Common Causes of T3 Errors

### Common Causes of T3 Errors

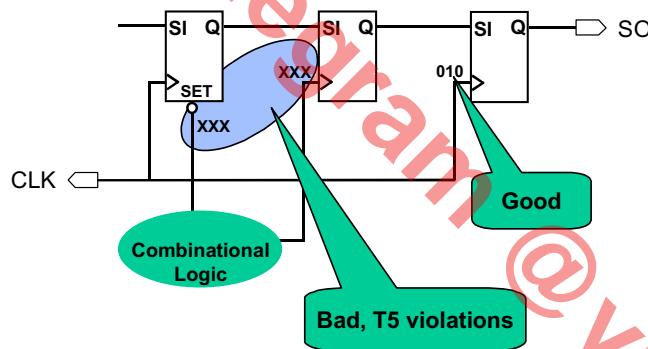
- ◆ Common causes of T3 errors:
  - Scan Enable (SE) controlled through TAP or state machine not initialized with test\_setup and constraints
  - Set or reset not properly initialized

### Notes:

## Scan Chain Trace Rules: T5

### Scan Chain Trace Rules: T5

- ◆ T5 trace rule:
  - During the shift procedure, scan clocks should pulse and set/resets should be disabled.



### Notes:

## Debugging T3 and T5 Violations

### Debugging T3 and T5 Violations

- ◆ Use DFTInsight to back trace on problem value.
- ◆ Optionally, use the REPort Gates command.
  - SETUP> SET TTrace Report ON
    - FastScan prints out all the gates it traces through when it traces the scan chains.
  - SETUP> SET GATE Report Trace
    - FastScan prints out the data for the “shift” procedure.
    - Optionally, use the SET Gate Report Drc shift command.
  - SETUP> SET SYstem Mode ATPG
  - SETUP> REPort GAtes <instance number>
    - Determine where scan chain is blocked.
  - SETUP> B
    - Trace back the blocking value through the design and correct the problem(s).

### Notes:

## Testbenches

### Testbenches

- ◆ FastScan provides serial and parallel Verilog or VHDL testbenches.
- ◆ They drive time-based simulations to verify FastScan's expected values against the simulated values.
- ◆ When these values do not match, there is a simulation mismatch:
  - Functional discrepancies.
  - Timing issues (FastScan is event-driven).

### Notes:

## Serial Testbench

### Serial Testbench

- ◆ In the serial testbench, the scan chain is operated like a tester:
  - Data is shifted serially through the chain.
  - Takes a long time to simulate test vectors.
- ◆ Reports the scan output pin and time of mismatch.
- ◆ Normal practice is to simulate 2 or 3 serial patterns.

ATPG> SAVe PAtterns -Verilog -Serial -Sample 2

### Notes:

## Parallel Testbench

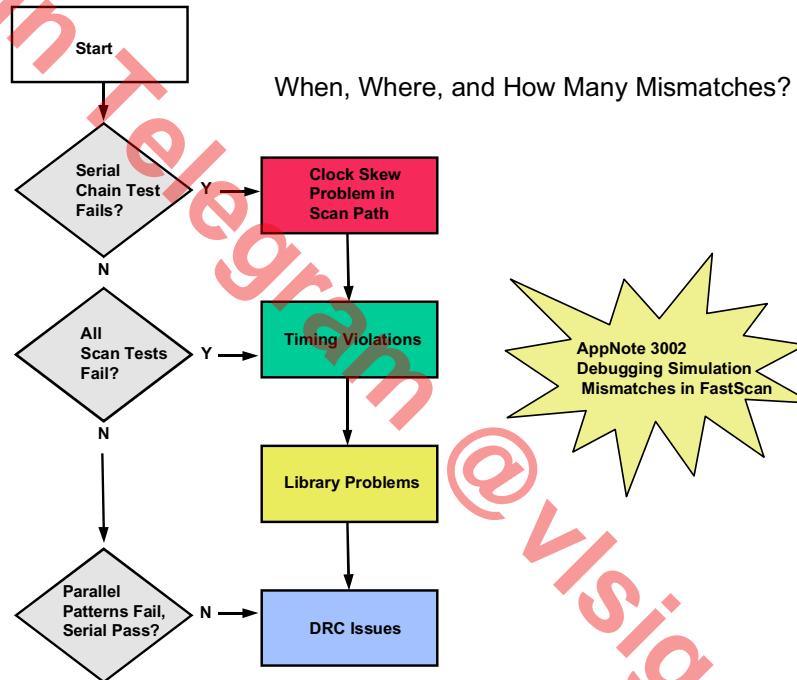
### Parallel Testbench

- ◆ The parallel testbench simulator has access to the internal nodes in the design:
  - Data is loaded in parallel.
    - Data is directly “forced” at scan cell inputs.
    - Shift procedure is applied once.
    - Loads and unloads the entire scan chain in one clock cycle.
  - Reduced simulation time.
- ◆ Reports the following:
  - Time.
  - Pattern number
  - Name/location of scan cell (or PO) where mismatch is observed.
- ◆ Normal practice is to simulate all parallel patterns

### Notes:

# Debugging Simulation Mismatches in FastScan

## Debugging Simulation Mismatches in FastScan



9-29 • Design-for-Test: Scan and ATPG:  
Troubleshooting DRC and Simulation Mismatch

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## DRC Violations: Simulation Mismatches

### DRC Violations: Simulation Mismatches

- ◆ DRC violations that are most likely to cause simulation mismatches are:
  - C3
  - C4
  - C6
- ◆ If mismatches occur, first check if DRCs were reported.

### Notes:

# When, Where, and How Many Mismatches

---

## When, Where, and How Many Mismatches

- ♦ If DRC violations are not the problem, check the following:
  - Mismatches reported on POs, scan cells, or both?
    - Mismatches on scan cells are related to capture ability and timing.
    - Mismatches on POs related to an incorrect value loaded into scan cells.
  - Mismatches reported on a few or most of the patterns?
    - Mismatches on a few patterns indicate a problem with certain patterns
    - Mismatches on most patterns indicate a more generalized problem.
  - Mismatches observed on a few pins/cells or most pins/cells?
    - Mismatches on a few pins/cells indicate a problem related to a few specific instances or one part of the logic.
    - Mismatches on most pins/cells indicate a more general problem.

## Notes:

# When, Where, and How Many Mismatches (Cont.)

## When, Where, and How Many Mismatches (Cont.)

- ♦ **Checking for mismatches:**
  - Do both the serial and parallel test bench fail or just one of them?
    - Serial failure indicates mismatch is related to scan shifting.
    - A shadows problem may cause serial test bench to pass and the parallel test bench to fail.
  - Does the chain test fail?
    - If the serial pattern fails the chain test also fails.
  - Do only certain pattern types fail?
    - If only ram sequential patterns fail, the problem is related to RAMs.
    - If only clock\_sequential patterns fail, the problem is related to non-scan flip-flops and latches.

## Notes:

## Clock Skew Problems

### Clock Skew Problems

- ◆ Clock delays are caused by the following:
  - Routing.
  - Gates (muxes, buffers) on clock lines.
- ◆ Do the following to detect this problem:
  - Run a time-based simulation.
    - ModelSim or HDL simulator.
  - Run a critical timing analysis in scan mode.
    - SST Velocity or another static timing analyzer

### Notes:

## Timing Violations

### Timing Violations

- ◆ The following timing issues cause mismatches:
  - Setup and hold violations during testbench simulation.
  - Timeplate with timing too tight.
    - Timing events need more separation.
- ◆ Do the following to correct this problem:
  - Examine the simulation data and compare the values observed with values expected by FastScan.
  - Expand timeplate and/or test procedure files.
    - Default timing has 10 ns separation of events.

### Notes:

## Library Problems

### Library Problems

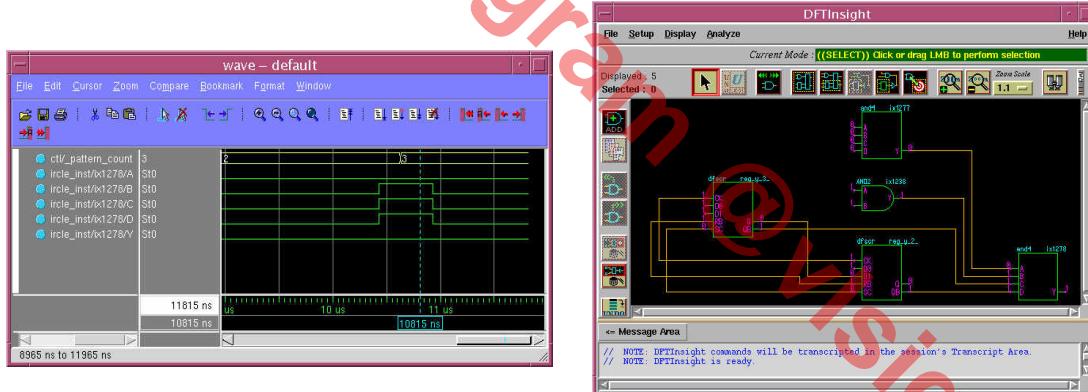
- ◆ **Incorrect library model:**
  - For combinational and sequential elements, this causes mismatches for all patterns.
  - For instances such as RAM, this causes mismatches for a few patterns (such as RAM sequential).
  - Non-equal values on the inputs to non-tristate multi-driven nets
- ◆ **To check for this problem:**
  - Run simulation on library.
    - Has the library been validated (verified) ?

### Notes:

# Automatic Analysis of Simulation Mismatch

## Automatic Analysis of Simulation Mismatch

- ◆ Automatic analysis of simulation mismatch:
  - Traces the design, automatically
  - Locates sources of mismatch between FastScan and ModelSim
  - Displays mismatch sources(s) in:
    - DFTInsight
    - ModelSim



9-36 • Design-for-Test: Scan and ATPG:  
Troubleshooting DRC and Simulation Mismatch

Copyright © 2003 Mentor Graphics Corporation

## Notes:

## Automatic Analysis of Simulation Mismatch (Cont.)

### Automatic Analysis of Simulation Mismatch (Cont.)

- ♦ Automatic analysis of simulation mismatch:

```
ATPG> SAVe PAttern <filename.v> -Verilog -Debug  
      -Display
```

- Saves test bench in parallel Verilog format
- Compiles test bench
- Invokes vsim (ModelSim)
- Compares simulation result between vsim and FastScan
  - Locates the source(s) of mismatch
- Displays the first mismatch source in DFTInsight and ModelSim
- Reports mismatch sources

Performs automatic troubleshooting

### Notes:

# Debugging Serial Simulation Mismatches: Chain Test

## Debugging Serial Simulation Mismatches: Chain Test

- ◆ In most cases where serial simulation mismatches and parallel simulation passes, the serial chain test will also fail.
- ◆ Confirm that the chain test fails:
  - Save out a pattern with chain test.  
SETUP> SAVE PATterns <filename> -Serial -CHain\_test
  - Verify pattern using a time-based simulator.
- ◆ If the chain test fails, do the following:
  - Edit the procedure file so that it contains two independent shifts.
  - Use the parallel test bench.

## Notes:

## Clock Skew in Chain Test

### Clock Skew in Chain Test

- ♦ Clock skew occurs in chain test when the scan cells are clocked at slightly different times:
  - Scan cells should capture “old” data of D.
    - But, Q is updated with a “new” value.
- ♦ Solution:
  - Add buffers.
  - Redo clock synthesis.

### Notes:

# Lab: Troubleshooting DRC and Simulation Mismatch

## Objectives

- Use DFTAdvisor to insert scan chains.
- Identify blocks of low coverage.
- Determine the cause of ATPG Untestable (AU) faults.
- Use DFTInsight to debug faults.
- Check for simulation mismatch.
- Correct DRC violations.

## List of Exercises

- Exercise 21: Troubleshooting DRC and Simulation Mismatch — Full Tool Flow
- Exercise 22: Debugging Mismatches with Automated Tools

## Getting Started

1. Log in to your workstation if you are not already logged in.
2. Change to the \$ATPGNW/lab 9/exercise\_21 directory.

```
shell> cd $ATPGNW/lab9/exercise_21
```



Note

Remember that for the exercises in this lab you use the libraries found in the **libraries\_7\_to\_9** directory.

## Exercise 21: Troubleshooting DRC and Simulation Mismatch — Full Tool Flow

This exercise takes you through multiple runs of DFTAdvisor, FastScan, and ModelSim to do the following:

- Insert scan chains and test logic.
- Improve test coverage.
- Check for simulation mismatch.
- Correct DRC violations.
- Generate high quality patterns.

In the following activities, you run the DFT applications from batch mode by using dofiles to pipe commands into the applications. Recall that a dofile is a text file that enables you to automatically control the operations of the tool because it contains application commands. If you have a large number of commands, or a common set of commands that you use frequently, you can save time by placing these commands in a dofile.

If you place all commands, including the **Exit** command, in a dofile, you can run the entire session as a batch process that is run at invocation.

1. Run the runDFTA invocation script.

```
shell> runDFTA
```



You may have to use `./runDFTA` depending on how the account path is defined

**Note**

- a. What does the script instruct DFTAdvisor to do? (*Look in the Shell window.*)

b. What is the name of the .do file that the runDFTA script references? \_\_\_\_\_

c. What nets are not driven? \_\_\_\_\_

d. What are FN1 violations and how many are there? \_\_\_\_\_

e. How many scannability failures are there? \_\_\_\_\_

f. Which scannability rule fails? \_\_\_\_\_

g. Which clock rule fails? \_\_\_\_\_ How many times? \_\_\_\_\_

The runDFTA script runs the following commands from the dfta.do file:

```
add clocks 0 clk
add clocks 0 rst
set drc handling cl warning
set system mode dft
run
insert test logic -number 2 -clock merge -edge merge
report scan chains
write netlist netlists/gate_scan.v -verilog -replace
write atpg setup atpg -replace
exit
```

You instructed DFTAdvisor to insert two scan chains and to write a netlist — gate\_scan.v.

2. Run the runFS invocation script.

```
shell> runFS  
OR  
shell> ./runFS
```

- a. What does this script instruct DFTAdvisor to do? \_\_\_\_\_

---

---

---

The runFS script runs the following commands from the fs.do dofile:

```
dofile atpg.dofile //dofile written by DFTAdvisor  
set system mode atpg  
add faults -all  
run  
rep statistics  
save pat verilog_ser.pat -verilog -serial -proc -end 1 \  
-rep  
save pat verilog_par.pat -verilog -parallel -proc -rep  
exit
```

The atpg.dofile dofile runs the following commands:

```
add scan groups grp1 atpg.testproc  
add scan chains chain1 grp1 scan_in1 scan_out1  
add scan chains chain2 grp1 scan_in2 scan_out2  
add clocks 0 rst  
add clocks 0 clk  
SET DRC Handling C01 WARNING
```

- b. Fill in the following tables:

**Table 9-1. Report Statistics ATPG****Table 9-2.**

| Fault Class            | # faults (coll.) | # faults (total) |
|------------------------|------------------|------------------|
| FU (full)              |                  |                  |
| UO (unobserved)        |                  |                  |
| DS (det_sim)           |                  |                  |
| DI (det_imp)           |                  |                  |
| PU (posdet_untestable) |                  |                  |
| UU (unused)            |                  |                  |
| TI (tied)              |                  |                  |
| BL (blocked)           |                  |                  |
| RE (redundant)         |                  |                  |
| AU (atpg_untestable)   |                  |                  |

**Table 9-3. Report Coverage/Effectiveness ATPG****Table 9-4.**

| # sim_pat. | #test_pat. | test_cov. | fault_cov. | atpg_effect. |
|------------|------------|-----------|------------|--------------|
|            |            |           |            |              |

Note that after the script finishes, you are in FastScan interactive command mode.

- c. Observe the S1 violations from the dfta.log file.

```
ATPG> cat results/dfta.log
```

FastScan generated test patterns, but you have a problem — the test coverage is too low because of S1 violations and other problems.

In the next activity, you open DFTInsight to view the S1 violations.

### 3. Open DFTInsight.

ATPG> **open schematic viewer**

- a. View S1 violations, selecting the **Analyze > DRC Violation** menu option.

- i. Analyze the D5-1 DRC violation. Click on D5 in the Failed DRCs Qty area, then double click on D5-1 in the Specific IDs area.

What element is displayed graphically? \_\_\_\_\_

What warning message appears in the Message Area?

---

---

- ii. Use the E-Z Trace Mode button to trace back one level from pin “R”.

What gate drives “R”? \_\_\_\_\_

- iii. Trace back one level on “A0”.

What gate drives “A0”? \_\_\_\_\_

- iv. Trace back one level on “A1”.

What gate drives “A1”? \_\_\_\_\_

- b. Exit FastScan. This closes DFTInsight as well.

4. It is possible to produce a netlist that corrects the violations. Run the runDFTA1 invocation script.

```
shell> runDFTA1  
      OR  
shell> ./runDFTA1
```

- a. What is the difference between this script and the last one?

b. What does this achieve? \_\_\_\_\_  
\_\_\_\_\_

The runDFTA1 script runs the following commands from the dfta1.do dofile:

```
add clo 0 clk
add clo 0 rst
set test logic -reset on //to prevent S1 violations
set drc handling c1 warning
set sys mode dft
run
insert test logic -number 2 -clock merge -edge merge
report scan chains
write netlist netlists/gate_scan.v -verilog -replace
write atpg setup atpg -replace
exit
```

DFTAdvisor inserted test logic on the resets to make them controllable and inserted scan chains. A new netlist file — gate\_scan.v was written, as well as new ATPG setup files.

c. Run the runFS script to generate test patterns.

```
shell> runFS
```

d. Fill in the following tables:

**Table 9-5. Report Statistics ATPG****Table 9-6.**

| Fault Class            | # faults (coll.) | # faults (total) |
|------------------------|------------------|------------------|
| FU (full)              |                  |                  |
| UO (unobserved)        |                  |                  |
| DS (det_sim)           |                  |                  |
| DI (det_imp)           |                  |                  |
| PU (posdet_untestable) |                  |                  |
| PT (posdet_testable)   |                  |                  |
| UU (unused)            |                  |                  |
| TI (tied)              |                  |                  |
| BL (blocked)           |                  |                  |
| RE (redundant)         |                  |                  |
| AU (atpg_untestable)   |                  |                  |

**Table 9-7. Report Coverage/Effectiveness ATPG****Table 9-8.**

| # sim_pat. | #test_pat. | test_cov. | fault_cov. | atpg_effect. |
|------------|------------|-----------|------------|--------------|
|            |            |           |            |              |

The test coverage has improved, but it can be improved further.

5. In the next activity, you assess the reasons for low coverage, determine why faults are classified as untestable, and use DFTInsight to analyze the design.
  - a. Analyze the AU fault class to find a summary of the reasons for faults that are allocated to this class.

```
ATPG> report testability data -class au
```

Fill in the following:

**Table 9-9. Report Testability Data Class AU****Table 9-10.**

| Total No. Faults AU class | No. Faults tied by constraints | No. Faults blocked by constraints | No. Faults Connect. to clk/set/reset | No. Faults connected from tiex | No. Faults unclassified |
|---------------------------|--------------------------------|-----------------------------------|--------------------------------------|--------------------------------|-------------------------|
|                           |                                |                                   |                                      |                                |                         |

- b. Report and list all AU faults.

What command do you use? \_\_\_\_\_

- c. Analyze the AU fault /U\_CTR/U14238/A0 (*About 5th up from the bottom.*)

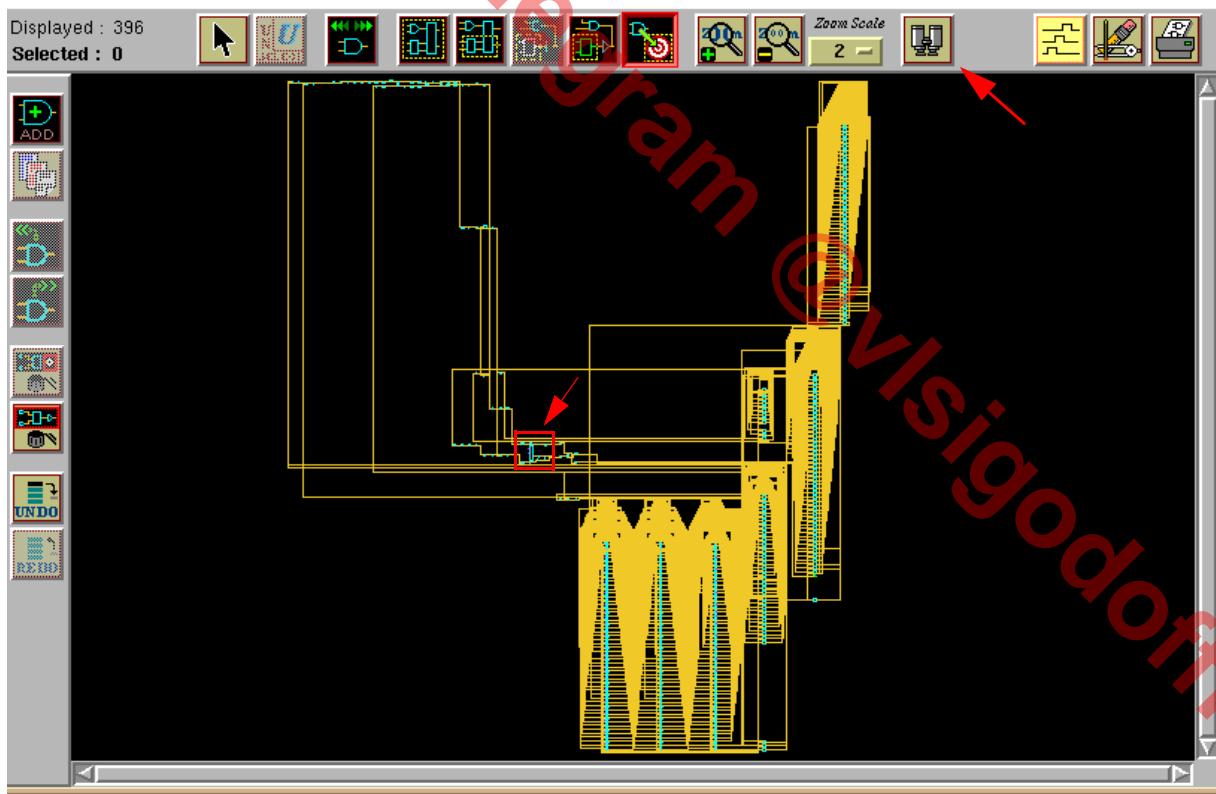
What command do you use? \_\_\_\_\_

What is causing the problem ? \_\_\_\_\_  
\_\_\_\_\_

6. Open DFTInsight.

- What command do you use? \_\_\_\_\_
- Select **Analyze > Faults** to bring up the DFTInsight Faults Analysis dialogue box.
  - Analyze the same node as above by typing it into the Fault entry field (in the middle of the dialogue box).
  - Set the options to Stuck-at-1/Slow-to-Fall (Options... button).
  - Click on the Analyze button.
  - You should see the same message as in 5.c above.

- c. Set up the reporting detail to work with constrained pins and scan cells.
  - i. Choose **Setup > Reporting Detail** in the menu item.
  - ii. Select — Simulated Values Resulting From: Constrained Pins and Scan Cells. Click OK.
  - iii. Click on the Analyze (Graphical) button in the DFTInsight Fault Analysis window. Continue to DFTInsight.
- d. Zoom in to find the instance that is a black box. It is located somewhere to the top and left of the banks of RAMs, ROMs and registers.



- i. Click on the Zoom In button multiple times to expand the displayed view (or use I and O to zoom in and out). Look for a register whose inputs and most of its outputs are undefined.

What is the instance called? \_\_\_\_\_

- ii. You can trace this instance all the way to U\_CTR/U14238/A0. Click on the Zoom In button until instance U\_CTR/U14238 (far right nand gate) is clearly displayed, or click on the Find button and type U\_CTR/U14238 in the Name or Gate ID to be Viewed field.
  - iii. Select the wire connected to gate A0.
  - iv. Trace it back to U\_CTR/U12976, and continue tracing it back through and gates, inverters and buffers until instance I8051\_ALU is clearly displayed.
  - v. Zoom In and Zoom Out as needed for best view.
- e. When you have finished, exit from DFTInsight.
7. Instance I8051\_ALU is a black box that is causing the loss of controllability on instance /U\_CTR/U14238/A0. However, if we make the outputs of this module directly controllable from primary input (PI) faults in its logic cone then it would be controllable. It is possible from within FastScan to determine how much this would improve test coverage before going back to DFTAdvisor to make the improvements.

- a. Go to system mode setup.
- b. Add the extra primary inputs. A dof file, addpi.dof does this for you. Run this file.

What commands are in the dof file? \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

- c. Set system mode to atpg and add all faults.

What two commands do you use?

- i. \_\_\_\_\_

ii. \_\_\_\_\_

d. Run.

e. Report statistics and fill in the following:

**Table 9-11. Report Statistics ATPG**

**Table 9-12.**

| Fault Class            | # faults (coll.) | # faults (total) |
|------------------------|------------------|------------------|
| FU (full)              |                  |                  |
| UO (unobserved)        |                  |                  |
| DS (det_sim)           |                  |                  |
| DI (det_imp)           |                  |                  |
| PU (posdet_untestable) |                  |                  |
| UU (unused)            |                  |                  |
| TI (tied)              |                  |                  |
| BL (blocked)           |                  |                  |
| RE (redundant)         |                  |                  |
| AU (atpg_untestable)   |                  |                  |

**Table 9-13. Report Coverage/Effectiveness ATPG**

**Table 9-14.**

| # sim_pat. | #test_pat. | test_cov. | fault_cov. | atpg_effect. |
|------------|------------|-----------|------------|--------------|
|            |            |           |            |              |

This gives a much more acceptable test coverage.

f. It is possible to report on what happened by using the **report gate** command.

Run the dofile reppi.dof to view the status of the inputs.

The reppi.dof dofile runs the following commands:

```
rep gate U_ALU/des_1[0]
rep gate U_ALU/des_1[1]
rep gate U_ALU/des_1[2]
rep gate U_ALU/des_1[3]
rep gate U_ALU/des_1[4]
rep gate U_ALU/des_1[5]
rep gate U_ALU/des_1[6]
rep gate U_ALU/des_1[7]
rep gate U_ALU/des_2[0]
rep gate U_ALU/des_2[1]
rep gate U_ALU/des_2[2]
rep gate U_ALU/des_2[3]
rep gate U_ALU/des_2[4]
rep gate U_ALU/des_2[5]
rep gate U_ALU/des_2[6]
rep gate U_ALU/des_2[7]
```

- g. Exit FastScan without saving patterns.

Why not save any patterns? \_\_\_\_\_

Next, DFTAdvisor inserts test logic to improve controllability of the logic in the output cone of I8051\_ALU.

DFTAdvisor inserts control logic driven from the newly added scan cells.

8. The modifications necessary for DFTInsight to correct the problem are in the runDFTA2 invocation script.

Run the script runDFTA2.

- a. What is the difference between this script and the last one?

---

---

b. What does this achieve? \_\_\_\_\_  
\_\_\_\_\_

The runDFTA2 script runs the following commands from the dfta2.do dofile:

```
add clo 0 clk rst
set test logic -reset on
set drc handling cl warning
dofile control.dof
set system mode dft
run
insert test logic -number 2 -clock merge -edge merge
report scan chains
write netlist /netlists/gate_scan.v -verilog -replace
write atpg setup atpg -replace
```

The control.dof dofile runs the following commands:

```
add cell model sff -type scan CLK D
add test point /U15/A control mux21_macro test_en -new \
sff
add test point /U30/A control mux21_macro test_en -new \
sff
add test point /U37/A control mux21_macro test_en -new \
sff
add test point /U35/A control mux21_macro test_en -new \
sff
add test point /U29/A control mux21_macro test_en -new \
sff
add test point /U34/A control mux21_macro test_en -new \
sff
add test point /U28/A control mux21_macro test_en -new \
sff
add test point /U23/A control mux21_macro test_en -new \
sff
```

```

add test point /U17/A control mux21_macro test_en -new
sff
add test point /U24/A control mux21_macro test_en -new
sff
add test point /U44/A control mux21_macro test_en -new
sff
add test point /U11/A control mux21_macro test_en -new
sff
add test point /U13/A control mux21_macro test_en -new
sff
add test point /U12/A control mux21_macro test_en -new
sff
add test point /U10/A control mux21_macro test_en -new
sff
add test point /U9/A control mux21_macro test_en -new sff

```

- c. Exit DFTAdvisor.
- d. Run the runFS script to generate patterns.

Fill in the following:

**Table 9-15. Report Statistics ATPG**

**Table 9-16.**

| Fault Class            | # faults (coll.) | # faults (total) |
|------------------------|------------------|------------------|
| FU (full)              |                  |                  |
| UO (unobserved)        |                  |                  |
| DS (det_sim)           |                  |                  |
| DI (det_imp)           |                  |                  |
| PU (posdet_untestable) |                  |                  |
| UU (unused)            |                  |                  |
| TI (tied)              |                  |                  |
| BL (blocked)           |                  |                  |
| RE (redundant)         |                  |                  |

**Table 9-16.**

| Fault Class          | # faults (coll.) | # faults (total) |
|----------------------|------------------|------------------|
| AU (atpg_untestable) |                  |                  |

**Table 9-17. Report Coverage/Effectiveness ATPG****Table 9-18.**

| # sim_pat. | #test_pat. | test_cov. | fault_cov. | atpg_effect. |
|------------|------------|-----------|------------|--------------|
|            |            |           |            |              |

e. Exit FastScan.

Next you compile and simulate the test patterns using ModelSim to check for simulation mismatch.

9. Run ModelSim.

```
shell> run_sim
      OR
shell> ./run_sim
```

What does this script do? \_\_\_\_\_

\_\_\_\_\_

This is the run\_sim invocation script:

```
rm -r work
vlib work
vlog netlists/gate_scan.v
vlog lib/adk.v
vlog verilog_par.pat
vsim I8051_ALL_verilog_par_pat_ctl
```

- Run the simulation. Choose the **Simulate > Run > Run -All** menu item.
  - Select ‘No’ in the Finish Vsim dialogue box after simulation finishes.

- ii. Expand the ModelSim Main window to see all the messages.
  - iii. Observe the mismatch messages in the ModelSim Transcript window.
- b. Exit ModelSim.
10. It is recommended that you start troubleshooting mismatches by first checking to see if any of the failing pattern cells correlate to DRC errors. In this case the place to start is in the log file for the last run in DFTAdvisor. Open /results/dfta2.log and study the DRC violations in this file.

```
shell> cat results/dfta2.log
```

What type of DRC violations are causing the problems?

---

11. There is an easy way of ‘fixing’ patterns to cope with these errors within FastScan. The correction has been made in the script runFS1.
- a. Study the script and identify the change(s) made
- 

- b. Run the runFS1 script.

The runFS1 script runs the following commands from the fs1.do dofile:

```
dofile atpg.dofile
set system mode atpg
add fault -all
set clock_off simulation on
set atpg compression on
run
rep stat
sav pat verilog_ser.pat -verilog -serial -proc -end 1
-rep
sav pat verilog_par.pat -verilog -parallel -proc -rep
exit
```

c. Fill in the following:

**Table 9-19. Report Statistics ATPG****Table 9-20.**

| Fault Class            | # faults (coll.) | # faults (total) |
|------------------------|------------------|------------------|
| FU (full)              |                  |                  |
| UO (unobserved)        |                  |                  |
| DS (det_sim)           |                  |                  |
| DI (det_imp)           |                  |                  |
| PU (posdet_untestable) |                  |                  |
| UU (unused)            |                  |                  |
| TI (tied)              |                  |                  |
| BL (blocked)           |                  |                  |
| RE (redundant)         |                  |                  |
| AU (atpg_untestable)   |                  |                  |

**Table 9-21. Report Coverage/Effectiveness ATPG****Table 9-22.**

| # sim_pat. | #test_pat. | test_cov. | fault_cov. | atpg_effect. |
|------------|------------|-----------|------------|--------------|
|            |            |           |            |              |

You generated patterns with clock\_off simulation on. Ideally, you would find that the failing cells in the simulation correspond to the C6 warnings. The clock\_off simulation will change the ATPG behavior for these cells.

- d. Re-run the simulation to verify that there are no simulation mismatches.
- e. Exit ModelSim.

## Exercise 22: Debugging Mismatches with Automated Tools

In this exercise, you enable FastScan to perform automatic analysis of simulation mismatches when saving patterns using the **-debug** switch.

Automatic analysis of simulation mismatch traces back through the design, locates mismatches between FastScan and ModelSim, and displays mismatch sources in DFTInsight and FastScan.

Automatic analysis does the following:

- Compiles the testbench
- Invokes **vsim** (ModelSim)
- Compares simulation result between **vsim** and FastScan
  - Locates the source(s) of mismatch
- Displays the mismatch source in DFTInsight and ModelSim

### Getting Started

1. Log in to your workstation if you are not already logged in.
2. Change to the \$ATPGNW/lab 9/exercise\_22 directory.

```
shell> cd $ATPGNW/lab9/exercise_22
```

In the first activity, you create test patterns using FastScan. Next, you prepare a simulation library area and compile the design. Then you save patterns and automatically simulate them to check for mismatch.

3. Invoke the design in FastScan from the shell prompt.

```
shell> fastscan gate_scan_8.v -verilog -lib \
.../..../libraries_7_to_9/atpglib -sensitive -nogui -log \
good_run.log -replace
```

- a. Run the dofile atpg\_8.dofile.

What commands does the dofile execute? \_\_\_\_\_

By default, FastScan simulates a single event per test cycle. Default simulation is not appropriate if FastScan is reporting C3/C4/C6 DRC violations when you exit setup mode. This design does have circuitry issues that cause C6 DRC violations. A C6 DRC violation occurs when a clock input of a scannable element and its data line are in the same cone. FastScan assumes that the clock “ON” (active) value is captured into these cells. The exact behavior cannot be predicted without timing information. If this is not correct, you can change the C6 handling by using the **set clock\_off simulation on** command.

- b. What does the **set clock\_off simulation on** command do?

i. \_\_\_\_\_

ii. \_\_\_\_\_

iii. \_\_\_\_\_

- c. Set Clock\_off Simulation ON.

What command do you use? \_\_\_\_\_

4. Go to ATPG system mode, load faults and generate patterns.

- a. Set system mode ATPG.

- b. Load external faults.

```
ATPG> load faults fs_sample_faults.flt -restore
```

What does the **-restore** switch do? \_\_\_\_\_

- c. Generate patterns using the **run** command.
- d. You need to prepare a work library and compile the design.

```
ATPG> system rm -r work  
ATPG> system vlib work  
ATPG> system vlog gate_scan_8.v -v adk.v -v ram.v \  
      -work work
```

- e. Save patterns and automatically simulate them.

```
ATPG> save patterns testpat_p.v -verilog -replace \  
      -debug -display -begin 0 -end 20
```

- f. What does the message say in the shell window? \_\_\_\_\_

Any mismatches? \_\_\_\_\_

- g. Exit FastScan.

5. In this activity, you again save patterns and perform automatic analysis of simulation mismatch. But, this time the aoi22 library cell has had its internal connections changed to produce simulation mismatch.

Invoke FastScan at the shell prompt:

```
shell> fastscan gate_scan_8.v -verilog \  
      ../../libraries_7_to_9/atpglib_e -sensitive -nogui \  
      -log bad_run.log -replace
```

- a. Run the dofile atpg\_8.dofile.
- b. Set Clock\_off Simulation ON.
- c. Go to ATPG mode.
- d. Load external faults.
- e. Generate patterns.

- f. Prepare a simulation library area (work) and compile the design as you did in step 4.d.
- g. Save patterns and automatically simulate them.



**Note**

This takes some time before you observe results.

- h. What does the message say in the shell window? \_\_\_\_\_

Any mismatches? \_\_\_\_\_

FastScan automatically displays the simulation mismatch. The error is in the aoi22 library cell.

Compare the two DFT library models.

atpglib DFT library file:

```
model aoi22(A0, A1, B0, B1, Y) (
    input(A0, A1, B0, B1) ()
    intern(INT_RES_0) (primitive = _and(A0, A1, INT_RES_0));
    intern(INT_RES_1) (primitive = _and(B0, B1, INT_RES_1));
    output(Y) (primitive = _nor(INT_RES_0, INT_RES_1, Y));
)
```

atpglib\_e DFT library file:

```
model aoi22(A0, A1, B0, B1, Y) (
    input(A0, A1, B0, B1) ()
    intern(INT_RES_0) (primitive = _and(A0, A1, INT_RES_0));
//intern(INT_RES_1) (primitive = _and(B0, B1, INT_RES_1));
    intern(INT_RES_1) (primitive = _and(B0, A1, INT_RES_1));
    output(Y) (primitive = _nor(INT_RES_0, INT_RES_1, Y));
)
```

What is the difference between the two DFT library models?

You can view the mismatched waveforms from this simulation in the Wave window in ModelSim.

- i. Select ‘No’ in the Finish Vsim dialogue box after simulation finishes.
- ii. Expand the ModelSim Main window to see all the messages.
- iii. Observe the mismatch messages in the Transcript window.
- iv. Experiment with the Wave window to view the waveforms of the mismatched simulation.
- i. When you have finished, exit ModelSim.
- j. Exit FastScan and exit the lab.

## Test Your Knowledge

1. List the steps you would take to troubleshoot areas of low coverage.  
\_\_\_\_\_  
\_\_\_\_\_
2. What command is used to analyze collapsed faults for the AU fault class?  
\_\_\_\_\_  
\_\_\_\_\_
3. Why are dofiles useful?  
\_\_\_\_\_  
\_\_\_\_\_
4. What command is used when FastScan reports C6 violations?  
\_\_\_\_\_  
\_\_\_\_\_
5. What command is used when FastScan is reporting C3 violations?  
\_\_\_\_\_  
\_\_\_\_\_
6. When saving patterns, what switch option enables automatic analysis of simulation mismatch?  
\_\_\_\_\_  
\_\_\_\_\_

### Lab Summary

Now that you have completed the Troubleshooting DRC and Simulation Mismatch lab, you should know how to do the following:

- Use DFTInsight to debug S1 violations.
- Identify blocks of low coverage.
- Determine the cause of ATPG Untestable (AU) faults.
- Check for simulation mismatch.
- Correct DRC violations.
- Enable FastScan to perform automatic analysis of simulation mismatch when saving patterns.

---

## **NOTES:**

VLSIGOD Join Telegram @vlsigodofficial

---