



Embedded Test Hardware Reference

Software Version 2013.4

December 2013

**© 2009-2013 Mentor Graphics Corporation
All rights reserved.**

This document contains information that is proprietary to Mentor Graphics Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary 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 OF 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 HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

U.S. GOVERNMENT LICENSE RIGHTS: The software and documentation were developed entirely at private expense and are commercial computer software and commercial computer software documentation within the meaning of the applicable acquisition regulations. Accordingly, pursuant to FAR 48 CFR 12.212 and DFARS 48 CFR 227.7202, use, duplication and disclosure by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in the license agreement provided with the software, except for provisions which are contrary to applicable mandatory federal laws.

TRADEMARKS: The trademarks, logos and service marks ("Marks") used herein are the property of Mentor Graphics Corporation or other parties. No one is permitted to use these Marks without the prior written consent of Mentor Graphics or the owner of the Mark, as applicable. The use herein of a third-party Mark is not an attempt to indicate Mentor Graphics as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A current list of Mentor Graphics' trademarks may be viewed at: www.mentor.com/trademarks.

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/

Send Feedback on Documentation: supportnet.mentor.com/doc_feedback_form

Table of Contents

Chapter 1

About This Manual	13
Manual Contents	13

Chapter 2

TAP Controller	15
IEEE 1149.1 TAP Architecture	19
TAP Controller	21
Test Data Registers	23
TAP Controller Ports	26
Ports for Required External Test Pins	28
Ports for Test Mode Functions	29
Ports for the Boundary-Scan Register	31
Ports for User-Defined and BIST Controller Scan Registers	33
Ports for BIST Controllers	34
Ports for User-Defined Test Data Registers	35
Other Ports	36
Test Data Register Timing Waveforms	39
Instruction Register	40
Instruction Register Bit Assignment	41
Status Register Bit Assignment	48
Reserved Instructions	49
Timing Waveforms	49
Internal Data Register	50
Parallel BIST Enable Bits	51
ResetTest	52
SetTest	52
TCKLow	52
HoldShiftHigh	52
SuppressUpdateDR	52
IOTest Bits	52
UserDR Bits	53
The CPU Interface	53
CPU Interface Ports on the TAP	56
TDI TDO Data Padding	59

Chapter 3

WTAP Controller	61
WTAP Controller Architecture	64
Interface With Top-Level TAP	66
WTAP Controller Ports	70
TAP Interface Ports	71

Ports for the Instruction Register	73
Ports for Test Mode Functions	73
Ports for Embedded Test Controllers	74
Ports for User-Defined Scan Registers	76
Output Ports for Lower-Level WTAPs	77
Other Ports	77
Instruction Register	78
Instruction Register Bit Assignment	79
Status Register Bit Assignment	85
Test Data Registers	85
Device ID Register	86
Bypass Register	86
Seed Register	87
Chapter 4	
Boundary-Scan Cells	89
Boundary Scan Overview	90
Boundary-Scan Operating Modes	91
Uses of Boundary Scan	91
Boundary-Scan Cell Features	93
Classes and Subclasses	93
Clocking	96
Boundary-Scan Cells Control and Data Pins	96
Basic Boundary-Scan Cells	98
I: Input Boundary -Scan Cells	99
O: Output Boundary-Scan Cells	99
IO: Bidirectional Boundary-Scan Cells	100
EN: Enable Boundary-Scan Cells	101
Subclasses for Complex Boundary-Scan Cells	102
S: Sample-Only	103
NM: No JTAG Multiplexer	103
H: Holding	104
XO: Auxiliary Data Multiplexer	105
Chapter 5	
logicTest Controller	107
logicTest Controller Architecture	107
The Scan-Chain Router	109
Scan-Chain Configurations	109
Embedded Deterministic Test Controller	117
Shared EDT and LogicBIST Hardware	117
Hardware Variations	117
The BurstMode Logic BIST Controller	122
Logic BIST Controller Modules	123
Clock Connections to the logicTest Controller	126
logicTest Controller Ports	126
Shift Clock Controller	132
Shift Clock Controller Ports	134

Table of Contents

Burst Clock Controller	137
Burst Clock Controller Ports	140
Scan Enable and Clock Enable Controllers	142
Scan Enable Controller Ports	143
Dedicated Isolation Cells	145
Test Point Types	149
Using Dedicated Flip-flops	149
Using Functional Flip-flops	150
 Chapter 6	
Memory BIST Controllers, Collars, and Interfaces	151
Tessent MemoryBIST BIST Architecture	152
NonProgrammable Memory BIST Architecture	153
NonProgrammable Architecture with Collared Memories	154
NonProgrammable Memory BIST Controller	155
Top-Level Ports	157
NonProgrammable Memory Interface	167
Non-Programmable Memory Collar	169
Programmable Memory BIST Architecture	171
Top-Level Signals of Programmable Memory BIST	174
Controller Design Object	181
Memory Interface Design Object	206
BIST Operating Protocol	209
Using Local Comparators	210
Comparators Within the Controller	211
Comparators Within the Memory Collar/Interface	211
Syntax for Local Comparators	211
 Appendix A	
Bit Fields in Instruction Word	213
Instruction Word Bit Field Descriptions	214
OperationSelect Bit Field	215
Add_Reg_A_Equals_B Bit Field	216
Y0AddressCmd Bit Field	216
Y1AddressCmd Bit Field	217
X0AddressCmd Bit Field	217
X1AddressCmd Bit Field	218
ZAddressCmd Bit Field	219
AddressSelectCmd Bit Field	219
WriteDataCmd Bit Field	221
ExpectDataCmd Bit Field	222
RepeatLoop Bit Field	224
InhibitLastAddressCount Bit Field	224
InhibitDataCompare Bit Field	225
InhibitRefresh Bit Field	225
CounterACmd Bit Field	226
DelayCounterCmd Bit Field	226
DisableMemoriesWithoutGroupWriteEnable	226

DisableMemoriesWithoutOutputEnable	227
DisableMemoriesWithoutReadEnable.	227
DisableMemoriesWithoutSelect	228
DisableMemoriesWithoutWriteEnable	228
BranchToInstruction Bit Field	229
NextConditions Bit Field	229
Y0_EndCount	230
Y1_EndCount	230
X0_EndCount	231
X1_EndCount	231
Z_EndCount	231
CounterAEndCount.	232
DelayCounterEndCount	232
RepeatLoopCmd	232
 Appendix B	
Getting Help	233
Documentation.	233
Mentor Graphics Support.	233
 Third-Party Information	
End-User License Agreement	

List of Figures

Figure 2-1. High-Level Diagram of TAP Architecture	20
Figure 2-2. State Diagram for the TAP Controller	21
Figure 2-3. Structure of the Device ID Code	24
Figure 2-4. Mentor Graphics TAP Controller	27
Figure 2-5. Internal Scan Register Shift Timing	40
Figure 2-6. Default Instruction Register Bit Assignments	41
Figure 2-7. Extended Instruction Register Bit Assignments	41
Figure 2-8. 16-Bit Status Register	49
Figure 2-9. Instruction Register Shift Timing	50
Figure 2-10. Internal Data Register Bit Assignments	51
Figure 2-11. Using the Internal Data Register for Parallel BIST	51
Figure 2-12. Controlling the TAP from a CPU Interface	54
Figure 2-13. TAP Controller with a TDR Register without a CPU	55
Figure 2-14. TAP Controller with a TDR Register with a CPU Interface	56
Figure 2-15. DataIn Port Data Format	58
Figure 2-16. DataOut Port Data Format	59
Figure 2-17. ITDR Data Padding to Become a Multiple of CPUDataBudWidth-2	59
Figure 3-1. The WTAP Controller Architecture	65
Figure 3-2. Connections Between WTAP Controllers and TAP Controller	67
Figure 3-3. State Diagram for the WTAP Controller	69
Figure 3-4. WTAP Controller Ports	71
Figure 3-5. WTAP Instruction Register Control Bit Assignments	79
Figure 3-6. Wrapper Instruction Register Status Bit Assignments	85
Figure 3-7. Structure of the Device ID Code	86
Figure 3-8. Structure of the Seed Register	87
Figure 4-1. IC with Boundary Scan	90
Figure 4-2. A Boundary-Scannable Board Design	92
Figure 4-3. I: Basic Input Boundary-Scan Cell	99
Figure 4-4. O: Basic Output Boundary-Scan Cell	100
Figure 4-5. IO: Basic I/O Boundary-Scan Cell	101
Figure 4-6. EN: Basic Enable0 Boundary-Scan Cell	102
Figure 4-7. EN(1): Basic Enable1 Boundary-Scan Cell	102
Figure 4-8. I(S): Sample-Only Input Boundary-Scan Cell	103
Figure 4-9. IO(NM): IO Bcell with JTAG Multiplexers Inside the Pad	104
Figure 4-10. O(H): Output BCell with Holding	105
Figure 4-11. O(XO): Output Bcell with an Auxiliary Data Multiplexer	106
Figure 5-1. The Block Diagram of the logicTest Controller	108
Figure 5-2. Graphical View of Single-Chain Configuration	110
Figure 5-3. Text Presentation of Single-Chain Configuration	111
Figure 5-4. Graphical View of Multi-Chain Configuration	112

Figure 5-5. Text Presentation of Multi-Chain Configuration	113
Figure 5-6. Graphical View of Logic BIST Configuration	114
Figure 5-7. Textual View of Logic BIST Configuration	115
Figure 5-8. Text Presentation of Controller Chain Configuration	115
Figure 5-9. Graphical View of Controller Chain Configuration	116
Figure 5-10. EDT and LogicBIST Hardware Created by Tessent TestKompress and LogicBIST	118
Figure 5-11. Shared EDT and LogicBIST Hardware	119
Figure 5-12. Low-Power Controller	120
Figure 5-13. BurstMode logicBIST Controller Functional Diagram	122
Figure 5-14. Scalable PRPG Architecture	125
Figure 5-15. Scalable MISR Architecture	126
Figure 5-16. Shift Clock Controller	133
Figure 5-17. Burst Clock Controller (Single Clock)	138
Figure 5-18. Burst Clock Controller (Synchronous Clock Group)	139
Figure 5-19. Launch Aligned Synchronous Clocks in Burst Clock Controller	139
Figure 5-20. Capture Aligned Synchronous Clocks in Burst Clock Controller	139
Figure 5-21. Using Demotion Size and Rate in Burst Clock Controller	140
Figure 5-22. Multi-Cycle and False Path Waveforms During Burst Phase	140
Figure 5-23. Scan and Clock Enable Controller	143
Figure 5-24. Input Dedicated Isolation Cell	146
Figure 5-25. Output Dedicated Isolation Cell	147
Figure 5-26. External Scan-Only Dedicated Isolation Cell	147
Figure 5-27. Internal Scan-Only Dedicated Isolation Cell	147
Figure 5-28. Set/Reset Dedicated Isolation Cell	148
Figure 5-29. Force 1 Dedicated Isolation Cell	148
Figure 5-30. Force 0 Dedicated Isolation Cell	148
Figure 5-31. Control Points Implemented Using Dedicated Flip-flops	149
Figure 5-32. Control Points Implemented Using Functional Flip-flops	150
Figure 6-1. High-Level View of the NonProgrammable Memory BIST Architecture with Collared Memories	154
Figure 6-2. NonProgrammable Memory BIST Controller Functional Diagram for a Synchronous Dual Port SRAM	156
Figure 6-3. Connections to the Interface for a Synchronous (Clocked) SRAM Memory	168
Figure 6-4. ROM Connection to the Interface	169
Figure 6-5. Connections to the Collar for a Synchronous (Clocked) SRAM Memory	170
Figure 6-6. ROM Connection to the Collar	171
Figure 6-7. High-Level View of the Programmable Memory BIST Architecture	172
Figure 6-8. Programmable Controller Block Diagram	182
Figure 6-9. BIST FSM States	183
Figure 6-10. Next Instruction Decision Tree	185
Figure 6-11. Independent RepeatLoop Type Example	200
Figure 6-12. Nested-type A RepeatLoop Example	202

List of Figures

Figure 6-13. Nested-type B RepeatLoop Example	205
Figure 6-14. Expanding Write and Expect Data onto the Memory Data Bus	207
Figure 6-15. Memory Interface Block Diagram	208
Figure 6-16. ROM Connection to the Programmable Interface	209
Figure 6-17. The BIST Operating Protocol	210
Figure 6-18. Shared Comparators and Local Comparators	212

List of Tables

Table 2-1. Behavior of the Boundary-Scan Register	24
Table 2-2. Behavior of the Device ID Register	25
Table 2-3. Behavior of the Bypass Register	25
Table 2-4. Optional TAP Ports	26
Table 2-5. State encoding and TAP states	37
Table 2-6. Abbreviations and TAP Controller States	39
Table 2-7. Specifying the bistEnable[M-1:0] Instruction Register Bits	42
Table 2-8. Specifying the clkRoatio[2:0] Instruction Register Bits	43
Table 2-9. Specifying the OP[2:0] Instruction Register Bits	44
Table 2-10. Specifying the selectJtagIn_BAR and the selectJtagOut_BAR Instruction Register Bits	45
Table 2-11. Specifying the setupMode_BAR[1:0] Instruction Register Bits	46
Table 2-12. Specifying the testMode_BAR,setupMode_BAR[1:0] Instruction Register Bits	47
Table 2-13. IEEE 1149.1 Reserved Instructions	49
Table 3-1. The Control Ports of WTAP Interfacing with TAP	66
Table 3-2. Seven (7) Connections of Control Signals Between WTAP and TAP	67
Table 3-3. TAP FSM States and Corresponding WTAP Control Signal Values	70
Table 3-4. Optional WTAP Ports	70
Table 3-5. Specifying the DRSel[1:0] Instruction Register Bits	80
Table 3-6. Specifying the setupMode[2:0] Instruction Register Bits	81
Table 3-7. Specifying the clkRoatio[2:0] Instruction Register Bits	82
Table 3-8. Behavior of the Device ID Register	86
Table 3-9. Behavior of the Bypass Register	87
Table 3-10. Behavior of the Seed Register	87
Table 4-1. Boundary-Scan Cell Class Attributes	94
Table 4-2. Boundary-Scan Cell Subclass Attributes	94
Table 4-3. Boundary-Scan Data and Control Signals	97
Table 5-1. Configuration Mode Encoding	108
Table 5-2. Shift Clock Selection Control	133
Table 5-3. Dedicated Isolation Cell Types	145
Table 6-1. Port List for the NonProgrammable Memory BIST Controller	158
Table 6-2. Port List for the Programmable Memory BIST Controller	174
Table 6-3. Port List for Memory Interfaces	179
Table 6-4. Triggers To Be Tested	185
Table 6-5. Repeat1 Register Bit Field Assignment	198
Table A-1. Instruction Register Field	214
Table A-2. Number of Bits in the OperationSelect bit field	215
Table A-3. A_Add_Reg_Equals_B Bit Field Decode	216
Table A-4. Y0AddressCmd Bit Field Decode	216
Table A-5. Y1AddressCmd Bit Field Decode	217

List of Tables

Table A-6. X0AddressCmd Bit Field Decode	218
Table A-7. X1AddressCmd Bit Field Decode	218
Table A-8. ZAddressCmd Bit Field Decode	219
Table A-9. AddressSelectCmd Bit Field Programming	220
Table A-10. WriteDataCmd Bit Field Decode	221
Table A-11. ExpectDataCmd Bit Field Decode	222
Table A-12. RepeatLoop Bit Field Decode	224
Table A-13. InhibitLastAddressCount Bit Field Decode	224
Table A-14. InhibitDataCompare Bit Field Decode	225
Table A-15. InhibitRefresh Bit Field Decode	225
Table A-16. CounterACmd Bit Field Decode	226
Table A-17. DelayCounterCmd Bit Field Decode	226
Table A-18. DisableMemoriesWithoutGroupWriteEnable Bit Field Decode	227
Table A-19. DisableMemoriesWithoutOutputEnable Bit Field Decode	227
Table A-20. DisableMemoriesWithoutReadEnable Bit Field Decode	228
Table A-21. DisableMemoriesWithoutSelect Bit Field Decode	228
Table A-22. DisableMemoriesWithoutWriteEnable Bit Field Decode	229
Table A-23. Number of Bits in the BranchToInstruction bit field	229
Table A-24. NextConditions Bit Field Assignment	230
Table A-25. RepeatLoopCmd Decode	232

Chapter 1

About This Manual

The Mentor Graphics tools provide a complete, automated embedded test solution for at-speed testing of logic, embedded memories, mixed-signal blocks, and legacy cores at the chip level; and interconnects and memories at the board level. The Mentor Graphics tools also generate a complete IEEE 1149.1 test access port (TAP), memory BIST controllers, WTAP controllers, and boundary-scan implementation, as well as timing-robust scan.

Mentor Graphics' flexible design objects, coupled with powerful automation tools, reduce test development time and cost, while yielding high-quality tests. This manual describes the hardware used for the LV Flow.

For the complete list of Mentor Graphics Tessent-specific terms, refer to the [Tessent Glossary](#).

Refer to “[Getting Help](#)” for information on support options and related documentation.

Manual Contents

This manual comprises the following sections:

- Chapter 2, “[TAP Controller](#)” describes the IEEE 1149.1 test access port (TAP) architecture, the TAP controller ports, and the instruction register bit assignments. It as well describes CPU interface which allows controlling the TAP and everything connected to the TAP from a CPU bus.
- Chapter 3, “[WTAP Controller](#)” describes the WTAP architecture, the WTAP controller ports, and the Bypass, Device ID, and Instruction Registers.
- Chapter 4, “[Boundary-Scan Cells](#)” provides an overview of boundary scan (its operating modes and uses), describes general boundary-scan cell features (such as clocking and data and control signals) and presents sample boundary-scan cell schematics.
- Chapter 5, “[logicTest Controller](#)” describes the logicBIST controller and scan-chain router, ports for this architecture, and test mode configurations.
- Appendix 2, “[Memory BIST Controllers, Collars, and Interfaces](#)” provides a description of the Tessent MemoryBIST hardware architecture. It provides a high-level view of the memory BIST controller types—NonProgrammable and Programmable, as well as the memory collar/interface.

Chapter 2

TAP Controller

The Mentor Graphics ETAssemble tool create an IEEE 1149.1- compliant test-access port (TAP), including the required instruction register, the required bypass register, the optional device ID register, and an optional Mentor Graphics-specific internal data register.

This chapter describes the 1149.1 TAP architecture, the TAP controller ports, and the bypass, device ID instruction, and internal data registers.

This chapter covers the following topics:

- IEEE 1149.1 TAP Architecture
 - TAP Controller
 - Test-Logic-Reset
 - Run-Test/Idle
 - Select-DR-Scan, Select-IR-Scan
 - Capture-DR, Capture-IR
 - Shift-DR, Shift-IR
 - Exit1-DR, Exit1-IR
 - Pause-DR, Pause-IR
 - Exit2-DR, Exit2-IR
 - Update-DR, Update-IR
 - Test Data Registers
 - Boundary-Scan Register
 - Device ID Register
 - Bypass Register
- TAP Controller Ports
 - Ports for Required External Test Pins
 - tck
 - tdi

- tdo
- tdoEnable
- tms
- trst
- Ports for Test Mode Functions
 - clkRatio_2, clkRatio_1, clkRatio_0
 - diagMode
 - forceDis
 - freezeMode
 - functMode
 - multiEnk
 - setupMode[2:0]
 - tckMode
 - testMode
- Ports for the Boundary-Scan Register
 - bscanFaultDirection
 - bscanFaultEnable
 - bscanFaultSetup
 - bscanOnly
 - captureBscan
 - clockBscan
 - enableClkBscan
 - fromBscan
 - forceDis
 - selectJtagInput
 - selectJtagOutput
 - shiftBscan
 - shiftBscan2Edge
 - toBscan

-
- updateBscan
 - enableUpdateBscan
 - Ports for User-Defined and BIST Controller Scan Registers
 - clockIsCan
 - enableClkIsCan
 - fromBistk
 - shiftIsCan
 - shiftIsCan2Edge
 - toIsCan
 - updateIsCan
 - Ports for BIST Controllers
 - bistEnk
 - holdBist
 - shiftBist
 - shiftBist2Edge
 - Ports for User-Defined Test Data Registers
 - captureDR
 - captureDR2Edge
 - enableClkDR
 - shiftDR
 - shiftDR2Edge
 - updateDR
 - updateDREnable
 - Other Ports
 - devIDIn[31:0]
 - devIDOut[31:0]
 - instruction[N-1:0]
 - LVLogicHigh
 - LVLogicLow

- resetTest
- setTest
- state[3:0]
- status[N-3:0]
- testLogicReset
- updateIR
- updateIREnable
- userBits[P-1:0]
- userBitsNotLatched[P-1:0]
- userDRBits[L-1:0]
- userDRBitsNotLatched[L-1:0]
- userDRReset
- tckInvOut
- tckInvIn
- <userSignals>
- waferBurnInEnable#
- Test Data Register Timing Waveforms
- Instruction Register
 - Instruction Register Bit Assignment
 - ACPULSE, ACTRAIN
 - bistEnable[M-1:0]
 - clkRatio[2:0]
 - forceDis
 - freezeMode
 - invertAsyncTCK
 - OP[2:0]
 - selectJtagIn, selectJtagOut
 - setupMode[1:0]
 - testMode

- user[P-1:0]
 - Status Register Bit Assignment
 - Reserved Instructions
 - Timing Waveforms
- Internal Data Register
 - Parallel BIST Enable Bits
 - UserDR Bits
- The CPU Interface
 - CPU Interface Ports on the TAP
 - Clock
 - ResetN
 - Enable
 - WriteEn
 - DataIn
 - DataOut
 - TDI TDO Data Padding

IEEE 1149.1 TAP Architecture

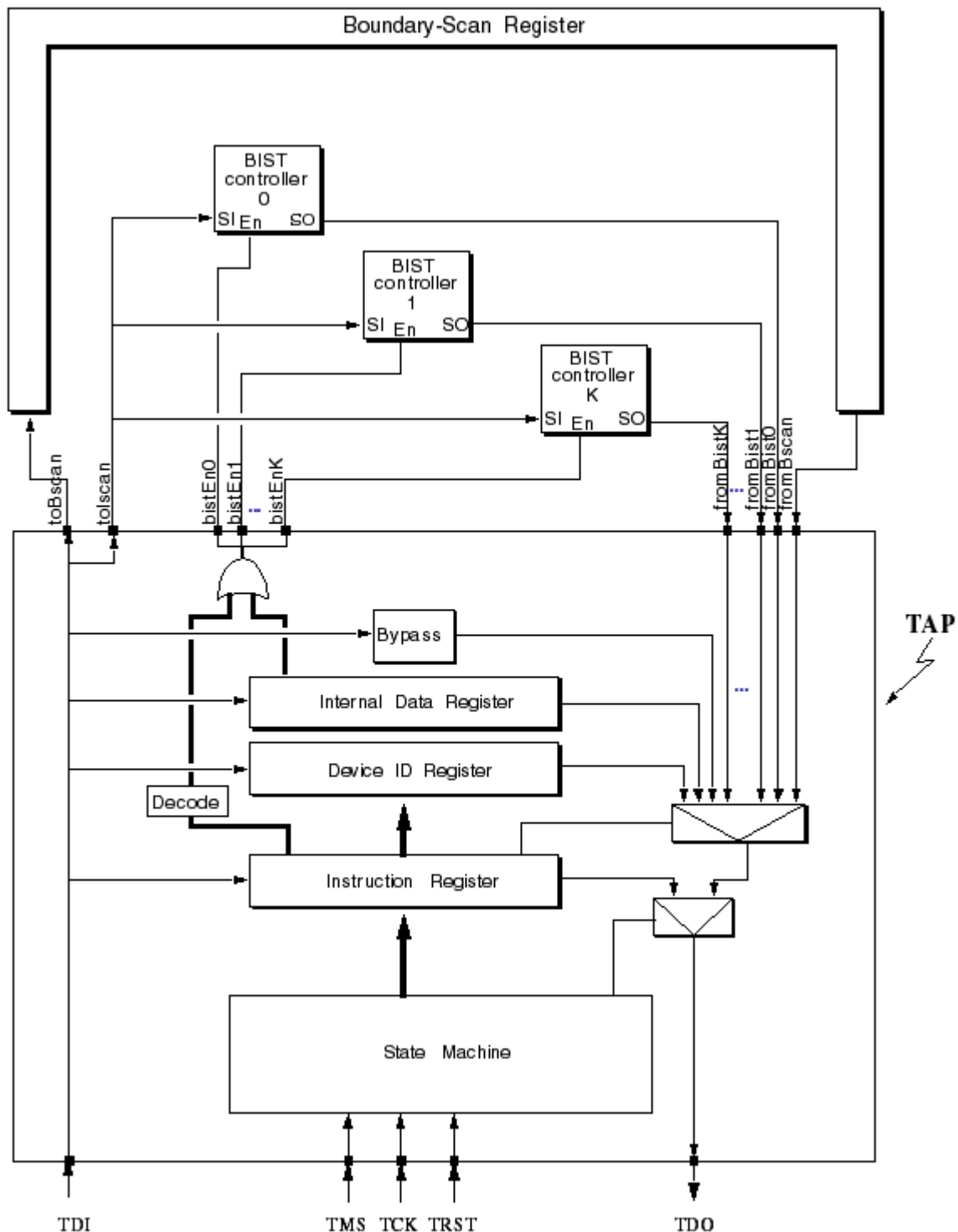
The Mentor Graphics ETAssemble tool creates an IEEE 1149.1 compliant TAP. [Figure 2-1](#) provides a high-level diagram of this TAP along with its interface to a number of BIST controllers (or other embedded test functions), and scan chains on the chip. As shown, the TAP contains the following key blocks:

- State Machine
- Instruction register
- Internal data register
- Device ID register
- Bypass register
- Internal scan chain multiplexing logic

The TAP can individually access and enable up to 128 BIST controllers or embedded test functions. You can scan-initialize and enable these controllers or functions either individually or in parallel with one or more other BIST controllers or embedded test functions.

The following sections of this chapter provide a detailed description of the TAP blocks, as well as a description of how the TAP controls and interfaces with the BIST controllers and the scan registers on the chip.

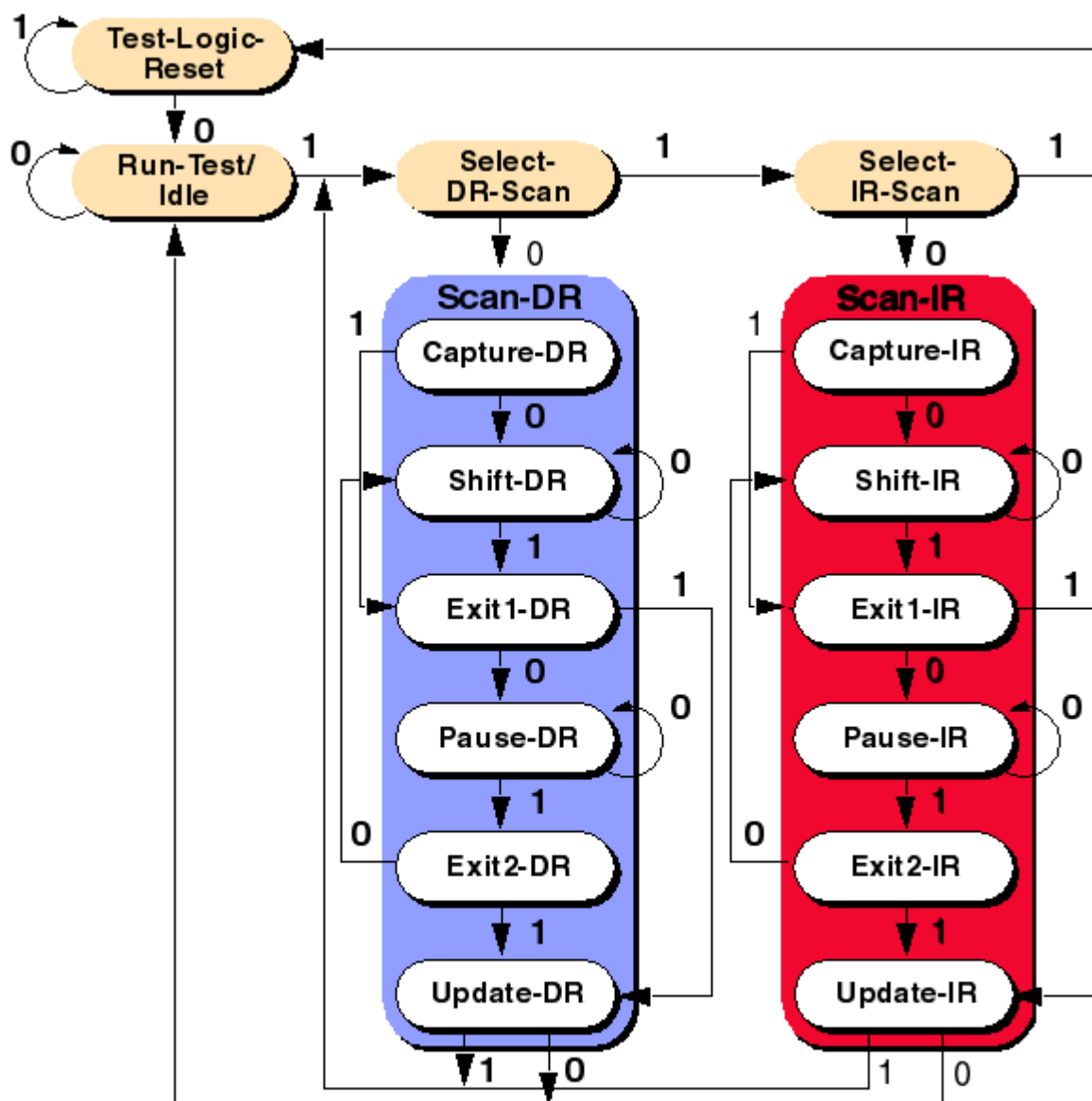
Figure 2-1. High-Level Diagram of TAP Architecture



TAP Controller

The TAP controller contains a finite state machine (FSM) that manages access to all the instruction and data registers within the TAP and within the chip. This state machine cycles through the sixteen states illustrated in Figure 2-2, based on the value present on the TMS signal at each subsequent TCK clock cycle. Each of these states are described following the illustration.

Figure 2-2. State Diagram for the TAP Controller



The values adjacent to each state transition correspond to the TMS signal. The signal TMS, sampled by the TAP controller on the rising edge of TCK, controls the state transitions. For more information on the TMS TAP signal, refer to tms.

Test-Logic-Reset

The Test-Logic-Reset controller state disables all test logic and resets the instruction TAP instruction register to “1...110”. That instruction selects the device ID register if present, or else it selects bypass register.

When the TMS signal remains high for at least five rising edges of the TCK signal, the TAP controller transitions to the Test-Logic-Reset controller state.

Run-Test/Idle

The Run-Test/Idle controller state retains the last state of the test logic. During this state, the TAP controller can execute an internal test, such as BIST, previously selected by the instruction register.

Select-DR-Scan, Select-IR-Scan

The Select-DR-Scan and Select-IR-Scan are temporary controller states, which retain the last state of the test logic. During this state, if the TMS signal is low, the TAP controller initiates a scan sequence for either the selected test data register or the instruction register.

Capture-DR, Capture-IR

The Capture-DR and Capture-IR controller states parallel-load data into either a selected test data register or the instruction register on the rising edge of the TCK signal.

Shift-DR, Shift-IR

The Shift-DR and Shift-IR controller states shift either the selected test data register or the instruction register one stage towards its serial output.

Exit1-DR, Exit1-IR

The Exit1-DR and Exit1-IR are temporary controller states, during which all test data registers and the instruction register retain their previous state. During this state, if the TMS signal is high, the TAP controller terminates the scanning process. If the TMS signal is low, the TAP controller transitions the selected test data register or instruction register to the corresponding Pause controller state.

Pause-DR, Pause-IR

The Pause-DR and Pause-IR controller states temporarily halt the shifting of either the selected test data register or the instruction register. The TAP controller remains paused until the TMS signal goes high.

Exit2-DR, Exit2-IR

The Exit2-DR and Exit2-IR are temporary controller states, during which all test data registers and the instruction register retain their previous state. During this state, if the TMS signal is high, the TAP controller terminates the scanning process. If the TMS signal is low, the TAP controller returns either the selected test data register or the instruction register to the corresponding Shift controller state.

Update-DR, Update-IR

The Update-DR and Update-IR controller states transfer data from each shift-register stage into the corresponding parallel output latch on the falling edge of the TCK signal.

Test Data Registers

The test data registers, shown in [Figure 2-1](#), are user-defined scan chains. The six types of test data registers are as follows:

- **Boundary-scan register** — an optional test data register used to test either logic in the IC or at the board-level interconnect
- **Device ID register** — an optional test data register that provides a unique code to identify the device
- **Bypass register** — a required test data register that is always included in Mentor Graphics' TAP and that shifts data from the test data input port of the TAP to the test data output port of the TAP
- **Internal scan registers** — optional test data registers that consist of user core logic scan chains and are used to perform scan and logic BIST testing
- **Internal data register** — a Mentor Graphics-specific, optional test data register that supplements the instruction register by providing additional BIST control and result gathering capabilities
- **Internal fault insertion register** — a Mentor Graphics-specific, optional test data register used to drive fault insertion control signals

For information about the test data register sequencing, refer to “[invertAsyncTCK](#).”

The following subsections describe the boundary scan, device ID, bypass, and internal scan test data register types. A description of the internal data register is provided in the “[Internal Data Register](#)” section.

Boundary-Scan Register

The optional Boundary-scan test data register tests either the logic within your IC or the board-level interconnect. This register is a single scan chain that contains boundary-scan cells that

connect to all input and output pins, as well as output enable or direction control signals associated with the I/O pins. For information on the provided boundary-scan cells, refer to “Boundary-Scan Cells.”

The following OP[2:0] instruction register bit values provide access to the Boundary-scan register:

- 000
- 001
- 011

For more information, refer to OP[2:0].

Table 2-1 describes how the TAP controller states affect the operation of the Boundary-scan register.

Table 2-1. Behavior of the Boundary-Scan Register

During this TAP Controller State...	The Boundary-Scan Register...
Test-Logic-Reset	Retains the last state
Capture-DR	Loads data at cell input into the shift-register
Shift-DR	Shifts data through the boundary-scan cells
Update-DR	Transfers data from the shift-register to the output latch
All other states	Retains the last state

Device ID Register

The optional Device ID test data register provides a unique code that identifies the device. The Device ID register contains 32 parallel-in, serial-out shift-register stages that identify the version number, the part number, and the manufacturer of your IC.

Figure 2-3 illustrates the structure of the Device ID code.

Figure 2-3. Structure of the Device ID Code

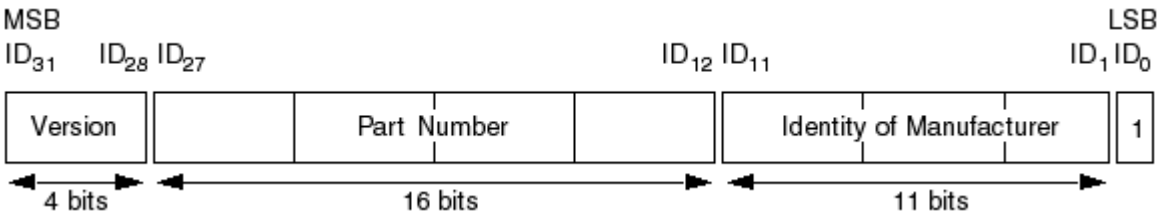


Table 2-2 describes how the TAP controller states affect the behavior of the Device ID register.

Table 2-2. Behavior of the Device ID Register

During this TAP Controller State...	The Device ID Register...
Test-Logic-Reset	Retains the last state
Capture-DR	Loads the device ID code
Shift-DR	Shifts the ID towards the tdo TAP port
All other states	Retains the last state

When the Device ID register is present in a design, the Test-Logic-Reset controller state automatically selects this register. If your design does not contain a Device ID register, the TAP controller selects the Bypass register. By examining the first bit that the TAP controller shifts out of an IC, you can tell whether or not a Device ID register is present. If the first bit is 0, the IC does not include a Device ID register and the TAP controller selected the Bypass register instead.

Bypass Register

The required Bypass test data register, consisting of a single shift-register stage, shifts information from the tdi port to the tdo TAP port without interfering with the normal operation of your IC.

The Bypass register is selected by assigning 111 to the OP[2:0] instruction register bits.

For more information, refer to [OP\[2:0\]](#).

Once selected, the Bypass register loads a constant logic 0 into the shift-register stage on a rising edge of TCK during the Capture-DR controller state. [Table 2-3](#) describes how the TAP controller states affect the shift-register stage.

Table 2-3. Behavior of the Bypass Register

During this TAP Controller State...	The Bypass Register...
Test-Logic-Reset	Retains the last state
Capture-DR	Loads a logic-zero
Shift-DR	Shifts towards the tdo TAP port
All other states	Retains the last state

The Bypass register also provides a one-cycle delay between the tdi TAP port and the tdo TAP port. During a board-level test, the Bypass register reduces the access time to the test data registers on other ICs.

TAP Controller Ports

Mentor Graphics' ETAssemble tool creates an IEEE 1149.1-compliant TAP, as illustrated in [Figure 2-4](#), based upon user input. However, depending on which properties you specify, your generated TAP could be different from the TAP illustrated in [Figure 2-4](#). Descriptions of the TAP ports follow the illustration.

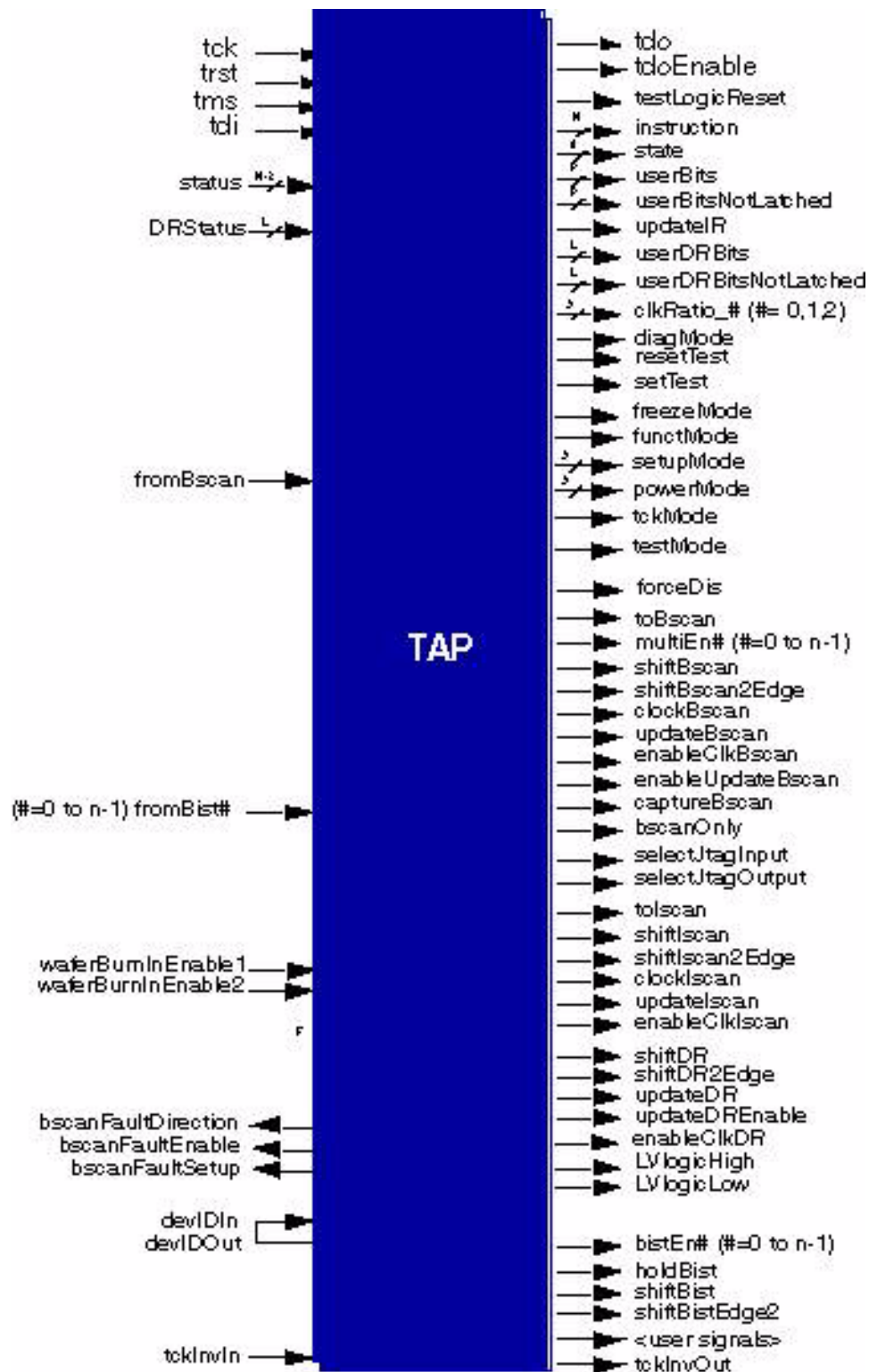
[Table 2-4](#) provides a correlation between file properties and resulting TAP ports.

Table 2-4. Optional TAP Ports

Property	Resulting Ports
NumberUserBits P	userBits[P-1:0] userBitsNotLatched[P-1:0] updateIR status[P-1+J:J] ¹
NumberUserDRBits L	userDRBits[L-1:0] userDRBitsNotLatched[L-1:0] DRStatus[L+K+6:0] ²
NumberBistPorts K ³	bistEn0, bistEn1,..., bistEnK-1 fromBist0, fromBist1,..., fromBistK-1 multiEn0, multiEn1,...,multiEnK-1
UserSignal wrapper	<userSignal#>

1. J depends on the value of other properties. See the [Instruction Register Bit Assignment](#) for details.
2. Default for DRStatus is [K+6:0].
3. K defaults to 5.

Figure 2-4. Mentor Graphics TAP Controller



Ports for Required External Test Pins

IEEE 1149.1 requires five dedicated test pins, TCK, TDI, TDO, TMS, and TRST, to which the TAP signals TCK, TDI, TDO, TMS, and TRST connect. This subsection discusses the TAP ports that correspond to these TAP signals.

The Assemble tool connects the tck, tdi, tdo, tms, and trst TAP ports to the respective test pins TCK, TDI, TDO, TMS, and TRST.

tck

The tck TAP port controls the following actions:

- Sampling the TDI and TMS TAP signals
- Updating the TDO TAP signal

The TAP controller samples the TDI and TMS TAP signals on the rising edge of the TCK TAP signal and updates the TDO TAP signal on the falling edge of TCK.

The test clock has the following properties:

- Single phase
- Free-running
- Frequency range from 0 to 10 MHz (as hard coded in the BSDL file)

tdi

The tdi TAP port is the serial input for the instruction register and the test data registers. The TAP controller samples tdi on the rising edge of the TCK TAP signal.

tdo

The tdo TAP port is the serial output for the instruction register and the test data registers, and is at high-impedance except during shifting. The TAP controller updates tdo on the falling edge of the TCK TAP signal.

tdoEnable

The tdoEnable TAP port connects to the enable port of the tdo pad. This signal goes active one full clock cycle before valid data are output on tdo, and remains active until one clock cycle after the last bit is available on tdo.

tms

The tms TAP port controls the finite state machine (FSM). The TAP controller samples tms on the rising edge of the TCK TAP signal. For more information, refer to the “[TAP Controller](#)” section.

trst

The trst TAP port connects to the external pin TRST, an active-low, asynchronous reset for the TAP controller. When the TRST TAP signal is low, the TAP controller immediately enters the Test-Logic-Reset state.

Note



If you do not plan to specify a TRST primary pin in your design, you must set the **TRST** property in the **TestPortConnections** wrapper to point to the output pin of your embedded power-on reset circuit. This is a requirement of the IEEE 1149.1 standard.

Ports for Test Mode Functions

This subsection defines ports for test mode functions.

clkRatio_2, clkRatio_1, clkRatio_0

The clkRatio_# TAP ports are output ports that selects the test clock source. The TAP controller generates the clkRatio_# TAP signals by inverting the output from the clkRatio[2:0] instruction bits. For more information on the clkRatio[2:0] instruction register bits, refer to [clkRatio\[2:0\]](#).

diagMode

The diagMode TAP port controls when the TAP controller FSM suppresses the capture state. When the capture state is suppressed, the TAP controller (during scan-Thru-TAP testing) or the logicBIST controller (during logic BIST testing) scans out the contents of the test data registers without corrupting the data.

The TAP controller generates the diagMode TAP signal by inverting the output from the instruction bit diagMode.

freezeMode

The freezeMode TAP port controls whether or not the clock prescaler stops the clock after the capture of the last vector during logic BIST mode.

The TAP controller generates the freezeMode TAP signal by inverting the output from the freezeMode instruction bit. For more information on the freezeMode instruction register bit, refer to “[freezeMode](#).”

functMode

The functMode TAP port generates an output signal, functMode, which indicates the circuit is in normal operating mode.

multiEnk

The multiEnk ports on the TAP indicates that the logicTest controller connected to the bistEnk port is in multi mode. It is used to control the auxiliary scan-in and scan-out ports of the corresponding logicTest controller.

setupMode[2:0]

The setupMode[2:0] TAP ports control the run or setup mode of the currently selected BIST controller.

The TAP controller generates the setupMode[2:0] TAP signals by inverting the output from the setupMode[1:0] instruction bits and the inverted testMode instruction bit. The setupMode[1:0] instruction bits control the ports setupMode[1:0] while the testMode instruction bit controls the port setupMode[2].

For more information on the setupMode[1:0] instruction register bits, refer to “[setupMode\[1:0\]](#)”. The port setupMode[2] is only used by the logicTest controller. Refer to [logicTest Controller Ports](#) for information on how the complete setupMode port is used.

tckMode

The tckMode TAP port generates an output signal, tckMode, which indicates that the clkRatio[2:0] TAP signal selected the test clock.

testMode

The testMode TAP port generates an output signal, testMode, which indicates that the circuit is in test mode. This signal is not used when a top-level logicTest controller is used, but is provided for use with third-party scan/ATPG.

If the design has a top-level logicTest controller, then the INT_TM output port on the logicTest controller is used instead of the testMode port. For information on the logicTest controller, refer to Chapter 5, “[logicTest Controller](#)” in this manual.

Ports for the Boundary-Scan Register

This subsection defines ports that control the Boundary-scan register.

bscanFaultDirection

The bscanFaultDirection TAP port controls in which direction bidirectional fault insertion boundary scan cells are faulted.

bscanFaultEnable

The bscanFaultEnable TAP port enables faulting on all fault insertion boundary scan cells.

bscanFaultSetup

The bscanFaultSetup TAP port sets all fault insertion cells into setup mode.

bscanOnly

The bscanOnly TAP port indicates that only the boundary-scan chain is currently selected. It is used to feed the proper clock to the boundary- scan chain.

captureBscan

The captureBscan TAP port feeds the clock input of the capture latch within synchronous boundary-scan cells. The generated captureBscan TAP signal is high whenever the TAP is in the Capture_DR state and the boundary-scan chain is selected.

clockBscan

The clockBscan TAP port feeds the boundary-scan cell clock input. The generated clockBscan TAP signal is a gated, inverted version of the TCK TAP signal.

enableClkBscan

The enableClkBscan TAP port enables the boundary-scan cell clock inputs. The generated enableClkBscan TAP signal can be routed directly to any flip-flop on the boundary-scan chain to gate the TCK TAP signal.

fromBscan

The fromBscan TAP port connects to the scan out port of the last cell in the boundary-scan chain.

forceDis

The forceDis TAP port tri-states all of the three state pads.

The TAP controller generates the forceDis TAP signal by inverting the output from the forceDis instruction bit. For more information on the forceDis instruction register bit, refer to “[forceDis](#).”

selectJtagInput

The selectJtagInput TAP port switches control of the core between the Boundary-scan register and the primary input pins. (When the selectJtagInput TAP signal is active-high, the Boundary-scan register sources data to the core.)

The TAP controller generates the selectJtagInput TAP signal by inverting the output from the selectJtagInput instruction bit. For more information on the selectJtagInput instruction register bit, refer to “[selectJtagIn](#), [selectJtagOut](#).”

selectJtagOutput

The selectJtagOutput TAP port switches control of the primary output pins between the Boundary-scan register and the chip logic. (When the selectJtagOutput TAP signal is active-high, the Boundary-scan register controls the primary output pins.)

The TAP controller generates the selectJtagOutput TAP signal by inverting the output from the selectJtagOutput instruction bit. For more information on the selectJtagOutput instruction register bit, refer to “[selectJtagIn](#), [selectJtagOut](#).”

shiftBscan

The shiftBscan TAP port configures the Boundary-scan register into a shift register when the generated shiftBscan TAP signal is active-high. The shiftBscan TAP signal is a delayed version of the shiftBscan2Edge TAP signal; it is delayed by half a TCK-signal period.

shiftBscan2Edge

The shiftBscan2Edge TAP port configures the Boundary-scan register into a shift register when the generated shiftBscan2Edge TAP signal is active-high.

toBscan

The toBscan TAP port connects to the scan in port of the first cell in the boundary-scan chain.

updateBscan

The updateBscan TAP port controls when the TAP updates the Boundary-scan register. When the generated updateBscan TAP signal is active-high, the TAP updates the boundary-scan

output latch. This reduces race conditions by holding the data in the Boundary-scan register while the TAP shifts the data.

enableUpdateBscan

The enableUpdateBscan TAP port is the same as updateBscan except that it is decoded one tck clock cycle earlier to use with boundary-scan cells which do not use a latch for the update register but a flip-flop which runs off tckInv with a holding multiplexer controlled by enableUpdateBscan.

Ports for User-Defined and BIST Controller Scan Registers

This subsection defines ports that control User-Defined and BIST Controller Scan registers.

clockIsScan

The clockIsScan TAP port is used to clock user-defined scan chains and BIST Controller Scan registers. The generated clockIsScan TAP signal is a gated, inverted version of the TCK TAP signal.

clockIsScan is used when [TestClockSource](#) is set to TCK, as is the case for scanVerify. clockIsScan is a gated clock in accordance with 1149.1 protocol:

- When the clock is running and shiftIsScan is low, then clockIsScan captures.
- When the clock is running and shiftIsScan is high, then clockIsScan shifts.
- When the clock is not running, then clockIsScan holds.

Mentor Graphics implemented clockIsScan as an inverted gate. 1149.1 is a two-edge protocol. Inputs are retimed on the positive edge of TCK. Outputs come out on the negative edge. Mentor Graphics chose to place retiming flip-flops on inputs. Because retiming flip-flops are not placed on the outputs, clockIsScan had to be inverted.

enableClkIsScan

The enableClkIsScan TAP port enables the clock inputs to User-Defined and BIST Controller Scan registers. The generated enableClkIsScan TAP signal can be routed directly to any flip-flop on a scan chain to gate the TCK TAP signal.

fromBistk

The fromBistk TAP port connects to the Q port of the last flip-flop of the scan chain within the BIST controller enabled by the bistEnk TAP signal. The fromBist0 port is reserved for use with the top-level logicTest controller when a top-level logicTest controller is present.

shiftIsScan

The shiftIsScan TAP port configures the User-Defined and BIST Controller Scan registers into shift registers when the generated shiftIsScan TAP signal is active-high. The shiftIsScan TAP signal is a delayed version of the shiftIsScan2Edge TAP signal; it is delayed by half a TCK-signal period.

shiftIsScan2Edge

The shiftIsScan2Edge TAP port configures the User-Defined and BIST Controller Scan registers into shift registers when the generated shiftIsScan2Edge TAP signal is active-high.

toIsScan

The toIsScan TAP port connects to the scan-data (SD) port of the first flip-flop of the user-defined and all BIST controller scan chains.

updateIsScan

The updateIsScan TAP port reduces race conditions by holding the data in the User-Defined and BIST Controller Scan registers while the TAP captures data.

Ports for BIST Controllers

This subsection defines ports that control the BIST controllers or other embedded test functions.

bistEnk

The bistEnk TAP port, when selected by the bistEnable[M-1:0] instruction register bits or the BIST_EN[k] internal data register bit, generates an output signal, bistEnk, which enables the BIST controller connected between the toIsScan and fromBistk TAP ports. The bistEn0 signal is reserved for use with the top-level logicTest controller when the top-level logicTest controller is present.

For more information on the bistEnable[M-1:0] instruction register bits, refer to “[bistEnable\[M-1:0\]](#).” For more information on the BIST_EN[k] internal data register bit, refer to “[Parallel BIST Enable Bits](#).”

holdBist

The holdBist TAP port controls the holding mode of the BIST controllers. The holdBist TAP signal is used to instruct the selected BIST controller to retain its state while its internal registers are being scanned.

shiftBist

The shiftBist TAP port controls the shifting mode of the logic BIST and memory BIST controllers. The shiftBist signal is a delayed version of the shiftBist2Edge TAP signal; it is delayed by half a TCK-signal period.

shiftBist2Edge

The shiftBist2Edge TAP port controls the shifting mode of the logic BIST and memory BIST controllers which flops operate on the Leading Edge (LE) of clockIsCan.

Ports for User-Defined Test Data Registers

This subsection defines ports that control all User-Defined test data registers.

captureDR

The captureDR TAP port enables the capture of the selected User-Defined test data register when the generated captureDR TAP signal is active high. The captureDR signal is a delayed version of the captureDR2Edge TAP signal. It is delayed by half a TCK signal period.

captureDR2Edge

The captureDR2Edge TAP port enables the capture of the selected User-Defined test data register when the generated captureDR2Edge TAP signal is active high.

enableClkDR

The enableClkDR TAP port enables the clock inputs of the selected User-Defined test data register. The generated enableClkDR TAP signal can be routed directly to any flip-flop on the selected User-Defined test data register to gate the TCK TAP signal

shiftDR

The shiftDR TAP port configures the selected User-Defined test data register into a shift register when the generated shiftDR TAP signal is active-high. The shiftDR TAP signal is a delayed version of the shiftDR2Edge TAP signal; it is delayed by half a TCK-signal period.

shiftDR2Edge

The shiftDR2Edge TAP port configures the selected User-Defined test data register into a shift register when the generated shiftDR2Edge TAP signal is active-high.

updateDR

The updateDR TAP port reduces race conditions by holding either the data in the selected User-Defined test data register while the TAP shifts the data.

updateDREnable

The updateDREnable TAP output port is the same as updateDR, except that it is decoded one-half TCK clock cycle earlier. It can be used in custom test data register implementations where the update latch stage is replaced by a flip-flop with a holding multiplexer.

Other Ports

This subsection defines other ports that relate to the TAP controller.

devIDIn[31:0]

The devIDIn[31:0] TAP ports are a bused input port that supplies the ID code to the device ID register.

devIDOut[31:0]

The devIDOut[31:0] TAP ports is a bused output port that allows the device ID to be observed or programmed.

DRstatus[L-1:0]

The DRstatus[L-1:0] TAP port is a bused input port that feeds the userDR bits of the internal data register. This port can be connected to any external signals for capture and examination.

instruction[N-1:0]

The instruction[N-1:0] TAP ports are N instruction register outputs that control test mode functions. Note that the port carries the raw Instruction register values (without inversion). For more information on the instruction register, refer to “[Instruction Register](#).”

LVLogicHigh

LVLogicHigh is a logic “1” TAP output port.

LVLogicLow

LVLogicLow is a logic “0” TAP output port.

resetTest

The resetTest TAP port provides a reset signal for flip-flops during asynchronous reset tests.

setTest

The setTest TAP port provides a set signal for flip-flops during asynchronous set tests.

state[3:0]

The state[3:0] TAP ports indicate the current TAP controller state. The state encoding is as follows.

Table 2-5. State encoding and TAP states

Encode Value	Corresponding TAP State
1111	Test-Logic-Reset
1100	Run-Test/Idle
0111	Select-DR-Scan
0110	Capture-DR
0010	Shift-DR
0001	Exit1-DR
0011	Pause-DR
0000	Exit2-DR
0101	Update-DR
0100	Select-IR-Scan
1110	Capture-IR
1010	Shift-IR
1001	Exit1-IR
1011	Pause-IR
1000	Exit2-IR
1101	Update-IR

Figure 2-2 shows the state diagram for the TAP controller.

status[N-3:0]

The status[N-3:0] TAP port is a bused input port that can be connected to any signal for capture and examination.

testLogicReset

The testLogicReset TAP port indicates when the TAP is in the Test-Logic_Reset state. The testLogicReset TAP signal can be used to reset user defined logic.

updateIR

The updateIR TAP port indicates when the instruction register bits are clocked into the register's update latches. The updateIR TAP signal is provided to clock user-provided latches fed by the userBitsNotLatched[] signals.

updateIREnable

The updateIREnable TAP port is the same as updateIR except that it is decoded one-half tck clock cycle earlier.

userBits[P-1:0]

The userBits[P-1:0] TAP port is a bused output port connected inside the TAP to all of the user bits that you added. The ETAssemble tool adds user bits to the instruction register, if you specified [NumberUserBits](#) in the ETAssemble configuration file.

userBitsNotLatched[P-1:0]

The userBitsNotLatched[P-1:0] TAP port provides non latched versions of the userBits[P-1:0] signals. That is, these signals originate from before the instruction register update latches.

userDRBits[L-1:0]

The userDRBits[L-1:0] TAP port is a bused output port connected inside the TAP to all of the userDR bits that you added. For details on userDR bits, refer to [NumberUserDRBits](#) in the ETAssemble configuration file.

userDRBitsNotLatched[L-1:0]

The userDRBitsNotLatched[L-1:0] TAP port provides non latched versions of the userDRBits[L-1:0] signals. That is, these signals originate from before the internal data register update latches.

userDRReset

The userDRReset TAP port provides an external synchronous reset signal for the userDR bits.

tckInvOut

The tckInvOut TAP port is simply the tck input port inverted.

tckInvIn

The tckInvIn TAP port is used to clock all flip flop inside the TAP controller. This port is the only clock domain used inside the TAP. There are no gated clocks and the TAP is synthesized using a set_dont_touch_clock_network on this port. The default source of this port is tckInvOut. If you need to insert a clock distribution macro for the tckIncIn clock network, you can insert it between the tckInvOut and tckInvIn ports.

<userSignals>

The name of these ports are user specified using [UserSignal](#) in the ETAssemble configuration file. The signals are decoded off the user IR bits or the user DR bits and are registered before coming out of the TAP. A signal decoded off the user DR bits are updated using the updateDR signal. A signal decoded off the user IR bits is updated using the updateIR signal.

waferBurnInEnable#

The waferBurnInEnable# TAP input ports are generated when a chip select pin is present to enable the wafer-level or package-level burn-in tests. Up to two waferBurnInEnable ports will be created on the TAP controller.

Test Data Register Timing Waveforms

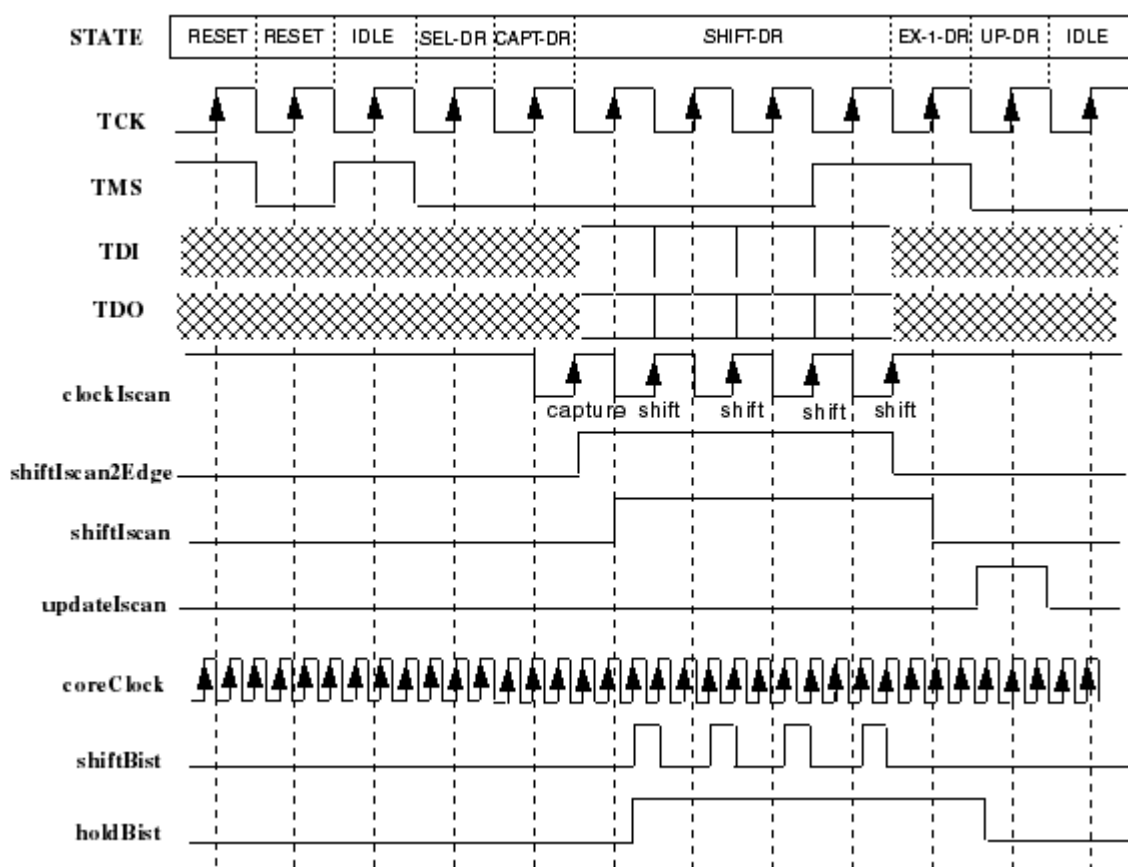
The TAP controller controls access to the internal scan register through 16 TAP controller states. These states affect when the TAP controller loads data into the internal scan register and when the TAP reads out the data.

[Figure 2-5](#) shows the shift timing for the internal scan register according to the TAP controller states. [Table 2-6](#) identifies the TAP controller states with the listed abbreviations. For more information, refer to the “[TAP Controller](#)” section.

Table 2-6. Abbreviations and TAP Controller States

Abbreviation	Corresponding TAP Controller State
RESET	Test-Logic-Reset.
IDLE	Run-Test/Idle
SEL-DR	Select-DR-Scan
SEL-IR	Select-IR-Scan TAP
CAPT	Capture-DR
SHIFT	Shift-DR
EX-1	Exit1-DR
UPDATE	Update-DR

Figure 2-5. Internal Scan Register Shift Timing



Instruction Register

The instruction register selects test data registers and controls internal test data structures. [Figure 2-6](#) illustrates the register bit assignments for the default 16-bit instruction register. The instruction register is expanded beyond 16-bits if certain user options are specified. A fully expanded instruction register is illustrated in [Figure 2-7](#).

Finally, up to 16-user-bits also can be added to the instruction register. Descriptions of the default and extended instruction register bits are provided in this section.

Figure 2-6. Default Instruction Register Bit Assignments

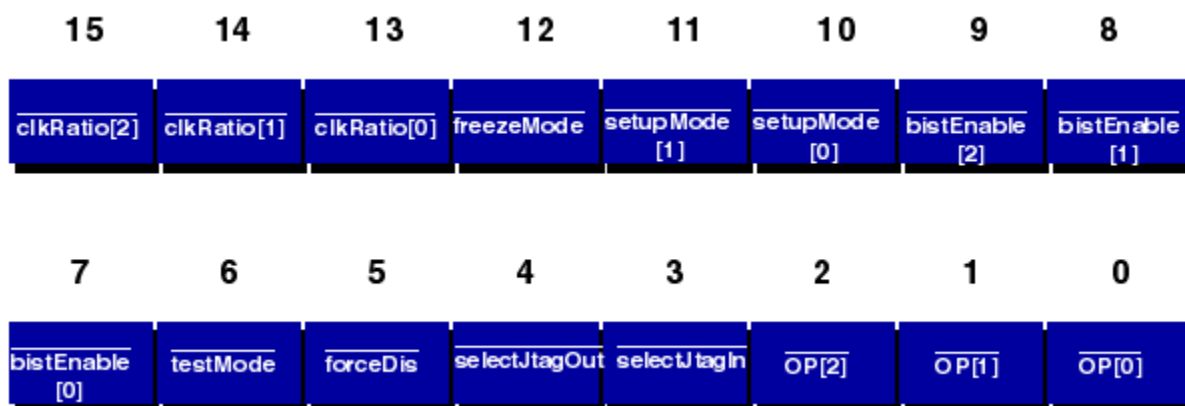


Figure 2-7. Extended Instruction Register Bit Assignments



Instruction Register Bit Assignment

This section describes the assignment for instruction register bits.

ACPULSE, ACTRAIN

These two optional IR bits only exist for designs featuring Mentor Graphics' 1149.6 product. For details, refer to [Support for IEEE 1149.6 Boundary Scan](#).

bistEnable[M-1:0]

As described in “[IEEE 1149.1 TAP Architecture](#),” the Mentor Graphics TAP can individually address and enable multiple BIST controllers. The bistEnable[M-1:0] instruction register bits are used to select one of these controllers. The number of bistEnable bits defaults to 3, but can be expanded to 8-bits depending on the number of BIST ports required.

[Table 2-7](#) lists the bistEnable[M-1:0] bit values necessary to address various BIST ports. Refer to the HSDL file if you have more BIST ports than shown and want to find out the bit encoding.

The first three bistEnable codes are unused and the bistEn0 port is reserved for use with a top-level logicBIST controller (if present). The other bistEn ports can be assigned to any other BIST controllers in the chip.

Table 2-7. Specifying the bistEnable[M-1:0] Instruction Register Bits

BIST Port Enabled	bistEnable[M-1:0] bit values		
	M 3	M 4	M 5
bistEn0	0 1 1	1 0 1 1	1 1 0 1 1
bistEn1	1 1 0	1 1 1 0	1 1 1 1 0
bistEn2	1 0 0	1 1 0 0	1 1 1 0 0
bistEn3	0 1 0	1 0 1 0	1 1 0 1 0
bistEn4	0 0 0	1 0 0 0	1 1 0 0 0
bistEn5		0 1 1 1	1 0 1 1 1
bistEn6		0 1 0 1	1 0 1 0 1
bistEn7		0 0 1 1	1 0 0 1 1
bistEn8		0 0 0 1	1 0 0 0 1
bistEn9		0 1 1 0	1 0 1 1 0
bistEn10		0 1 0 0	1 0 1 0 0
bistEn11		0 0 1 0	1 0 0 1 0
bistEn12		0 0 0 0	1 0 0 0 0
bistEn13			0 1 1 1 1
bistEn14			0 1 1 0 1
bistEn15			0 1 0 1 1
bistEn16			0 1 0 0 1
bistEn17			0 1 1 1 0
bistEn18			0 1 1 0 0
bistEn19			0 1 0 1 0
bistEn20			0 1 0 0 0
bistEn21			0 0 1 1 1
bistEn22			0 0 1 0 1
bistEn23			0 0 0 1 1
bistEn24			0 0 0 0 1
bistEn25			0 0 1 1 0

Table 2-7. Specifying the bistEnable[M-1:0] Instruction Register Bits (cont.)

BIST Port Enabled	bistEnable[M-1:0] bit values		
	M 3	M 4	M 5
bistEn26			0 0 1 0 0
bistEn27			0 0 0 1 0
bistEn28			0 0 0 0 0

clkRatio[2:0]

The clkRatio[2:0] instruction register bits specify the clock that is used to perform a scan test or run any of the BIST controllers,

Table 2-8 lists the clkRatio[2:0] values and the corresponding clock.

Table 2-8. Specifying the clkRatio[2:0] Instruction Register Bits

clkRatio_BAR[2:0]	Selected Clock
1 1 1	functionalMode
1 1 0	sysClock
1 0 1	sysClock/2
1 0 0	sysClock/4
0 1 1	clkIsCan
0 1 0	sysClock PLL
0 0 1	sysClock/2 PLL
0 0 0	sysClock/4 PLL

Specifying the Clock for Logic BIST

The clock source that you select for logic BIST depends upon the [TestClockSource](#) property option in the ETVVerify configuration file.

- If **-clockPeriod TestClk**, the testbench sets clkRatio[2:0] to 011.
- If **-clockPeriod SysClk**, the testbench sets clkRatio[2:0] to 110.
- If **-clockPeriod SysClk_2**, the testbench sets clkRatio[2:0] to 101.
- If **-clockPeriod SysClk_4**, the testbench sets clkRatio[2:0] to 100.
- If **-clockPeriod PLLSysClk**, the testbench sets clkRatio[2:0] to 010.
- If **-clockPeriod PLLSysClk_2**, the testbench sets clkRatio[2:0] to 001.

- If **-clockPeriod PLLSysClk_4**, the testbench sets clkRatio[2:0] to 000.

forceDis

The forceDis instruction register bit tri-states all of the three state pads. This instruction register bit is typically used in logic BIST or scan-through-TAP testing.

freezeMode

The freezeMode instruction register bit controls whether or not the clock prescaler stops the clock after the capture of the last vector.

invertAsyncTCK

This IR bit controls the polarity inversion of the TAP asyncIntTck output fanning out to each LV controller's TAP asynchronous interface.

Control of this bit is transparent to you, and is set up by the LV testbenches. Wherever a test is run with a tckRatio greater or equal to 8, this bit is asserted and inverts the asyncIntTck. Doing this increases the hold margin of the TAP BIST_SETUP and BIST_SI signals fanning out to the Mentor Graphics embedded test controllers. You can always increase that margin further by specifying a higher tckRatio in their ETVerify configuration file. The default tckRatio for all controllers is set to 16, which means the TCK clock frequency is 16 times smaller than the operating frequency of the controller, therefore, allowing plenty of both setup and hold timing margin for all TAP interface signals.

OP[2:0]

The OP[2:0] instruction register bits define a serial path from the tdi TAP controller port to the tdo TAP controller port through the test data registers.

Note



If the OP[2:0] values select a test data register that is absent from your design, the TAP controller selects the bypass register instead. For example, if your design does not include a Device ID register and OP[2:0] 110, the instruction that selects the Device ID register, then 110 selects the Bypass register instead.

Table 2-9 lists the OP[2:0] values and the corresponding test data register sequence.

Table 2-9. Specifying the OP[2:0] Instruction Register Bits

OP[2]	OP[1]	OP[0]	Selected Test Data Register Sequence
0	0	0	tdi > Boundary Scan > tdo

Table 2-9. Specifying the OP[2:0] Instruction Register Bits (cont.)

OP[2]	OP[1]	OP[0]	Selected Test Data Register Sequence
0	0	1	tdi > Boundary Scan > Internal Scan > Device ID > Bypass > tdo
0	1	0	tdi > Internal Scan > tdo
0	1	1	tdi > Boundary Scan > Internal Scan > Device ID > Bypass > tdo ¹
1	0	0	tdi > Internal Scan > tdo ²
1	0	1	tdi > Internal Data Register > tdo ³
1	1	0	tdi > Device ID > tdo
1	1	1	tdi > Bypass > tdo

1. The 011 OP[2:0] bit combination differs from the 001 combination in that it also places the TAP in diagnostic mode. In this mode, the TAP controller suppresses the capture state. By suppressing this state, you can scan out the contents of the internal data registers without corrupting the data.

2. The 100 OP[2:0] bit combination differs from the 010 combination in that it also places the TAP in hold BIST mode. This forces any selected BIST controller to not reset its state while its internal registers are being scanned.

3. The 101 OP[2:0] bit combination is also used to select the internal fault insertion register when the first bistEnable code is selected and the testMode bit in the instruction register is set.

selectJtagIn, selectJtagOut

The selectJtagIn and selectJtagOut instruction register bits control the boundary-scan operating mode. (The ETAssemble tool inverts the instruction-register bits selectJtagIn and selectJtagOut to create the selectJtagIn and selectJtagOut signals that control the individual boundary-scan cells.)

Table 2-10 lists the selectJtagIn and the selectJtagOut values and the corresponding operating mode. Refer to Chapter 4, “Boundary-Scan Cells” for more information on boundary-scan operating modes.

Table 2-10. Specifying the selectJtagIn_BAR and the selectJtagOut_BAR Instruction Register Bits

selectJtagIn_BAR	selectJtagOut_BAR	Selected Mode
0	0	SAMPLE ¹
0	1	EXTEST
1	0	INTEST
1	1	SAMPLE

1. The {00} selectJtagIn_BAR and the selectJtagOut_BAR instruction register bits correspond to the EXTEST operating mode when an 1149.1-1993 compliant TAP is generated.

setupMode[1:0]

The setupMode[1:0] instruction register bits specify the run or setup mode for the currently selected BIST controller. [Table 2-11](#) lists the setupMode[1:0] values and the corresponding run and setup modes. Descriptions of these modes are provided below:

- The Short Setup mode places the BIST controller in scan mode and provides access to the controller's short setup scan chain. The contents of the short setup scan chain differs among controller types, but generally contains a relatively small number of registers used to set various run time test options.
- The Long Setup mode places the BIST controller in scan mode and provides access to the controller's long setup scan chain. The content of the long setup scan chain differs among controller types, but generally contains a relatively large number of registers used to set various runtime test and diagnostic options, as well as provide access to detailed test and diagnostic result data.
- The Default Run mode places the BIST controller in run mode. In this mode, the controller uses default initialization values and disregards any values scanned in using either the Short or Long Setup modes.
- The Normal Run mode places the BIST controller in run mode. In this mode the controller makes use of the initialization values scanned in using either the Short or Long Setup modes.

Table 2-11. Specifying the setupMode_BAR[1:0] Instruction Register Bits

setupMode_BAR[1:0] Instruction Bits	Selected Mode
1 1	Short Setup
1 0	Long Setup
0 1	Default Run
0 0	Normal Run

The output setupMode[1:0] port on the TAP controller is formed by the concatenation of the testMode instruction bit with the setupMode[1:0] instruction bits. The inverse of the testMode bit directly drives the setupMode[2:0] port. The inverse of the setupMode bits drive the setupMode[1:0] port. The 3-bits of the setupMode port are used only by the logicTest controller

with the encoding described in [Table 2-12](#). Other controller types use only the `setupMode[1:0]` ports with the encoding described in [Table 2-11](#).

Table 2-12. Specifying the `testMode_BAR,setupMode_BAR[1:0]` Instruction Register Bits

<code>testMode_BAR, setupMode_BAR[1:0]</code>	Selected Mode
0 1 0	Multi Scan mode
0 1 1	Single Scan mode
1 1 1	Short Setup
1 1 0	Long Setup
1 0 1	Default Run
1 0 0	Normal Run

testMode

The `testMode` instruction register bit generates a global `testMode` TAP signal that places your design into test mode. This signal is not generated when a top-level `logicTest` controller is used. It is only provided when third-party scan/ATPG is used.

user[P-1:0]

The `user[P-1:0]` instruction register bits are for user-defined purposes and as such have no value assignment definitions.

However, depending on your design configuration, some `userBits` are strictly reserved for Mentor Graphics' use. When both free `userIRbits` and reserved `IRbits` are present, the reserved ones are always put on the most significant bits side.

For example, user-specified `numberUserBits = 5` for the TAP. In this case:

- `UserIRBit[4:0]` are allowed for user-defined signals
- `UserIRBit[13:5]` might be reserved for Mentor Graphics' use.

The number of Mentor Graphics reserved bits vary from one design to the another, and those bits are usually used for controlling some `logicTest` controller or memory BIST controller configuration inputs. Here is an example list of reserved user bits. Offset is the `userIRbit` index of the first reserved `IR` bit. All `IR` bits are set to Off (meaning they drive a `logicLow` into the core) upon a TAP reset.

userIRBit[Offset]

ETClockEnable — This optional bit only exists when there is at least one Mentor Graphics embedded test controller in the design (besides the TAP and the boundary-scan). This bit is

asserted by the Mentor Graphics testbenches whenever a Mentor Graphics embedded test controller runs. You can use the ETClockEnable signal to customize their BIST mode clock configuration when it differs from their functional mode configuration.

userIRBit[Offset+1]

BistClockGating — This optional bit gates the clock feeding all memory BIST controllers if you specified the following property in your *.etplan* configuration file:

```
GateBistClockInFuncMode: Yes;
```

userIRBit[Offset+2]

BurstEnable — This optional bit only exists when a Mentor Graphics 2005 logicTest controller is present. It controls the enabling of the BurstMode capturing method.

userIRBit[Offset+7: Offset+3]

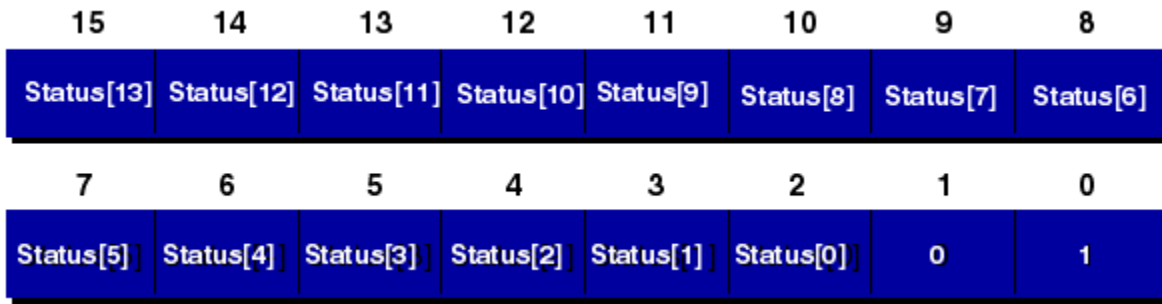
Those optional bits control test configuration inputs that are exclusive to Tessent MemoryBIST NonProgrammable controllers. The list below shows an example of possible control signals, but more might be added depending on the design. For details refer to the [Tessent MemoryBIST User's and Reference Manual](#).

- MBIST_RST_MEM — instructs a Non-Programmable memory BIST controller to reset the memories writing zeros to all memory locations.
- BIST_DIAG_EN — activates the diagnostic mode of the NonProgrammable controller during CompStat and Stop-On-Nth-Error diagnostics.
- BIST_ALGO_MODE — specifies the algorithm phase to be applied by the NonProgrammable controller during parallel retention testing.

Status Register Bit Assignment

During the Capture-IR TAP controller state, the TAP captures the values present on the status register inputs. In the following TAP controller state, the Shift-IR TAP controller state, the status register contents are serially shifted out through the tdo port—while the TAP serially loads a new instruction register opcode through tdi.

In this capacity, the instruction register is referred to as the status register. An N-bit instruction register contains N-2 status bits. (The first 2-bits of the status register are always 01, where the 1 is the first bit seen at the TDO pin.) Thus, a 16-bit instruction register has 14-bits available for your use, Status[13:0]. [Figure 2-8](#) shows a 16-bit status register.

Figure 2-8. 16-Bit Status Register

Use the IR_STATUS property in the [TAP: Connections](#) wrapper of the ETAssemble configuration file.

Reserved Instructions

The IEEE 1149.1-2001 specification includes a number of reserved instructions. [Table 2-13](#) lists the bit assignments for the default 16-bit instruction register that corresponds to the reserved instructions. Set all extended instruction register bits to 1 for all instructions listed below, except for the 0 BYPASS instruction.

Table 2-13. IEEE 1149.1 Reserved Instructions

Instruction[15:0]	Reserved Instruction
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	BYPASS
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	BYPASS ¹
1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0	EXTEST
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0	SAMPLE/PRELOAD
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0	IDCODE
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0	INTEST
1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1	HIGHZ
1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1	CLAMP

1. When generating a TAP compliant with the IEEE 1149.1-1993 standard, the all-zeros {000...000} instruction corresponds to the EXTEST reserved instruction.

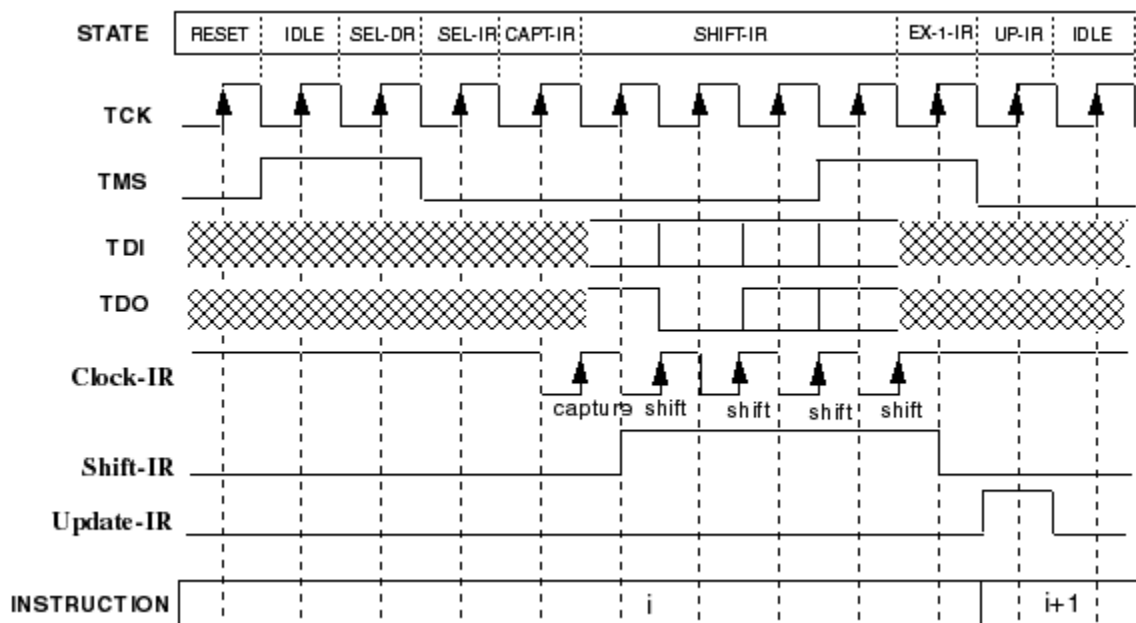
Timing Waveforms

The TAP controls access to the instruction register through 16 TAP controller states. [Figure 2-9](#) shows the shift timing for the instruction register according to the TAP controller states and identifies the TAP controller states with the following abbreviations:

- RESET corresponds to the Test-Logic-Reset TAP controller state.
- IDLE corresponds to the Run-Test/Idle TAP controller state.
- SEL-DR corresponds to the Select-DR-Scan TAP controller state.
- SEL-IR corresponds to the Select-IR-Scan TAP controller state.
- CAPT corresponds to the Capture-IR TAP controller state.
- SHIFT corresponds to the Shift-IR TAP controller state.
- EX-1 corresponds to the Exit1-IR TAP controller state.
- UPDATE corresponds to the Update-IR TAP controller state.

For more information about the TAP controller states, refer to “[TAP Controller](#).”

Figure 2-9. Instruction Register Shift Timing



Internal Data Register

The optional internal data register supplements the instruction register by providing additional BIST control and result gathering capabilities. The internal data register bit assignments are illustrated in [Figure 2-10](#). The register consists of two sections: parallel BIST enable bits and userDR bits. Descriptions of both sections follow [Figure 2-10](#).

Figure 2-10. Internal Data Register Bit Assignments

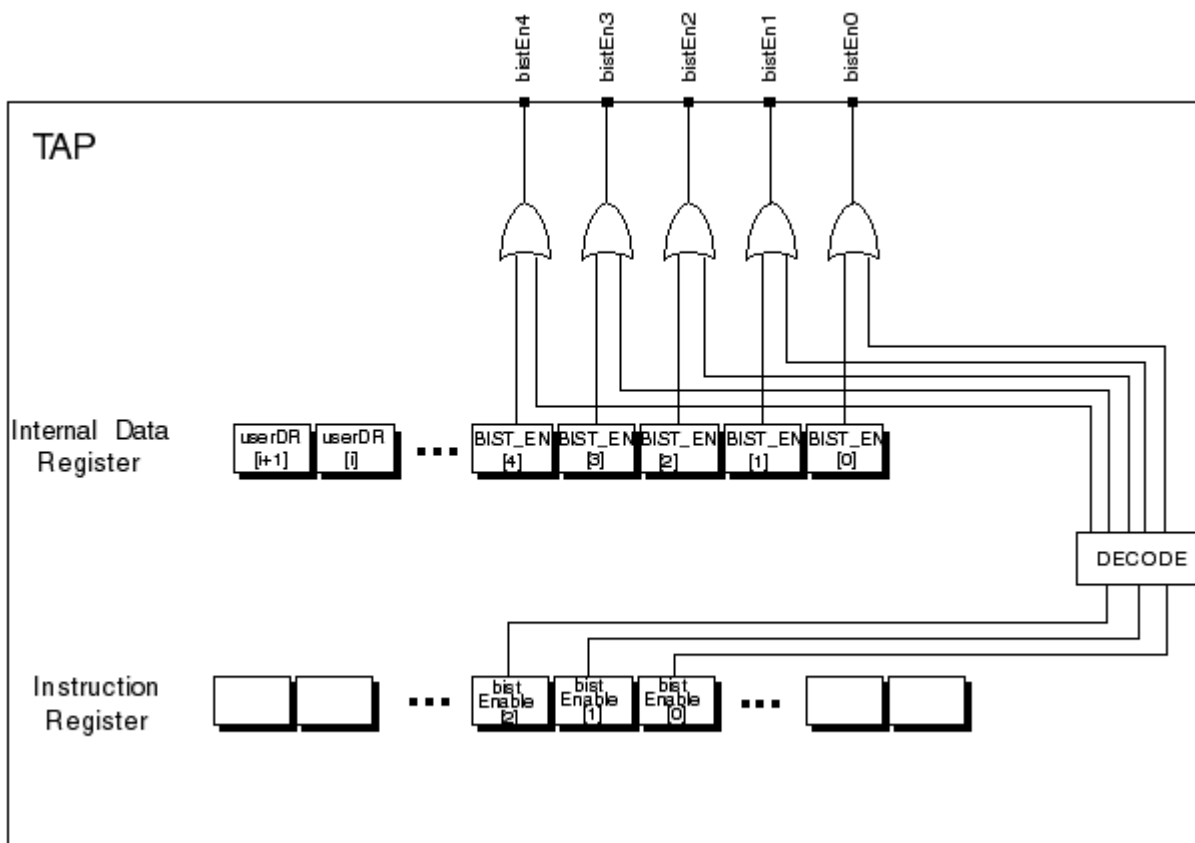


Parallel BIST Enable Bits

If the design has multiple BIST controllers, then an internal data register with parallel BIST enable bits is added to the TAP. Each of these bits corresponds to a BIST enable port on the TAP. As a result, the internal data register can contain up to 2048 parallel BIST enable bits.

Recall that the `bistEnable[M-1:0]` bits in the instruction register select a single BIST controller at a time. The parallel BIST enable bits provide the additional capability of enabling more than one BIST controller at a time. This is accomplished as illustrated in [Figure 2-11](#).

Figure 2-11. Using the Internal Data Register for Parallel BIST



In the above illustration, the value of the parallel BIST enable bits are ORed with the decoded value of the `bistEnable[M-1:0]` bits. As a result, placing a logic 1 in a particular parallel BIST

enable bit enables the associated BIST controller irrespective of the BIST controller selected by the `bistEnable[M-1:0]` bits. With this approach, multiple BIST controllers can be enabled in parallel. Note, however, that only the BIST controller selected by the `bistEnable[M-1:0]` bits has its scan chain connected between `tdi` and `tdo`.

The parallel BIST enable bits can also be used to capture data from outside the TAP. Typically, each of these bits is used as a status bit for the corresponding BIST controller.

ResetTest

The `ResetTest` bit of the Internal data register forces asynchronous reset pins on flip-flops to be asserted when performing an asynchronous reset test. This bit is normally OFF (logic 0).

SetTest

The `SetTest` bit of the Internal data register forces asynchronous set pins on flip-flops to be asserted when performing an asynchronous set test. This bit is normally OFF (logic 0).

TCKLow

The `TCKLow` bit of the Internal data register forces the `clockIsCan` signal to be held low instead of high. This bit is normally OFF (logic 0) but is used for retention tests and asynchronous set/reset tests.

HoldShiftHigh

The `HoldShiftHigh` bit of the Internal data register forces the `shiftBist` signal to be held high. This bit is normally OFF (logic 0) but is used for retention tests and asynchronous set/reset tests.

SuppressUpdateDR

The `SuppressUpdateDR` bit of the Internal data register forces the `updateDR` signal to stay de-asserted. This bit is normally OFF (logic 0) but is used for IO leakage `TriStateEnable_NC`.

IOTest Bits

The `IOTest` bits of the Internal data register modify the behavior of the TAP output signals `BscanUpdate` and `BscanUpdateEnable` so that their assertion occurs during a different TAP state than the normal `UPDATE -DR` state. They also de-assert the TAP output signal `ForceDisable` during the `CAPTURE-DR` state. These bits are effective only when the `SuppressUpdateDr` bit is ON. They are normally OFF (logic 0) but are used for the IO test `TriStateEnable_NC`. The value 01 selects the global enable path when the `TriStateEnable_NC` test is executed while the value 10 selects the local enable path when the `TriStateEnable_NC` test is executed. The value 11 has no effect, as 00.

UserDR Bits

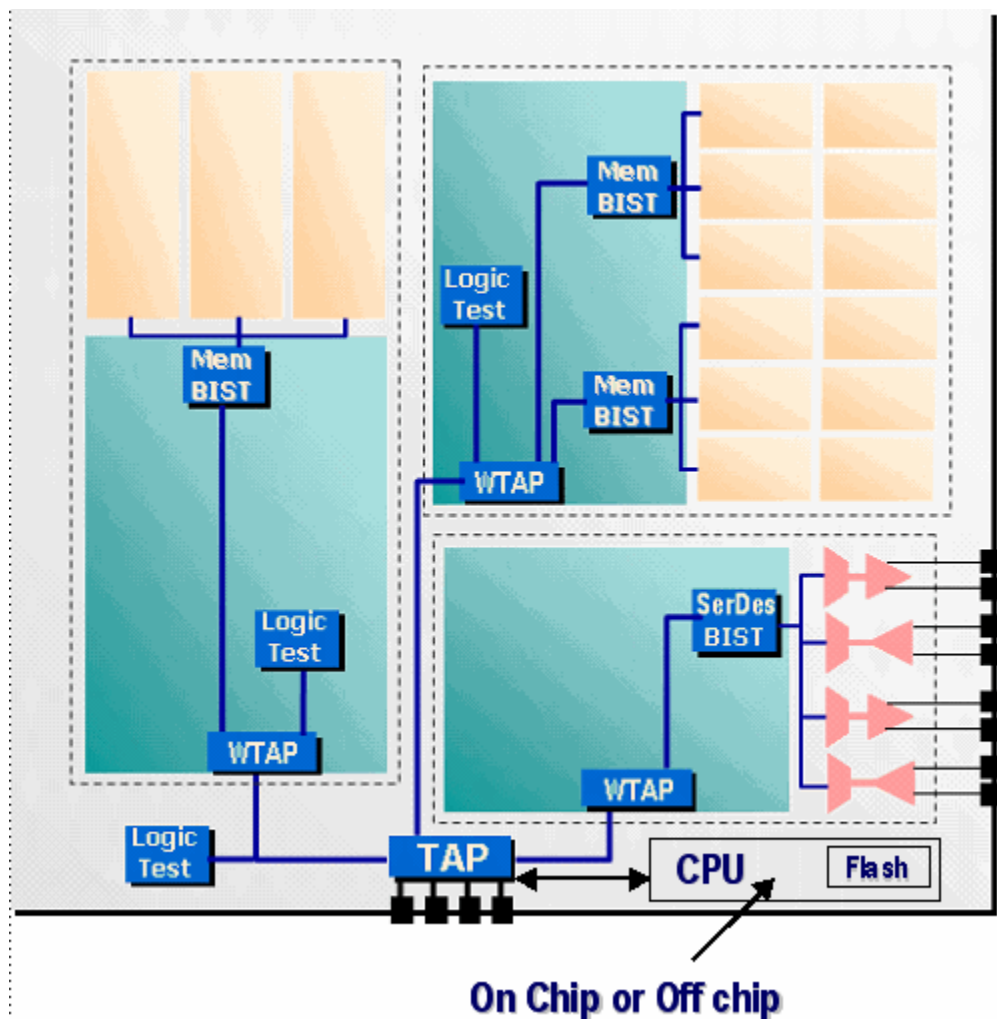
If the [NumberUserDRBits](#) property is specified in the configuration file of the ETAssemble tool, then user bits are added to the internal data register. Up to 2048 userDR bits can be specified. These bits are different from the user bits in the instruction register in that they can hold their values while different instructions are scanned into the TAP. As with the parallel BIST enable bits, the userDR bits can also be used to capture values from outside the TAP.

The CPU Interface

The TAP controller can be created with a CPU Interface which allows controlling the TAP and everything connected to the TAP from a CPU bus as illustrated in [Figure 2-12](#).

The creation of the CPU Interface is controlled by the presence of the [CPUInterface](#) wrapper inside the [GlobalOptions](#) wrapper of the *.etplan* file. Its width is controlled by the [DataBusWidth](#) property which can be set to an integer between 4 and 64.

Figure 2-12. Controlling the TAP from a CPU Interface



The hardware in your chip which drives the CPU Interface on the TAP can be any CPU bus you have in your chip including an i²c bus or an embedded CPU core.

Figure 2-13 shows a TAP controller with a TDR register connected to it. Figure 2-14 shows the same TAP controller with the same TDR register connected to it when the TAP is equipped with a CPU interface. Notice how tdi and tck connect directly the TDR in the configuration without the CPU interface, and how tck_INT and tdi_INT are used when the CPU interface is present. This is to allow the CPU interface to take full control of the TDR registers connected to the TAP and be in a position to supply the tdi data and tck to them.

Figure 2-13. TAP Controller with a TDR Register without a CPU

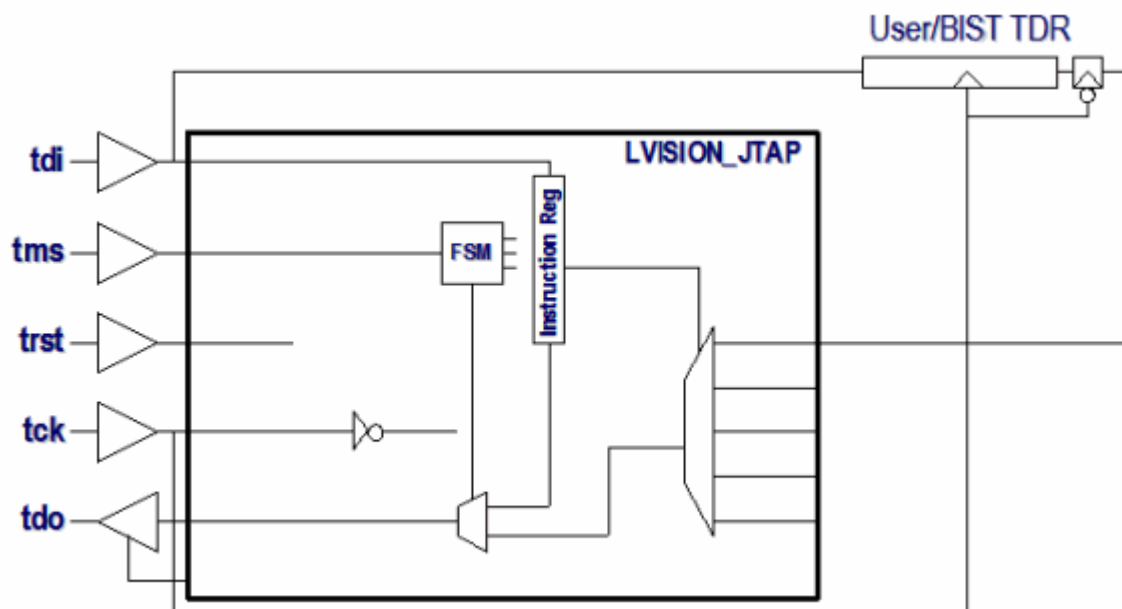
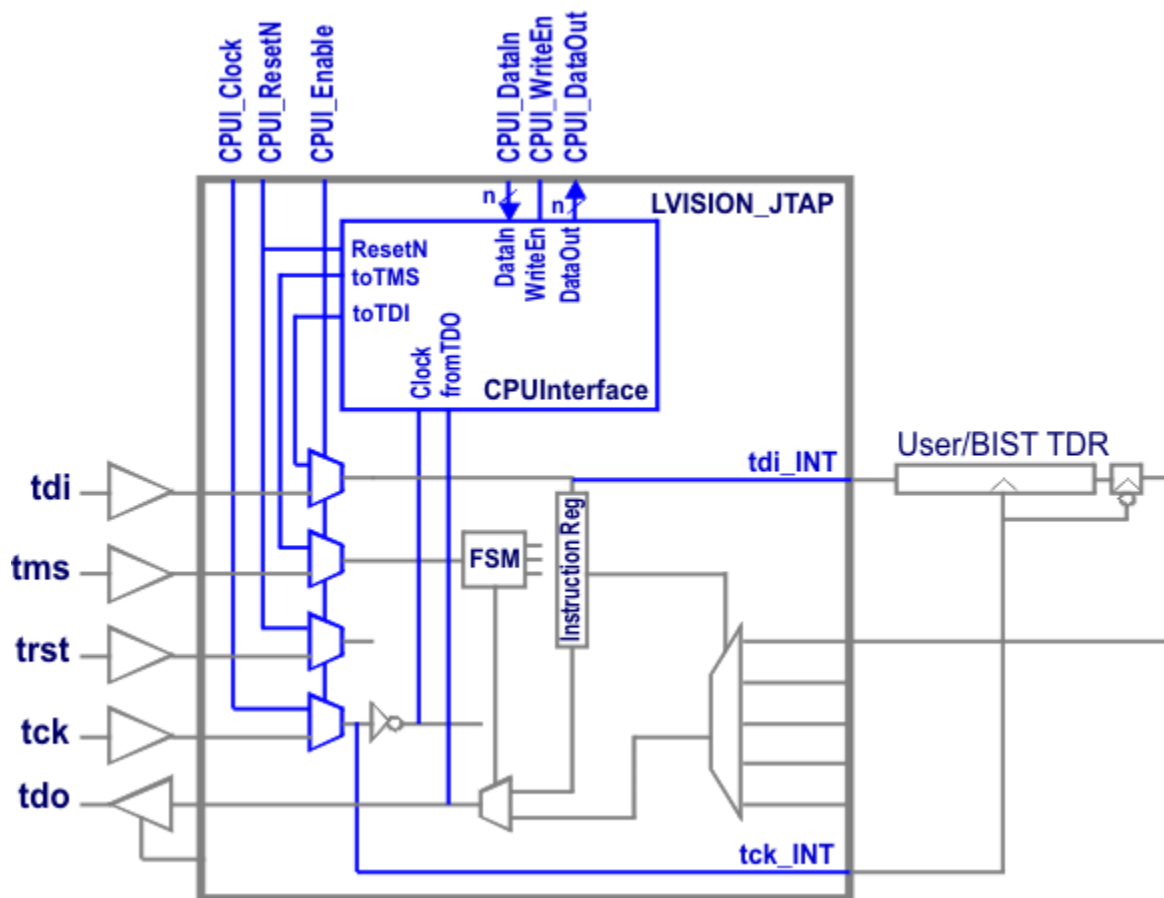


Figure 2-14. TAP Controller with a TDR Register with a CPU Interface



CPU Interface Ports on the TAP

Following is a description of the six CPU interface ports on the TAP controller. The port names on the TAP controller have the names listed below with the extra CPUInterface_ prefix.

Clock

This port is the clock that operates the CPU Interface, the TAP, and any TDR connected to the TAP when the CPU Interface is enabled. For this reason, you need to supply a clock with a frequency that is equal or slower than the TCKPeriod you designed the TAP and WTAPs to work at. This is typically in the 10 to 50 MegaHertz range.

ResetN

This port is an active low asynchronous reset signal. Once deasserted, the CPU interface and the TAP are parked in the RunTestIdle state waiting for a write command. The timing of this signal

is not important, including its deassertion time relative to the clock as long as the WriteEn port is low when the ResetN signal is brought up.

Enable

This port is active high and, when asserted high, gives control of the TAP to the CPU interface. It must remain high for as long as the CPU interface needs to remain in control of the TAP.

WriteEn

This port is active high and specifies that valid data on the DataIn bus is ready to be used by the CPU interface. This signal has the following timing specifications:

- It is asynchronous to the clock and does not need to meet any setup and hold relationship with respect to the clock as its rising edge is asynchronously detected by a no-timing flip-flop.
- The WriteEn signal is allowed to go up one clock cycle after the DataIn data is stable and must remain high for at least 3 clock cycles.
- The WriteEn must go back down for at least 3 clock cycles before it is brought back up.

DataIn

This port is an N-bit bus the size of the specified [DataBusWidth](#) and carries the command to be executed as well as the Data to be shifted in.

This port has the following timing requirements:

- It is asynchronous to the clock and does not need to meet any setup and hold relationship with respect to the clock.
- The DataIn must be stable at least one clock cycle before the rising edge of the WriteEn port and must remain stable for at least 3 clock cycles after the rising edge of WriteEn.

The DataIn bits are separated into two sections as illustrated in [Figure 2-15](#). The two most significant bits represent the command to be executed, and the rest of the bits corresponds to the data bits that are to be shifted in by the TAP. The least significant bit is shifted in first while the most significant bit is shifted in last. The 2-bit command encoding is as follow:

- **Command 00** — This command instructs the TAP to go and shift the tdi data into the TAP instruction register and then go park its state machine in the PauseIR state. Depending on the current state of the TAP, this will represent one of the following state transition:
 - RunTestIdle > ShiftIR > PauseIR
 - PauseIR > ShiftIR > PauseIR

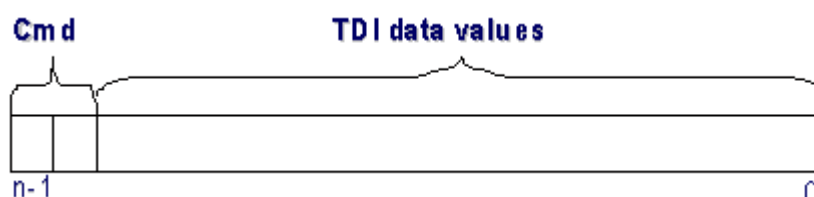
- **Command 01** — This command instructs the TAP to go and shift the tdi data into the test data register currently selected by the TAP instruction register and then go park its state machine in the PauseDR state. Depending on the TAP current state, this will represent one of the following state transition:
 - RunTestIdle > ShiftDR > PauseDR
 - PauseDR > ShiftDR > PauseDR
- **Command 10** — This command instructs the TAP to go and shift the tdi data into the test data register currently selected by the TAP instruction register or the Instruction register and then go park the TAP state machine in the RunTestIdle state. Depending on the TAP current state, this will represent one of the following state transition:
 - PauseIR > ShiftIR > RunTestIdle
 - PauseDR > ShiftDR > RunTestIdle

If the current state of the TAP is TestLogicReset then the TAP State machine is simply taken to RunTestIdle. The tdi data values are not used.

If the current state of the TAP is already RunTestIdle then the TAP State machine is simply left in RunTestIdle. The tdi data values are not used.

- **Command 11** — This command instructs the TAP to go to the Test LogicReset state. The current state is irrelevant, and the tdi data values are not used.

Figure 2-15. DataIn Port Data Format



DataOut

This port is an N-bit bus the size of the specified [DataBusWidth](#) and carries the read status bits as well as the Data that was shifted out.

This port has the following timing requirements:

- It is asynchronous to the clock and does not meet any setup and hold relationship with respect to the clock.
- The DataOut values remain stable until the next rising edge of WriteEn.

The DataOut bits are separated into two sections as illustrated in [Figure 2-16](#). The two most significant bits represent the Read status, and the rest of the bits corresponds to the data bits that

were shifted out by the TAP. The least significant bit was shifted out first while the most significant bit was shifted out last.

The meaning of the Read status bits is as follow:

- **Status X1**—The data value on the DataOut is valid and corresponds to the shifted out values. The WriteEn port needs to be 0 for the Status bit to be set to 1.
- **Status X0**—The data value on the DataOut is not valid. The TAP is most likely still busy shifting out the data, or the WriteEn port has not yet been set to 0.

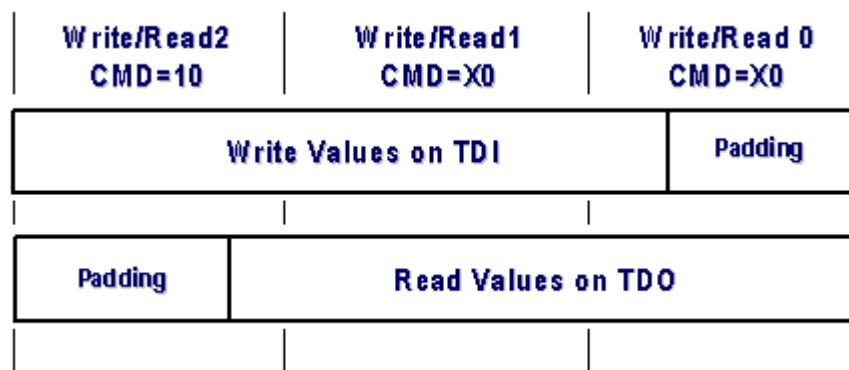
Figure 2-16. DataOut Port Data Format



TDI TDO Data Padding

The length of the TDR register can be any size and might be shorter or longer than the CPU word size minus 2. The TDI data is padded on the right side (LSB) such that the number of tdi bits is exactly a multiple of CPU word size minus 2. The TDO data is padded on the left side (MSB) by the same amount. This is illustrated in [Figure 2-17](#). Note that the minimum number of WriteReads needed to load any TDR is 2 even if the TDR is shorter than the CPU word size minus 2. This is because you must first run a command 01 (RTI>shiftDR>PauseDR) followed by a command 10 (PauseDR>ShiftDR>RTI). The padding size simply needs to be adjusted accordingly.

Figure 2-17. ITDR Data Padding to Become a Multiple of CPUDatBudWidth-2



Chapter 3

WTAP Controller

This chapter describes the WTAP architecture, its controller ports, and its registers.

This chapter covers the following topics:

- [WTAP Controller Architecture](#)
 - [Interface With Top-Level TAP](#)
 - [Operation Control From the TAP](#)
- [WTAP Controller Ports](#)
 - [TAP Interface Ports](#)
 - [WSI](#)
 - [WSO](#)
 - [WRSTN](#)
 - [updateWR](#)
 - [shiftWR](#)
 - [captureWR](#)
 - [WRCK](#)
 - [selectWIR](#)
 - [enableWR](#)
 - [extTM](#)
 - [Ports for the Instruction Register](#)
 - [instruction\[q-1:0\]](#)
 - [status\[q-3:0\]](#)
 - [userIRBits\[j-1:0\]](#)
 - [userIRBitsNotLatched\[j-1:0\]](#)
 - [updateIR](#)
 - [updateIREnable](#)

- Ports for Test Mode Functions
 - clkRatio[2:0]
 - freezeMode
 - functMode
 - setupMode[2:0]
 - tckMode
 - testMode
 - diagMode
 - resetTest
 - setTest
- Ports for Embedded Test Controllers
 - clkBistTCK
 - asyncIntTCK
 - fromBist[k-1:0]
 - toBist
 - bistEn[k-1:0]
 - multiEn[k-1:0]
 - holdBist
 - shiftBist
 - shiftBist2Edge
- Ports for User-Defined Scan Registers
 - enableClkDR
 - captureDR
 - captureDR2Edge
 - shiftDR
 - shiftDR2Edge
 - updateDR
 - updateDREnable
 - testLogicReset

-
- Output Ports for Lower-Level WTAPs
 - captureWRGated
 - shiftWRGated
 - updateWRGated
 - Other Ports
 - WRCKInvOut
 - WRCKInvIn
 - Instruction Register
 - Instruction Register Bit Assignment
 - DR_Sel[1:0]
 - BISTEn[k:1]
 - holdBIST
 - setupMode[2:0]
 - freezeMode
 - diagMode
 - freeRunMode
 - clkRatio[2:0]
 - enableStatus
 - resetTest
 - resetTest
 - setTest
 - holdShiftHigh
 - tckLow
 - invertAsyncTCK
 - userIRBit[J-1:0]
 - Status Register Bit Assignment
 - Test Data Registers
 - Device ID Register
 - Bypass Register

- [Seed Register](#)

WTAP Controller Architecture

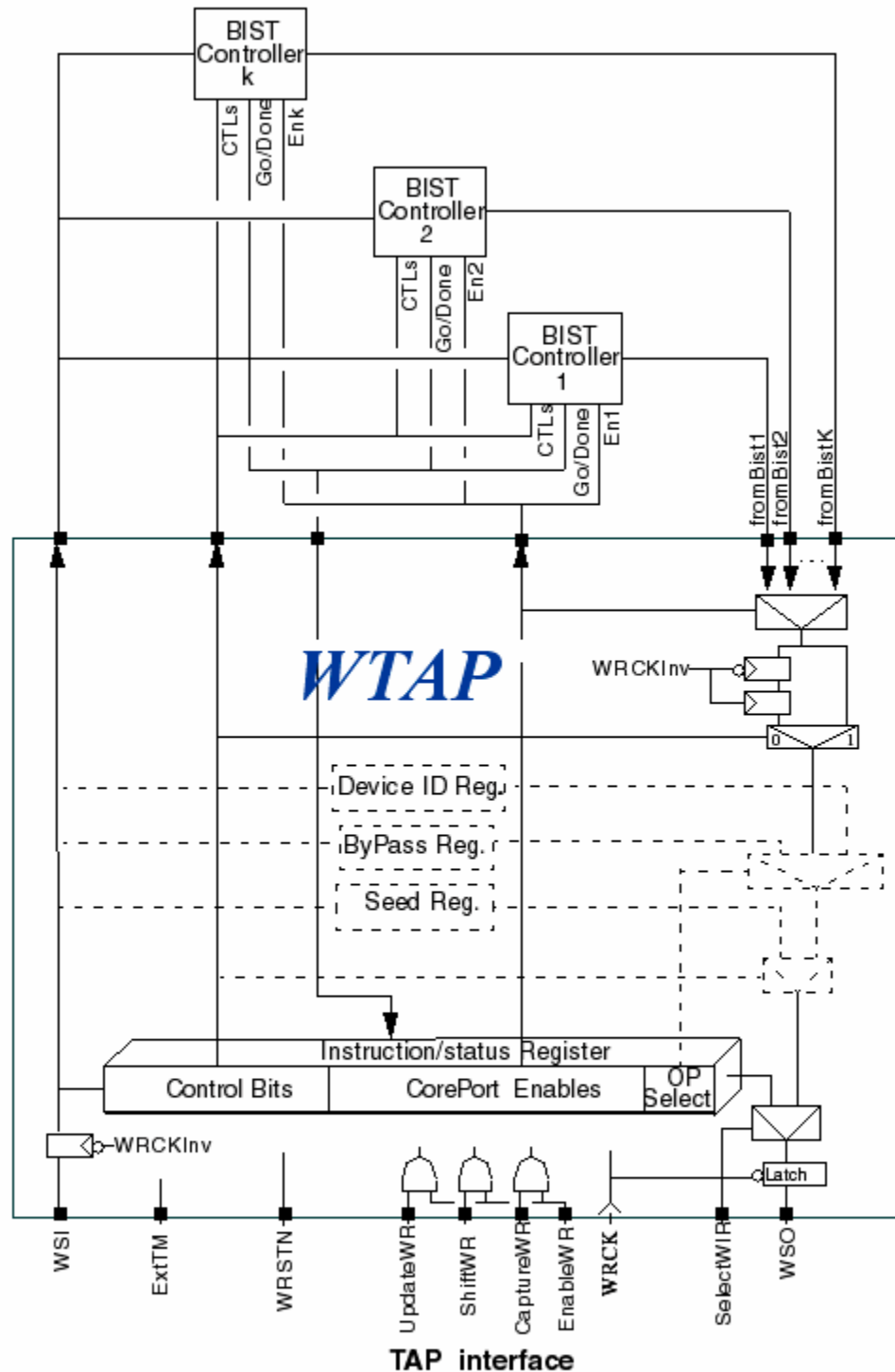
The WTAP circuit is similar to the TAP circuit except that it has no Finite State Machine (FSM).

[Figure 3-1](#) shows a block diagram of the Mentor Graphics' WTAP controller and its connection to Mentor Graphics' embedded test controllers. The WTAP controller includes the following key blocks:

- Instruction register (includes status register)
- Device ID register (optional)
- Bypass Register (optional)
- BIST Setup and Internal Scan Chain Multiplexing Logic

The following sections of this chapter provide descriptions of the WTAP blocks and interfaces. [Table 3-1](#) provides descriptions of the WTAP ports that interface with the top-level TAP controller.

Figure 3-1. The WTAP Controller Architecture



Notes:

Ports marked in bold are extra Mentor Graphics-specific ports.
Hardware drawn with ---- is optional.

Interface With Top-Level TAP

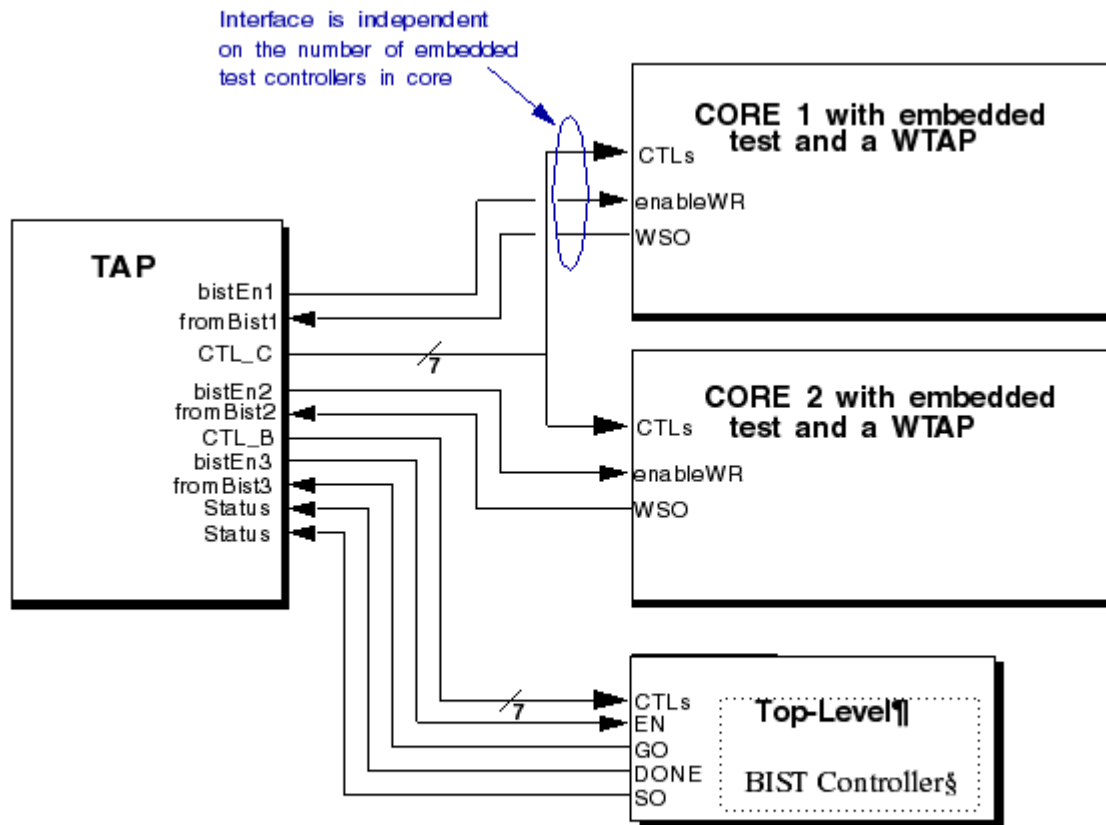
Table 2-1 provides the descriptions of the WTAP ports that interface with the top-level TAP.

Table 3-1. The Control Ports of WTAP Interfacing with TAP

Port Name	Direction	Description
WSO	Output	Scan out port of the WTAP controller.
WRSTN	Input	Active low asynchronous reset of the WTAP controller.
UpdateWR	Input	Update enable signal. When UpdateWR is high, the update register loads the value of the shift register on the next falling edge of WRCK.
ShiftWR	Input	Shift enable signal. When ShiftWR is high, the selected shift register moves data closer to WSO by one bit on the rising edge of WRCK.
CaptureWR	Input	Capture enable signal. When CaptureWR is high, the selected shift register captures the values of its corresponding status bits on the rising edge of WRCK.
SelectWIR	Input	Select WTAP Instruction Register. When high, the WTAP controller instruction register is accessed. When low, the data register specified by the instruction register contents is accessed.
EnableWR	Input	Enable WTAP access. When low, all WTAP registers hold their states. When high, the WTAP registers follow the action specified by UpdateWR, ShiftWR, and CaptureWR control signals. This port is a Mentor Graphics-specific addition to the IEEE 1500 standard.
ExtTM	Input	External Test Mode. Forces the WTAP controller to external test mode. When ExtTM is high, the WTAP is reset and the periphery scan chains are isolated from contribution of the core inside, and their control is released to external scan ports on the collared core. This port is Mentor Graphics-specific addition to the IEEE 1500 standard.

Each WTAP controller is connected as a separate Data Register of the TAP controller. As shown in [Figure 3-2](#), there are five global control signals that fan out from the TAP controller to each WTAP controller, and only two dedicated signals in the opposite direction. Each WTAP EnableWR pin connects to its own dedicated TAP controller bistEn# output pin. Each WTAP controller is accessed like any other top-level Mentor Graphics embedded test controller from the TAP controller. [Table 3-2](#) lists the connections between the TAP and WTAP controllers.

Figure 3-2. Connections Between WTAP Controllers and TAP Controller



CTL_C = Tdi, shiftDR_2EDGE, captureDR_2EDGE, updateDREnable, testLogicResetInv, tck, setupMode[0]

CTL_B = tolscan, shiftBist, setupMode[1:0], tck, tckMode, holdBist

Table 3-2. Seven (7) Connections of Control Signals Between WTAP and TAP

Port on WTAP	Port on TAP
WRSTN	testLogicResetInv
updateWR	updateDREnable
shiftWR	shiftDR2Edge
captureWR	captureDR2Edge
selectWIR	setupMode[0]
WSO	fromBist#
enableWR	bistEn#

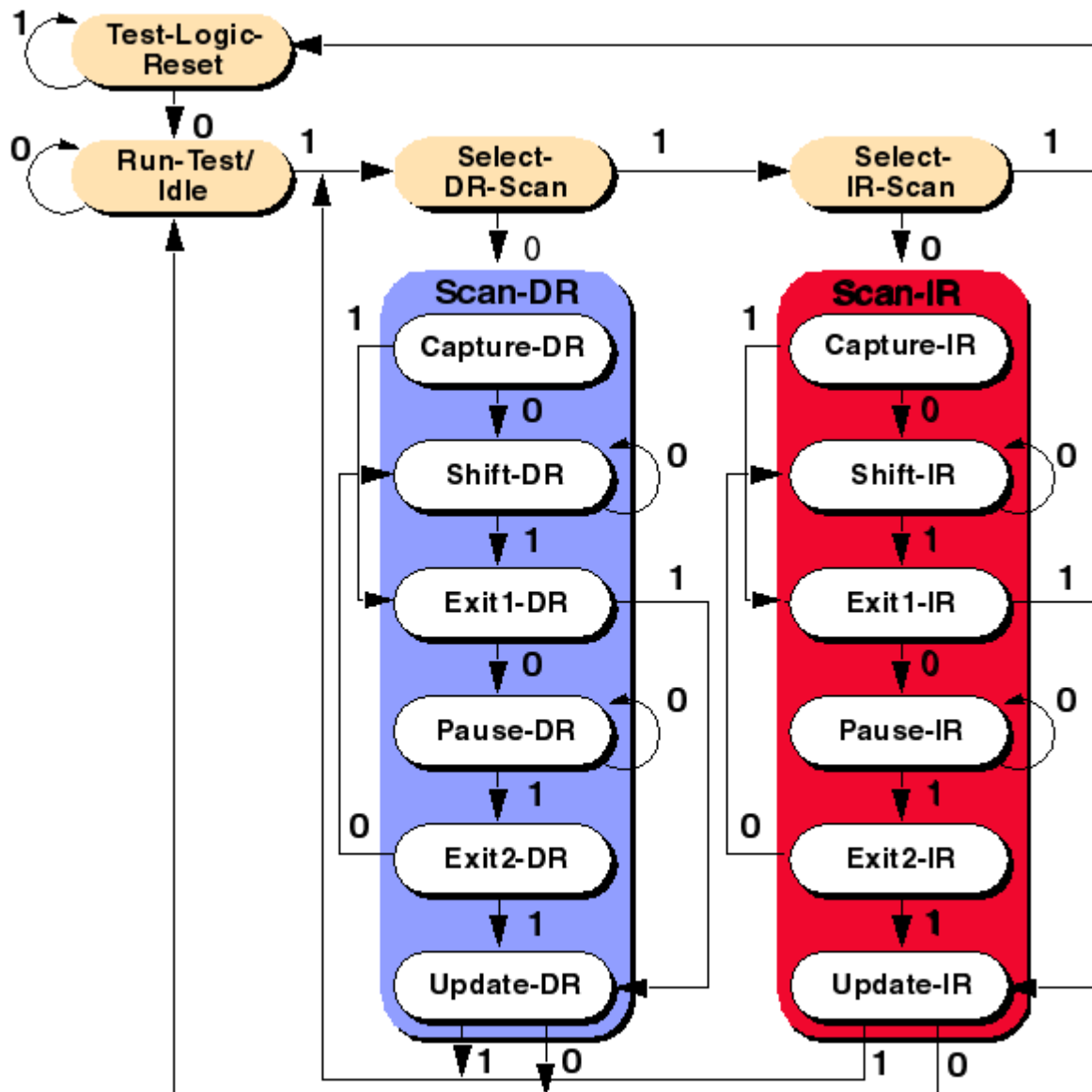
Mentor Graphics chose to connect all lower-level blocks WTAP to the unique top-level TAP using a star topology instead of a daisy chain (ring) topology, as prescribed in the P1500 standard. The advantages of the star topology are:

- It allows connecting a core with embedded test to the TAP as soon as one is ready, and its verification with the testbench generation can be completed without waiting for the other cores to be ready.
- The shift path does not go through other WTAP controllers. Entire sections of the chip can be “blacked out” when simulating test features within a given embedded test core.
- Addressing of each WTAP is much easier as they are directly associated with a unique BIST port of the TAP controller.

Operation Control From the TAP

As mentioned earlier, a WTAP has no internal Finite State Machine (FSM). The 1149.1 TAP FSM controls all WTAP operations through the selectWR, captureWR, shiftWR, and updateWR signals. [Figure 3-3](#) provides a state diagram of the TAP FSM, and [Table 3-3](#) lists the values of these signals for each state of the TAP FSM.

Figure 3-3. State Diagram for the WTAP Controller



The values adjacent to each state transition correspond to the TMS signal. The signal TMS, sampled by the TAP controller on the rising edge of TCK, controls the state transitions.

Table 3-3 lists the values of the WTAP control signals for each state of the TAP FSM. Access to the WTAP only occurs during the TAP DR-states. IR-states only access the TAP's own Instruction Register. Since the SelectWIR signal state is directly connected to the TAP IR "bist_setup0" bit, it only changes when a new TAP instruction is loaded. All control signals stay OFF when the WTAP's corresponding bistEn# bit is OFF.

Table 3-3. TAP FSM States and Corresponding WTAP Control Signal Values

TAP FSM state	WTAP control signal values		
	captureWR	shiftWR	updateWR
TestLogicReset	0	0	0
RunTestIdle	0	0	0
Select-DR-Scan	0	0	0
Capture-DR	1	0	0
Shift-DR	0	1	0
Exit1-DR	0	0	0
Pause-DR	0	0	0
Exit2-DR	0	0	0
updateDR	0	0	1

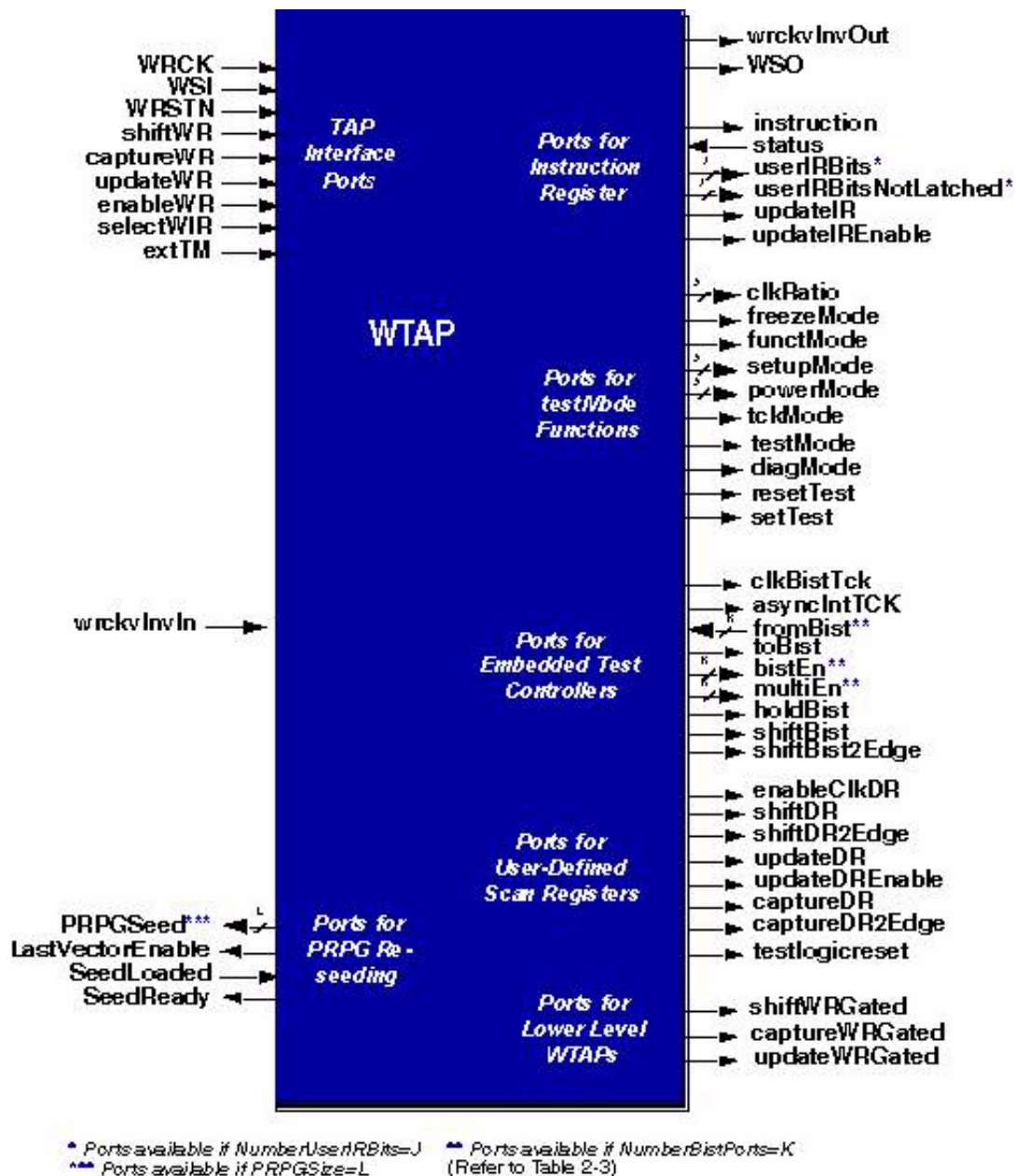
WTAP Controller Ports

Mentor Graphics' WTAP ports are shown in [Figure 3-4](#). The presence or number of some WTAP ports might vary depending on property values that you can specify in either ETPlanner or ETAssemble configuration files. [Table 3-4](#) lists those properties and their associated ports. A detailed description of the ports then follows.

Table 3-4. Optional WTAP Ports

Property	Resulting Ports
NumberUserIRBits : J;	userBits[J-1:0] userBitsNotLatched[J-1:0]
NumberBistPorts : K;	bistEn[K-1:0] fromBist[K-1:0] multiEn[K-1:0]
PRPGSize: L;	PRPGSeed[L-1:0]

Figure 3-4. WTAP Controller Ports



TAP Interface Ports

The following sections describe all TAP interface ports.

WSI

Input. WSI is the scan-in port to the WTAP controller registers. This port will be connected directly to the TDI pad.

WSO

Output. WSO is the scan-out port from the WTAP controller registers.

WRSTN

Input. WTAP active low asynchronous reset.

updateWR

Input. When UpdateWR is high, the update register loads the value of the shift register on the next **falling edge** of WRCK.

shiftWR

Input. When ShiftWR is high, the selected shift register moves data closer to WSO by one bit on the **rising edge** of WRCK.

captureWR

Input. When CaptureWR is high, the selected shift register captures the values of its corresponding status bits on the **rising edge** of WRCK.

WRCK

Input. This is the WTAP controller clock. Note that in Mentor Graphics' implementation, all WTAP sequential elements of the WTAP circuits are clocked by the input signal WRCKInvIn. This port will be connected directly to the TCK pad.

selectWIR

Input. When high, the WTAP instruction register is accessed. When low, the data register selected by the instruction register of the WTAP is accessed.

enableWR

Input. Enable WTAP access. When low, all WTAP registers hold their states. When high, the WTAP registers follow the action specified by UpdateWR, ShiftWR, and CaptureWR control signals. This port is a Mentor Graphics-specific addition to the IEEE 1500 standard.

extTM

Input. External Test Mode. Forces the WTAP controller to external test mode. When ExtTM is high, the WTAP is reset and the periphery scan chains are isolated from contribution of the core inside, and their control is released to external scan ports on the collared core. This port is Mentor Graphics-specific addition to the IEEE 1500 standard.

Ports for the Instruction Register

The following sections describe ports that interface with the WTAP Instruction Register.

instruction[q-1:0]

That output port carries the latched WTAP Instruction Register bit-by-bit values.

status[q-3:0]

Input port. The current value of signals connected to the status[] port are captured during a WIR access during the TAP capture-DR state, and then can shifted out during the following shift-DR states. Typically the Done and Go status outputs of the BIST controllers are connected to this port.

userIRBits[j-1:0]

This bussed output port carries a copy of the userIRbits portion of the WTAP Instruction Register latched output. The number of user IR bits depends on the value of the [NumberUserIRBits](#) property in either the *.etassemble* file.

userIRBitsNotLatched[j-1:0]

This bussed port carries a copy of the userIRbits portion of the WTAP Instruction Register, as they appear in the shift-register stage of the IR shadow register, before they get latched by the IR latching stage during capture-DR.

updateIR

The updateIR WTAP port indicates when the Instruction Register bits are clocked into the Register's update flops. The updateIR WTAP signal is provided to be used as a clock for the userBitsNotLatched[j-1:0] signals.

updateIREnable

The updateIREnable WTAP output port is the same as updateIR, except that it is decoded one- half TCK clock cycle earlier.

Ports for Test Mode Functions

The following sections describe the ports for test mode functions.

clkRatio[2:0]

The clkRatio[2:0] WTAP output port select the test clock source. These signals are the outputs of the Instruction bits clkRatio[2:0].

freezeMode

The freezeMode WTAP output port controls whether or not the clock is stopped after the capture of the last vector during logic BIST mode. This signal is the outputs of the Instruction bits freezeMode.

functMode

The functMode WTAP output port indicates the circuit is in normal operating mode, i.e., the Instruction Register bits clkRatio[2:0] are set to functional.

setupMode[2:0]

The setupMode[2:0] WTAP output port controls the run or setup mode of the currently selected embedded test controller. These signals are the outputs of the Instruction bits setupMode[2:0].

tckMode

The tckMode WTAP output port indicates the circuit is operating with the test clock, i.e., the Instruction Register bits clkRatio[2:0] are set to TCK.

testMode

The testMode WTAP output port indicates the circuit is in scan test mode. This signal is the output of the *most significant bit* of the Instruction bits SetupMode[2:0]. i.e., SetupMode[2].

diagMode

The diagMode output port indicates when the WTAP controller suppresses the capture cycle on the clock output clkBistTCK. When the capture cycle is suppressed, the logic BIST controller scans out the contents of the test data registers without corrupting the data. This signal is the outputs of the Instruction bits diagMode.

resetTest

The resetTest WTAP output port provides a reset signal for flip-flops during asynchronous reset tests. This signal is the outputs of the Instruction bits resetTest.

setTest

The setTest WTAP output port provides a set signal for flip-flops during asynchronous set tests. This signal is the outputs of the Instruction bits setTest.

Ports for Embedded Test Controllers

The following sections describe all ports embedded test controllers.

clkBistTCK

The clkBistTCK WTAP port is used to clock user-defined and embedded test controller at TCK speed. This output signal is a gated version of the WTAP input wrckInvIn.

asyncIntTCK

The AsyncIntTCK WTAP port connects to the TCK input of Mentor Graphics controller's Asynchronous Interface, and is a gated version of the WTAP input WRCK. The WTAP Instruction register bit InvertAsyncClock conditionally inverts the polarity of that signal respective to WRCK. Doing this increases the hold timing margin of the TAP BIST_SETUP and BIST_SI signals fanning out to the Mentor Graphics embedded test controllers. Refer to "[Instruction Register](#)" for more details.

fromBist[k-1:0]

Each of the elements of the fromBist[k-1:0] WTAP ports is connected to the Q port of the last flip-flop of the scan chain within the embedded test controller enabled by the corresponding element of the bistEn[k-1:0] WTAP output signal.

toBist

The toBist WTAP output port connects to the scan-data (SD) port of the first flip-flop of all embed test controller scan chains.

bistEn[k-1:0]

the bistEn[k-1:0] WTAP port is a bussed output port on which each element enables a specific embedded test controller. These signals are the outputs of the Instruction bits bistEn[k-1:0].

multiEn[k-1:0]

The multiEn[k-1:0] WTAP port is a bussed output port on which each element multiEn[i] indicates that the logicTest controller connected to the bistEn[i] port is in multi mode. It is used to control the auxiliary scan-in and scan-out ports of the logicTest controllers.

holdBist

The holdBist WTAP output port controls the holding mode of the embedded test controllers. This signal is used to instruct the selected embedded test controller to retain its state while its internal registers are being scanned.

shiftBist

The shiftBist WTAP port controls the shifting mode of the logic BIST and memory BIST controllers. The shiftBist signal is a delayed version of the shiftBist2Edge WTAP signal; it is delayed by half a WRCK-signal period.

shiftBist2Edge

The shiftBist2Edge TAP port controls the shifting mode of the logic BIST and memory BIST controllers which flops operate on the Leading Edge (LE) of clkBistTCK.

Ports for User-Defined Scan Registers

The following sections describe each of the user-defined scan register ports.

enableClkDR

The enableClkDR WTAP output port enables the clock input of the selected user-defined test data register. This signal can be routed directly to any flip-flop on the selected user-defined test data register to gate the WRCK WTAP signal.

captureDR

The captureDR WTAP output port enables the capture of the selected user-defined test data register when asserted. The captureDR signal is a delayed version of the captureDR2Edge WTAP signal. It is delayed by half a TCK period.

captureDR2Edge

The captureDR2Edge WTAP output port enables the capture of the selected user-defined test data register when asserted.

shiftDR

The shiftDR WTAP output port configures the selected user-defined test data register into a shift register when asserted. The signal is a delayed version of the shiftDR2Edge WTAP signal. It is delayed by half a WRCK-signal period.

shiftDR2Edge

The shiftDR2Edge WTAP output port configures the selected user-defined test data register into a shift register when asserted.

updateDR

The updateDR WTAP output port can be used as the clock signal for update latches in user-defined registers.

updateDREnable

The updateDREnable WTAP output port is the same as updateDR, except that it is decoded one-half TCK clock cycle earlier. It can be used in custom test data register implementations where the update latch stage is replaced by a flip-flop with a holding multiplexer.

testLogicReset

The testLogicReset WTAP output port indicates that it is reset by either the WRSTN input signal (when the TAP is in the Test-Logic-Reset state) or by the etxTM input signal (when the WTAP is configured in external test mode). The testLogicReset TAP signal can be used to reset user-defined logic.

Output Ports for Lower-Level WTAPs

The following sections describe the ports for lower-level WTAPs.

captureWRGated

The captureWRGated WTAP port is a gated version of the input captureWR. When a design contains WTAPs at different hierarchy levels, the input captureWR of lower-level WTAPs should all connect to the captureWRGated output of the higher-level WTAP.

shiftWRGated

The shiftWRGated WTAP port is a gated version of the input captureWR. When a design contains WTAPs at different hierarchy levels, the input shiftWR of lower-level WTAPs should all connect to the shiftWRGated output of the higher-level WTAP.

updateWRGated

The updateWRGated WTAP port is a gated version of the input captureWR. When a design contains WTAPs at different hierarchy levels, the input updateWR of lower-level WTAPs should all connect to the updateWRGated output of the higher-level WTAP.

Other Ports

The following sections describe the rest of the WTAP ports.

WRCKInvOut

The WRCKInvOut TAP output port is simply the WRCK input port inverted. This can be used to feed to the WTAP clock distribution circuit/buffer.

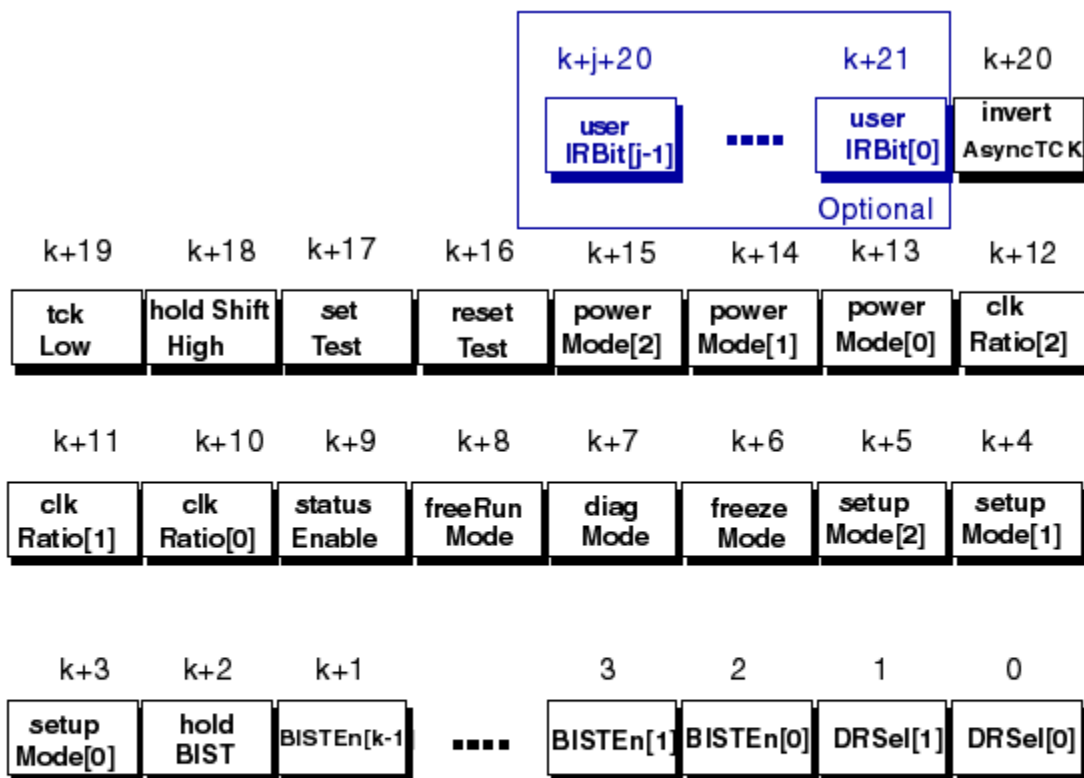
WRCKInvIn

The WRCKInvIn WTAP port is used to clock all flip-flops inside the WTAP controller. This port is the only clock domain used inside the TAP. The WTAP is synthesized using a `set_dont_touch_clock_network` on this port. The default source of this port is WRCKInvOut. If you need to insert a clock distribution macro for the WRCKInvIn clock network, you can insert it between the WRCKInvOut and WRCKInvIn ports.

Instruction Register

Figure 3-5 illustrates how the WTAP instruction register bits are organized.

Figure 3-5. WTAP Instruction Register Control Bit Assignments



Notes:

k—is the number of embedded test controllers.
j—is the number of UserIRBits.

Instruction Register Bit Assignment

This section describes the use of each WTAP instruction register bits.

DR_Sel[1:0]

The DRSel[1:0] Instruction Register bits define which Data Register is accessed. These bits are present in the Instruction Register only if the DeviceID and Bypass Registers are present in the *.etassemble* file.

The only exception to this selection scheme occurs when the SetupMode Instructions bits are “110”, in which case, the Seed Register is selected.

Table 3-5. Specifying the DRSel[1:0] Instruction Register Bits

SetupMode[2:0]	DRSel[1:0]	Selected Data Register
110	XX	Seed Register ¹
other values	00	Embedded Test Controller
	01	Bypass Register ¹
	10	Device ID Register ¹
	11	Bypass Register ¹

1. These registers are optional and their associated codes should not be used if the register is not present.

After a WTAP reset, The DRSel[1:0] selects the Device ID register if present. If not, the Bypass register is selected instead. If neither the Device ID nor the Bypass register is present, the embedded test controller register is selected.

BISTEn[k:1]

The BISTEn[K-1:0] Instruction Register bits are used to select the embedded test controllers to be active. More than one controller might be active in parallel during the execution of a test as long as they are the same type of controller (memory BIST, logic BIST, etc.), and that they are run with the same operation mode as defined by the other Instruction Register bits.

holdBIST

The holdBist Instruction register bit controls the holding mode of the selected embedded test controllers. This signal is used to instruct the selected embedded test controller to retain its state while its internal registers are being scanned.

setupMode[2:0]

The setupMode[2:0] Instruction Register bits specify the run, setup, or scan mode for the currently selected embedded test controller. [Table 3-6](#) lists the setupMode[2:0] values and the corresponding modes. Descriptions of these modes are provided below:

- The Short Setup mode places the embedded test controller in scan mode and provides access to the controller’s short setup scan chain. The contents of the short setup scan chain differ among controller types, but, generally, contain a relatively small number of registers used to set various runtime test options.
- The Long Setup mode places the embedded test controller in scan mode and provides access to the controller’s long setup scan chain. The contents of the long setup scan chain differ among controller types, but, generally, contain a relatively large number of

registers used to set various runtime test and diagnostic options, as well as provide access to detailed test and diagnostic result data.

- The Default Run mode places the embedded test controller in run mode. In this mode, the controller uses default initialization values and disregards any values scanned in using either the Short or Long Setup modes.
- The Normal Run mode places the embedded test controller in run mode. In this mode, the controller makes use of the initialization values scanned in using either the Short or Long Setup modes.
- The Single Chain Scan mode places the embedded logic test controller in scan mode and provides access to the flop scan chains as one concatenated scan chain for diagnose and single mode ATPG.
- The Multiple Chain Scan mode places the embedded logic test controller in scan mode and provides access to the flop scan through multiple scan chains for multi mode ATPG.
- The PRPG Reseed Run (ATPG Compression run) mode places the embedded logic test controller in ATPG Compression mode. In this mode, PRPG seeds are shifted in the WTAP through its WSI input and transferred in parallel to the embedded logic test controller.
- The Controller Chain Run mode places the embedded logic test controller in a mode where a portion of embedded logic test controller and the Seed register of the WTAP (if present) are configured in a single scan chain for controller chain mode ATPG.

Table 3-6. Specifying the `setupMode[2:0]` Instruction Register Bits

<code>setupMode[2:0]</code> Instruction Bits	Selected Mode
000	Short Setup
001	Long Setup
010	Default Run
011	Normal Run
100	Single Chain Scan ¹
101	Multi Chain Scan ¹
110	PRPG Reseed Run ¹
111	Controller Chain Run ¹

1. Mode used for logic BIST controllers only

freezeMode

The freezeMode instruction register bit controls whether or not the clock prescaler stops the clock after the capture of the last vector.

diagMode

The diagMode Instruction register bit specifies that the capture cycle will be suppressed. When the capture cycle is suppressed, the contents of the Data Register or scan chain can be shifted out without corrupting the data. This mode is used for flop diagnose.

freeRunMode

The freeRunMode Instruction Register bit forces the WTAP gated clock output clkBistTck to run freely. This is used when lower-level ELT scan chain needs to be initialized.

clkRatio[2:0]

The clkRatio[2:0] instruction register bits specify the clock that is used to perform a scan test or run any of the embedded test controllers.

Table 3-7 lists the clkRatio[2:0] values and the corresponding clock.

Table 3-7. Specifying the clkRatio[2:0] Instruction Register Bits

clkRatio[2:0]	Selected Clock
0 0 0	functionalMode
0 0 1	sysClock ¹
0 1 0	sysClock/2 ¹
0 1 1	sysClock/4 ¹
1 0 0	TCK
1 0 1	sysClock PLL ¹
1 1 0	sysClock/2 PLL ¹
1 1 1	sysClock/4 PLL ¹

1. Supported values but have no effect on the circuit

enableStatus

The statusEnable Instruction Register bit is used to select between forcing the instruction shift register to capturing the associated update flop or the IRStatus signals. This allows capturing the output of the IR update flop to structurally test them including their reset states.

resetTest

The resetTest Instruction Register bit forces asynchronous reset pins on flip-flops to be asserted when performing an asynchronous reset test. This bit is normally OFF (logic 0).

setTest

The setTest Instruction Register bit forces asynchronous set pins on flip-flops to be asserted when performing an asynchronous set test. This bit is normally OFF (logic 0).

holdShiftHigh

The holdShiftHigh Instruction register bit forces the shiftBist signal to be held high. Normally, this bit is OFF (logic 0) but is used for retention tests and asynchronous set/reset tests.

tckLow

The tckLow Instruction Register bit forces the clkBistTck output clock signal to be held low instead of high. This bit is normally OFF (logic 0) but is used for retention tests and asynchronous set/reset tests.

invertAsyncTCK

This IR bit controls the polarity inversion of the TAP asyncIntTck output fanning out to each Mentor Graphics controller's TAP asynchronous interface. AsyncIntTCK is a buffered version of the input WRCK.

Control of this bit is transparent to you, and is set up by the Mentor Graphics testbenches. Wherever a test is run with a tckRatio greater or equal to 8, this bit is asserted to one and inverts the asyncIntTck. Doing this increases the hold margin of the TAP BIST_SETUP and BIST_SI signals fanning out to the Mentor Graphics embedded test controllers. You can always increase that margin further by specifying a higher tckRatio in your ETVerify configuration file. The default tckRatio for all controllers is set to 16, which means the TCK clock frequency is 16 times smaller than the operating frequency of the controller, therefore allowing an extended setup and hold timing margin for all TAP interface signals.

userIRBit[J-1:0]

The userIRBits[J-1:0] Instruction Register bits are for user-defined purposes and as such have no value assignment definitions. However, depending on your design configuration, some userBits are strictly reserved for Mentor Graphics' use. The number of **free** available userBits will always equal the number you specified in your *.etassemble* file [NumberUserBits](#) property. When both free userIRbits and reserved IRbits are present, the reserved ones are always put on the most significant bits side.

For example, user-specified **NumberUserBits = 5** for the TAP. In this case:

- UserIRBit[4:0] are allowed for user-defined signals.
- UserIRBit[13:5] might be reserved for Mentor Graphics' use.

ETAssemble configuration file properties that refer to a combination of user bits, such as [TAP: UserSignal: Code](#) only refer to the free user bits. Reserved user bits are ignored.

The number of Mentor Graphics reserved bits vary from one design to the next, and those are typically used for controlling some logicTest controller or memory BIST controller configuration inputs. Here is an example list of reserved user bits. Offset is the userIRbit index of the first reserved IR bit. All IR bits are set to Off (meaning they drive a logicLow into the core) upon a WTAP reset.

userIRBit[Offset] ETClockEnable

This bit is present only when there is at least one Mentor Graphics embedded test controller in your design (besides the TAP and the boundary-scan). ETClockEnable is low in functional mode, and ETVerify asserts it high during all Mentor Graphics controllers run modes.

Use the ETClockEnable signal to customize your clock distribution network if your functional mode clocking differs from your test mode clocking.

For example, you might want to run your memoryBist mode at a different frequency than the functional clock domain to which it belongs, or you might want to avoid using some of your functional clocks in LogicBist mode. Refer to “[lv.ETClockEnable](#)” on how to use ETClockEnable in the *ETChecker User’s and Reference Manual*.

userIRBit[Offset+1] BistClockEnable

This bit is present only when the WTAP controls one or many memory BIST controllers, and if you specified the following property in your ETPlanner configuration file:

```
GateBistClockInFuncMode: Yes;
```

The memory BIST hardware can then save power in functional mode by blocking the clock feeding all flops in the controller.

BistClockEnable is low in functional mode, and ETVerify asserts it high during memoryBist mode.

userIRBit[Offset+2] BurstEnable

This optional bit is added only when a logicTest controller is present in your design. It controls the enabling of the at-speed BurstMode.

userIRBit[Offset+7: Offset+3]

Those optional bits control test configuration inputs that are exclusive to Tessent MemoryBIST NonProgrammable controllers. The list below shows an example of possible control signals, but more might be added depending on the design. For details refer to the *Tessent MemoryBIST User’s and Reference Manual*.

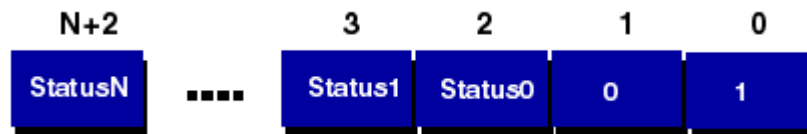
- **MBIST_RST_MEM** — instructs a NonProgrammable memory BIST controller to reset the memories writing zeros to all memory locations.
- **BIST_DIAG_EN** — activates the diagnostic mode of the NonProgrammable controller during CompStat and Stop-On-Nth-Error diagnostics.
- **BIST_ALGO_MODE** — specifies the algorithm phase to be applied by the NonProgrammable controller during parallel retention testing.

Status Register Bit Assignment

During the Capture-IR TAP controller state, when the Instruction Register bit—enable Status—has been previously asserted, the WTAP controller captures the values present on the Status Register inputs. In the following TAP controller state, the Shift-IR TAP controller state, the Status Register contents are serially shifted out through the WSO port—while the WTAP controller serially loads a new instruction register opcode through WSI.

In this capacity, the Instruction Register is referred to as the Status Register. As shown on [Figure 3-6](#), an N-bit Instruction Register, where $N = K + J + 19$, contains N-2 status bits. The first 2 bits of the Status Register are always 01, where 1 is the first bit seen at the TDO pin.

Figure 3-6. Wrapper Instruction Register Status Bit Assignments



Test Data Registers

The Test Data Registers, shown in [Figure 3-1](#) are as follows:

- **Device ID Register** — an optional test data register that provides a unique code to identify the core or block.
- **Bypass Register** — an optional bypass register that shifts data from the WSI input port of the WTAP controller to the WSO output port of the TAP controller.
- **Seed Register** — an optional register used for the ATPG Compression mode. In this mode the register is serially loaded through the WSI input port of the WTAP controller and loaded in parallel in the logic BIST controller.

Device ID Register

The optional device ID test data register provides a unique code that identifies the core or the block. The device ID register contains 32 parallel-in, serial-out shift-register stages that identify the version number, the part number, and the manufacturer of your IC.

Figure 3-7 illustrates the structure of the device ID code.

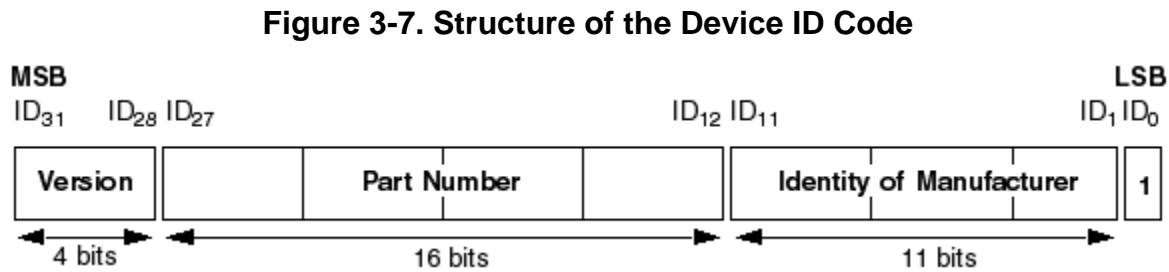


Table 3-8 describes how the TAP controller states affect the behavior of the Device ID Register.

Table 3-8. Behavior of the Device ID Register

During this TAP Controller State...	The WTAP Device ID Register...
Test-Logic-Reset	Retains the last state
Capture-DR	Loads the device ID code
Shift-DR	Shifts the ID towards the tdo TAP port
All other states	Retains the last state

Bypass Register

The optional Bypass Test Data Register, consisting of a single shift-register stage, shifts information from the tdi port to the WSO TAP port without interfering with the normal operation of your IC.

The Bypass register is selected by assigning 10 to the DRSel Instruction Register bit.

For more information on the DRSel Instruction Register bit, refer to [DR_Sel\[1:0\]](#).

Once selected, the Bypass Register loads a constant logic 0 into the shift-register stage on a rising edge of WRCK during the Capture-DR controller state of the TAP. [Table 3-9](#) describes how the TAP controller states affect the shift-register stage.

Table 3-9. Behavior of the Bypass Register

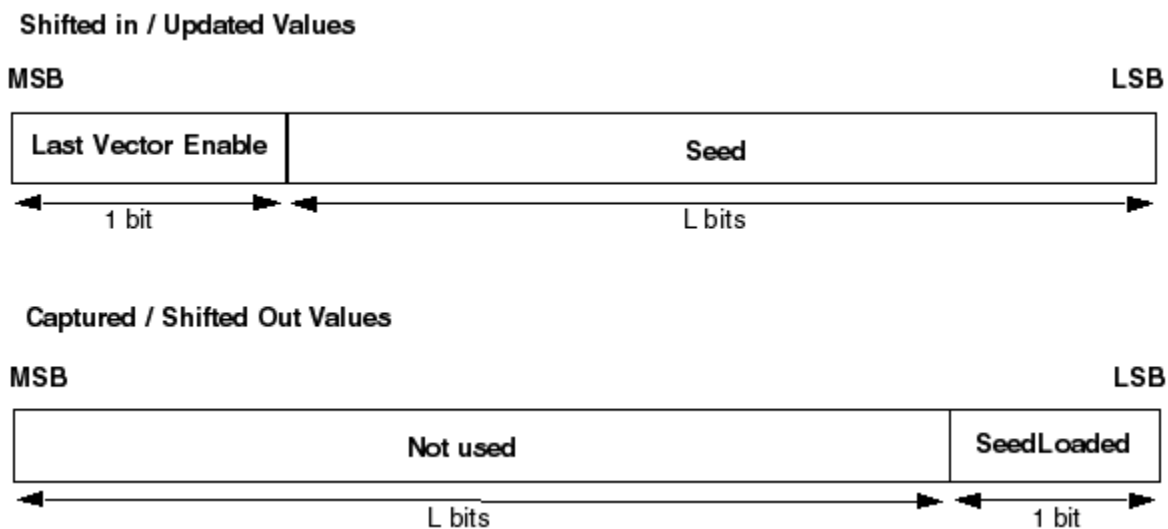
During this TAP Controller State...	The WTAP Bypass Register...
Test-Logic-Reset	Retains the last state
Capture-DR	Loads a logic-zero
Shift-DR	Shifts towards the tdo TAP port
All other states	Retains the last state

Seed Register

The optional Seed register is used to serially shift the PRPG seed values inside the WTAP and to provide the values in parallel to the logic BIST controller. The control signal LastVectorEnable and the status signal SeedLoaded are also transferred through this register.

[Figure 3-8](#) illustrates the structure of the Seed register.

Figure 3-8. Structure of the Seed Register



[Table 3-10](#) describes how the TAP controller states affect the behavior of the Seed Register.

Table 3-10. Behavior of the Seed Register

During this TAP Controller State...	The WTAP Seed Register...
Test-Logic-Reset	Retains the last state

Table 3-10. Behavior of the Seed Register (cont.)

During this TAP Controller State...	The WTAP Seed Register...
Capture-DR	Loads the input SeedLoaded in the Register
Shift-DR	Shifts out the SeedLoaded towards the WSO WTAP port and shifts in the Seed and LastVectorEnable values from the WSI WTAP port.
All other states	Retains the last state

Chapter 4

Boundary-Scan Cells

Mentor Graphics' ETAssemble tool creates boundary-scan cells that are parameterized to match the I/O requirements of the design and groups these boundary-scan cells into hierarchical assemblies called bgroups. The tool can create different types of boundary-scan cells and then can group the cells into bgroups according to user input. This chapter describes the features and the circuit designs of the most commonly used boundary-scan cells that can be created by ETAssemble. This chapter also gives an overview on the operation of boundary-scan in general.

This chapter covers the following topics:

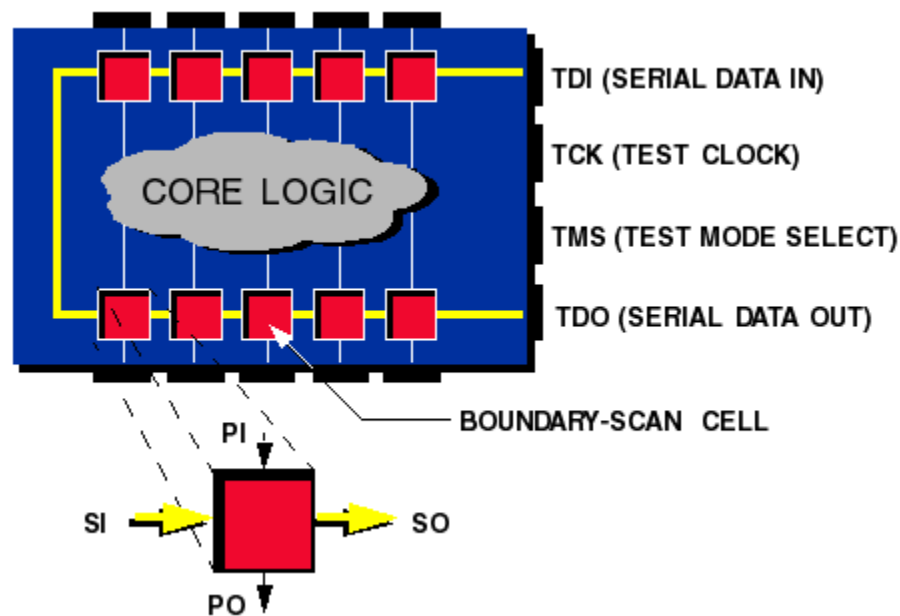
- [Boundary Scan Overview](#)
 - [Boundary-Scan Operating Modes](#)
 - [Uses of Boundary Scan](#)
- [Boundary-Scan Cell Features](#)
 - [Classes and Subclasses](#)
 - [Clocking](#)
 - [Boundary-Scan Cells Control and Data Pins](#)
- [Basic Boundary-Scan Cells](#)
 - [I: Input Boundary -Scan Cells](#)
 - [O: Output Boundary-Scan Cells](#)
 - [IO: Bidirectional Boundary-Scan Cells](#)
 - [EN: Enable Boundary-Scan Cells](#)
 - [S: Sample-Only](#)
- [Subclasses for Complex Boundary-Scan Cells](#)
 - [NM: No JTAG Multiplexer](#)
 - [H: Holding](#)
 - [XO: Auxiliary Data Multiplexer](#)

Boundary Scan Overview

Boundary scan is a structured design-for-test technique developed in the late 1980s by the Joint Test Action Group (JTAG) and standardized in 1990 by the IEEE as a fundamental component of the 1149.1 standard. When applied to an ASIC's input and output pins, boundary scan enhances the ASIC's accessibility and testability. With boundary scan, each input/output (I/O) bonding pad has an associated shift-register stage (contained in a boundary-scan cell).

Figure 4-1 shows an IC with boundary-scan cells inserted between the I/O pads and the core logic.

Figure 4-1. IC with Boundary Scan



Notice that each boundary-scan cell has four data port types:

- PI — Parallel input. Signal originates at either an I/O pad or the core.
- PO — Parallel output. Signal connects to either the core or an I/O pad.
- SI — Scan-In port. Signal originates at either the TDI pin or the previous boundary-scan cell.
- SO — Scan-out port. Signal connects to either the next boundary-scan cell or the TDO pin.

The SI and SO ports of each boundary-scan cell are connected between TDI and TDO to form the boundary-scan register. (The Mentor Graphics Assembly tool automatically inserts the boundary-scan cells between the pads and the core and connects the cells.)

Boundary-Scan Operating Modes

The following section discusses the boundary-scan operating modes.

The boundary-scan register has three operating modes:

- SAMPLE or bypass mode performs tests without interfering with the normal operation of the IC. The SAMPLE mode is valuable for design debugging and fault diagnosis since connections inaccessible to the tester can be examined.
- INTEST or internal test mode tests the logic within your IC.
- EXTEST or external test mode performs board-level interconnect testing.

During these operating modes, the TAP can configure the boundary-scan cells several ways to perform various functions, including the following:

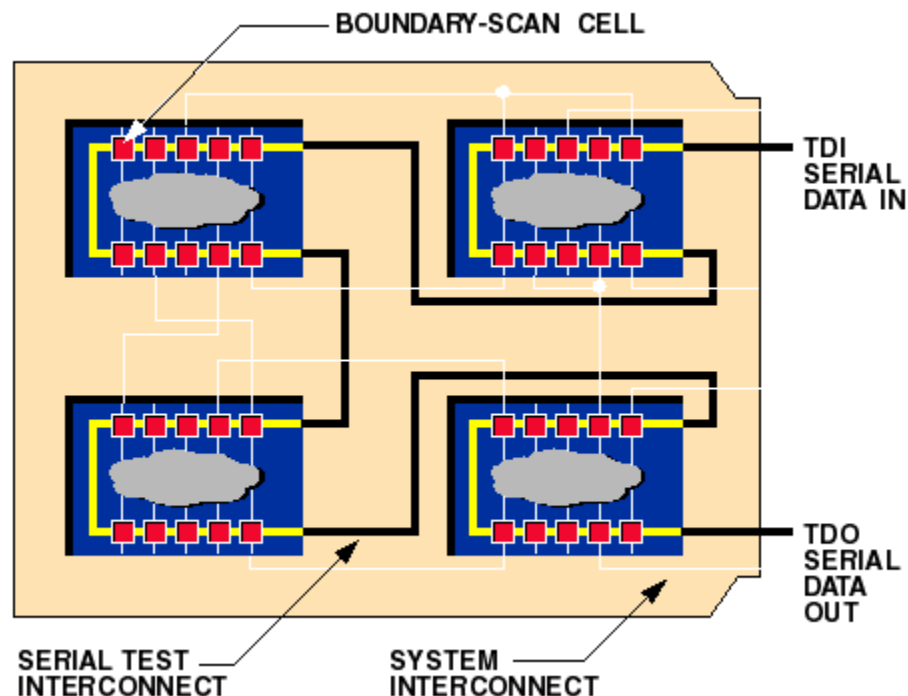
- Sampling or forcing of ASIC inputs
- Sampling or forcing of ASIC outputs
- Forcing of bidirectional pin inputs and sampling of bidirectional pin outputs

In the Mentor Graphics boundary-scan implementation, the selectJtagInput and selectJtagOutput TAP signals, which are inverted versions of the instruction register bits selectJtagIn_BAR and selectJtagOut_BAR, control the boundary-scan cells and select the boundary-scan operating mode. ([Table 2-10](#) lists the selectJtagIn_BAR and the selectJtagOut_BAR values and the corresponding operating mode.)

Uses of Boundary Scan

Boundary scan is primarily used for board-level interconnect testing and the integration of IC-level testing with board-level testing. [Figure 4-2](#) shows a boundary-scannable board design.

Figure 4-2. A Boundary-Scannable Board Design



When all of a board's components include boundary scan (as shown in [Figure 4-2](#)), the boundary-scan register in each component can be connected serially. The resulting serial data path allows the following:

- Testing of the connections between various components
- Testing of the individual components on the board

Specifically, boundary scan does the following:

- Detects ASIC bonding and I/O continuity defects
- Tests PCB connectivity, specifically shorts and opens
- Checks DC parametrics on input and output pins
- Disables outputs for in-circuit PCB tests
- Eliminates bed-of-nails testing
- Isolates ASICs at the PCB-level
- Takes snapshots of I/O states during PCB operation

Boundary-Scan Cell Features

The ETAssemble tool integrates technology-independent boundary-scan cells into your design. These boundary-scan cells comply with the IEEE 1149.1 standard and include observation and control circuits for both data and pad enable paths. Specific cells for the IEEE 1149.6 standard are also supported. For details, refer to [Support for IEEE 1149.6 Boundary Scan](#).

The following types of boundary-scan cells can be generated:

- Input or observe-only boundary-scan cells
- Output boundary-scan cells
- Bidirectional boundary-scan cells
- Enable boundary-scan cells

These technology-independent boundary-scan cells are created in RTL. Their Verilog description must be synthesized along with the rest of the chip design by the designer.

Note



Mentor Graphics does not recommend making functional modifications to the generated boundary-scan cells; however, modifications for timing correctness or optimization are acceptable.

Classes and Subclasses

The Generate tool creates boundary-scan cells based on user-input specifications. These input specifications use class and subclass identifiers to specify what features to build into the boundary-scan cells.

Only a sample set among the possible boundary-scan cell class and subclass combinations will be presented in this chapter. The list was chosen so as to show the maximum number of important features by using a minimum number of examples. Many other custom boundary-scan cell class/subclass combinations are possible.

Both basic and custom boundary-scan cells may be defined in the BCELL column of the *.table* file using the following general syntax:

```
<class>[(<subclass1>,[<subclass2>,[...<subclassN>]])]
```

Users can either use reserved class keywords to make the Generate tool create any of the Mentor Graphics boundary-scan cells or simply refer to the name of the Verilog or VHDL object.

There are four classes of boundary-scan cell that can be produced: I, O, EN, IO. A series of subclasses may also be added to the class label to complete the cell specification. See [Table 4-1](#) for a list of classes and [Table 4-2](#) for a description of the subclasses.

Table 4-1. Boundary-Scan Cell Class Attributes

Class	Description
I	Input cells Valid Subclasses: CK, I, NM, S.
O	Output cells Valid Subclasses: CK, H, I, NM, OD, OS, XO.
EN	Enable cells Valid Subclasses: 0, 1, H.
IO	Bidirectional cells Valid Subclasses: CK,C2, DI, DO, H, I/II/IO, NM, OD, OS, S, XO.

Table 4-2. Boundary-Scan Cell Subclass Attributes

Subclass	Description
0	The active level of the enable controlling the output pad associated to this enable cell is 0. This is the default.
1	The active level of the enable controlling the output pad associated to this enable cell is 1.
CK	The pin is a clock. If it's an input pin, it requires a "clock" type bscan cell, which only observes the pin without controlling it in logictest mode. If the pin is an output, the output or bidir cell must capture itself in logictest mode.
C2	The bidirectional boundary-scan cell has two separate boundary-scan registers—one for output direction data and one for input direction data. Without the C2 subclass, the Tessent bidirectional boundary-scan cell uses only one register for both directions.

Table 4-2. Boundary-Scan Cell Subclass Attributes (cont.)

Subclass	Description
DI	<p>This subclass is inferred for bidirectional pads which “fromPad” pin was left unconnected in the functional netlist, so as to serve as output-only pads in functional mode, despite their connection to a top-level port of direction “inout.” In the generated BSDL file, the package pin direction will stay “inout” and the cell function “bidir.” These functional output-only pins then become inout pins in boundary-scan mode, providing better test flexibility and diagnostic. For example, during wafer tests, the bidirectional configuration enables stuck-at testing of your pad logic with no tester contact to the pin by taking advantage of its inherent loopback structure.</p> <p>If instead you want your BSDL to document the pin and the cell as “output-only,” you need to change your netlist port direction to “output” as well.</p> <p>The IO(DI) cell does not include an SJI multiplexer because it does not need to intercept the pad-to-core data path.</p>
DO	<p>This subclass is inferred for bidirectional pads whose “toPad” pin was tied off in the functional netlist, so as to serve as input-only pads in functional mode, despite their connection to a top-level port of direction “inout.” In the generated BSDL file, the package pin direction will stay “inout” and the cell function “bidir.” These functional input-only pins then become inout pins in boundary-scan mode providing better test flexibility and diagnostic. For example, during wafer tests, the bidirectional configuration enables stuck-at testing of your pad logic with no tester contact to the pin by taking advantage of its inherent loopback structure.</p> <p>If instead you want your BSDL to document the pin and the cell as “input-only,” you need to change your netlist port direction to “input” as well.</p> <p>The IO(DO) cell does not include an SJO multiplexer because it does not need to intercept the core-to-pad data path.</p>
H	<p>Applies to output (O), bidirectional (IO), and control (EN) classes of boundary-scan cells. This cell always captures the output of its own update latch in all possible modes, including SAMPLE, EXTEST, scan, and logicBist. (A normal output boundary scan cell captures the value that comes from the core in scan and logicBist modes.) Note that a bidirectional boundary scan cell with subclass H captures its update latch output in scan and logicBist modes and captures the pin value in SAMPLE and EXTEST modes.</p>
I/II/IO	<p>Assumes that the pad inverts its data at the fromPad/toPad pin, requiring the boundary-scan cell to do the same internally, so as to comply with the 1149.1 standard which forces the boundary-scan register polarity to be the same as the pin. II and IO apply to bidir cells: II means “invert the input path,” IO means “invert the output path.”</p>

Table 4-2. Boundary-Scan Cell Subclass Attributes (cont.)

Subclass	Description
NM	No Mux. The 2:1 mux controlled by selectJtagInput/Output is inside the pad cell.
OS/OD	The cell provides the HIGHZ instruction support to an open-drain or open-source pad which has no forceDisable signal. The boundary scan cell would force the pad data to its disabling state when the TAP controller's ForceDisable signal is asserted.
S	When the S subclass is added to an input (I) boundary-scan cell, the cell converts to a Sample-Only cell (observe-only cell, by the standard).
XO	The boundary-scan cell has an inner extra mux on the test side of the selectJtag mux. This allows you to multiplex auxiliary output test signals and functional signals without adding any further mux on the functional path. XO is automatically inferred by the Generate tool when needed by some options.

Clocking

The clock input for the boundary-scan cells connects to the clockBscan TAP port. This clock is a gated, inverted version of the TCK TAP signal. The active clock edge for both the TCK and clockBscan TAP signals is the rising edge.

Note



During logicTest, the shift clock controller drives clockBscan at the frequency of its selected shiftClock. The SCC also generates an UpdateBscan waveform that makes the cell's updateLatch stretch the hold of all paths from the bscan cells to the core.

Serial Path Retiming

Because the boundary-scan chain runs around the periphery of the device and can be several centimeters in length, each of the boundary-scan flip-flops is susceptible to clock skew. To eliminate this problem, all generated boundary-scan cells insert a retiming flop on their serial input path. That flop is also used when capturing the chip's pin value, making it happen on the rising edge of TCK, as prescribed by the standard.

Boundary-Scan Cells Control and Data Pins

Mentor Graphics Boundary-scan cells control signals generally come from the TAP, although the logicBist controller may override some of them during test mode. They are as follows:

- clockBscan
- updateBscan

- shiftBscan2Edge
- selectJtagInput
- selectJtagOutput
- BscanShiftIn, BscanShiftOut
- ForceDisable

The other pins carry data signals that interface with the pads or the core. [Table 4-3](#) describes the various boundary-scan data and control signals that appear in the schematics.

Table 4-3. Boundary-Scan Data and Control Signals

Signal Name	Signal Description	Connection	Description
clockBscan	Clock	clockBscan TAP port	The boundary-scan cell clock input.
forceDisable	Force disable	instruction[5] TAP port	Disables all three-state pads simultaneously when instruction[5] 0.
fromPad	Pad output (ASIC input)	Output from the input buffer	The data output from the input buffer.h
padEnable0	Pad enable active low	Pad enable port	The three-state enable port for the pad.
padEnable1	Pad enable active High	Pad enable port	The three-state enable port for the pad.
bscanShiftIn	Scan in	Connection to the next/previous cell, to the TAP, or to the Router.	Boundary-scan serial input connection, which is dependent on the chip's configuration and the cell's position inside the boundary-scan segments.
bscanShiftOut	Scan Out	Connection to the next/previous cell, to the TAP, or to the Router.	Boundary-scan serial output connection, which is dependent on the chip's configuration and the cell's position inside the boundary-scan segments.
shiftBscan	Bscan Shift Control input	shiftBscan TAP port	Configures the boundary-scan register into a shift register when shiftBscan 1.
shiftBscan2Edge	Bscan Shift Control input	From the output of the TAP controller port by the same name.	The shiftBscan2Edge TAP port configures the boundary-scan register when the generated shiftBscan2Edge TAP signal is active-high.
selectJtagInput	Select in	selectJtagInput TAP port.	Switches control of the core between the boundary-scan register and the primary input pins: <ul style="list-style-type: none"> • When selectJtagInput 1, the boundary-scan data are the input data to the core. • When selectJtagInput 0, the pad input data are the input data to the core.

Table 4-3. Boundary-Scan Data and Control Signals (cont.)

Signal Name	Signal Description	Connection	Description
selectJtagOutput	Select out	selectJtagOutput TAP port.	Switches control of the primary output pins between the chip logic and the boundary-scan register: <ul style="list-style-type: none"> When selectJtagOutput 1, the boundary-scan data are the output data for the pad. When selectJtagOutput 0, the ASIC core data are the output data for the pad.
toPad	Pad input (ASIC output)	Input to the output buffer.	The data input to the output buffer.
padIO	Pad input/output (ASIC front-end)	Input to the input buffer and the Output to the output buffer.	The data input/output for the chip external interface.
updateBscan	Load	updateBscan TAP port	Controls when the TAP updates the boundary-scan register: <ul style="list-style-type: none"> When updateBscan 1, the TAP updates the boundary-scan output latch. This prevents the output from rippling as the TAP shifts the boundary-scan register. When updateBscan 0, the TAP shifts data through the boundary-scan output latch.
<auxiliary data enable signal>	User enable signal for selecting the auxiliary MUX	Can be a signal name derived from a pad input specified in the .btable or the output of the TAP, such as bistEn port or any signal from the user core to control the auxiliary mux.	This is the enable signal that is used to control the auxiliary mux that resides in the output Bcell, for selecting between the functional data or scan data, going to the pad.
auxMuxData	Auxiliary data input to the auxiliary MUX within the output Bcell.	Can be a signal name derived from a pad input specified in the .btable or the output of the TAP, such as bistEn port or any signal from the user core to control the auxiliary mux.	This data can be coming from the scan-out ports of the internal scan chains during the multi chains mode or from the observation of some internal test or diagnostic data, such as CMP_STAT from the memBist controllers at the pads.
userEnable	User enable	Core functional enable	The three-state enable.

Basic Boundary-Scan Cells

The basic boundary-scan cells are those used for simple input, output and bidirectional pad cells, as well as those used for bidirectional pad enables and those used on input or output pads that should only be sampled because of performance issues.

I: Input Boundary -Scan Cells

The input boundary-scan cells intercept the data input from input-only pads. Based on the boundary-scan operating mode, either the boundary-scan cell or the input only pad sends data to the core. Refer to [Figure 4-3](#).

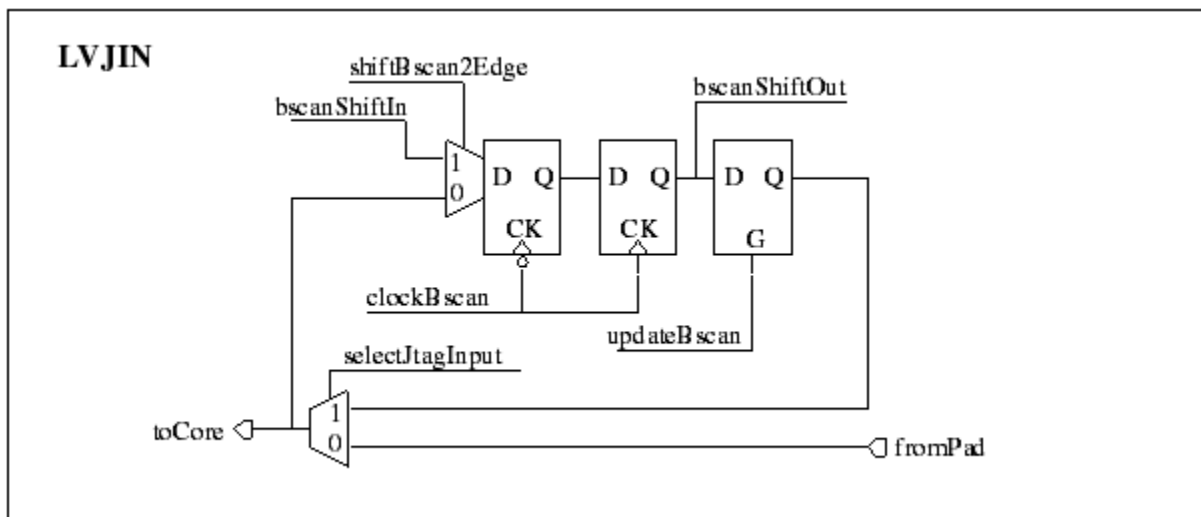
The Mentor Graphics input boundary-scan cells sample the data after the input multiplexer allowing full testing of the logic in the boundary-scan cell without having to probe the pad.

Note



None of the Mentor Graphics input boundary-scan cells can hold the signal driven into the core at a constant value during board-level interconnect testing.

Figure 4-3. I: Basic Input Boundary-Scan Cell



O: Output Boundary-Scan Cells

The output boundary-scan cells intercept the core logic data output to the output-only pads. Based on the boundary-scan operating mode, either the boundary-scan cell or the core sends data to the output-only pad. Refer to [Figure 4-4](#).

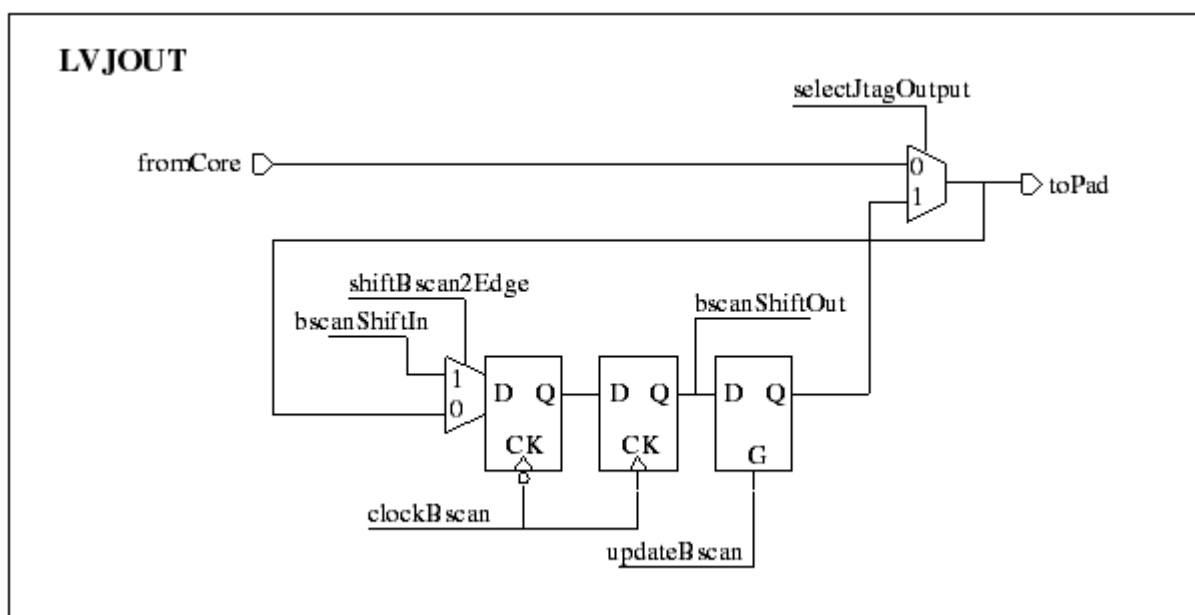
These cells sample the data after the output multiplexer (except when specifying subclass H), allowing full testing of the logic in the boundary-scan cell without having to probe the pad.

Note



The Mentor Graphics output boundary-scan cells cannot hold the value driven out from the cell at a constant value during internal scan testing and logic BIST. By default, assuming a three-state pad, the forceDisable TAP signal places the pad in a high impedance state during an internal test.

Figure 4-4. O: Basic Output Boundary-Scan Cell



IO: Bidirectional Boundary-Scan Cells

The bidirectional boundary-scan cells intercept both the data output to and the data input from bidirectional pads. Based on the boundary-scan operating mode, on the output side, either the boundary-scan cell or the core transmits data to the pad. On the input side, the source of data is either the boundary-scan cell or the pad. Refer to [Figure 4-5](#).

The Mentor Graphics bidirectional boundary-scan cells sample the data after the input and output multiplexers (except when subclass H is specified) allowing full testing of the logic in the boundary-scan cell without having to probe the pad.

Note

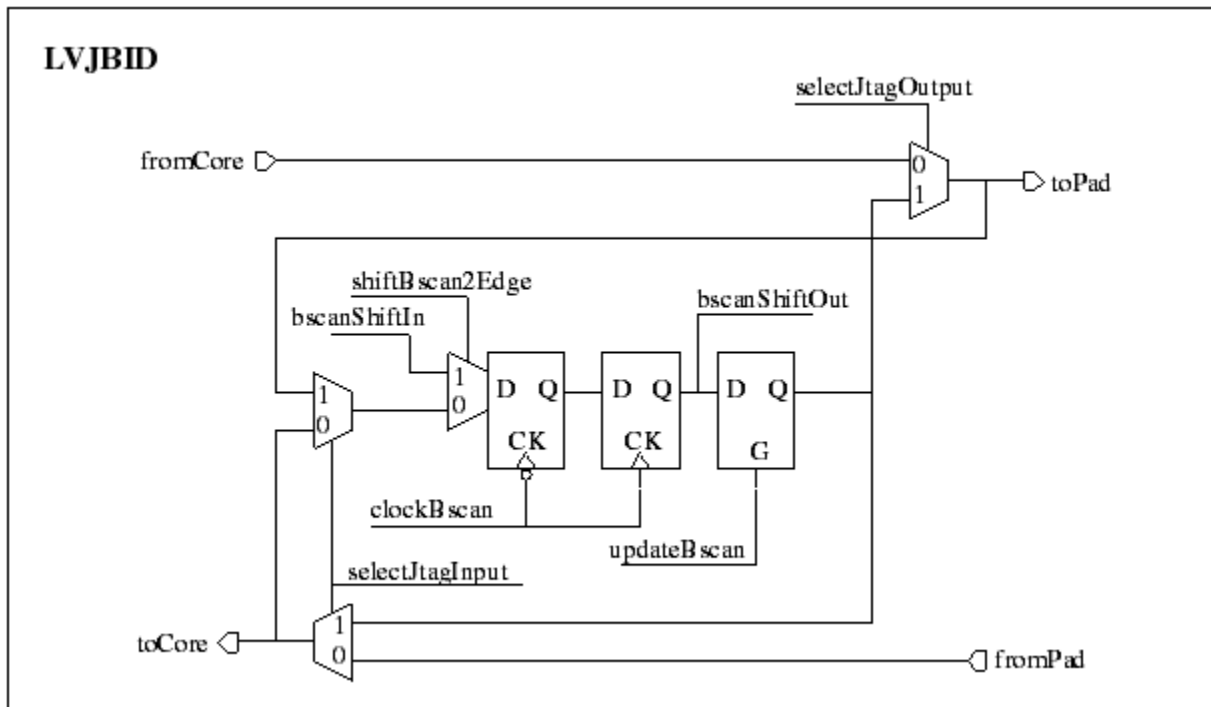


The Mentor Graphics bidirectional boundary-scan cells cannot hold the following:

- The signal driven into the core at a constant value during board-level interconnect testing.

- The value driven out from the boundary-scan cell at a constant value during internal scan testing and logic BIST. By default, assuming a three-state pad, the forceDisable TAP signal places the pad in a high impedance state during an internal test.

Figure 4-5. IO: Basic I/O Boundary-Scan Cell



EN: Enable Boundary-Scan Cells

The enable boundary-scan cells sample and control the three-state enables on three-state output pads and bidirectional pads. LVJEN0 is for zero-enabled pad; LVJEN1 is for positive-enabled pads. LVJEN0 is the default, whereas LVJEN1 is obtained by specifying subclass “1.” Refer to [Figure 4-6](#) and [Figure 4-7](#).

During the EXTEST TAP instruction, selectJtagOutput is asserted, and the bscan register value controls the output of the pad. forceDisable forces padEnable0/1 into the disable state (HIGHZ), which allows the TAP to implement the HIGHZ test. The selected logic depends upon the polarity of the padEnable0/1 signal.

During multi-chain scanning, the forceDisable signal is asserted for all enable cells. However, any pad used as an auxiliary serial port (AUX_SO) will be individually forced enabled by some extra logic located inside the pad’s own bgroup.

Figure 4-6. EN: Basic Enable0 Boundary-Scan Cell

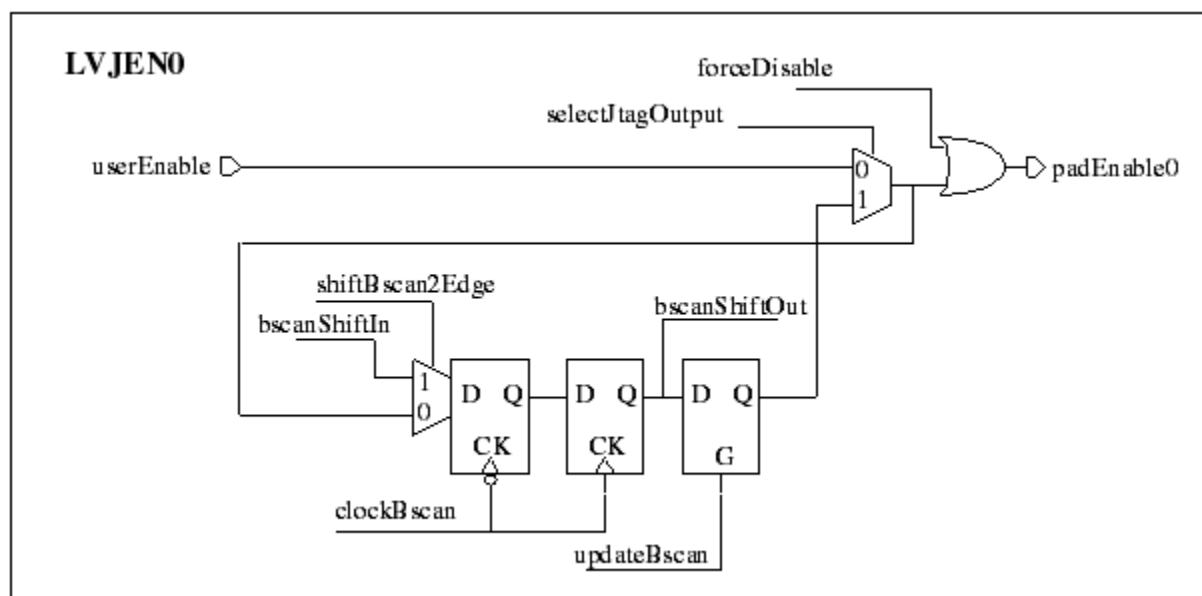
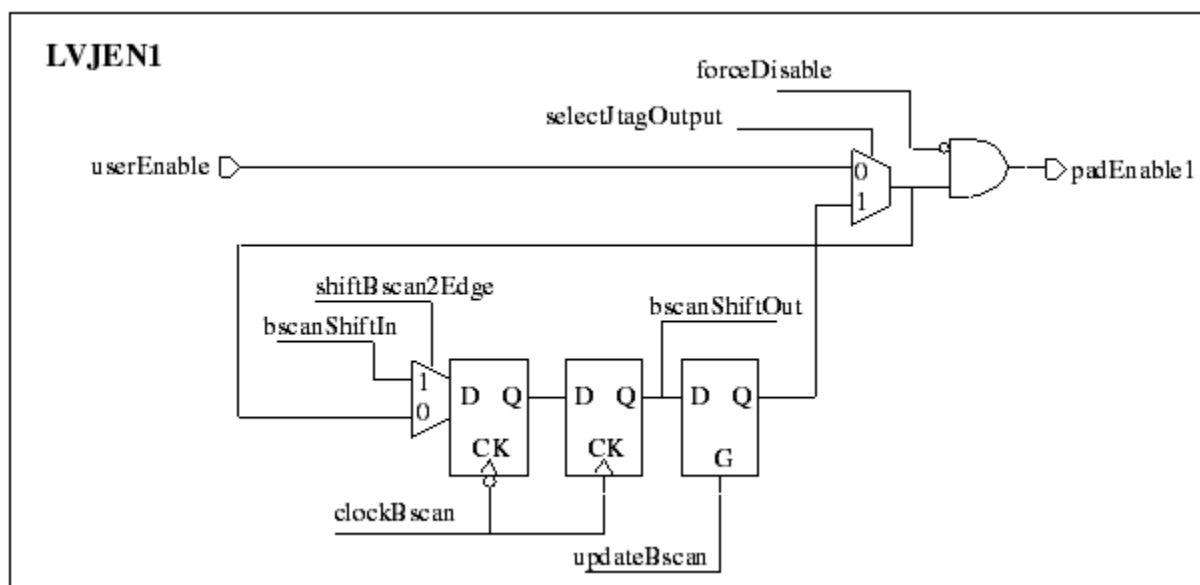


Figure 4-7. EN(1): Basic Enable1 Boundary-Scan Cell



Subclasses for Complex Boundary-Scan Cells

Many subclasses can be used for most or all the different boundary-scan cell classes. Often, the effect is the same from one class to the other. Therefore, all the subclasses are shown here in the context of one class only. Exceptions to that rule occur only when necessary. Each schematic shows the following:

- Logic added by the subclass is in blue or black.
- Logic that was present without the subclass is in gray.

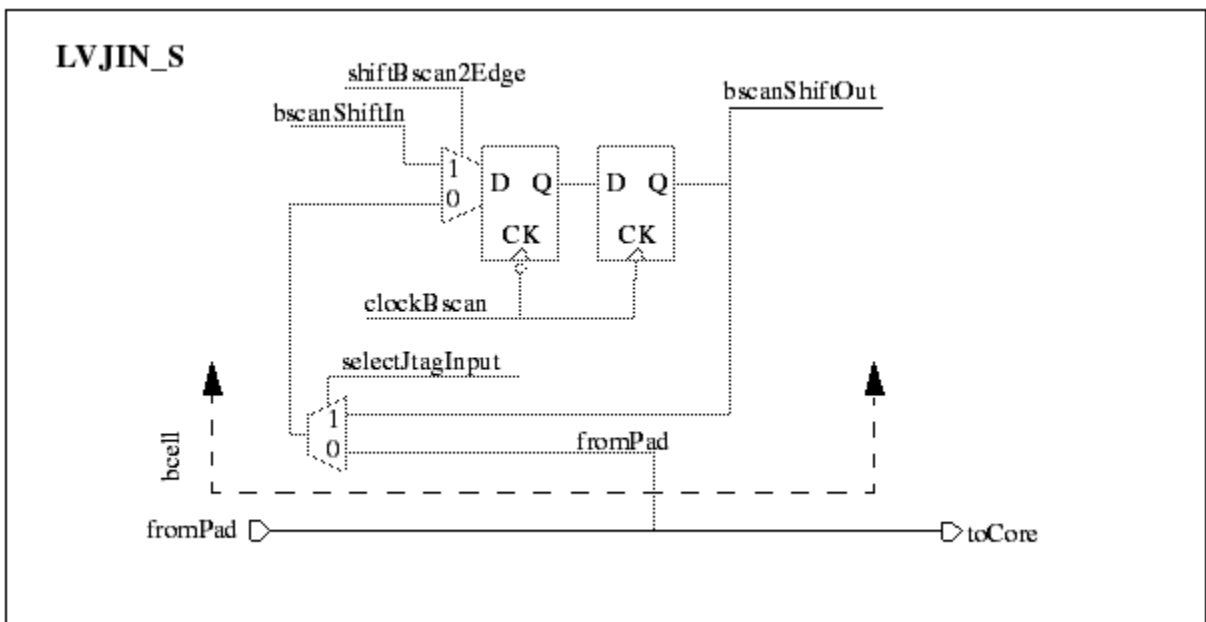
S: Sample-Only

The sample-only boundary-scan cells observe their input pin value without intercepting them with a mux going to the core. You can specify that subclass whenever:

- The boundary-cells are not used during the chip internal logic test mode, hence, they do need to control data going to the core.
- When your data signals are too performance-critical to accept the one multiplexer delay associated with full boundary-scan cells.
- When your logictest mode requires your pin signal to drive the core directly without interference, such like a clock or a reset signal.

For an example of this cell, refer to [Figure 4-8](#).

Figure 4-8. I(S): Sample-Only Input Boundary-Scan Cell



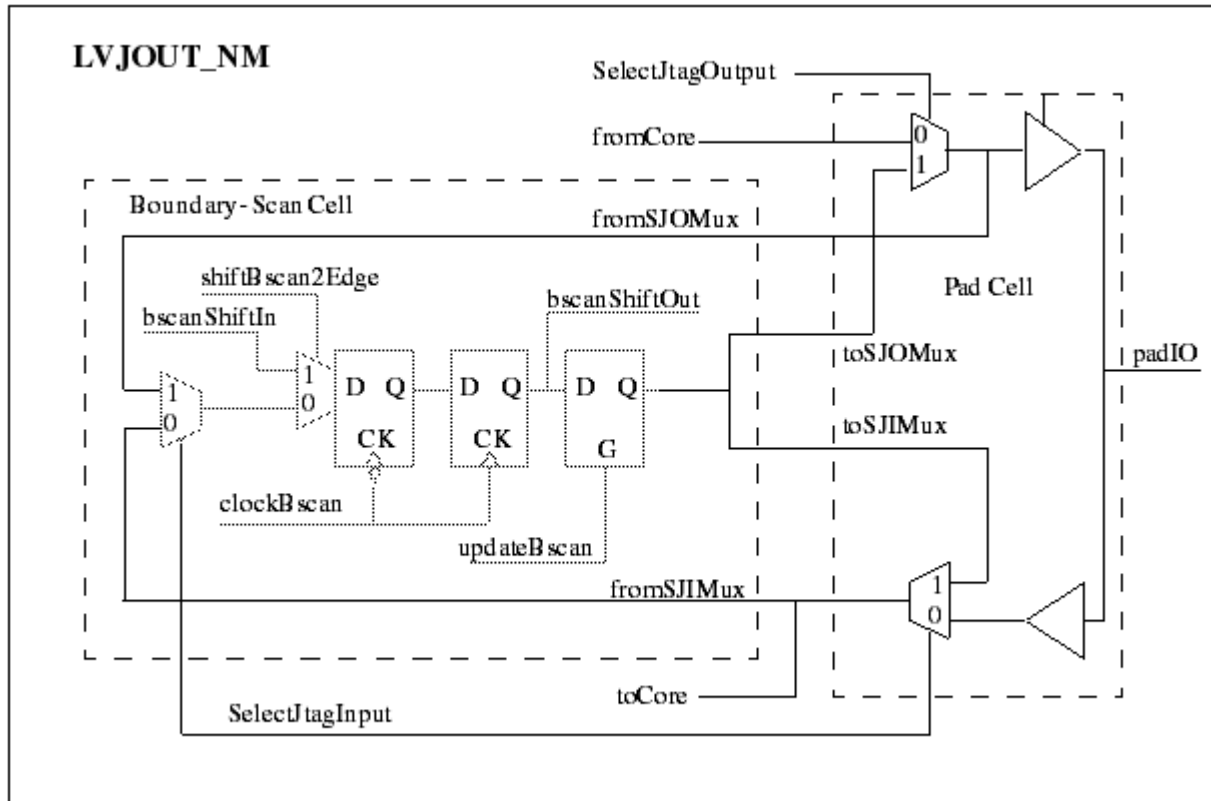
NM: No JTAG Multiplexer

When the 2:1 JTAG muxes controlled by SelectJtagInput and SelectJtagOutput are embedded inside the pad cells, the NM subclass boundary-scan cell takes advantage of them and create connections to their test input and output pins. The toCore and fromCore connections now branch directly to/from the pad. Refer to [Figure 4-9](#). The advantages are as follows:

- The pad JTAG mux can be optimized for timing.

- The boundary-scan cell insertion process introduces no change to the original functional path timing.

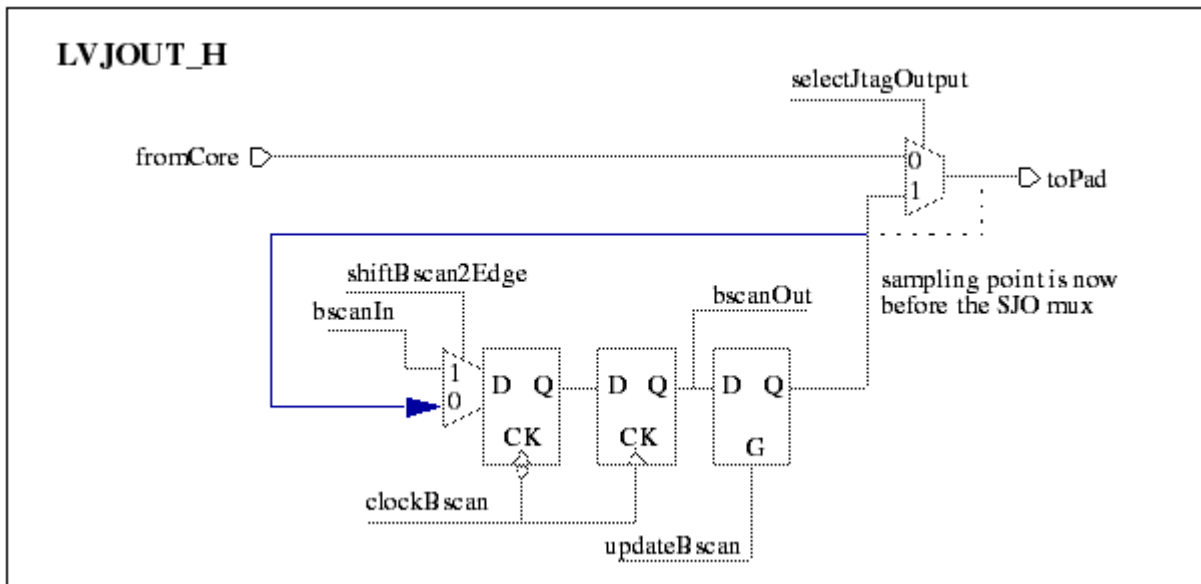
Figure 4-9. IO(NM): IO Bcell with JTAG Multiplexers Inside the Pad



H: Holding

The sample point is moved from the pad to the output from the boundary-scan cell to create a holding cell. Refer to [Figure 4-10](#).

Figure 4-10. O(H): Output BCell with Holding



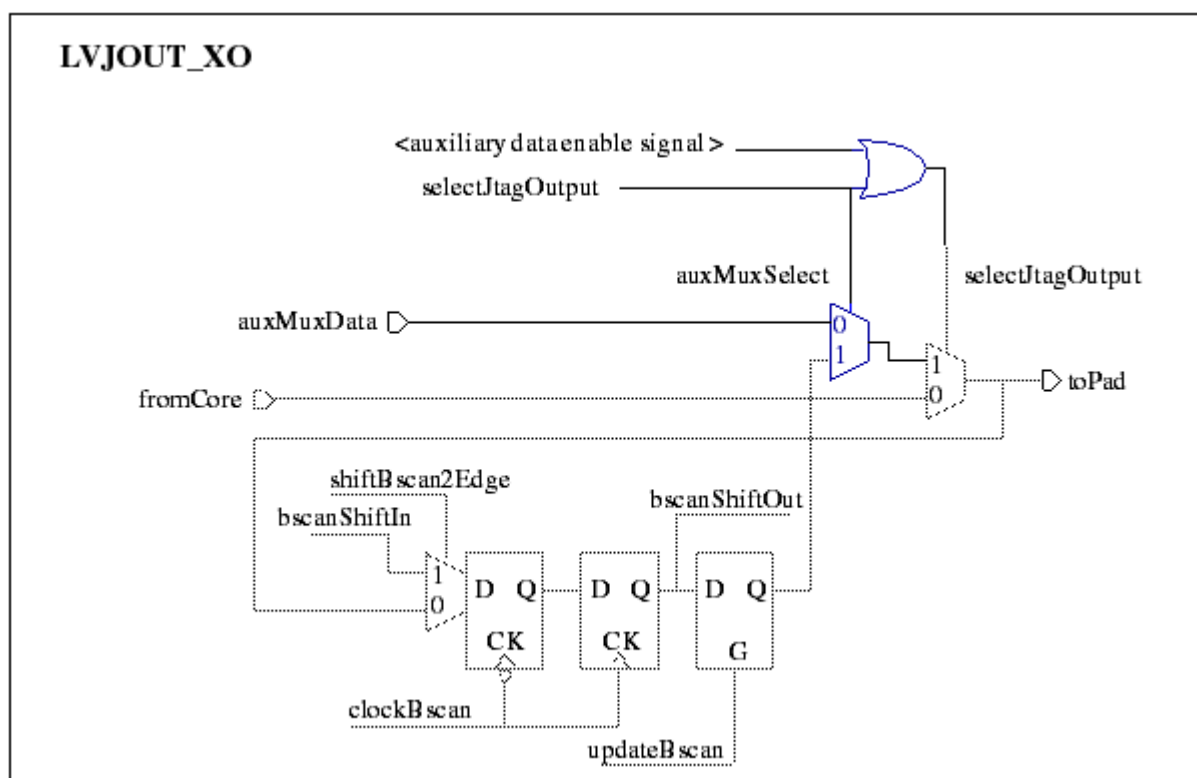
XO: Auxiliary Data Multiplexer

This boundary-scan cell has an extra multiplexer on the test side of the selectJtag multiplexer. Having this multiplexer on the test side, as opposed to placing it before the normal fromCore signal, prevents the Generate tool from adding another multiplexer delay to the normal pin's functional path. Refer to [Figure 4-11](#).

The XO subclass is automatically selected by the Mentor Graphics tools whenever it is needed. It is used for the implementation of the scan output ports (multi-chain scan mode) or for the observation of some internal test or diagnostic signal at the pads. In all cases, the pad is shared with a functional port.

If a port is defined as an auxiliary serial output (AUX_SO) or as an auxiliary output (diagnostic) port, then the Generate tool automatically infers the XO subclass.

Figure 4-11. O(XO): Output Bcell with an Auxiliary Data Multiplexer



Chapter 5

logicTest Controller

The ETAssemble tool generates the logicTest controller for your design. This chapter discusses the logicTest controller architecture, ports, and configuration modes.

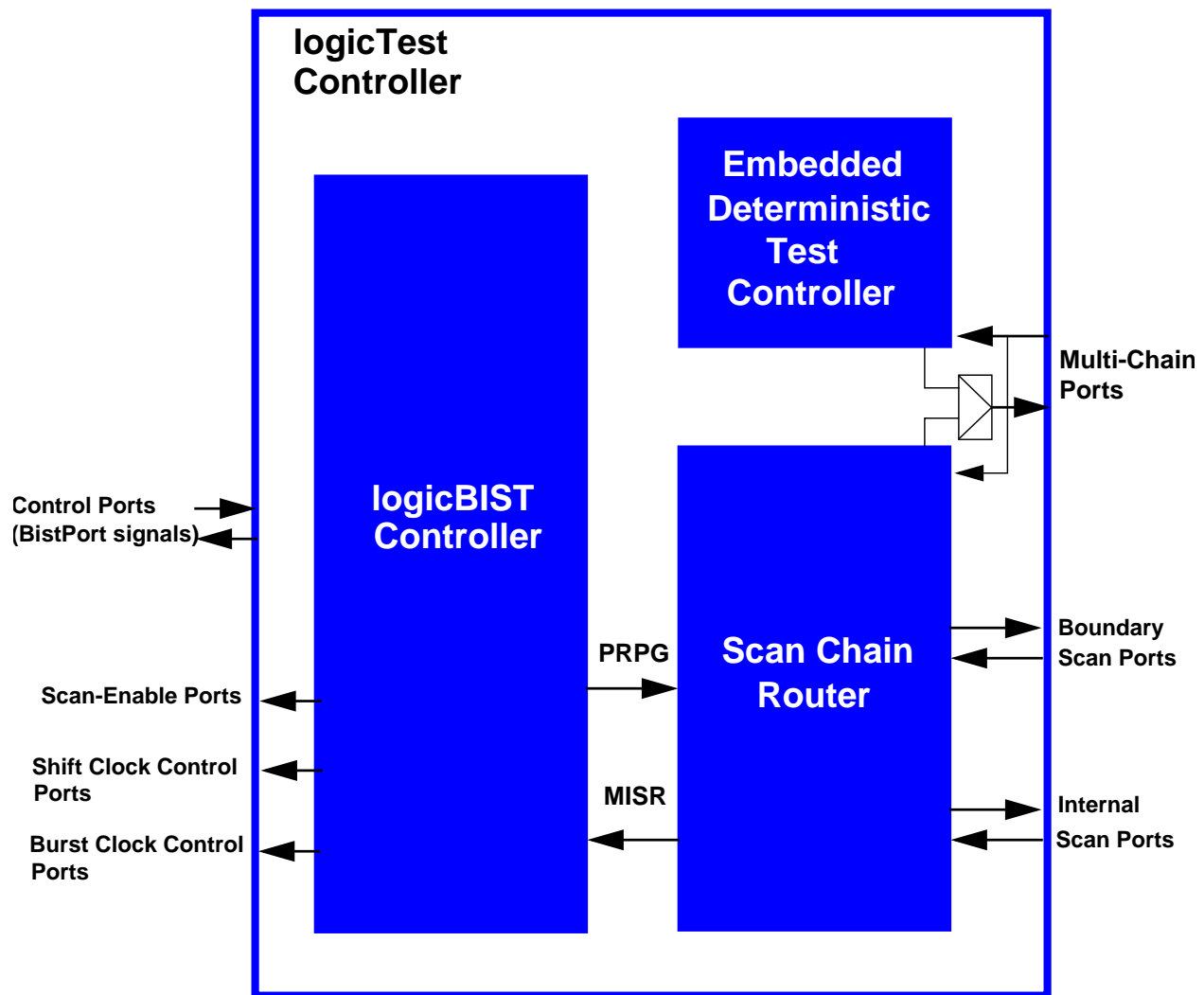
This chapter covers the following topics:

- [logicTest Controller Architecture](#)
- [The Scan-Chain Router](#)
- [Embedded Deterministic Test Controller](#)
- [Shared EDT and LogicBIST Hardware](#)
- [The BurstMode Logic BIST Controller](#)
- [Shift Clock Controller](#)
- [Burst Clock Controller](#)
- [Scan Enable and Clock Enable Controllers](#)
- [Dedicated Isolation Cells](#)
- [Test Point Types](#)

logicTest Controller Architecture

The ETAssemble tool creates a logicTest controller according to your custom needs. [Figure 5-1](#) shows a block diagram of the logicTest controller, which is essentially a simple assembly module composed of a scan-chain router, an optional logic BIST controller, and some decoding logic to control the mode configurations.

Figure 5-1. The Block Diagram of the logicTest Controller



The bistport signals are identical to any other Mentor Graphics' BIST controller, except that the port BIST_SETUP is 3-bits wide instead of 2-bits wide. The extra bit is used to separate those configuration modes that involve the logicBIST controller from those that involve the scan tests. Random logic gates inside the logicTest assembly module perform this decoding.

Table 5-1 lists configuration mode encoding as a function of the input ports BIST_EN and BIST_SETUP[2:0].

Table 5-1. Configuration Mode Encoding

BIST_SETUP[2:0]	BIST_EN	Mode
XXX	0	Idle
00X	1	Long setup

Table 5-1. Configuration Mode Encoding (cont.)

BIST_SETUP[2:0]	BIST_EN	Mode
010	1	Run in default mode
011	1	Run in normal mode
100	1	Scan-Thru-TAP
101	1	Multi-Chain Scan
111	1	Contoller's Chain Mode

The Scan-Chain Router

This section describes the scan-chain router.

Scan-Chain Configurations

The scan-chain router is always present within the logicTest controller but the number of test configurations that this hardware supports depends on user input:

- The single-chain scan configuration is always present, refer to “[Single-Chain Configuration](#).”
- The multi-chain scan configuration is optional and is only present when the number multi-chains is specified in ETPlanner. For details, refer to “[Multi-Chain Configuration](#).”
- The logic BIST configuration. For details, refer to “[Logic BIST Configuration](#).”
- The BurstMode logicBIST controller, Burst Clock controllers, and shift clock controller have their own internal scan chain that are not part of any of the circuit scan chains. For details, refer to “[Logic BIST Controller Chain Mode](#).”

The following sub-sections describe these test configurations in detail.

Single-Chain Configuration

In single-chain test configuration, the scan-chain router concatenates all existing scan segments connected to the scan-chain router into one scan chain (shift register) between the BIST_SI and BIST_SO ports of the logicTest controller.

[Figure 5-2](#) shows the concatenation of the internal scan chains and of the boundary-scan chains. The flip-flops used to generate capture disable signals inside the logicBIST controller and inside the router are also part of the single scan-chain configuration. The order of the scan segment concatenation is described in the file `<designName>_LVISION_LOGICTEST.config`, which is generated along with the RTL modules.

Figure 5-3 shows a section of the file which describes the scan configuration during single-chain mode. It uses the same example circuit that is used in Figure 5-2.

Figure 5-2. Graphical View of Single-Chain Configuration

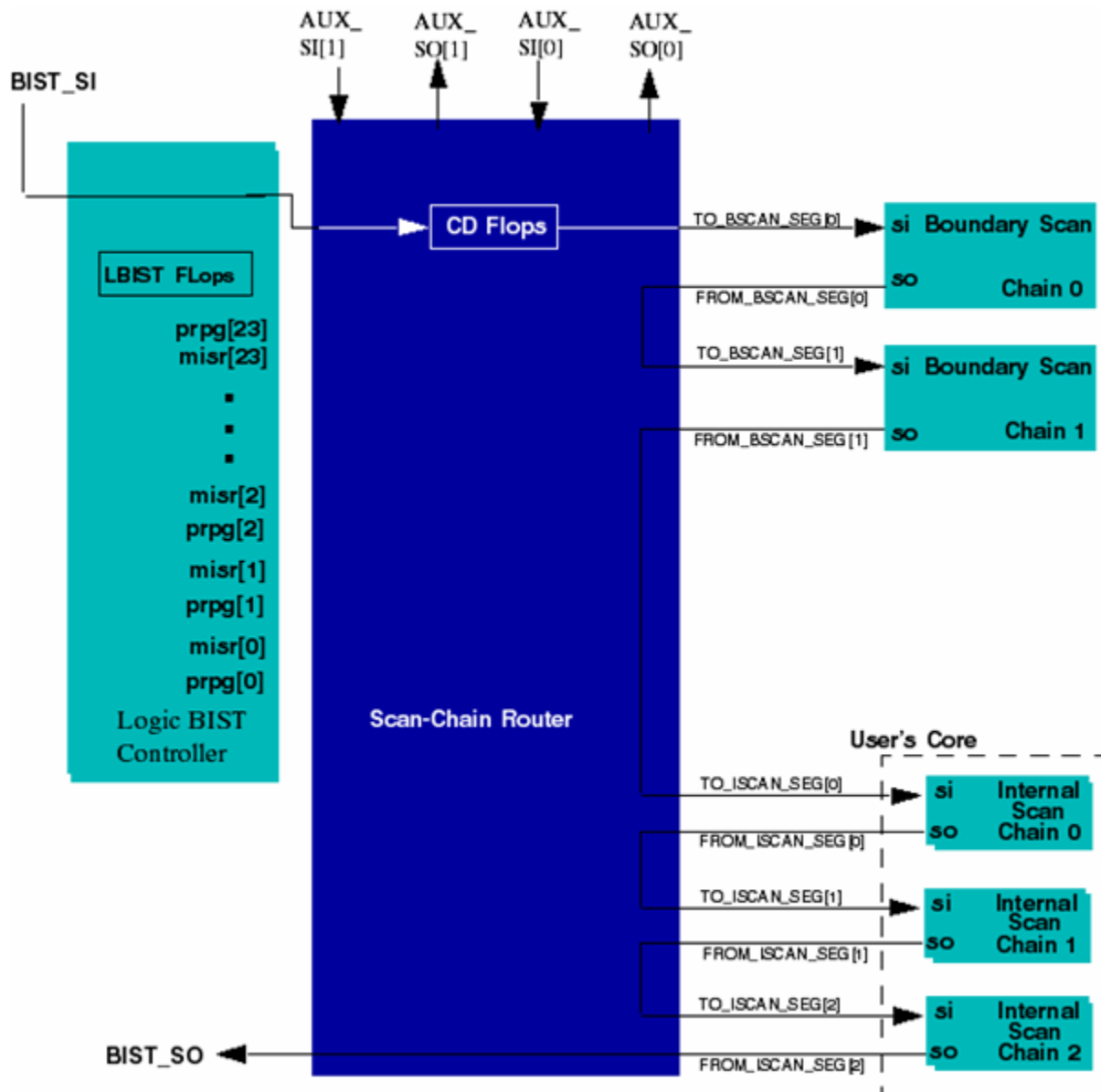


Figure 5-3. Text Presentation of Single-Chain Configuration

Scan Configuration in Single Chain Mode

```

BIST_SI  ->    CD_FLOPS
              BSCAN[0]
              BSCAN[1]
              ISCAN[0]
              ISCAN[1]
              ISCAN[2] -> BIST_SO

```

Multi-Chain Configuration

In the multi-chain configuration, the scan-chain router connects some scan segments to the auxiliary scan-in and scan-out ports. The scan-chain router also concatenates some of the scan segments between the BIST_SI and the BIST_SO ports. This scan chain ends up between TDI and TDO port when a TAP is used. The flip-flops used to generate capture disable signals inside the logicBIST controller and inside the router are also part of the multi chain configuration. The boundary and internal scan segments are automatically distributed between the BIST_SI/BIST_SO chain and the AUX_SI/AUX_SO chains to balance the length of all scan chains.

[Figure 5-4](#) shows a graphical representation of the scan configuration during the multi-scan chain configuration. [Figure 5-5](#) illustrates how the same example looks in a section of the file *<designName>_LVISION_LOGICTEST.config*. This file is generated along with the RTL modules.

Figure 5-4. Graphical View of Multi-Chain Configuration

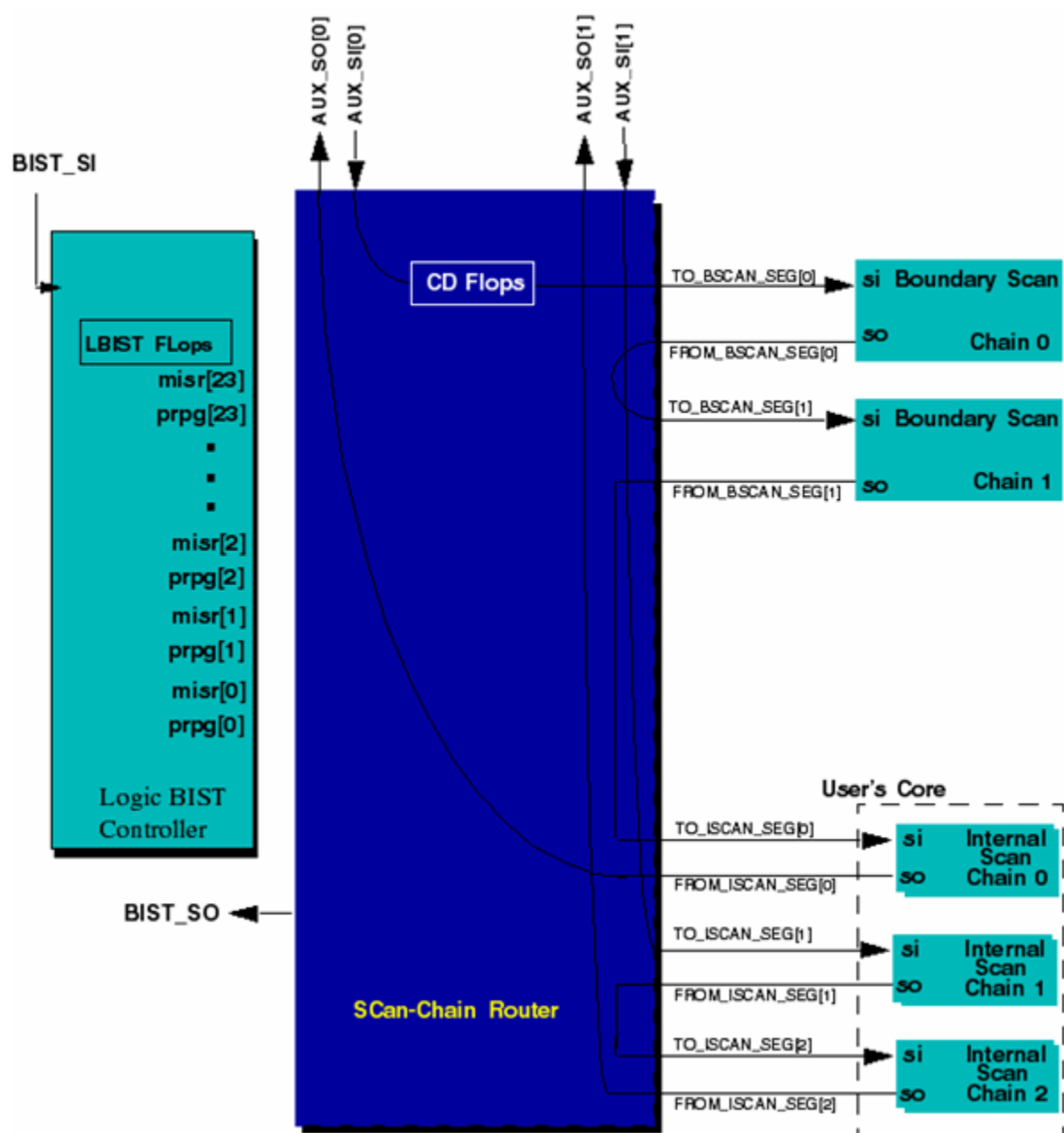


Figure 5-5. Text Presentation of Multi-Chain Configuration

Scan Configuration in Multi-Chain Mode

```
-----
AUX_SI[0] ->  CD_FLOPS
              BSCAN[0]
              BSCAN[1]
              ISCAN[0] -> AUX_SO[0]
AUX_SI[1] ->  ISCAN[1]
              ISCAN[2] -> AUX_SO[1]
```

Logic BIST Configuration

In the logic BIST configuration, the scan-chain router connects all scan segments, including the CD flip-flops, when they exist, to the MISR and PRPG ports of the logic BIST controller.

In addition to configuring the scan-chain connections, the scan-chain router also adds retiming flip-flops between the scanout port of scan segments and the MISR. [Figure 5-6](#) shows a graphical view of the scan-chain configuration during logic BIST mode. [Figure 5-7](#) illustrates how the same example looks in a section of the file

<designName>_LVISION_LOGICTEST.config generated along with the RTL modules.

Figure 5-6. Graphical View of Logic BIST Configuration

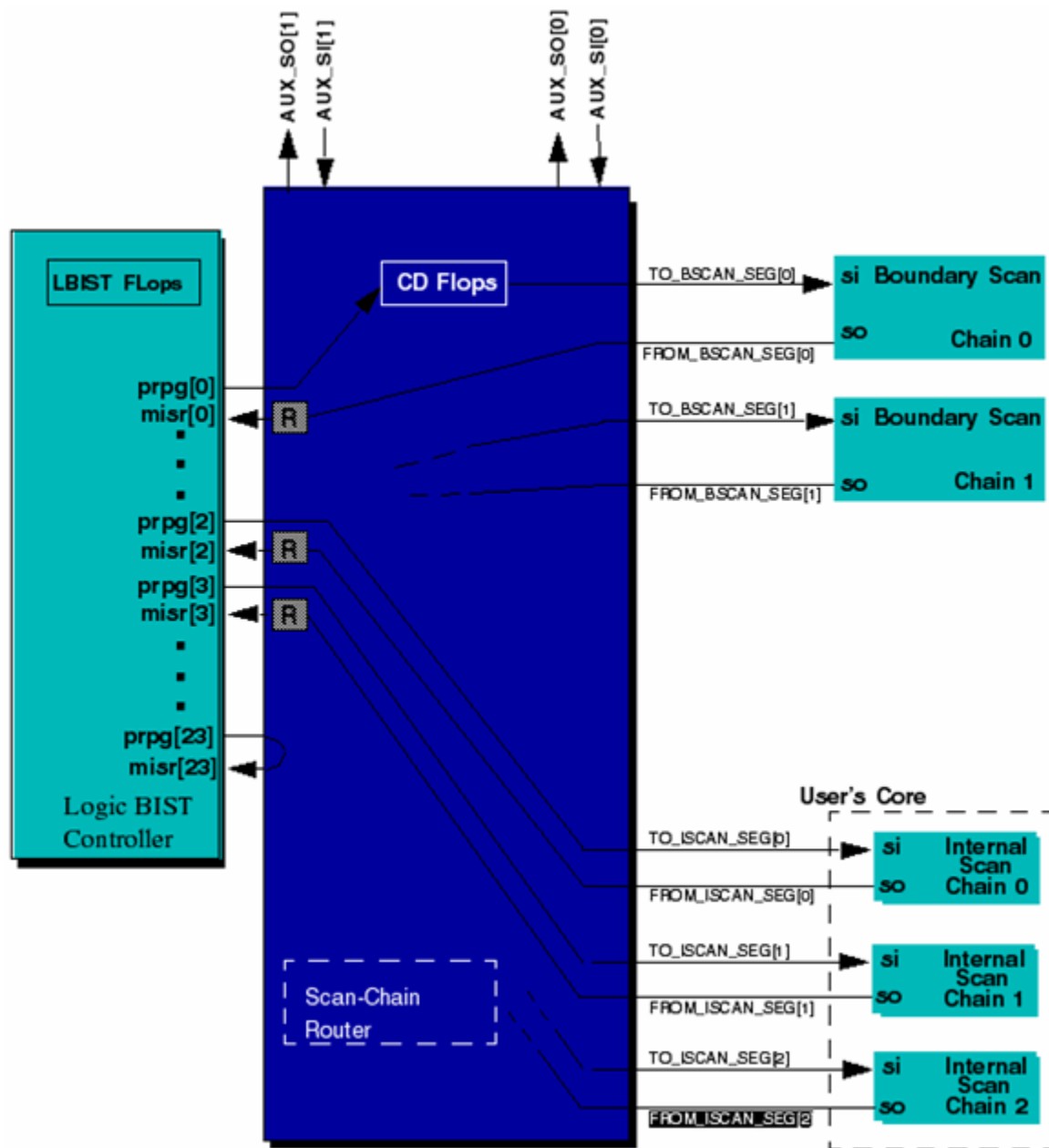


Figure 5-7. Textual View of Logic BIST Configuration

```
Scan Configuration in LogicBist Mode
-----

PRPG[0] -> CD_FLOPS
          BSCAN[0] -> R MISR[0]
PRPG[1] -> BSCAN[1] -> R MISR[1]
PRPG[2] -> ISCAN[0] -> R MISR[2]
PRPG[2] -> ISCAN[1] -> R MISR[2]
PRPG[3] -> ISCAN[2] -> R MISR[3]
```

Logic BIST Controller Chain Mode

The BurstMode logicBIST controller has its own internal scan chain. So does the Burst Clock controllers and shift clock controller. These internal chains are not part of any of the circuit scan chains. In controller chain mode, a special scan chain is formed from all controllers' chains and connected to the TAP/WTAP BIST_SI and BIST_SO ports.

Figure 5-8 shows the textual form of the concatenation of the controllers' scan chains.

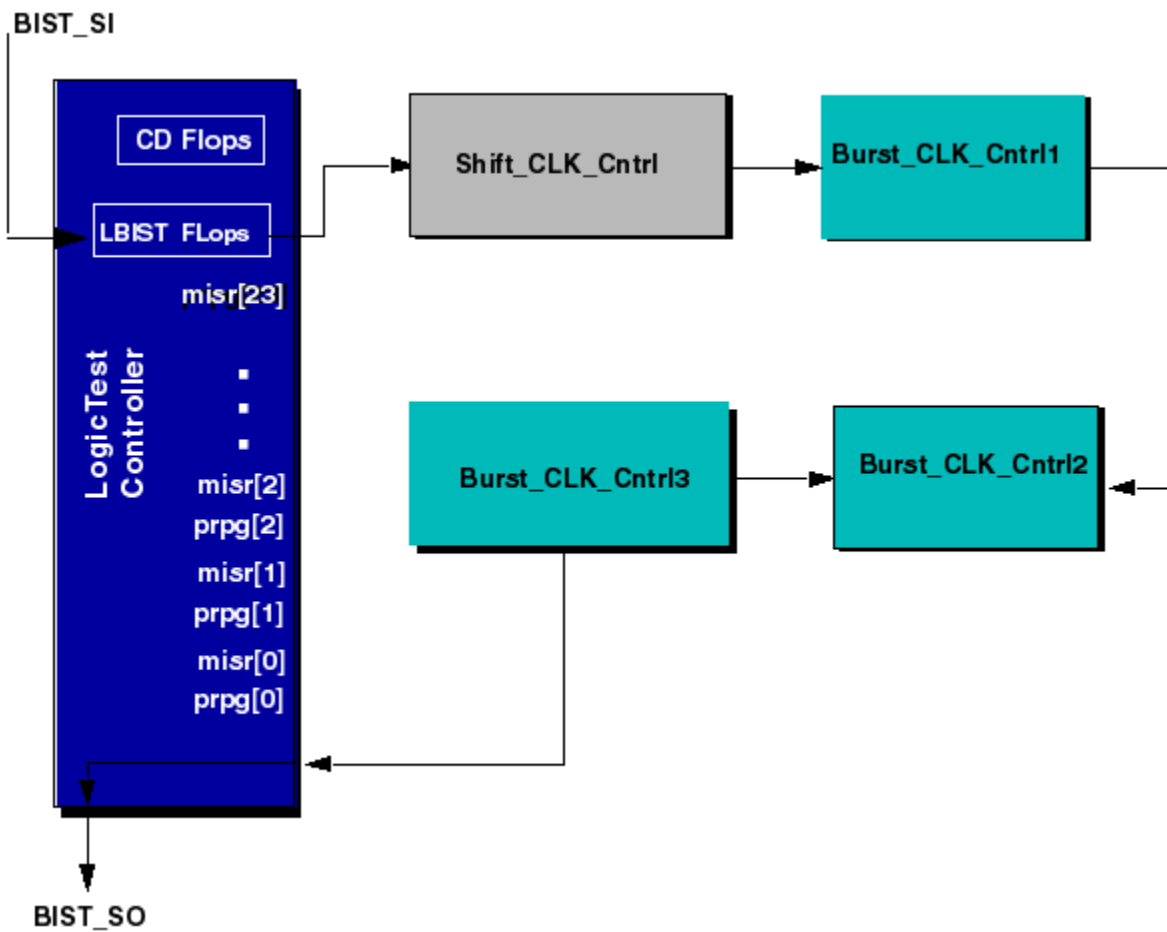
Figure 5-8. Text Presentation of Controller Chain Configuration

```
Scan Configuration in Controller Chain Mode
-----

BIST_SI -> LOGICTEST_CONTROLLER
          SHIFT_CLK_CNTRL
          BURST_CLK_CNTRL1
          BURST_CLK_CNTRL2
          .....
          BURST_CLK_CNTRLn -> BIST_SO
```

Figure 5-9 shows a graphical representation of the scan configuration during the controller chain configuration. The figure shows a design with three Burst Clock controllers.

Figure 5-9. Graphical View of Controller Chain Configuration



Embedded Deterministic Test Controller

The Embedded Deterministic Test controller (EDT), as shown in [Figure 5-1](#), is optional and is only present when the property in ETPlanner input files is used:

```
TestKompRESSPresent: Yes;
```

The input and output channels of the EDT controller are connected to the auxiliary scan-in and scan-out ports. The value specified for the [DedicatedMultiChainNumber](#) property in ETPlanner input files determines the number of input/output channels that will be created.

The EDT decompressor and compactor are connected to the scan-chain router module that allows for the single- and multi-scan chain configurations to be used when EDT bypass is being performed.

For detailed description of the EDT logic, refer to Appendix B [EDT Logic Specifications](#) in the *Tessent TestKompRESS User's Manual*.

Shared EDT and LogicBIST Hardware

The Tessent LV Flow allows you to combine the EDT and LogicBIST controllers to reduce the overhead by sharing duplicated IPs. Using the Hybrid IP also provides access to a new low power LogicBIST architecture that can control the switching activity during “shift” to reduce power consumption.

Hardware Variations

In the Tessent LV Flow, you can currently insert a Tessent EDT controller and a Tessent LogicBIST controller into the same design. Having both controllers in the same design increases the hardware overhead because of duplicate logic.

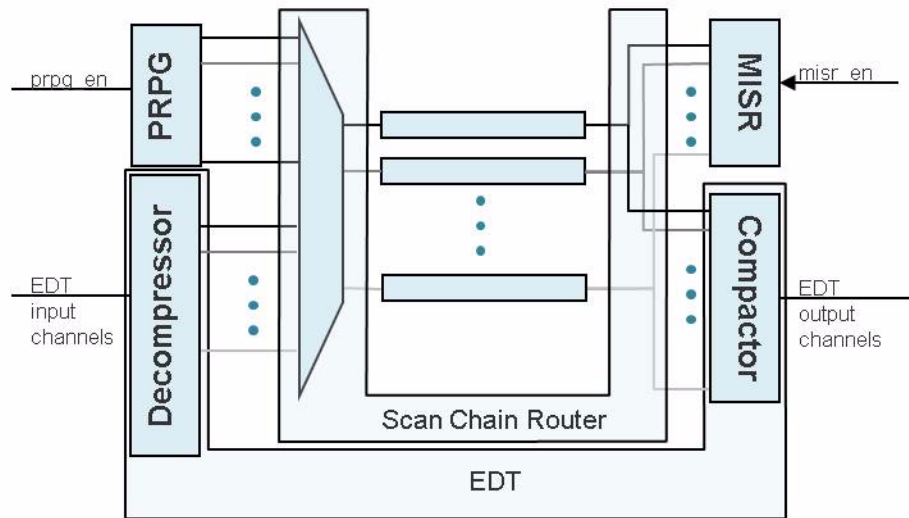
These are possible hardware variations:

- [Classic IP](#)
- [Hybrid/Merged IP](#)
- [Low-Power Shift](#)

Classic IP

The block diagram in [Figure 5-10](#) illustrates how the EDT controller is shared with the LogicBIST; this is referred to as the *Classic IP*.

Figure 5-10. EDT and LogicBIST Hardware Created by Tessent TestKompress and LogicBIST



Hybrid/Merged IP

The block diagram in [Figure 5-11](#) shows the Hybrid IP which can be enabled by setting the ETPlanner *LogicBistIP* property to *Hybrid*. The hardware with the Hybrid IP has the following changes:

- In the Hybrid IP, the LogicBIST MISR is still functionally equivalent to the Classic MISR but it is now instantiated within the EDT controller and has the module name `<designName>_edt_misr`.

In the Classic IP, the MISR operates on *TE* of *BIST_CLK*, but in the Hybrid IP, the MISR operates on *LE* of the *BIST_CLK*, thereby, allowing the removal of the MISR retiming flops in the scan chain router.

- In the Classic IP, the scan chain router contains a single register (*INPUT_MASK*) to enable input chain masking and a multi-bit register (*OUTPUT_MASK*) to perform output chain masking.

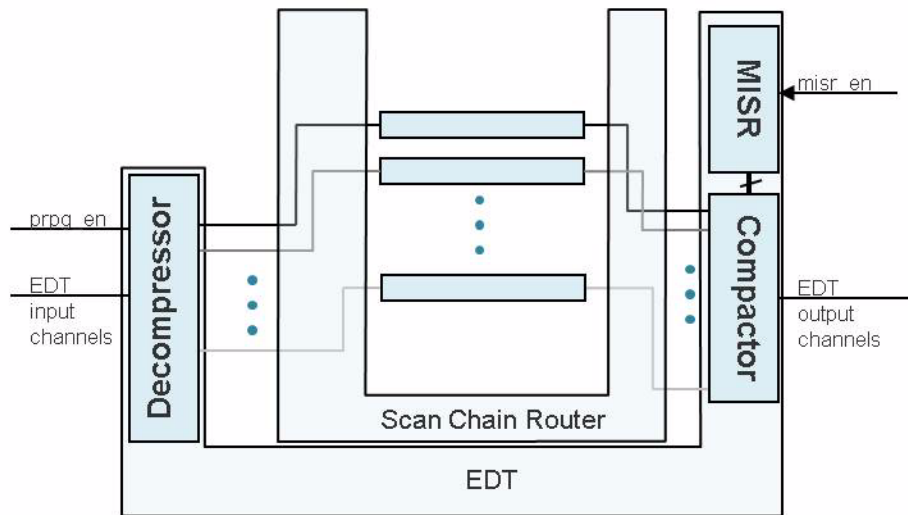
In the Hybrid IP, these registers have been removed, and instead the equivalent registers, *edt_chain_mask_load_en* and *edt_chain_mask*, in the EDT controller are reused.

- The Parallel Load simulations have changed:

In the Classic IP, the Parallel Load simulation would apply the parallel load and perform two-shift cycles before performing the Burst phase.

In the Hybrid IP, the two-shift cycles are no longer performed. Because no cycles of the Shift Phase are simulated during parallel load, it is now important that at least one Serial Load pattern is simulated to ensure that there are no issues during shift.

Figure 5-11. Shared EDT and LogicBIST Hardware



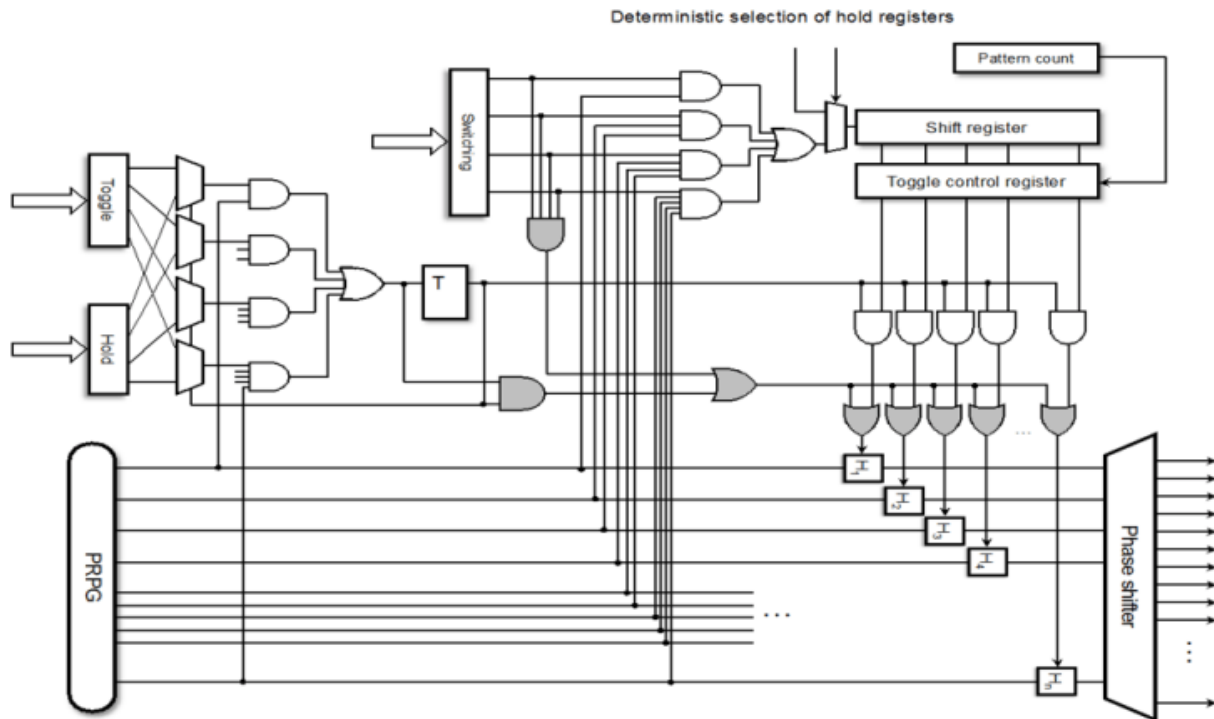
To switch between the Classic IP and the Hybrid IP, use the ETPlanner **LogicBistIP** property. When **LogicBistIP** is set to *Hybrid*, the EDT IP is created by ETAssemble.

Low-Power Shift

With the Hybrid IP, it is possible to have additional hardware added that will control the switching activity during “shift” to reduce power consumption. To enable the low-power LogicBIST, you must set the ETPlanner **LogicBistLowPowerShiftMode** property to *Yes*. The low power controller will be synthesized to support a range of switching percentages and the minimum value, between 1 and 50 with a default of 15, that can be set using the ETPlanner **LogicBistLowPowerShiftMinSwitching** property. This low power scheme is only available when using the Hybrid LogicBIST IP.

Figure 5-12 shows the overall architecture of the low-power BIST controller. The existing LFSM lockup cells are replaced by a hold register which load select signal is controlled by the low power control logic. By controlling when these hold registers update versus when they hold previous cycle values, the overall switching at the scan chain inputs can be controlled.

Figure 5-12. Low-Power Controller



The performance of the low-power BIST controller depends on the following three factors:

- Switching code (SC),
- Hold value (HV),
- Toggle value (TV).

The SC, HV, and TV values are automatically calculated by the tool based on the switching threshold the you set during simulation.

The 4-bit switching code might assume one of 15 different binary values ranging from *0001* up to *1111*. The last value can be alternatively used to disable the control register and enter the poorly pseudo-random test pattern generation (unless you activate the Hold mode). All codes are used to enable certain combinations of AND gates forming biasing logic, and, hence, to produce *1*s with probabilities *0.5* (*0001*), *0.25* (*0010*), *0.125* (*0100*), *0.0625* (*1000*), plus their combinations obtained due to an additional OR gate. The resultant *0*s and *1*s are shifted into the mask shift register, and, subsequently, they are reloaded to the mask hold register at the beginning of a test pattern to enable/disable the hold latches placed between the ring generator and its phase shifter.

The duration of how long the entire generator remains in the Hold mode with all latches temporarily disabled regardless of the hold register content.

In the Toggle mode (its duration is determined by TV), the latches enabled through the control register can pass test data moving from the ring generator to the scan chains. In order to switch between these two modes, a weighted pseudorandom signal is produced by a module Encoder H/T based on the content of different stages of the ring generator.

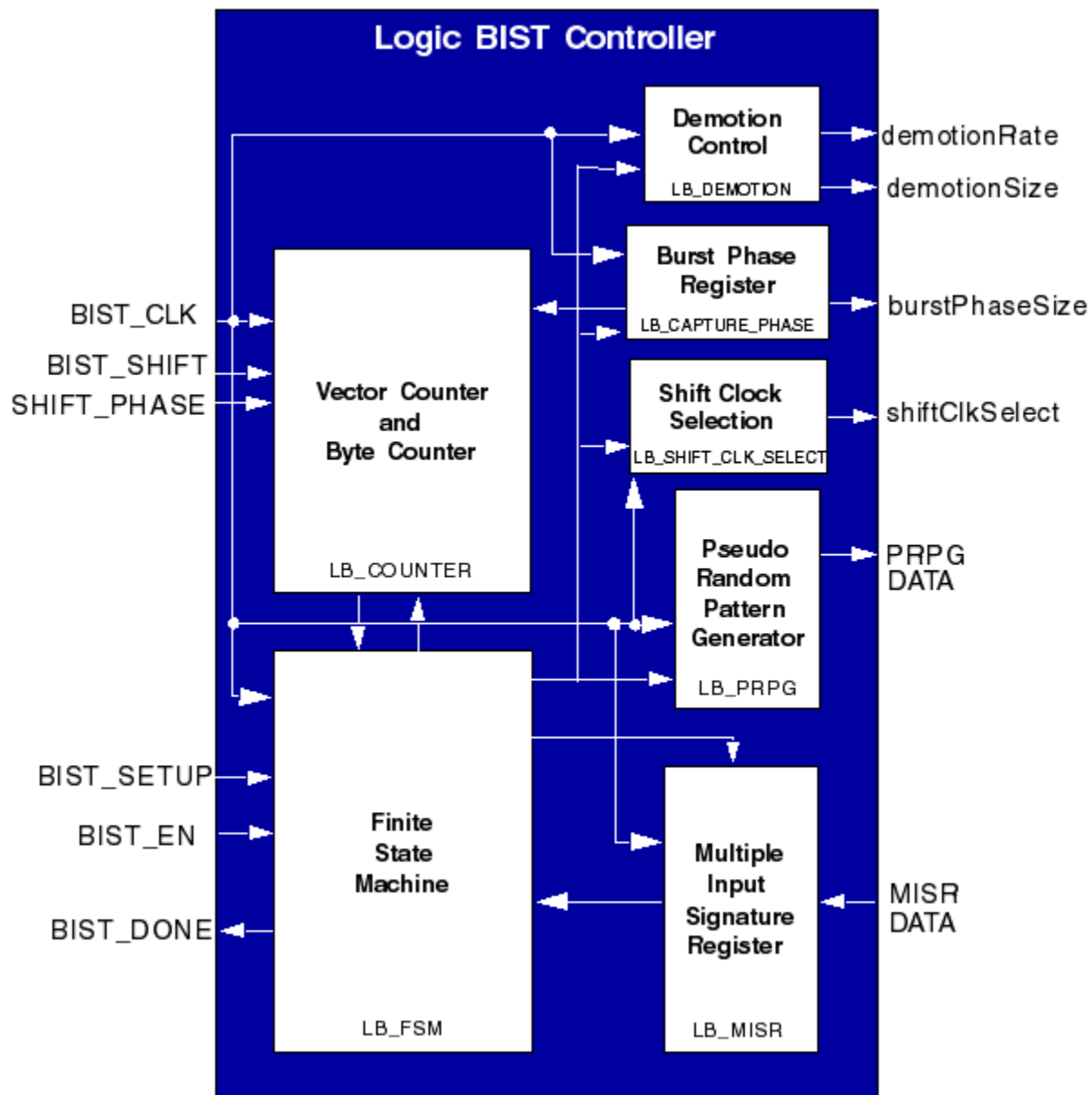
Disabling of the Low-Power shift during pattern generation can be performed by setting the Tcl variable ***DisableLogicBistLowPowerShiftMode*** to 1 in the dofile
<designName>.fastscan_setup_commands or <designName>.fastscan_setup_lbist_commands.

To dynamically change the switching threshold during pattern generation, meaning, while running the ***make fastscan_lbist_full_vectors*** target, you can add the command
set_power_control shift -switching_threshold X, where **X** is the new switching threshold, to the <designName>.fastscan_atpg_commands or
<designName>.fastscan_atpg_commands_lbist file.

The BurstMode Logic BIST Controller

The logic BIST controller is optional and is only present when the ETPlanner option **LogicBistPresent On** is used. [Figure 5-13](#) shows a block diagram of the logic BIST controller and includes all modules that form the controller.

Figure 5-13. BurstMode logicBIST Controller Functional Diagram



Logic BIST Controller Modules

The logicBIST controller contains the following sub-modules:

- A finite state machine module (LB_FSM) to control the flow of operation of the BurstMode logicBIST controller, refer to [Finite State Machine Module](#).
- A counter module (LB_COUNTER) that contains the vector counter and the byte counter. The byte counter in the waveform generator circuit counts the length of the vector while the vector counter counts the number of vectors, refer to [Counter Module](#).
- A pseudo-random pattern generator (LB_PRPG) that is used to load the scan chains with the random patterns, refer to [PRPG Module](#).
- A multiple-input signature register (LB_MISR) that is used to compress the returning captured data from the scan chain into a predictable signature, refer to [MISR Module](#).
- A burst phase register (LB_CAPTURE_PHASE), refer to [Burst Phase Register](#).
- A shift clock control selection circuit (LB_SHIFT_CLK_SELECT), refer to [Shift Clock Selection](#).
- A demotion control circuit (LB_DEMOTION), refer to [Demotion Control](#).

The following sub-sections describe these sub-modules in greater detail.

Finite State Machine Module

The LB_FSM module manages the flow of execution for the logicBIST controller. It controls when to start generating random patterns, when to start accumulating capture results into the MISR, and when to compare the final signature when running in default mode.

Counter Module

The counter module LB_COUNTER of the logicBIST controller contains a vector counter and a byte counter.

Vector_Count

The Vector_Count is part of the counter sub-module and is used to count the number of vectors to apply when the logic BIST controller is running. The size of the vector counter is 500K and cannot be changed.

Byte Counter

The byte counter is used to count the length of each vector. The length of the vector corresponds to the length of the longest chain rounded up to the nearest 8. The byte counter counts a multiple of eight. The size of the byte counter that ETAssemble includes in the logic BIST controller is set to minimum of 2K. This value is automatically increased by ETAssemble if necessary.

Burst Phase Register

The burst phase register is used to control the length of the burst phase of the shift clock controller. The burst phase length is run time programmable and can be up to 64 shift clock cycles in length. This is the default.

Shift Clock Selection

The shift clock selection module is used to select the clock to be used during the shift phase. The user can program the controller to use TCK or a divided version of up to two clock sources as shift clocks. Details on the shift clock selection is given in “[Shift Clock Controller](#).”

Demotion Control

This module controls the burst clock speed reduction and the number of clock cycles for which that applies. It does not affect the number of cycles in the burst phase. The number of demoted cycles can be from 0 to 3, and the demotion rate can be 1/2, 1/3, 1/4, or can use the shift clock as a burst clock.

The following properties in the ETVerify configuration file allow demotion control:

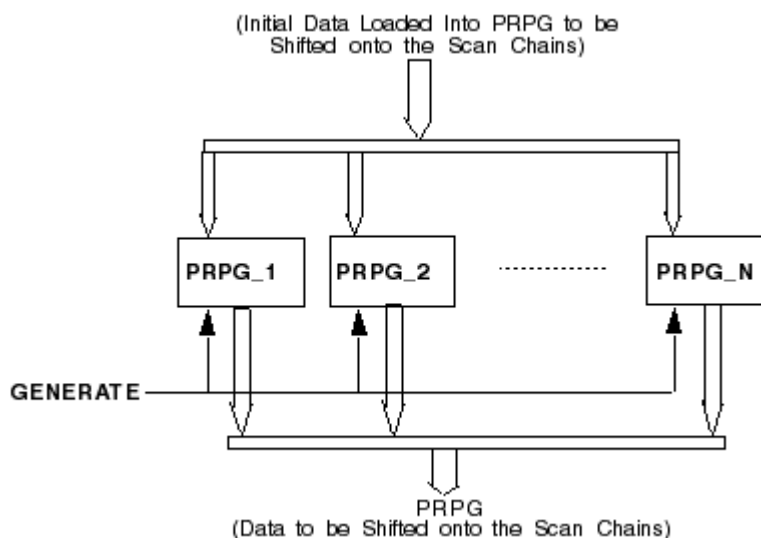
- [SlowedDownBurstCycles](#)
- [EffectiveSlowedDownFrequency](#)

PRPG Module

The PRPG (pseudo-random pattern generator) is a multiple-output device that generates pseudo-random output patterns with which to load the scan chains.

The PRPG in the Mentor Graphics logicBIST controller is scalable. [Figure 5-14](#) shows the PRPG architecture. The ETAssemble tool combines 24- and 32-bit PRPGs. The PRPG size depends upon the total number of scan chains.

Figure 5-14. Scalable PRPG Architecture



In addition to scaling the PRPG, ETAssemble selects a cellular automaton (CA) as the PRPG in your logicBIST controller.

The CA provides patterns that exhibit more random behavior than those generated by the LFSR.

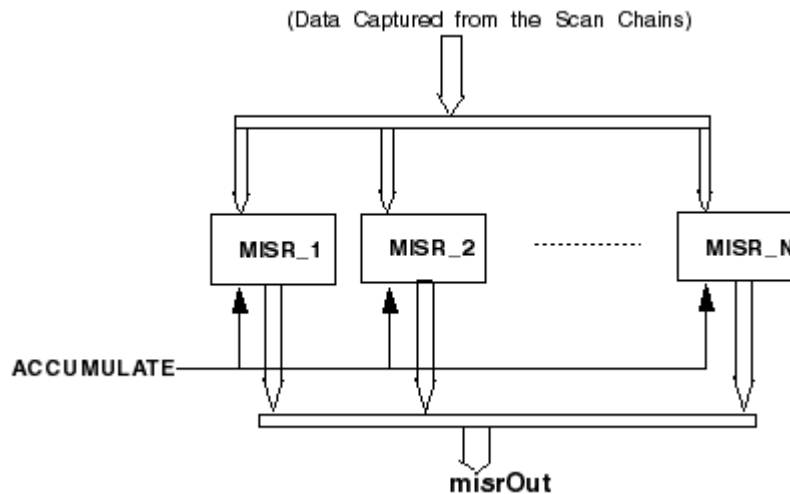
By default, ETAssemble seeds the CA with all 1s for the first copy of a 24 or a 32-bit PRPG. The nth copy of a 24- or a 32- bit PRPG has a seed with all 1s with bit n-1 inverted to 0.

MISR Module

The MISR (multiple-input signature register) module LB_MISR of the logicBIST controller is a multiple-input device that compresses a series of input patterns into a unique signature.

[Figure 5-15](#) shows the MISR architecture.

Figure 5-15. Scalable MISR Architecture



The MISR in the Mentor Graphics logicBIST controller is scalable. The ETAssemble tool creates a MISR, matching the PRPG size, by combining 24- and 32-bit MISRs.

Clock Connections to the logicTest Controller

With the BurstMode logic BIST architecture, the test clock muxing requirements are more complex, and test clock muxing is automated. Changes to the flow are made to extract internal clock sources and insert the functional clock mux multiplexing. The extraction is done by ETChecker, and the clock insertion is done by ETAssemble.

logicTest Controller Ports

The Mentor Graphics ETAssemble tool creates a logicTest controller based on user-defined property values and the information generated by ETChecker which describes the scan-chain segments to be tested. Following is a description of all input and output ports that can exist on the logicTest controller. The conditions in which these ports are present are also listed.

TP_ENABLE_FF[#]

The TP_ENABLE_FF[#] port is a multi-bit output port that is used by scanGenerate to create an enable signal for testpoints that are shared with functional flip-flops. This port is only created when the ETPlanner [ControlPointImplementation](#) property is set to *FunctionalFFs*.

Scan-Chain Ports

The following list details ports on the scan-chain

TO_BSCAN_SEG[#]

The TO_BSCAN_SEG[#] output ports are used to connect to the scanin ports of the boundary-scan segments.

FROM_BSCAN_SEG [#]

The FROM_BSCAN_SEG[#] input ports are used to connect to the scanout ports of the boundary-scan segments.

TO_ISCAN_SEG[#]

The TO_ISCAN_SEG[#] output ports are used to drive the scanin ports of the internal scan-chain segments.

FROM_ISCAN_SEG [#]

The FROM_ISCAN_SEG[#] input ports are used to connect to the scanout ports of the internal scan-chain segments.

TO_CTRL_CHAIN

The TO_CTRL_CHAIN output port is used to connect the scanin port of the ShiftClkCtrl for the controller chain test.

FROM_CTRL_CHAIN

The FROM_CTRL_CHAIN input port is used to connect the scanout port of the last Burst Clock controller for the controller chain test.

EXT_SI[#]

The EXT_SI[#] input ports are only present when the logicTest controller is in an ELTCore. They are used to drive the scanin signals of the periphery scan chains during the external test mode. They connect directly to primary inputs of the ELTCore module.

EXT_SO[#]

The EXT_SO[#] output ports are only present when the logicTest controller is in an ELTCore. They are used to carry the scanout signals of the periphery scan chains during the external test mode. They connect directly to primary outputs of the ELTCore module.

CDSE

When high the CDSE output port makes the scanEnable controllers force the scanEnables high during the capture cycle. When the logicTest controller is not enabled this output port is low.

AUX_SI[#]

The AUX_SI[#] input ports are only present when the multi-chain scan mode is implemented. They are used to drive the *scanin* signals of the multi-scan segments. They connect directly to primary inputs of the chip.

AUX_SO[#]

The AUX_SO[#] output ports are only present when the multi-chain scan mode is implemented. They are used to carry the *scanout* signals of the multi-scan segments. They connect to primary outputs of the chip through an auxiliary output connection in the boundary-scan cells.

CDFF[#]

The CDFF[#] output ports drive the *Capture Disable* signals. They are decoded by *scang*, in the **ETScan** flow, in such a way that only *Capture Disable* signal is low with all others high when the logicTest controller is enabled. When the logicTest controller is not enabled then all the decoded *CD* signals are low.

TO_BSCAN_TAP

The TO_BSCAN_TAP input port is only present on the top-level logicTest controller. It connects directly to the TAP controller port toBscan and is used to return control of the boundary-scan chain to the TAP controller when the logicTest controller is not enabled.

FROM_BSCAN_TAP

The FROM_BSCAN_TAP output port is only present on the top-level logicTest controller. It connects directly to the TAP controller port fromBscan and is used to return control of the boundary-scan chain to the TAP controller when the logicTest controller is not enabled.

BIST Ports

BIST ports are signals that are used to control the logicTest controller. These signals either connect to the TAP/WTAP controller or to primary input or output pins.

BIST_EN

The BIST_EN port is an active-high input port that enables the logicTest controller. It is controlled by the BIST_EN signal from the TAP/WTAP. Refer to [Table 5-1](#) for a description of how BIST_EN is used to control the modes of the logicTest controller.

BIST_SETUP[2:0]

The BIST_SETUP port along with the BIST_EN port control the modes of the logicTest controller as shown in [Table 5-1](#).

The BIST_SETUP[2:0] port will be directly connected to the setupMode[2:0] port of the TAP/WTAP.

BIST_SHIFT

The BIST_SHIFT port is an input port to the logicTest controller that places the controller in the shifting mode. It is connected to the TAP/WTAP SHIFT_BIST output port.

BIST_SI

The BIST_SI port connects to the toIsCan port on the TAP controller or directly to a primary input port. It is used to carry the scanin data during single chain scan test, the scanin data to the logicBIST controller during the logic BIST setup mode the scanin data of the controller chain scan test, and the scanin data of the first multi-chain segment during multi-chain scan test.

BIST_CLK

The BIST_CLK port is an input port that connects directly to the ltestClk port of the Shift Clock controller output. It is used as the only clock for the logic BIST controller.

BIST_SO

The BIST_SO port connects to the fromBist# signal of a BIST port on the TAP controller or directly to a primary output port. It is used to carry the scanout data during single chain scan test, the scanout data from the logicBIST controller during the logic BIST setup mode the scanout data during the controller chain scan test, and the scanout data of the first multi-chain segment during multi-chain scan test.

LBIST_DONE

The LBIST_DONE port is only present when [LogicBistPresent](#) is Yes. The LBIST_DONE port is an output port that indicates the status of the logicBIST controller.

- When the LBIST_DONE signal is low, the logicBIST controller is still running.
- When the LBIST_DONE signal is high, the logicBIST controller is done.

MISR_OUT[#]

The MISR_OUT[#] port is a multi-bit output port that can be used to observe the generated signature from the MISR module to generate a GO/NO-GO test. This port is only generated if you set the [LogicBistPresent](#) property to *Yes* in the .etplan file.

MISR_ON

The MISR_ON port is an output port that can be used to monitor when the MISR has begun accumulating data. This port is only generated if you set the [LogicBistPresent](#) property to *Yes* in the .etplan file.

RESET_TEST_TAP

The RESET_TEST_TAP is an input port used to control the asynchronous resets of functional flops during an Asynchronous Set/Reset scan test.

RESET_TEST

This is an output port that can be used to control the asynchronous reset in the functional logic.

SET_TEST_TAP

This is an input port used to control the asynchronous sets of functional flops during an Asynchronous Set/Reset scan test.

SET_TEST

This is an output port that is used to control the asynchronous set in the functional logic.

TP_RESET0

This is an output port this is used to control the asynchronous resets of any inserted observation cells or testpoints.

SELECT_JTAG_OUTPUT_TAP

An input port is to be used to control boundary scan cells to control the selection of the output enable of IO pads. This input is connected to the TAP selectJtagOutput output port.

SELECT_JTAG_OUPUT

This is an output port is decoded version of the SELECT_JTAG_OUTPUT_TAP port. It is used to control boundary scan cells to control the selection of the output of a boundary scan cell. This output is connected to the boundary scan enable cells in the design.

Mode Ports

These output ports are used to show when a given mode of operation of the logicTest controller is enabled.

INT_TM

The INT_TM active high output port is high when the single- or multi-chain scan mode or the logic BIST mode is enabled. It is used to control the dedicated isolation cells and a TestMode signal for scan.

EXT_TM

The EXT_TM port is an active high input port. It is only available for Embedded Logic Test (ELT) controller. It is used to configure the core into external test mode.

Note



The EXT_TM port is used as internal testmode at the next level up in hierarchical design.

GLOBAL_TM

The GLOBAL_TM port is an active high output port. It is high when the logicTest controller is placed in either external or internal test mode, i.e. it is just the OR of INT_TM and EXT_TM. This port is only created on a logicTest controller that is inserted inside an ELTCore module.

SINGLE_EN

The SINGLE_EN active high output port is high when the single scan-chain mode is enabled.

MULTI_EN

The MULTI_EN active high output port is high when the multi-chain scan mode is enabled.

LBIST_EN

The LBIST_EN active high output port is high when the logic BIST mode is enabled.

Clock Control Ports

There are several ports in the logicTest controller that are used to control clocking of the circuit in shift and capture modes. Note that with the BurstMode logic BIT architecture, the logic BIST controller does not generate the BIST clocks anymore. The clocks are generated by shift and Burst Clock controllers.

SHIFT_PHASE

This input port is used to control the scan chains in the design. When active it forces the output of the CD flops in the router high. It is also used to disable the flops from capturing during the shift phase.

CD_captureAligned

This output port is used to disable the fast to slow path of the synchronous clock domains when launch aligned waveforms are used.

CD_launchAligned

This output port is used to control the Burst Clock controller to align the launch of clock edge.

shiftClkSelect[3:0]

This multi-bit output port is used to selected the shift clock to be used during the shift phase. This connection is done by ETAssemble.

burstPhaseSize[5:0]

This six-bit output port is used to control the size (number of clocks) of the burst phase. The maximum duration of the burst phase is 64 shift clock cycles. This port is connected to the burstPhaseSize input port of the shift clock controller.

burstClkCtrl_#_demotionRate[1:0]

A two-bit output port per clock domain is used to control the demoted clock speed to be used during the burst phase of logic BIST. It is connected to the respective Burst Clock controller.

burstClkCtrl_#_demotionSize[1:0]

A two-bit output port per clock domain is used to control the number of burst phase clocks to use the demoted speed. It is connected to the respective Burst Clock controller.

END_OF_VECTOR

This output port is used to indicate that the full vector (test pattern) has been shifted into the scan chains. It is connected to the Shift Clock controller.

LAST_VECTOR

LAST_VECTOR is an output port that is used to indicate that all targeted test patterns have been applied. It is connected to the Shift Clock controller.

Shift Clock Controller

The main purpose of the shift clock controller is to control and select the clock used in the shift phase of logic BIST operation. It is also used to provide test clocks to run memory BIST tests when functional clocks are not available. [Figure 5-16](#) illustrates the conceptual schematic diagram of the shift clock controller circuit. The shift clock controller takes three clocks as inputs to generate the shift clock. The TAP clock TCK is the first source in addition to two other shift clock sources (shiftClkSrc1 and shiftClkSrc2). For logic BIST (bistEN=1), one of the three clock sources, or a divided version of this clock source, is selected using the shiftClkSelect signal. For memory BIST (bistEN=0), the selection is performed using the clkRatio input. [Table 5-2](#) shows the combinations of the inputs and the shift clock output for both the logic BIST and memory BIST cases.

This provides the user with capability to select a shift clock for the circuit based on required speed and power.

Figure 5-16. Shift Clock Controller

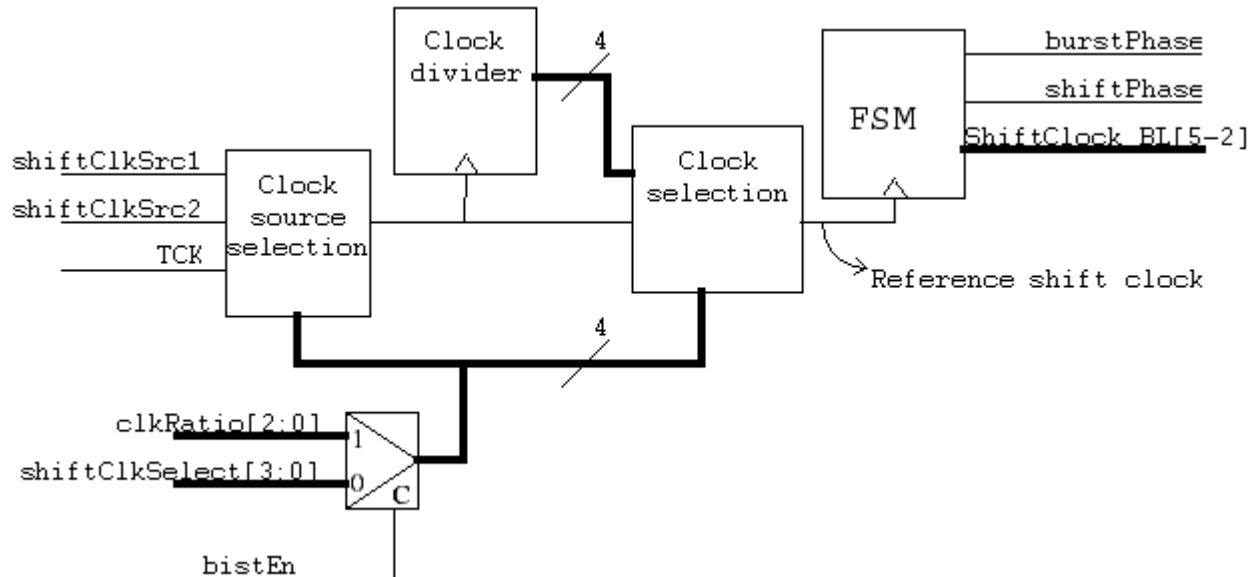


Table 5-2. Shift Clock Selection Control

Shift Clock (shiftClock_BLn)	shiftClkSelect[3:0]	clkRatio[2:0]
shiftClkSrc1	0001	001
shiftClkSrc1/2	0100	010
shiftClkSrc1/4	0101	011
shiftClkSrc1/8	0110	
shiftClkSrc1/16	0111	
TCK	1000	100
shiftClkSrc2	1001	101
shiftClkSrc2/2	1100	110
shiftClkSrc2/4	1101	111
shiftClkSrc2/8	1110	
shiftClkSrc2/16	1111	

Shift Clock Controller Ports

The following are the input and output ports of the shift clock controller:

TCK

The TCK input port is the TCK clock from the TAP/WTAP. This port is automatically connected to the TAP clockscan output port of the WTAP clkBistTck output port.

shiftClkSrc1

This input port is used as one of the shift clock sources that can be used (or a divided version of it) as the main shift clock in the design.

shiftClkSrc2

This input port is used as the second shift clock sources that can be used (or a divided version of it) as the main shift clock in the design.

shiftClkSelect[3:0]

The four-bit port is used to select the source of the shift clock as described in [Table 5-2](#). This port is connected to the shiftClkSelect output port of the logicTest controller.

clkRatio[2:0]

This three-bit input port is used to generate a divided version of the input shiftClkSelect1,2 as described in [Table 5-2](#). They are connected to the clkRatio0,1,2 ports of the TAP/WTAP.

killClock

This input port is used to force the shift clock to logic HIGH when asserted. It is connected to the freezeMode port of the TAP/WTAP.

funcMode

The funcMode input port is asserted to indicate that the controller forces the shift clock to logic HIGH. It is connected to the funcMode port of the TAP/WTAP.

bistEn


This active high input port indicates that the logic BIST controller is active. It is connected to bistEn# port of the TAP/WTAP.

shiftBist

This active high input port places the shift clock controller in shifting mode for the controller chain test. It is connected to the TAP/WTAP shiftBist output port.

extTM

This active high input port is only available for Embedded Logic Test (ELT) controller. It is used to configure the core into external test mode.

Note  The extTM port is used as internal testmode at the next level up in hierarchical design.

LV_SI

This input port is used to connect the scanout port of the logicTest controller for the controller chain test.

LV_SO

This output port is used to connect the scanin port of the first Burst Clock controller for the controller chain test.

LV_SE

This output port is connected to all Burst Clock controllers and acts as the scan enable for the controller chain test.

shiftDR2Edge

This active high input port is used to indicate that the shift of the data register is operating on the opposite edge. This port is automatically connected to the shiftDR2Edge output port of the TAP/WTAP.

setupMode[2:0]

The BIST_SETUP port along with the BIST_EN port controls the modes of the Shift Clock controller (setup shift, setup run, etc.) It is connected to the TAP/WTAP setupMode[2:0] output ports.

endOfVector

This active high input port indicates that the full vector (test pattern) has been shifted into the scan chains. It is connected to the END_OF_VECTOR output port of the logicTest controller.

lastVector

Active high input port is used to indicate that all targeted test patterns have been applied. It is connected to the LAST_VECTOR output port of the logicTest controller.

burstPhaseSize[5:0]

This six-bit input port is used to control the size (number of clocks) of the burst phase. The maximum duration of the burst phase is 64 shift clock cycles. This port is connected to the burstPhaseSize output port of the logicTest controller.

clockBscanTap

This is an input that is used to generated the clock for the boundary scan logic. It is connected to the clockBscan output port of the TAP.

updateBscanTAP

This is an input that is used to generated the update signal for the boundary scan logic. It is connected to the updateBscan output port of the TAP.

shiftBscanTap

This is an input that is used to generated the shift control signal for the boundary scan logic. It is connected to the shiftBscan output port of the TAP.

shiftBscan2EdgeTap

This is an input that is used to generated the shift control signal for the boundary scan logic. It is connected to the shift2EdgeBscan output port of the TAP.

BurstEnable

This active high input port indicates that the logic BIST controller is to operate in BurstMode. It is connected to a TAP/WTAP userIR bit.

clockBscan

This output port carries the clock the boundary scan register. This is connected to the bscanClock of the boundary scan register in the design.

updateBscan

This output port is the boundary scan register update signal. This is connected to the updateBscan of the boundary scan register in the design

shiftBscan

This is the output port that is used to control the shift operation of the boundary scan register in logic BIST mode. It is automatically connected to the shiftBscan port of the register.

shiftBscan2Edge

This is the output port that is used to control the shift operation of the boundary scan register in logic BIST mode. It is automatically connected to the shiftBscan2Edge port of the register.

AuxSiSoEn

This output port controls the auxiliary inputs and output that are used for multi-scan or memory BIST cmp_stat signals. It is automatically connected to the AuxSiSoEn port of the boundary scan instance(s).

shiftClock_BLn: n=2 to 5

During logic BIST, all shiftClock_BLn ports output the clock used by the functional logic during the shifting phase. The suffix n indicates the number of shift clock cycles generated during the burst phase. The shiftClock_BLn ports are automatically connected to the shiftClock input port of the corresponding Burst Clock controllers which have a burst length of n.

During memory BIST, the shiftClock_BLn ports output a free-running clock selected with the clkRatio input of the shift clock controller.

ltestClk

This output clock is used as the main clock to the logicTest controller. It is automatically connected to the BIST_CLK input port of the logicTest controller.

shiftPhase

This active high output port indicates that the circuit is operating in shift phase. It is automatically connected to the shiftPhase input port of all Scan Enable controllers and to the SHIFT_PHASE input port of the logicTest controller.

burstPhase

The burstPhase output port indicates that the logic BIST is in burst phase. This is used to control the Burst Clock controllers in the design. It is automatically connected to burstPhase input port of all Burst Clock controllers.

Burst Clock Controller

The BurstMode logic BIST architecture utilizes a number of controllers to generate the clocks while in BurstMode and to inject the shift clock during the shift phase. There will be one clock controller per clock domain in the design. [Figure 5-17](#) shows the conceptual schematic diagram of a Burst Clock controller for a domain with a single clock. [Figure 5-18](#) illustrates the case where the domain consists of a synchronous clock group. The Burst Clock controller is capable of aligning the active edges of a group of synchronous clocks.

The clock gating cells used to gate the functional clock and inject the shift clock is identical in both cases. However, the setup time at the input of the clock gating cell is a full period (T) of the clock input in the single clock case whereas it is half a period in the synchronous clock group case.

Figure 5-19 and Figure 5-20 illustrate the Launch Aligned and Capture Aligned clocking that are generated for a synchronous Burst Clock controller. The blue arrows in the figures identify the paths between the synchronous clocks that are tested, i.e., during a Launch Aligned burst phase the slow-to-fast synchronous clock paths are tested while during Capture Aligned the fast-to-slow synchronous clock paths are tested. The CD_LaunchAligned port on the logicTest controller determines when a synchronous Burst Clock controller will generate the Launch Aligned or Capture Aligned clock waveforms.

Figure 5-21 illustrates how demotion size and demotion rate are used to provide runtime adjustable burst clock waveforms. In this example, the demotion size is 2, and the demotion rate is 3. This effectively means that the first 2 clock cycles are slowed down (demoted) so that the effective clock rate is 1/4 of the nominal clock rate. This is achieved by suppressing 3-clock cycles after every demoted clock cycle.

Figure 5-22 shows how multi-cycle and false paths are handled during the burst phase.

Figure 5-17. Burst Clock Controller (Single Clock)

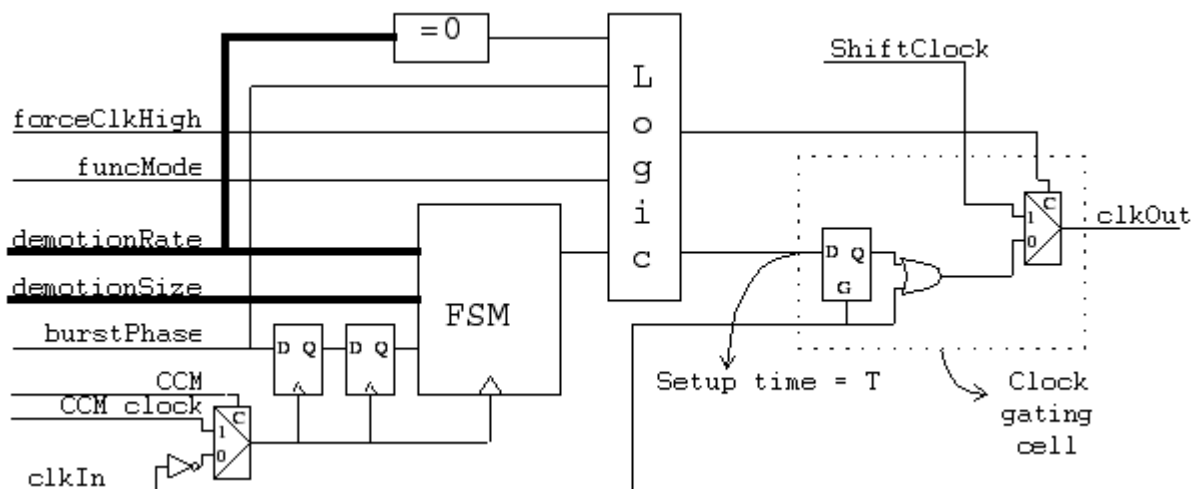


Figure 5-18. Burst Clock Controller (Synchronous Clock Group)

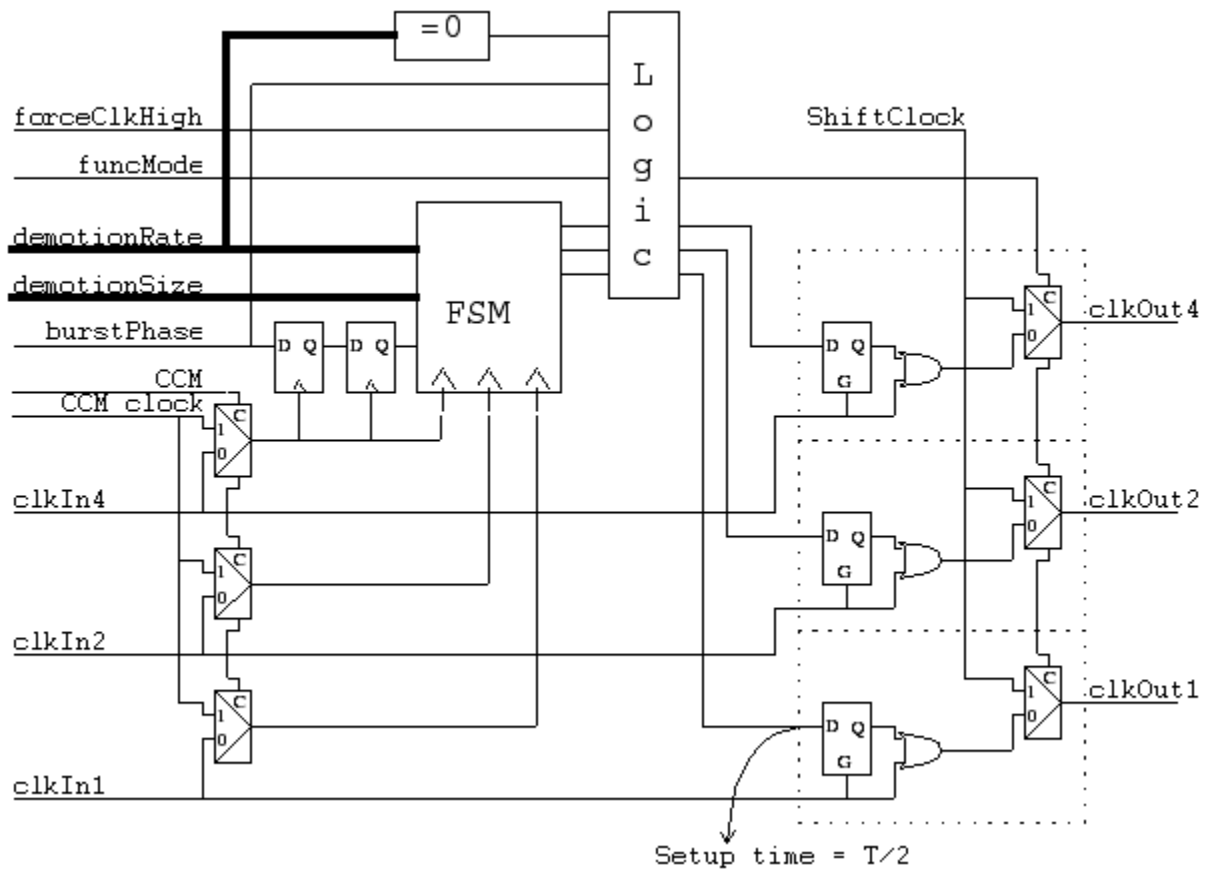


Figure 5-19. Launch Aligned Synchronous Clocks in Burst Clock Controller

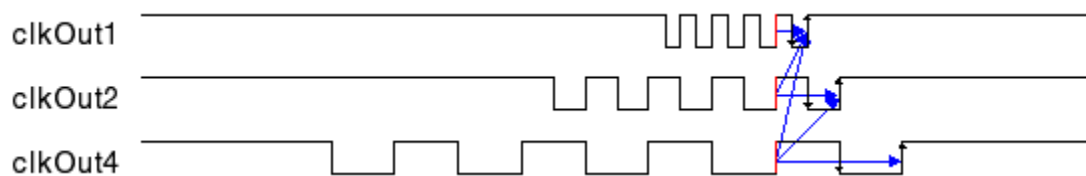


Figure 5-20. Capture Aligned Synchronous Clocks in Burst Clock Controller



Figure 5-21. Using Demotion Size and Rate in Burst Clock Controller

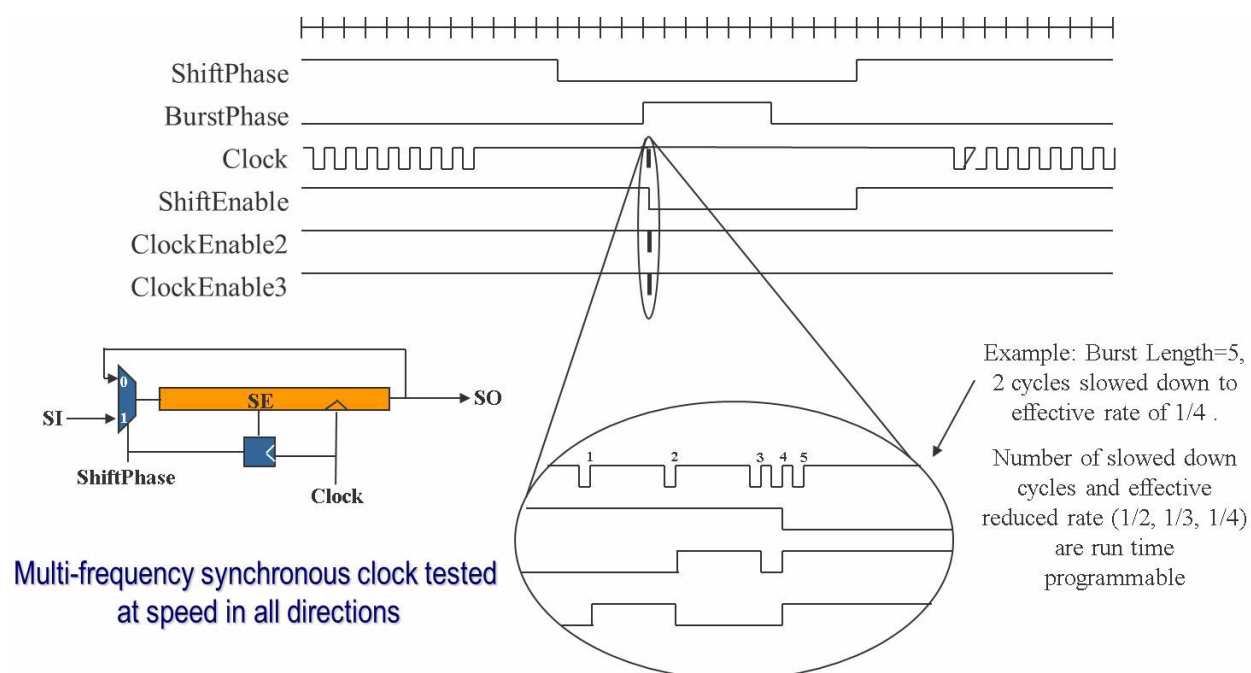
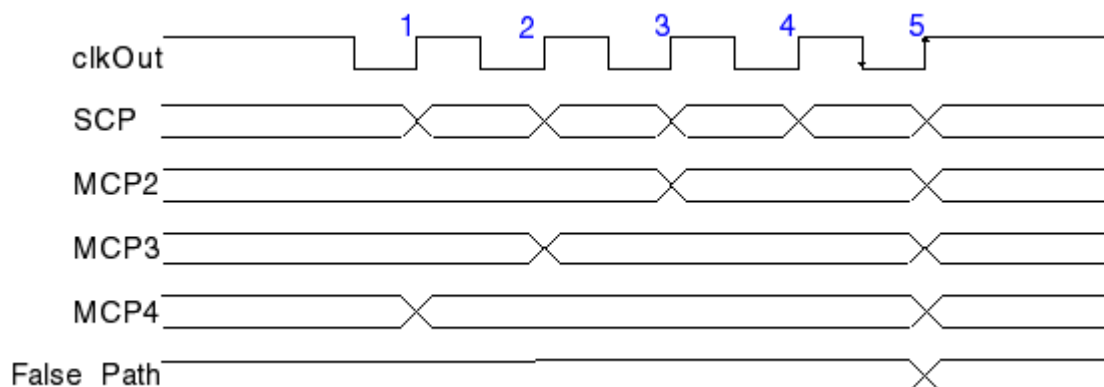


Figure 5-22. Multi-Cycle and False Path Waveforms During Burst Phase



Burst Clock Controller Ports

The following is a list of the Burst Clock controller ports:

TestClockSource

This input port is the clock source to be used during the burst phase by a Burst Clock controller whose functional clock is provided by clock divider that is not being reused during logicTest modes.

launchAligned

This input port is used by a synchronous burst clock controller to control whether the synchronous clocks will have a launch aligned or capture aligned waveform.

LV_SI

This input port is used to connect the scanout port of the Shift Clock controller for the controller chain test.

LV_SO

This output port is used to connect the scanin port of a Burst Clock controller for the controller chain test.

LV_SE

This input port is connected to the LV_SE output port on the Shift Clock controller and acts as the scanEnable for the controller chain test.

clk#

This input port is the functional clock source to be used during the burst phase. It is automatically connected to the clock source that is identified by ETChecker.

clkOut#

This is the output clock that will be used to clock the functional logic. In logicTest modes, this clock will be a burst of clocks depending on the run time options selected.

shiftClk

This input port provides the Burst Clock controller with the clock that will be used during the shift phase of logicTest modes. It is automatically connected to the shiftClock_BLn output port of the Shift Clock controller.

funcMode

The funcMode input port is asserted to indicate that the controller to forces the clock to logic HIGH. It is automatically connected to the funcMode port of the TAP/WTAP.

extTM

This active high input port is only available for Embedded Logic Test (ELT) controller. It is used to configure the core into external test mode.

Note



The extTM port is used as internal testmode at the next level up in hierarchical design.

killClock

This input port is used to force the burst clock to logic HIGH when asserted. It is automatically connected to the freezeMode port of the TAP/WTAP.

burstPhase

The burstPhase input port indicates that the logicTest is in burst phase. It is automatically connected to burstPhase output ports of all Shift Clock controllers.

bistEn

This active high input port indicates that the logicTest controller is active. It is automatically connected to the bistEn# port of the TAP/WTAP.

setupMode[2:0]

The setupMode[2:0] ports along with the BIST_EN port control the modes of the Burst Clock controller (setup shift, setup run, etc.). It is automatically connected to the TAP/WTAP setupMode[2:0] output ports.

demotionRate[1:0]

A two-bit input port is used to control the demoted clock speed to be used during the burst phase of logicTest modes. It is automatically connected to the respective logicTest controller port.

demotionSize[1:0]

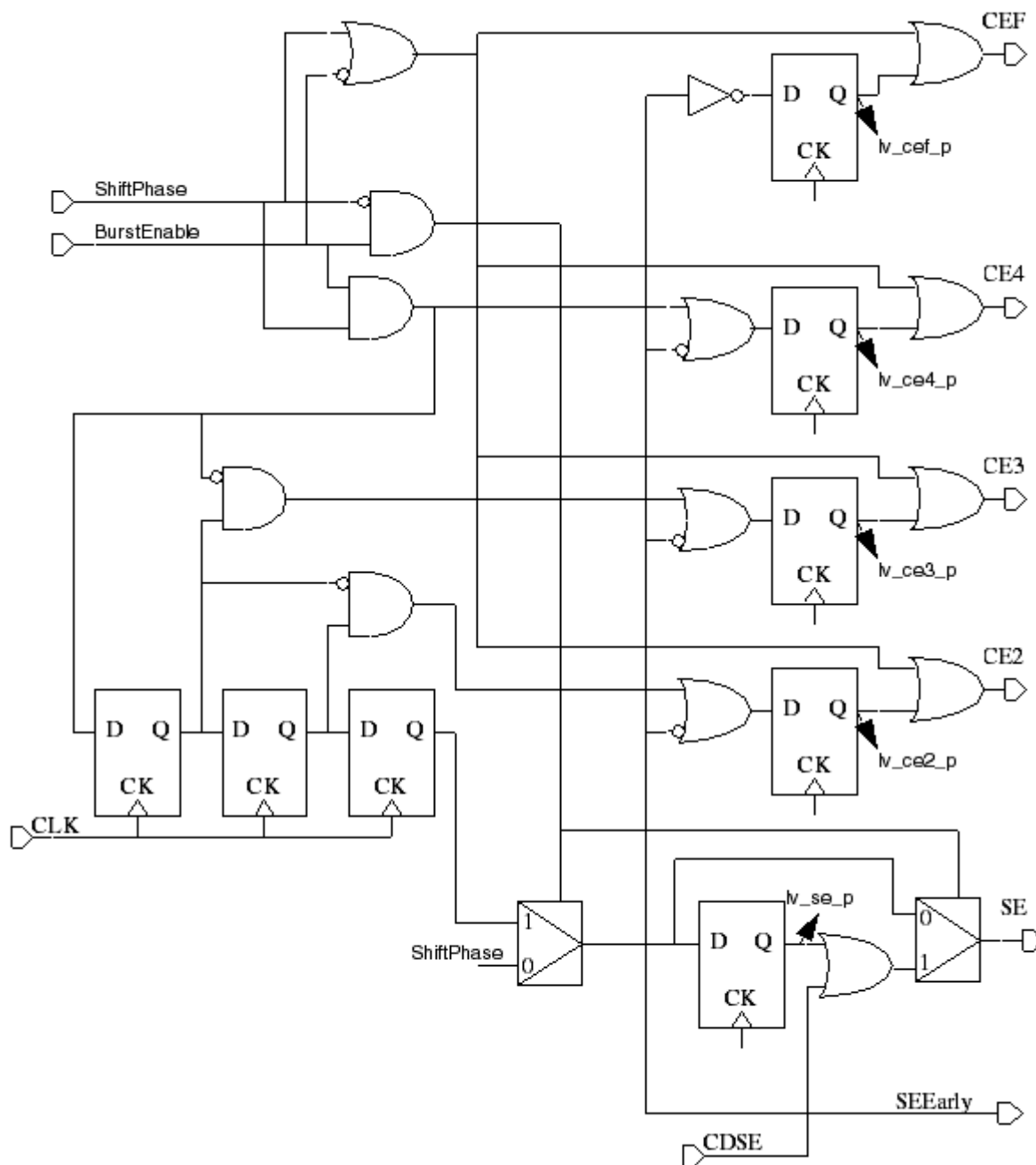
A two-bit input port used to control the number of burst phase clocks to use the demoted speed. It is automatically connected to the respective logicTest controller output ports.

Scan Enable and Clock Enable Controllers

In the BurstMode logicTest, shifting the data into the scan chains is controlled by a Scan Enable controller. The same controller is used to generate clock enable control logic. There would be one scan enable controller per clock domain per block.

Figure 5-23 shows the conceptual schematic diagram of the Scan Enable and Clock Enable controllers (Pipelined controlled signals: BL=5). The controller operates in burst logic BIST mode as well as in ATPG mode.

Figure 5-23. Scan and Clock Enable Controller



Scan Enable Controller Ports

The following are the ports associated with the Scan Enable controller:

Clk

This is an input source clock for the controller. It is automatically connected to the clkOut# output port of the respective Burst Clock controller.

ShiftPhase

This active high input port is used to indicate that the logicTest controller is operating in shift phase. It is automatically connected to the shiftPhase output port of the Shift Clock controller.

BurstEnable

This active input port is used to configure the controller to operate in BurstMode. When the signal is forced low, the controller will operate in scan mode. It is automatically connected to a TAP/WTAP userIR bit.

CEF

This output port is used to control flip-flops which are source of false paths contained within the clock domain of the input source clock.

CE4

This output port is used to control flip-flops which are source of multi-cycle paths contained within the clock domain of the input source clock. Up to 4 clock cycles are required to propagate signals from the multi-cycle path source to other flip-flops in the domain.

CE3

This output port is used to control flip-flops which are source of multi-cycle paths contained within the clock domain of the input source clock. Up to 3 clock cycles are required to propagate signals from the multi-cycle path source to other flip-flops in the domain.

CE2

This output port is used to control flip-flops which are source of multi-cycle paths contained within the clock domain of the input source clock. Up to 2 clock cycles are required to propagate signals from the multi-cycle path source to other flip-flops in the domain.

SE

This output port is used to configure flip-flops of the clock domain in shift (when high) or capture mode (when low).

CDSE

When high the CDSE output port makes the scanEnable controllers force the scanEnables high during the capture cycle. When the logicTest controller is not enabled this output port is low.

SEEarly

This output port is an early version of the SE Scan Enable controller signal; it is early by one clk-signal period.

Dedicated Isolation Cells

The hierarchical test approach requires a mechanism to partition the design into identifiable blocks that can be tested independently. The approach also requires a mechanism to facilitate the test of surrounding logic. These requirements are accomplished by an isolation mechanism. Embedded LogicTest utilizes two forms of isolation: Shared Isolation and Dedicated Isolation:

- Shared Isolation is a patented technique of making use of functional flip-flops to provide scan isolation of an ELT during internal or external test. This technique allows you to significantly reduce the amount of test circuitry required to implement a hierarchical test approach and to enable a true at-speed test of the periphery flip-flops made of functional flip-flops.
- Dedicated Isolation is the technique that makes use of test dedicated flip-flops to provide isolation of a ELT core during internal or external test. This technique is used to minimize the amount of periphery logic and periphery flip-flops. Input module pins with high fanout and output pins with high fanin are likely candidates to receive dedicated isolation circuitry. Dedicated Isolation can be inserted on input or output pins using the [lv.DedicatedIsolationPin](#), [lv.InternalScanInstance](#), [lv.ExternalScanOnlyPin](#), [lv.InjectControl](#) constraints in the *.etchecker* file.

The following table summarizes the different Dedicated Isolation cells that can be inserted by ETAssemble.

Table 5-3. Dedicated Isolation Cell Types

Dedicated Isolation Cell Types	Description
Input	Inserts a scannable flip-flop for observing and controlling an input port of the ELT core. Refer to Figure 5-24 . The ETChecker constraint lv.DedicatedIsolationPin on an input pin of an ELT core creates this cell.
Output	Inserts a scannable flip-flop for observing and controlling an output port of the ELT core. See Figure 5-25 . The ETChecker constraint lv.DedicatedIsolationPin on an output pin of an ELT core creates this cell.
External Scan-Only	Inserts a scannable flip-flop which takes part in the external test but does not contribute to the internal test. This cell is only inserted on output pins of the ELT core. Refer to Figure 5-26 . The ETChecker constraint lv.ExternalScanOnlyPin on an output pin of an ELT core creates this cell.

Table 5-3. Dedicated Isolation Cell Types (cont.)

Internal Scan-Only	Inserts a scannable flip-flop which takes part in the internal test but does not capture incoming data during the external test. This cell is only inserted on input pins of the ELT core. Refer to Figure 5-27 . The ETChecker constraint lv.InternalScanInstance on an input pin of an ELT core creates this cell.
Set Reset	Inserts the necessary logic to control the asynchronous set/reset input pin of an ELT core during the internal and external test. Refer to Figure 5-28 . Use of the ETChecker constraint lv.InjectControl of type AsyncSetRst on an input pin of an ELT core creates this cell.
Force 1	Inserts the necessary logic to force the value of an input pin of the ELT core to a logic 1 during internal and external test. Refer to Figure 5-29 . The ETChecker constraint lv.DedicatedIsolationPin with -force 1 on an input pin of an ELT core creates this cell.
Force 0	Inserts the necessary logic to force the value of an input pin of the ELT core to a logic 0 during internal and external test. Refer to Figure 5-30 . The ETChecker constraint lv.DedicatedIsolationPin with -force 0 on an input pin of an ELT core creates this cell.

Figure 5-24. Input Dedicated Isolation Cell

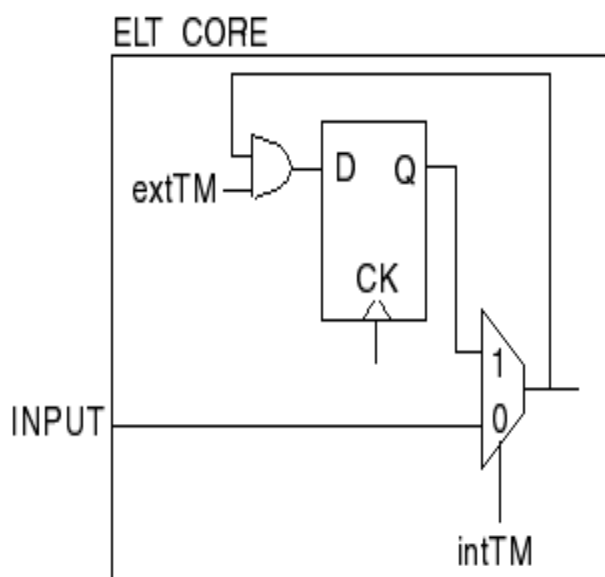


Figure 5-25. Output Dedicated Isolation Cell

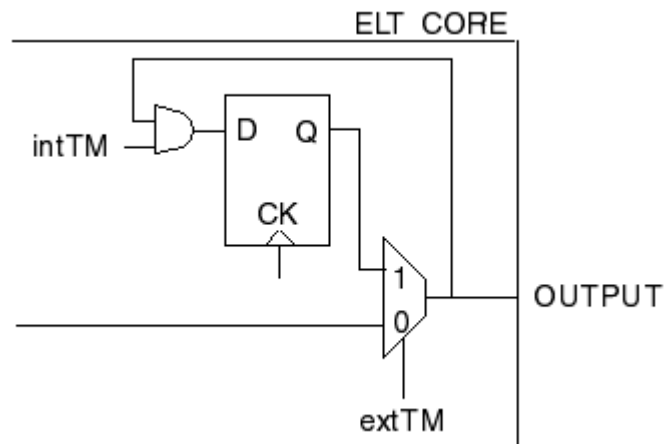


Figure 5-26. External Scan-Only Dedicated Isolation Cell

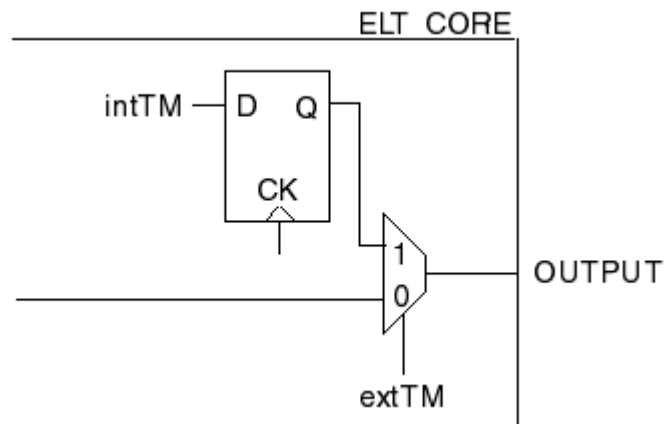


Figure 5-27. Internal Scan-Only Dedicated Isolation Cell

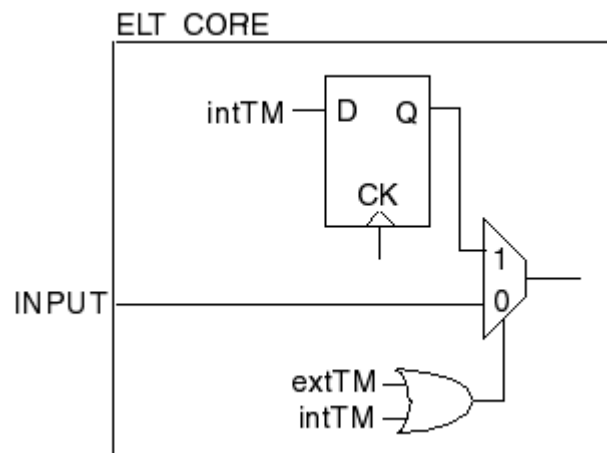


Figure 5-28. Set/Reset Dedicated Isolation Cell

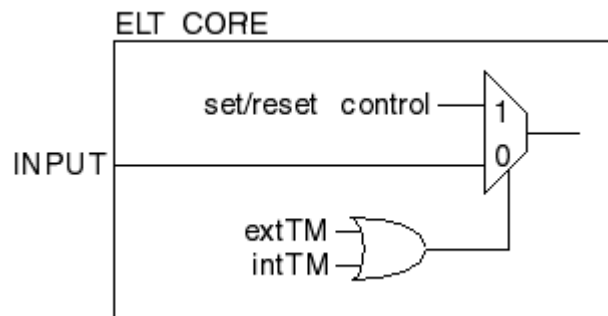


Figure 5-29. Force 1 Dedicated Isolation Cell

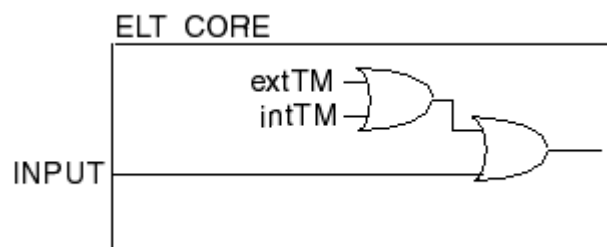
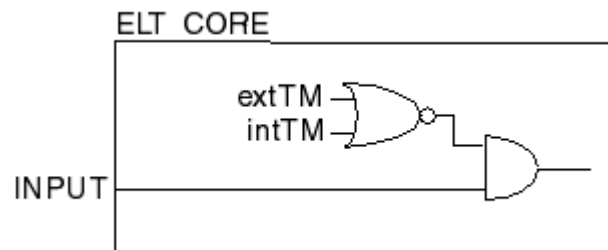


Figure 5-30. Force 0 Dedicated Isolation Cell



Test Point Types

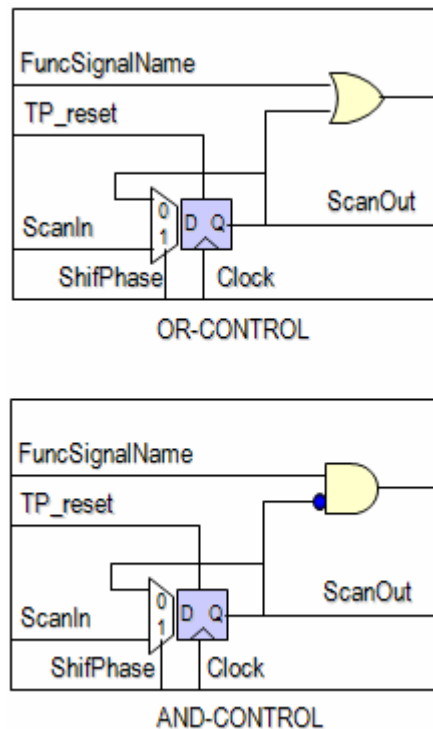
ScanGenerate processes the *.tpinfo* file and inserts the observation or control test points listed in the file. Observation test points are always implemented using dedicated flip-flops. However, control test points can be implemented using either dedicated or functional flip-flops depending on the value of the *ControlPointsImplementation* property in the *.etplan* file.

Setting the *ControlPointsImplementation* property to “*functionalFFs*” specifies that functional flip-flops should be used to implement control test points. However, an individual test point might still be implemented using a dedicated flip-flop if the tool is unable to identify a suitable candidate. For more information about the implementation of test points, see “[Selecting the Implementation Style of Testpoints](#)” in the *LV Flow User’s Manual*.

Using Dedicated Flip-flops

When you use a dedicated flip-flop to implement control points, the dedicated flip-flops are placed close to the test point to minimize routing as shown in [Figure 5-31](#). The flip-flops hold their state during the burst phase.

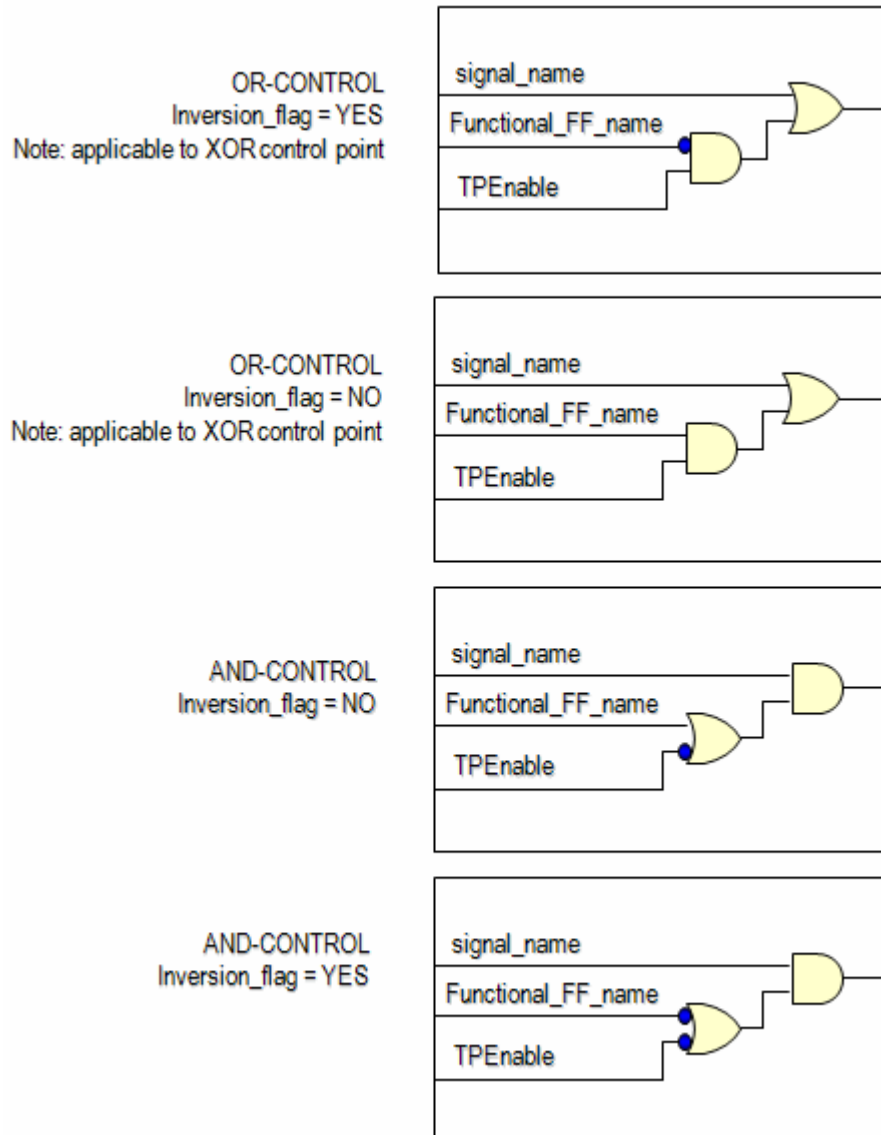
Figure 5-31. Control Points Implemented Using Dedicated Flip-flops



Using Functional Flip-flops

When you use a functional flip-flop to implement control points, the functional flip-flop is replaced by a two-input gate and an inversion is applied to the flip-flop output as indicated by the *inversion_flag* shown in [Figure 5-32](#).

Figure 5-32. Control Points Implemented Using Functional Flip-flops



Chapter 6

Memory BIST Controllers, Collars, and Interfaces

This chapter provides a description of the Tessent MemoryBIST hardware architecture. It provides a high-level view of the memory BIST controller types—NonProgrammable and Programmable, as well as the memory collar and interface.

This chapter covers the following topics:

- Tessent MemoryBIST BIST Architecture
- NonProgrammable Memory BIST Architecture
 - NonProgrammable Architecture with Collared Memories
 - NonProgrammable Memory BIST Controller
 - Step Counter
 - Port Counter
 - Address Counter
 - Bit Counter
 - Timing and Sequencing
 - Comparator
 - Algorithm
 - Top-Level Ports
 - NonProgrammable Memory Interface
 - SRAM
 - ROM
 - Non-Programmable Memory Collar
 - SRAM
 - ROM
- Programmable Memory BIST Architecture
 - Top-Level Signals of Programmable Memory BIST

- Controller Signal Descriptions
- Memory Interface Signal Descriptions
- Controller Signal Descriptions
- Controller Design Object
 - BIST FSM Block
 - Scannable Microcode Register Array Block
 - Pointer Control Block
 - Data Generator Block
 - Address Generator Block
 - Signal Generator Block
 - Dynamic Refresh Control Block
 - CounterA Block
 - DelayCounter Block
 - Repeat Loop Control Block
 - Step Counter Block
 - Port Counter
 - Bit Counter
 - Comparator
- Memory Interface Design Object
 - ROM Interface for Programmable Controller
- BIST Operating Protocol
- Using Local Comparators
 - Comparators Within the Controller
 - Comparators Within the Memory Collar/Interface
 - Syntax for Local Comparators

Tessent MemoryBIST BIST Architecture

The Tessent MemoryBIST product provides customers high flexibility while testing memories to achieve the target defect coverage and area overhead. The architecture supports two types of memory BIST controllers—Programmable and NonProgrammable:

- The NonProgrammable controller provides you with pre-defined memory test algorithms that are created for optimized controller and memory collars or interfaces size and performance. Details are provided in “[NonProgrammable Memory BIST Architecture](#).”
- The Programmable controllers enable you to define your own memory test algorithms. User-defined algorithms can be implemented in the hardware (HardProgrammable controller) or can be defined at tester time (SoftProgrammable controller). Details are provided in “[Programmable Memory BIST Architecture](#).”

This chapter provides a hardware overview of both types of controllers and their associated memory collar/interfaces. Note that both types of controllers use the same interface to the TAP/WTAP. However, they have different interfaces to the memories under test.

NonProgrammable Memory BIST Architecture

The NonProgrammable memory BIST uses a patented serial-access technique to provide a simple communication interface between a controller and the data path of the memory under test. This serial linkage minimizes the routing and gate overhead associated with memory BIST and offers greater versatility in testing several memories with a single memory BIST controller.

The memory BIST serial-data architecture is especially well suited for memories that have relatively wide data paths, where minimizing routing area and congestion are of concern.

- The memory BIST controller only needs to provide one bit of the test data-in to each memory that it tests.
- The memory BIST controller must only observe one output bit from each memory that it tests.

These two features significantly reduce the amount of routing between the memories and the memory BIST controller, compared to other schemes.

The NonProgrammable architecture supports two types of memory interfacing logic, namely, collar and interface.

The ETPlanner property

`GenerateMemoryInterface`: Yes | No;

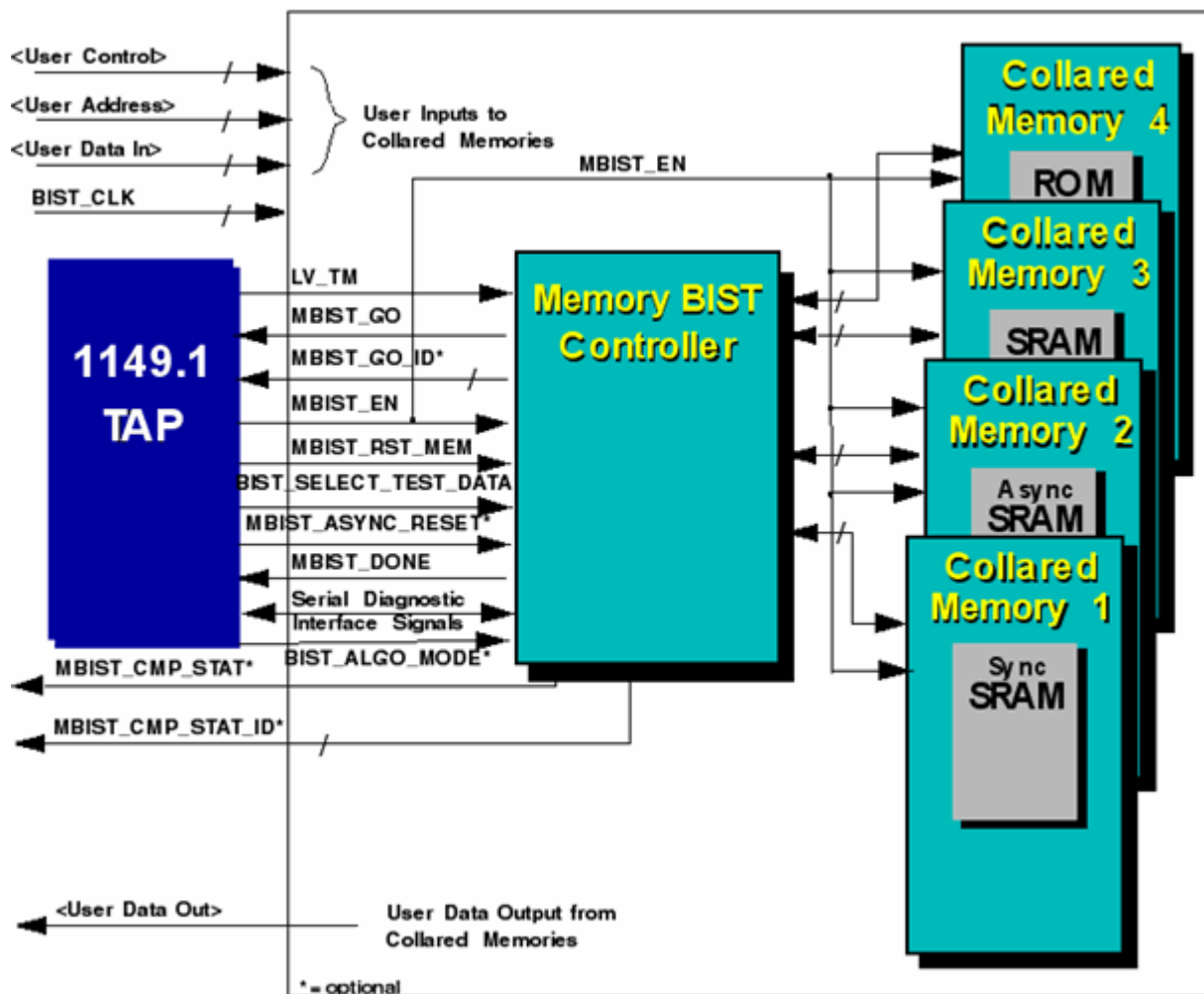
can be used to control the memory interface logic type. A memory interface is generated by default. The difference between the two is described in this chapter. Note that the BIST controller hardware is not different for both cases.

NonProgrammable Architecture with Collared Memories

When the ETPlanner property [GenerateMemoryInterface](#) is set to No, a collar is created around every memory. The memory collar contains all multiplexing and control logic needed for the BIST operation. Functional signal that do not need to be controlled in BIST mode are routed through the collar to the memory.

The simplified block diagram of BIST-enabled memories in [Figure 6-1](#) illustrates the basic architecture. The ETAssemble tool automatically creates the components of the BIST circuit—the memory BIST controller and collars.

Figure 6-1. High-Level View of the NonProgrammable Memory BIST Architecture with Collared Memories



As shown [Figure 6-1](#), the user signals in functional mode are as follows:

<User Control>

<UserControl>, such as the chip-enable and write-enable, controls a memory in the functional mode.

<User Address>

<UserAddress> means the address inputs to a memory in functional mode.

<User Data In>

<UserDataIn> denotes the data inputs to a memory in functional mode.

<User Data Out>

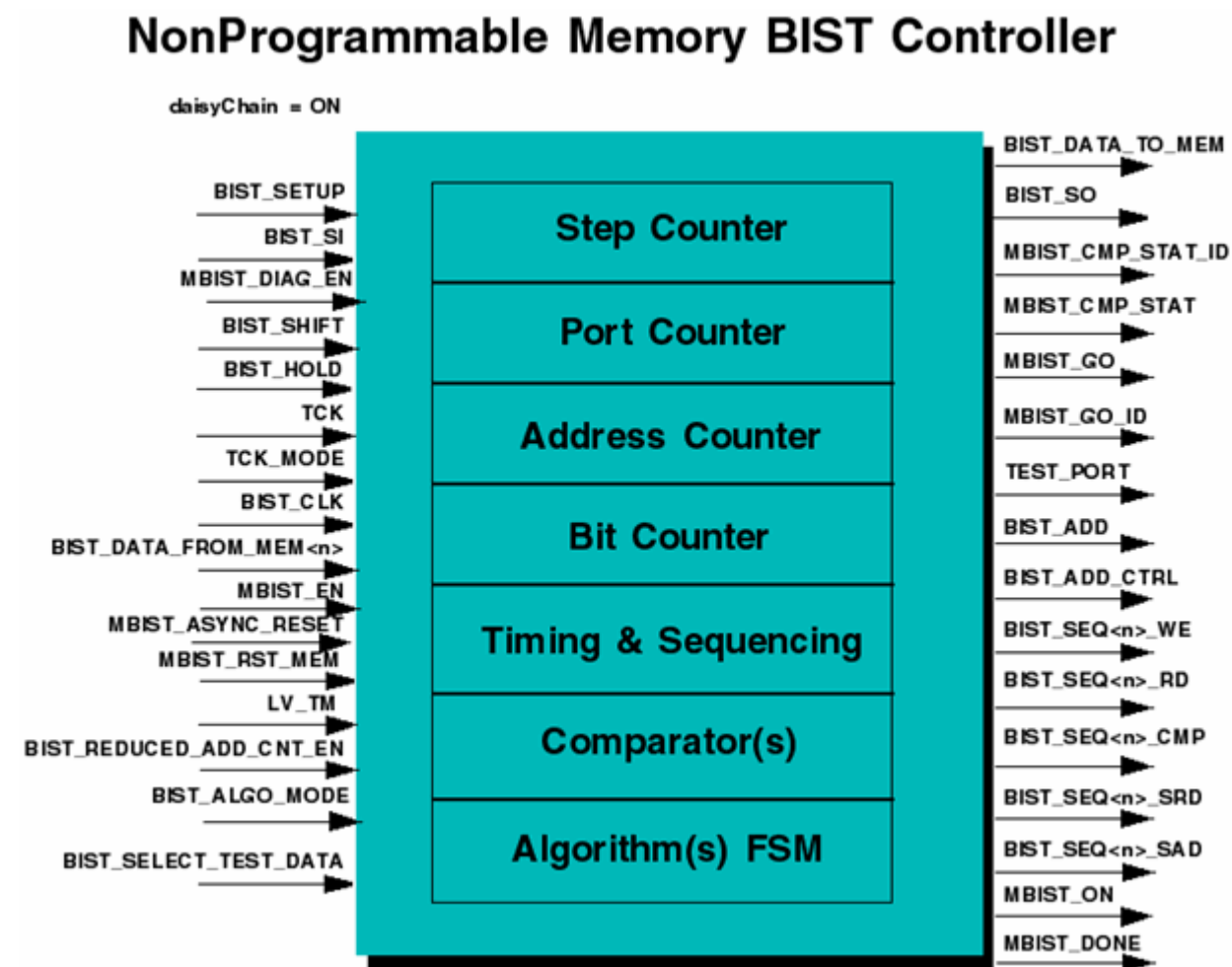
<UserDataOut> means the data outputs from a memory in functional mode.

For a description of memory BIST signals, refer to “[Top-Level Ports](#).”

NonProgrammable Memory BIST Controller

The main component of the *NonProgrammable* memory BIST architecture is the memory BIST controller, which controls the sequencing of control signals, address, and data to the collared memories. The *NonProgrammable* memory BIST controller consists of several functional modules, as shown in [Figure 6-2](#).

Figure 6-2. NonProgrammable Memory BIST Controller Functional Diagram for a Synchronous Dual Port SRAM



For a description of the NonProgrammable memory BIST controller signals, refer to “[Top-Level Ports](#).”

Step Counter

This module counts and controls the sequencing of memories when there are multiple steps in the memory BIST configuration file.

Port Counter

This module counts and controls the sequencing of the various ports of multi-port memories.

Address Counter

This module counts through the address space of the memory being tested according to the specified algorithm.

Bit Counter

This module counts and controls the sequencing of the test data to a word-slice as specified in the memory BIST configuration file.

Timing and Sequencing

This module controls the overall timing and sequencing of control signals necessary to access the memories. This module also controls the interaction of the other modules within the memory BIST controller.

Comparator

This module contains the comparator(s) used for checking the actual data from the memory against the expected data generated by the memory BIST controller. Note that parts of this module may reside in the memory collar when local comparators are used.

Algorithm

This module contains the hardware implementations of the selected algorithms necessary for the memory BIST configuration.

Top-Level Ports

The memory BIST RTL has three types of ports in its port list:

- Ports used for BIST control and must be connected to the TAP or WTAP controller.
- Ports used for internal communication between the memory controller and its interfaces and collars.
- Ports used for serial diagnostic interface that provides access to specific registers in the memory controller and collars.

All these ports are described in [Table 6-1](#).

Table 6-1. Port List for the NonProgrammable Memory BIST Controller

Signal Name	Type	Description
NonProgrammable Memory BIST Ports:		
BIST_ALGO_MODE	I	Inputs to the NonProgrammable memory BIST controller to control the execution of the BIST run when parallel retention test is required. This port is generated when the ETPlanner ParallelRetentionTest property is set to On or PerGroup. Using ETVerify, you can create a testbench for parallel retention testing by specifying the ParallelRetentionTest property to On in the ETVerify MembistVerify wrapper.
BIST_CLK	I	The clock input to the NonProgrammable memory BIST controller.
BIST_CLK_EN	I	An input to the NonProgrammable memory BIST controller and memory collar that allows the clock input BIST_CLK to be disabled. This port is generated when the ETPlanner GateBistClockInFuncMode property is set to <i>On</i> .
BIST_REDUCED_ADD_CNT_EN	I	An input to the NonProgrammable memory BIST controller that enables you to check the proper functionality of the BIST controller without having to simulate the test of the entire memory space. Using ETVerify you can generate a testbench for a reduced address simulation by specifying the ReducedAddressCount property to <i>On</i> in the ETVerify MembistVerify wrapper.
BIST_SELECT_TEST_DATA	I	An input to the NonProgrammable memory BIST controller that is connected to a UserIR bit output of the TAP controller. It is used to control the BIST multiplexers within the memory collar during parallel retention test. This port is generated when the ETPlanner ParallelRetentionTest property is set to <i>On</i> or <i>PerGroup</i> . Using ETVerify, you can create a testbench for parallel retention testing by specifying the ParallelRetentionTest property to <i>On</i> in the ETVerify MembistVerify wrapper.
LV_TM	I	LV_TM, when active, isolates the <userDataOut> from the surrounding logic. LV_TM must be held high during scan test or logic BIST. During memory BIST, LV_TM must be held low.

Table 6-1. Port List for the NonProgrammable Memory BIST Controller (cont.)

Signal Name	Type	Description
MBIST_ASYNC_RESET	I	An active-low input port to the NonProgrammable memory BIST controller and memory collar that asynchronously resets all flip-flops in the controller and collar.
MBIST_CMP_STAT	O	A BIST diagnostic output from the NonProgrammable memory BIST controller that provides the global comparator status. It is generated when the ETPlanner CompStat property is set to On. As the output of the comparator, this signal will normally be high during memory BIST. This signal is the logical AND of the individual MBIST_CMP_STAT_ID bits. If any memory in the configuration fails, MBIST_CMP_STAT falls for one cycle and then rises again. It continues to fall for one cycle for every additional memory failure.
MBIST_CMP_STAT_ID	O	BIST diagnostic outputs from the NonProgrammable memory BIST controller that provides the individual comparator status (one per comparator defined in the memory BIST configuration file). Each of the MBIST_CMP_STAT_ID signals will fall for one cycle if any memory using its associated comparator has a memory failure.
MBIST_CMP_STAT_ID_SEL	I	Inputs to the NonProgrammable memory BIST controller that select an individual comparator status in the controller to be monitored on the global MBIST_CMP_STAT or MBIST_GO output. These ports are generated when the ETPlanner CompStat property is set to On or sharedWithGo, and the CompStatMux property is specified to On. Normally, they are connected to the TAP UserIRBits.
MBIST_DIAG_EN	I	An input port used for diagnostics. It is generated when the ETPlanner CompStat property is set to SharedWithGo, or when one or more memory collars contain local comparators. This port is to be connected to the TAP. When asserted, the MBIST_GO signal becomes the CMP_STAT signal. Using ETVerify, you can create a testbench for diagnostic mode by specifying the Diagnostic property to On in the ETVerify MembistVerify wrapper.

Table 6-1. Port List for the NonProgrammable Memory BIST Controller (cont.)

Signal Name	Type	Description
MBIST_DONE	O	An output from the <i>NonProgrammable</i> memory BIST controller that provides the status indicating whether BIST is in progress. This signal is low during memory BIST. When MBIST_DONE rises, this indicates the completion of memory BIST. If the MBIST_GO signal is still high when MBIST_DONE rises, then the entire memory configuration has passed all tests.
MBIST_EN	I	An input to the NonProgrammable memory BIST controller that enables or disables the memory BIST mode. This input is held high throughout the duration of memory BIST. If this signal falls at any time, the controller will exit from the memory BIST mode.
MBIST_GO	O	An output from the NonProgrammable memory BIST controller that provides the global pass/fail result from all comparators in the controller. It is a logical AND of the MBIST_GO_ID signals. MBIST_GO is a status signal that is normally high throughout the duration of memory BIST. If a memory failure occurs, MBIST_GO is driven low and remains low for the duration of memory BIST. When the ETPlanner CompStat property is set to SharedWithGo, and MBIST_DIAG_EN is asserted, the MBIST_GO port will carry the waveform of the global comparator status MBIST_CMP_STAT.
MBIST_GO_ID	O	Outputs from the NonProgrammable memory BIST controller that provides the pass/fail result of each comparator. There is one GO bit assigned to each comparator according to the ETPlanner BitSliceWidth property. These status signals are normally high throughout the duration of memory BIST. If a memory failure occurs on any memory using the comparator, the corresponding MBIST_GO_ID will fall and will remain low for the duration of memory BIST. When the ETPlanner CompStat property is set to SharedWithGo, and MBIST_DIAG_EN is asserted, the MBIST_GO_ID ports will carry the waveform of the individual comparator status MBIST_CMP_STAT_ID.
MBIST_ON	O	A status output of the NonProgrammable memory BIST controller that indicates that the system is in memory BIST mode.

Table 6-1. Port List for the NonProgrammable Memory BIST Controller (cont.)

Signal Name	Type	Description
BIST_BIRA_EN	I	An input to the NonProgrammable memory BIST controller that enables the redundancy analysis mode. This port is generated when at least one repairable memory is present. Using ETVerify, you can activate the redundancy analysis mode by specifying the CheckRepairStatus or ExtractRepairFuseMap property in the MembistVerify wrapper.
MBIST_RA_PRSRV_FUSE_VAL	I	An input to the NonProgrammable memory BIST controller and memory collar that preserves the values logged in the repair analysis registers. This port is generated when at least one memory implements repair analysis. Using ETVerify, you can activate the preservation mode by specifying the PreserveFuseRegisterValues property to On in the ETVerify MembistVerify wrapper.
<mem>_REPAIR_STATUS	O	Reports the overall repair status of a repairable memory. The status values are as follows: 00 indicates no repair is required 01 indicates the memory is repairable 11 indicates the memory cannot be repaired
<mem>_<seg>_SCOL#_ALLOC_REG	O	Reports the allocation of a spare element for a memory with IO/column redundancy.
<mem>_<seg>_SCOL#_FUSE_REG	O	A group of outputs that reports the defective IO for a memory with IO/column redundancy.
<mem>_<seg>_SCOL#_FUSE_ADD_REG	O	A group of outputs that reports the defective address for a memory with IO/column redundancy.
<mem>_<seg>_SROW#_ALLOC_REG	O	Reports the allocation of a spare element for a memory with row redundancy.
<mem>_<seg>_SROW#_FUSE_ADD_REG	O	A group of outputs that reports the defective address for a memory with row redundancy.
MBIST_RST_MEM	I	An input to the NonProgrammable memory BIST controller that forces the BIST FSM to apply only the initialization portion of the algorithm to the memories. If this signal is driven high, the NonProgrammable memory BIST controller writes 0 to each address location of each SRAM. Using ETVerify, you can activate the memory reset mode by specifying the MemoryReset property to On in the ETVerify MembistVerify wrapper.

Table 6-1. Port List for the NonProgrammable Memory BIST Controller (cont.)

Signal Name	Type	Description
Internal Collar and Interface Ports:		
BIST_ADD	O	A group of outputs from the NonProgrammable memory BIST controller that provide the test addresses to a memory collar.
BIST_ADD_CTRL	O	A group of outputs from the NonProgrammable memory BIST controller that controls the address counter in the memory collar. The counter is inserted when the ETPlanner LocalAddressCounter property is set to On or PerPort.
BIST_DATA_FROM_MEM<n>	I	Inputs to the NonProgrammable memory BIST controller that are connected to the test data outputs from a memory collar. These ports are not generated when local comparators are used.
BIST_DATA_TO_MEM	O	The data output from the NonProgrammable memory BIST controller that connects to the test data inputs of a memory collar. These outputs provide the data to be written to a memory during a write operation.
BIST_SEQ<n>_WE	O	An output from the NonProgrammable memory BIST controller that generates the WriteEnable signal for writing to the active port of a memory.
BIST_SEQ<n>_RD	O	An output from the NonProgrammable memory BIST controller that generates the ReadEnable signal for reading from the active port of a memory.
BIST_SEQ<n>_STB	O	An output from the NonProgrammable memory BIST controller that controls when the memory output data is captured by the strobing flip-flops when they exist in the memory collar.
BIST_SEQ<n>_SRD	O	An output from the NonProgrammable memory BIST controller that generates the ShadowReadEnable signal for reading from an inactive port of a multi-port RAM.
BIST_SEQ<n>_SAD	O	An output from the NonProgrammable memory BIST controller that generates the ShadowReadAddress for reading from an inactive port of a multi-port RAM.
BIST_SEQ<n>_DATA_ON	O	An output from the NonProgrammable memory BIST controller that specifies when data is read from or written to the memory with a bi-directional data bus

Table 6-1. Port List for the NonProgrammable Memory BIST Controller (cont.)

Signal Name	Type	Description
BIST_SEQ<n>_STB_CLK	O	An output from the NonProgrammable memory BIST controller that generates the StrobeClock signal for (asynchronous) memories that require a strobe pulse to activate a read or write operation.
BIST_SEQ<n>_OE	O	An output from the NonProgrammable memory BIST controller that generates the OutputEnable signal for enabling the memory output drivers of the active port.
BIST_SEQ<n>_CMP	O	An output from the NonProgrammable memory BIST controller that specifies when the memory output data is sampled by the comparators. This port is generated when local comparators are used.
BIST_SEQ<n>_ATD	O	An output from the NonProgrammable memory BIST controller that generates the ATD signal for activating the address transitions for memories with address-transition-detect interfaces.
TEST_PORT	O	An output from the NonProgrammable memory BIST controller that selects the logical port to be tested (active port) in a multi-port memory.
BIST_CS<n>	O	Outputs from the NonProgrammable memory BIST controller that selects the memory collar(s) to be tested per Step. These ports are generated when at least two Step wrappers are present in the ETPlanner configuration file.
BIST_CS<n>_R<n>	O	Outputs from the NonProgrammable memory BIST controller that selects the memory collar(s) to be tested per Step . These ports are generated when the memories within a Step have different row and/or column address ranges.
BIST_CKB_EN	O	An output from the NonProgrammable memory BIST controller that formats the checkerboard data to/from the memory. This port is generated when the memory test algorithm uses the checkerboard pattern.
BIST_SHWRT	O	An output from the NonProgrammable memory BIST controller that enables the ShadowWrite operation on the inactive port of the memory. It is created only the memory is a multi-port memory and the memory library file ShadowWrite property is set to On.

Table 6-1. Port List for the NonProgrammable Memory BIST Controller (cont.)

Signal Name	Type	Description
BIST_COLUMN_SHWRT	O	An output from the NonProgrammable memory BIST controller that enables the ColumnShadowWrite operation on the inactive port of the memory. It is created only when the memory is a multi-port memory, and the memory library file ShadowWrite property is set to On. This port is generated when the SMarchCHKBvcd test algorithm is selected.
BIST_SELECT_TEST_DATA_TO_COLLAR	O	An output from the NonProgrammable memory BIST controller that controls the BIST multiplexers within the memory collar during parallel retention test. This port is generated when the ETPlanner ParallelRetentionTest property is set to On or PerPort
BIST_CLEAR	O	An output from the NonProgrammable memory BIST controller. During the initialization phase, it resets the registers in the serial diagnostic chain within the memory collar. This port is generated when local comparators are used.
BIST_CLEAR_DEFAULT	O	An output from the NonProgrammable memory BIST controller. During the initialization phase in HWDefault run mode, it resets the registers that select the individual comparator status within the memory collar. This port is generated when local comparators are used.
BIST_SHIFT_SHORT	O	An output from the NonProgrammable memory BIST controller that places the memory collar in serial shifting (scan) mode. This port is generated when local comparators are used.
BIST_TO_COLLAR_SO	O	An output from the NonProgrammable memory BIST controller that connects the serial diagnostic chain from the controller to the memory collar(s). This port is generated when local comparators are used.
BIST_FROM_COLLAR_SI	I	An input to the NonProgrammable memory BIST controller that connects the serial diagnostic chain from the memory collar(s) back to the controller. This port is generated when local comparators are used.
BIST_EXP_DATA	O	An output from the NonProgrammable memory BIST controller that provides the expected data value when reading the memory output. This port is generated when local comparators are used.

Table 6-1. Port List for the NonProgrammable Memory BIST Controller (cont.)

Signal Name	Type	Description
BIST_DIAG_EN_TO_COLLAR	O	An output from the NonProgrammable memory BIST controller that places the memory collar in diagnostic mode. The composite pass/fail result from the memory collar MBIST_GO<n> become the comparator status. This port is generated when local comparators are used.
MBIST_GO<n>	I	Inputs to the NonProgrammable memory BIST controller that connect to the BIST_GO output of memory collars with local comparators. This port provides the composite pass/fail result and becomes the comparator status during diagnostic mode.
BIST_CTL_OBSERVE_FLOPS_Q	O	Outputs from the scan observation points for memory control signals within the memory collar. These ports are generated when the ETPlanner property ObservationLogic is set to Yes.
BIST_PORT<n>_ADD_OBSERVE_FLOPS_Q	O	Outputs from the scan observation points for memory address signals within the memory collar. These ports are generated when the ObservationLogic property is specified in the ETPlanner configuration file to Yes.
BIST_LINK_BITSLICE	O	An output from the NonProgrammable memory BIST controller that configures serial or parallel access to the memory data input during the SMarchCHKBvcd test algorithm.
BIST_SHIFT_BITSLICE_FROM_MSB	O	An output from the NonProgrammable memory BIST controller that indicates the shift direction when the memory data port is accessed serially. This port is generated when the SMarchCHKBvcd Algorithm is selected.
BIST_DATAPATH_EN<n>	O	Outputs from the NonProgrammable memory BIST controller that enables serial access to the memory data input. This port is generated when the SMarchCHKBvcd Algorithm is selected.
BIST_WE_MASK	O	An output from the NonProgrammable memory BIST controller that controls the memory bit/group write enable port during the SMarchCHKBvcd Algorithm .
BIST_MEM_RD_OVERRIDE	O	An output from the NonProgrammable memory BIST controller that controls the memory read enable port during the SMarchCHKBvcd Algorithm .
BIST_MEM_RD_OVERRIDE_ADD_CTRL	O	Outputs from the NonProgrammable memory BIST controller that controls the memory address port during the SMarchCHKBvcd Algorithm .

Table 6-1. Port List for the NonProgrammable Memory BIST Controller (cont.)

Signal Name	Type	Description
BIST_MEM_CS_OVERRIDE	O	An output from the NonProgrammable memory BIST controller that controls memory select port during the SMarchCHKBvcd Algorithm .
BIST_MEM_CS_OVERRIDE_ADD_CTRL	O	An output from the NonProgrammable memory BIST controller that controls the memory address port during the SMarchCHKBvcd Algorithm .
BIST_RA_SUSPEND	O	An output from the NonProgrammable memory BIST controller that suspends the IO/Column repair analysis engine within the memory collar during specific phases of the SMarchCHKBvcd Algorithm .
BIST_IO_RA_PRSRV_FUSE_VAL_TO_COLLAR	O	An output from the NonProgrammable memory BIST controller that preserves the values logged in the IO/Column repair analysis registers within the memory collar.
<mem>_ERROR	I	An input to the NonProgrammable memory BIST controller that provides the comparator status from memory collar <mem> to the Row repair analysis engine.
SHORT_SETUP	O	An output from the NonProgrammable memory BIST controller that configures the serial diagnostic chain within the memory collar containing a memory with IO/Column repair analysis.
Serial Diagnostic Interface Ports:		The following ports are associated with the serial diagnostic interface.
BIST_HOLD	I	An input to the memory BIST controller that instructs the controller to retain its state while the internal register of the controller is shifted. It is normally connected to the TAP holdBist output port.
BIST_SHIFT	I	An input to the memory BIST controller that places the controller in serial shifting (scan) mode. It is normally connected to the TAP shiftBist output port.
BIST_SETUP	I	A two-bit input bus that specifies the run and setup modes for the memory BIST controller. It is normally connected to the TAP setupMode output ports.
BIST_SI	I	An input to the memory BIST controller that serves as the scan input for accessing the controller's internal registers. The internal registers are accessed for: <ul style="list-style-type: none"> • Setup of the controller before running BIST • Retrieval of test result and/or diagnostic data

Table 6-1. Port List for the NonProgrammable Memory BIST Controller (cont.)

Signal Name	Type	Description
BIST_SO	O	An output from the memory BIST controller that serves as the scan output during retrieval of test result and diagnostic data.
TCK	I	An input to the memory BIST controller that is connected to the test clock of the TAP.
TCK_MODE		An input to the memory BIST controller that indicates the test clock from the TAP is driving the system clock network. It is normally connected to the TAP tckMode output port.

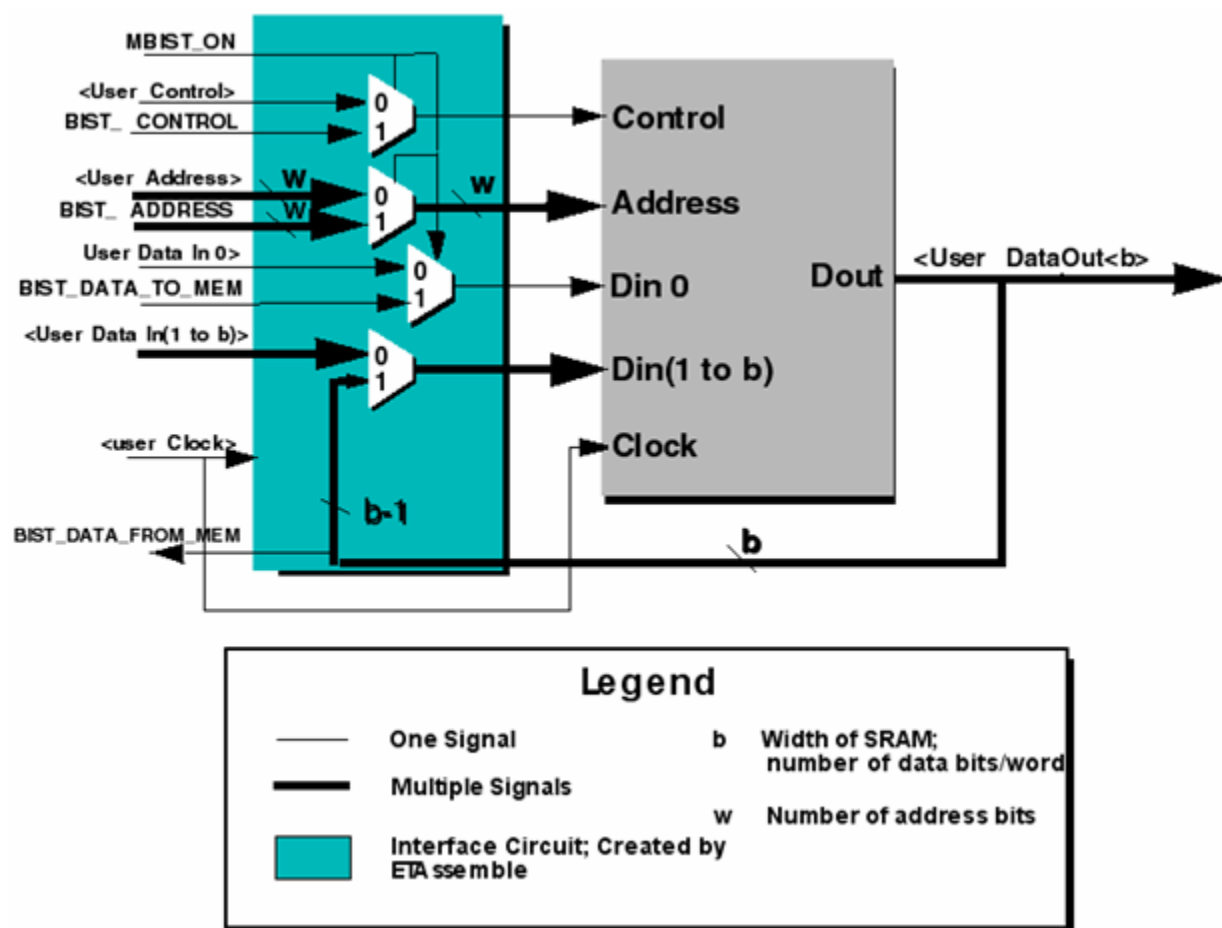
NonProgrammable Memory Interface

The next component of the NonProgrammable memory BIST architecture is the memory interface. The interface provides the link between the NonProgrammable memory BIST controller and the memory.

SRAM

Figure 6-3 shows how a synchronous (clocked) SRAM is connected to the interface. The memory is tested using the serial-access technique.

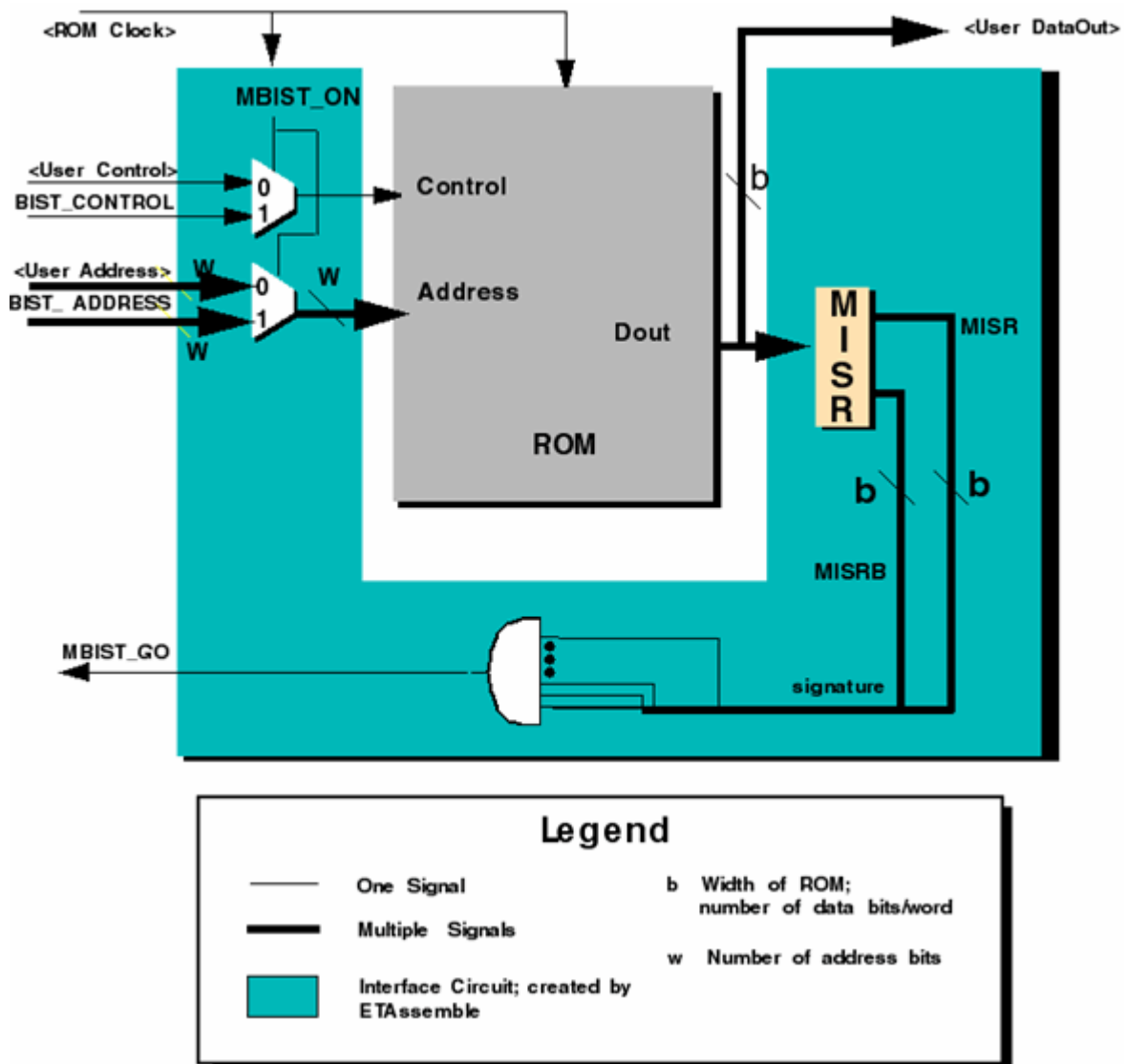
Figure 6-3. Connections to the Interface for a Synchronous (Clocked) SRAM Memory



ROM

Figure 6-4 shows how a ROM is connected to the interface.

Figure 6-4. ROM Connection to the Interface



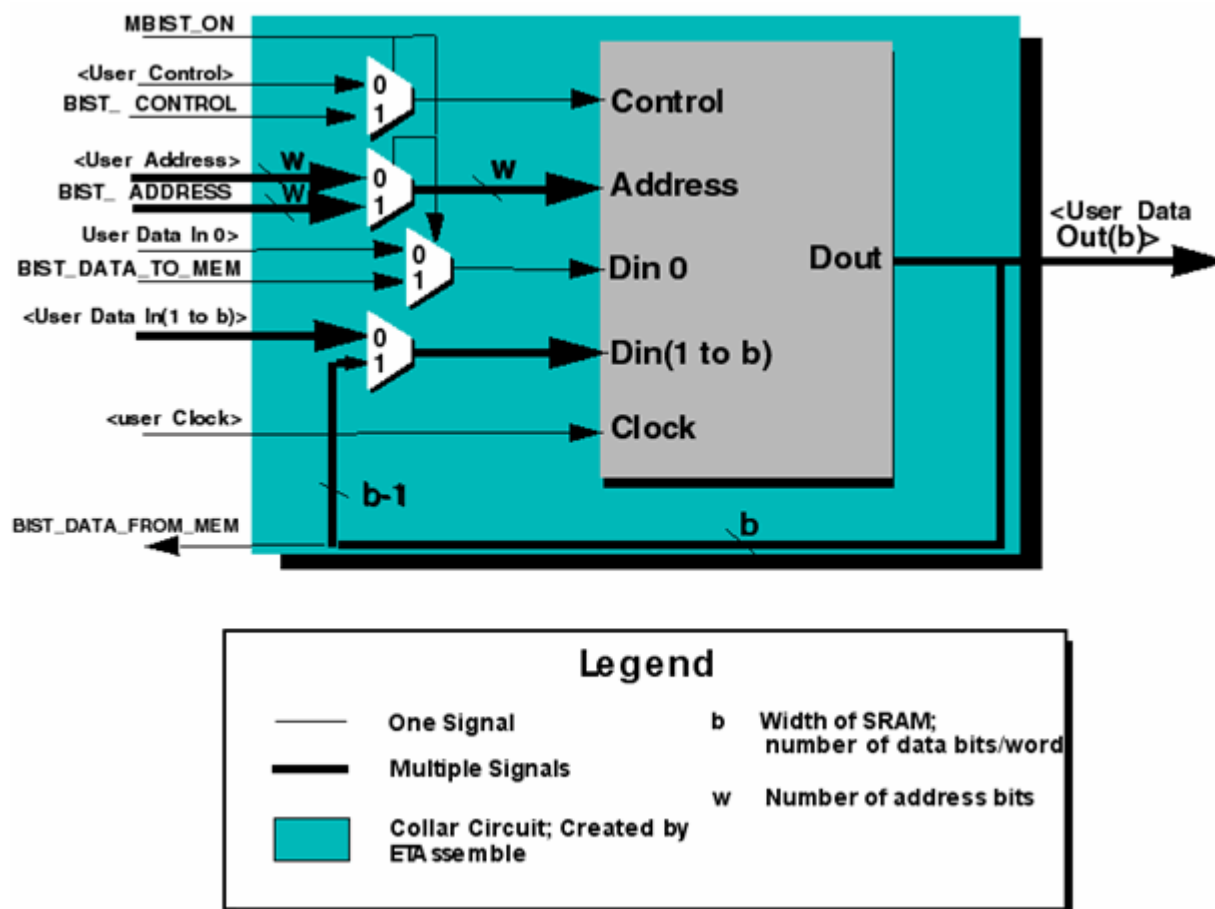
Non-Programmable Memory Collar

The NonProgrammable memory BIST architecture also supports the generation of collar from the memory. The collar provides the interface between the NonProgrammable memory BIST controller and the memory.

SRAM

Figure 6-5 shows how a synchronous (clocked) SRAM is connected to the collar. The memory is tested using the serial-access technique.

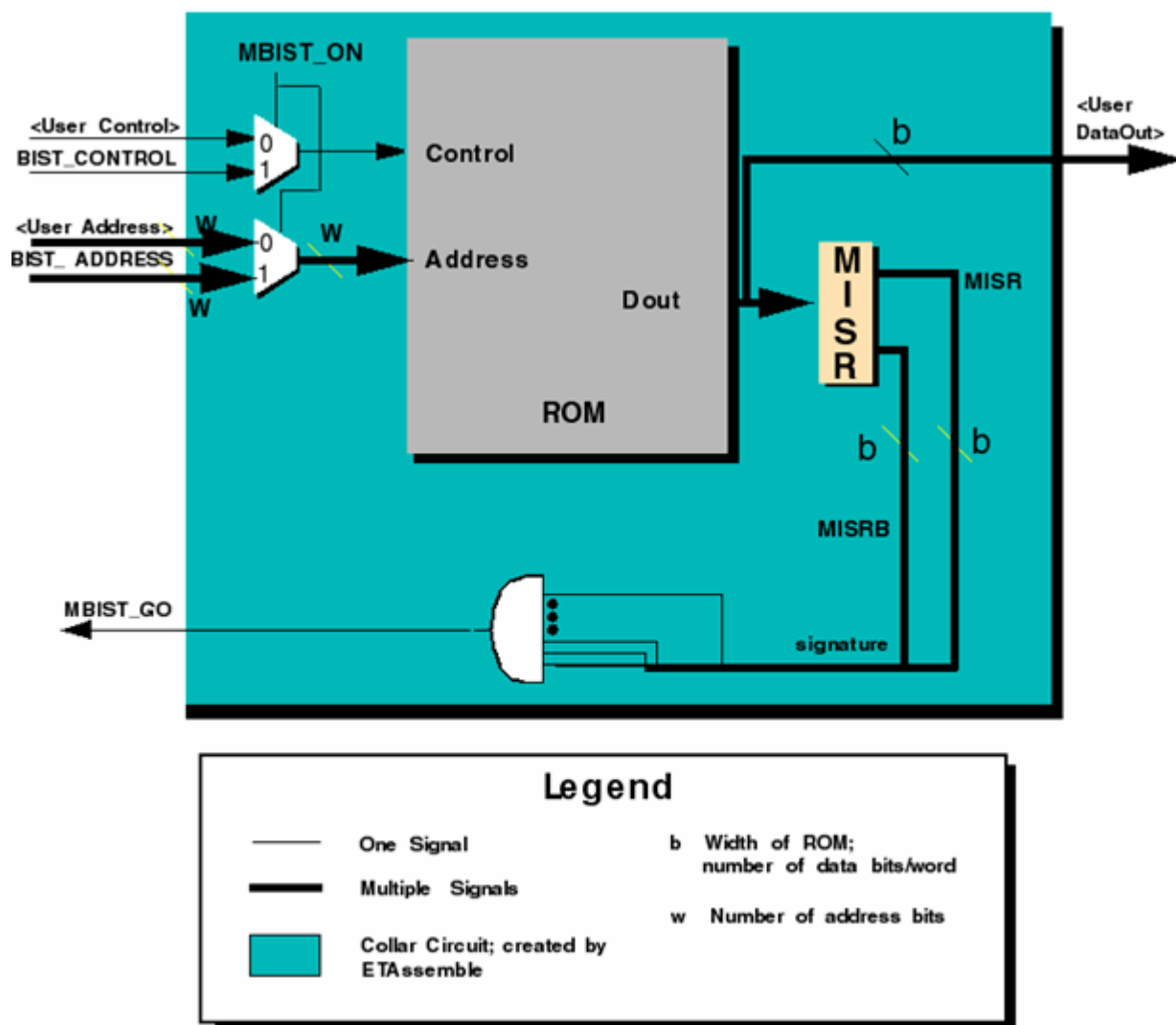
Figure 6-5. Connections to the Collar for a Synchronous (Clocked) SRAM Memory



ROM

Figure 6-6 shows how a ROM is connected to the collar.

Figure 6-6. ROM Connection to the Collar



Programmable Memory BIST Architecture

Tessent MemoryBIST Programmable controller architecture incorporates more hardware to enable algorithm programmability. It also uses different interface to the memories under test. The Programmable memory BIST consists of two design objects:

- Programmable memory BIST controller
- Memory Interface

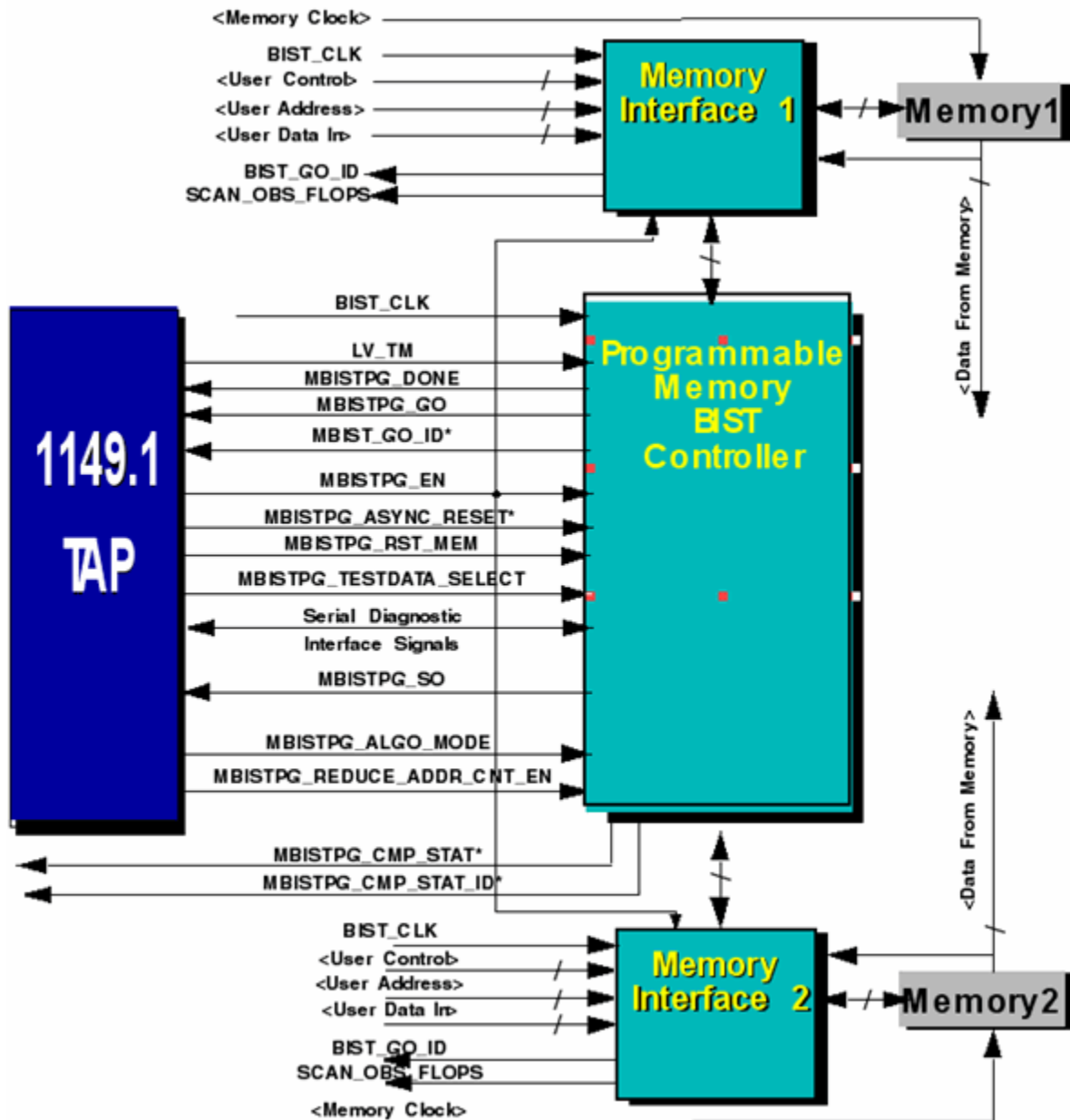
Note



For a description of the Programmable memory BIST assembly signals, refer to “[Memory Interface Signal Descriptions](#).”

A memory interface design object is instantiated for each memory. The simplified block diagram of BIST enabled memories in Figure 6-7 illustrates the basic Programmable memory BIST architecture.

Figure 6-7. High-Level View of the Programmable Memory BIST Architecture



The Tessent MemoryBIST Programmable memory BIST architecture includes the following and features:

- Fully configurable controller that applies a complete test to one or more embedded memories of different types and sizes

- Added functionality to support testing multi-port memories
- Simple controller interface allows access through a IEEE 1149.1 TAP or IEEE 1500 WTAP
- One or more user-programmable algorithms can be hard-coded into the controller to reduce controller size
- Support for a large number of library algorithms
- Run-time programmable algorithms:
- Supporting popular Order N march tests and many derivatives or variations of algorithms, such as March C, March Y, MATS++
- Supporting popular Order N and N^2 traditional test algorithms such as Moving Inversions, GalPat, GalRow, GalColumn, Walking ONEs, Walking ZEROs, Butterfly, Bit Surround Distrub
- Implementation of proprietary algorithms
- Support of true physical memory mapping for locating or detecting cell coupling faults
- Run-time programmable dynamic Auto Refresh or CBR Refresh:
- Refresh insertion during test execution
- Refresh memory to maintain data backgrounds during downloading of microcode for the next test or subtest
- Fully configurable memory interface timing supporting initialization sequences and sequential or interleave BurstModes.
- Run-time programmable address sequencing including support for address interleaving
- Diagnostic capabilities for debugging or bit mapping:
- Failure flag per bit for each compare cycle.
- At speed Stop-On-Nth-Error
- A default mode utilizing one or more hard-coded algorithms useful for production testing

The following sections describe in more detail components of the *Programmable* memory BIST architecture:

- [Top-Level Signals of Programmable Memory BIST](#)
- [Controller Design Object](#)
- [Memory Interface Design Object](#)

Top-Level Signals of Programmable Memory BIST

This section describes the signals on a programmable memory BIST controller and memory interfaces. Only actual top-level pin names are listed here.

Controller Signal Descriptions

All top-level signals for the programmable memory BIST controllers are listed in [Table 6-2](#) which includes the signal name, signal type (input or output), and description. Note that all signals are grouped into three categories:

- TAP/WTAP
- Status Signals
- Serial Interface Signals

Note



[Table 6-2](#) reflects information for a typical Programmable memory BIST controller. Extra signals generated for special configurations are not listed here. (For example, signals that control the memory interfaces.)

Table 6-2. Port List for the Programmable Memory BIST Controller

Signal Name	Type	Description
TAP/WTAP:		
BIST_CLK	I	Clock input. All inputs and outputs are sampled or asserted in the rising edge of this clock.
LV_TM	I	This signal is to be asserted during scan testing. The LV_TM signal must be de-asserted during BIST operation.
MBISTPG_EN	I	When asserted, this signal initiates the Programmable memory BIST controller in the mode specified by the BIST_SETUP signals.
MBISTPG_ASYNC_RESETN	I	An active-low input port to the Programmable memory BIST controller and memory interface that asynchronously resets all flip-flops in the BIST circuit.

Table 6-2. Port List for the Programmable Memory BIST Controller (cont.)

Signal Name	Type	Description
MBISTPG_REDUCED_ADDR_CNT_EN	I	MBISTPG_REDUCED_ADD_CNT_EN is an input to the Programmable memory BIST controller that enables the user to check the proper functionality of the BIST controller without having to simulate the test of the entire memory space. Using ETVerify you can generate a test bench for a reduced address simulation by specifying the ReducedAddressCount property to On in the ETVerify MembistPVerify wrapper. This signal is only created when a SMarchCHKB-based algorithm is used.
MBISTPG_MEM_RST	I	MBISTPG_MEM_RST is an input to the Programmable memory BIST controller that forces the BIST FSM to apply only the initialization portion of the SMarchCHKB-based algorithms to the memories. If this signal is driven high, the memory BIST controller writes 0 to each address location of each SRAM. Using ETVerify, you can activate the memory reset mode by specifying the MemoryReset property to On in the ETVerify MembistPVerify wrapper.
MBISTPG_ALGO_MODE	I	Inputs to the Programmable memory BIST controller to control the execution of the BIST run when parallel retention test is required. This port is generated when the ETPlanner ParallelRetentionTest property is set to On or PerGroup. Using ETVerify, you can create a testbench for parallel retention testing by specifying the ParallelRetentionTest property to On in the ETVerify MembistPVerify wrapper. This signal is only created when a SMarchCHKB-based algorithm is used.
MBISTPG_TESTDATA_SELECT	I	When this signal is asserted and when MBISTPG_EN is asserted and the BIST FSM block is in the IDLE or DONE state, the memory interface multiplexers will select the memory test signals. This signal is primarily used to maintain control of the memory interfaces during parallel retention test. This signal is also used to allow the controller to refresh the memory when scanning a test pattern into the memory BIST controller.

Table 6-2. Port List for the Programmable Memory BIST Controller (cont.)

Signal Name	Type	Description
MBISTPG_DIAG_EN	I	This input port used for diagnostics. It is generated when the ETPlanner CompStat property is set to SharedWithGo, or when one or more memory interfaces contain local comparators. This pin is sampled when the BIST_SETUP pins are in HWDefault mode. When asserted, the MBISTPG_GO signal becomes the CMP_STAT signal. Using ETVerify, you can create a test bench for diagnostic mode by specifying the Diagnostic property to On in the ETVerify MembistPVerify wrapper.
MBISTPG_CMP_STAT_ID_SEL	I	When asserted, this pin is used to select which controller comparator status is multiplexed onto the MBISTPG_GO port. This pin is sampled when the BIST_SETUP pins are in HWDefault mode and the BIST FSM block is in the BIST_SETUP1 state. Refer to “ BIST FSM Block ” and “ Pointer Control Block .”
MBISTPG_BIRA_EN	I	An input to the Programmable memory BIST controller that enables the redundancy analysis mode. This port is generated when at least one repairable memory is present. Using ETVerify, you can activate the redundancy analysis mode by specifying the CheckRepairStatus or ExtractRepairFuseMap property in the MembistPVerify wrapper.
MBIST_RA_PRSRV_FUSE_VAL	I	An input to the Programmable memory BIST controller and memory interface that preserves the values logged in the repair analysis registers. This port is generated when at least one memory implements repair analysis. Using ETVerify, you can activate the preservation mode by specifying the PreserveFuseRegisterValues property to On in the ETVerify MembistPVerify wrapper.

Table 6-2. Port List for the Programmable Memory BIST Controller (cont.)

Signal Name	Type	Description
MBISTPG_TEST_END_REF_ENABLE	I	This input pin is only created if there a DRAM in the design. When asserted, the Dynamic Refresh Control block is enabled while the BIST FSM block is in the IDLE or DONE states. This pin is sampled when the BIST_SETUP pins are in HWDefault mode, and the BIST FSM block is in the BIST_SETUP1 state. The MBISTPG_TESTDATA_SELECT pin must also be asserted. Refer to “ BIST FSM Block ” and “ Pointer Control Block .”
MBISTPG_RUN_TIME_REF_ENABLE	I	This input pin is only created if there a DRAM in the design. When asserted, the Dynamic Refresh Control block is enabled while the BIST FSM block is in the RUN state. This pin is sampled when the BIST_SETUP pins are in HWDefault mode and the BIST FSM block is in the BIST_SETUP1 state. Refer to “ BIST FSM Block ” and “ Pointer Control Block .”
MEM#_BIST_COLLAR_SO	I	An input to the Programmable memory BIST controller that connects the serial diagnostic chain from the memory interface(s) back to the controller. This port is generated when local comparators are used.
MEM#_BIST_COLLAR_SI	O	Output from the Programmable memory BIST controller that connects the serial diagnostic chain from the controller to the memory interface(s). This port is generated when local comparators are used.
Status Signals:		
MBISTPG_GO	O	Status signal when asserted (logic low) indicates a failure has occurred. This signal provides the composite pass/fail result of all comparators in the controller. If a fail occurs, this signal is driven low and remains low for the duration of the BIST execution. When the ETPlanner CompStat property is set to <i>SharedWithGo</i> , this output will carry the waveform of the global comparator status MBISTPG_CMP_STAT during diagnostic mode.
BIST_ON	O	Status signal when asserted indicates the system is in memory BIST mode.
MBISTPG_DONE	O	Status signal when asserted indicates the BIST FSM block has entered the DONE state and the algorithm has completed execution.

Table 6-2. Port List for the Programmable Memory BIST Controller (cont.)

Signal Name	Type	Description
MBISTPG_CMP_STAT	O	Status signal when asserted (logic low) indicates a failure has occurred. It is generated when the ETPlanner CompStat property is set to <i>Yes</i> . This signal provides the global comparator status during memory BIST. If any memory in the configuration fails, this signal falls for one cycle and then rises again. It continues to fall for one cycle for every additional memory failure.
<mem>_REPAIR_STATUS	O	Reports the overall repair status of a repairable memory. The status values are: 00 indicates no repair is required 01 indicates the memory is repairable 11 indicates the memory cannot be repaired
<mem>_<seg>_SCOL#_ALLOC_REG	O	Reports the allocation of a spare element for a memory with IO/column redundancy.
<mem>_<seg>_SCOL#_FUSE_REG	O	A group of outputs that reports the defective IO for a memory with IO/column redundancy.
<mem>_<seg>_SCOL#_FUSE_ADD_REG	O	A group of outputs that reports the defective address for a memory with IO/column redundancy.
<mem>_<seg>_SROW#_ALLOC_REG	O	Reports the allocation of a spare element for a memory with row redundancy.
<mem>_<seg>_SROW#_FUSE_ADD_REG	O	A group of outputs that reports the defective address for a memory with row redundancy.
Serial Interface Signals:		
TCK	I	Test clock used in the asynchronous TAP interface.
TCK_MODE	I	When asserted, the data shifted through the serial interface is performed at TCK clock rate. When de-asserted, indicates that data shifted through the serial interface is performed at BIST_CLK rate.
BIST_SI	I	Serial data input for initialization and scanning diagnostic information.
MBISTPG_SO	O	Serial data output for obtaining diagnostic information.
BIST_SHIFT	I	When asserted, this signal indicates that the serial interface is to begin shifting data in/out of the controller for initialization or diagnostics.

Table 6-2. Port List for the Programmable Memory BIST Controller (cont.)

Signal Name	Type	Description
BIST_HOLD	I	When asserted, the controller halts execution and internal registers maintain their state. Also asserted while the internal registers are scanned through the serial interface.
BIST_SETUP[1:0]	I	These signals determine the mode of the controller: <ul style="list-style-type: none">• ‘00’ — SHORT SETUP chain between BIST_SI and MBISTPG_SO• ‘01’ — LONG SETUP chain between BIST_SI and MBISTPG_SO• ‘10’ — HWDefault mode• ‘11’ — RunTimeProg mode

Memory Interface Signal Descriptions

All top-level signals for memory interfaces are listed in [Table 6-3](#) which includes the signal name, signal type (input or output), and description.

Note



[Table 6-3](#) provides a description of interface signals for a typical interface. Other signals might be created depending on the configuration and memory being tested (for example, multi-port memory and memories with row and column redundancy).

Table 6-3. Port List for Memory Interfaces

Signal Name	Type	Description
<User Control>	I	Functional mode memory control signals, such as chip enable or write enable.
<User Address>	I	Functional mode memory address signals.
<User Data In>	I	Functional mode memory input data signals.
<User Data Out>	I	Functional mode memory output data signals.
BIST_GO	O	Status signal when asserted (logic low) indicates a failure has occurred. This signal provides the pass/fail result of a memory tested using local comparators. This signal also carries the comparator status during diagnostic mode.

Table 6-3. Port List for Memory Interfaces (cont.)

Signal Name	Type	Description
< <i>BIST function</i> >	I	<p>These test control signals are used to perform transactions with the memory. The following is a typical list of available test signals:</p> <ul style="list-style-type: none"> • BIST_ACTIVATE • BIST_PRECHARGE • BIST_REFRESH • BIST_SELECT • BIST_OUTPUTENABLE • BIST_RAS • BIST_CAS • BIST_READENABLE • BIST_WRITEENABLE • BIST_USER0 • BIST_USER1 • BIST_USER2 • BIST_USER3 • BIST_USER4 • BIST_USER5 • BIST_USER6 • BIST_USER7
BIST_CMP	I	The Operation wrapper signal StrobeDataOut used to perform the compare between BIST_EXPECT_DATA and the data read from the memory.
BIST_DOUT_EN	I	The Operation wrapper signal Data used to control the output enable for bidirectional or tri-state data buses.
SCAN_OBS_FLOPS	O	Outputs of the observation flip-flops instantiated for address and control signals when ObservationLogic : Yes.
BIST_WRITE_DATA	I	A slice of the Write Data to be written to the memory. This signal is generated by the Data Generator Block of the controller. The number of bits in the BIST_WRITE_DATA signal is defined by the MemBistControllerOptions : NumberOfDataRegisterBits property of the ETPlanner configuration file.
BIST_EXPECT_DATA	I	Slice of the Expect Data to be compared with read data from the memory. This signal is generated by the Data Generator block of the controller. The number of bits in the BIST_EXPECT_DATA signal is defined by the MemBistControllerOptions : NumberOfDataRegisterBits property of the ETPlanner configuration file.

Table 6-3. Port List for Memory Interfaces (cont.)

Signal Name	Type	Description
BIST_BANK_ADD	I	Bank Address generated by the Address Generator block of the controller. The number of bits in the BIST_BANK_ADD signal is specified by the LogicalAddressMap wrapper of the memory library file.
BIST_ROW_ADD	I	Row Address generated by the Address Generator block of the controller. The number of bits in the BIST_ROW_ADD signal is specified by the LogicalAddressMap wrapper of the memory library file.
BIST_COL_ADD	I	Column Address generated by the Address Generator block of the controller. The number of bits in the BIST_COL_ADD signal is specified by the LogicalAddressMap wrapper of the memory library file.

Controller Design Object

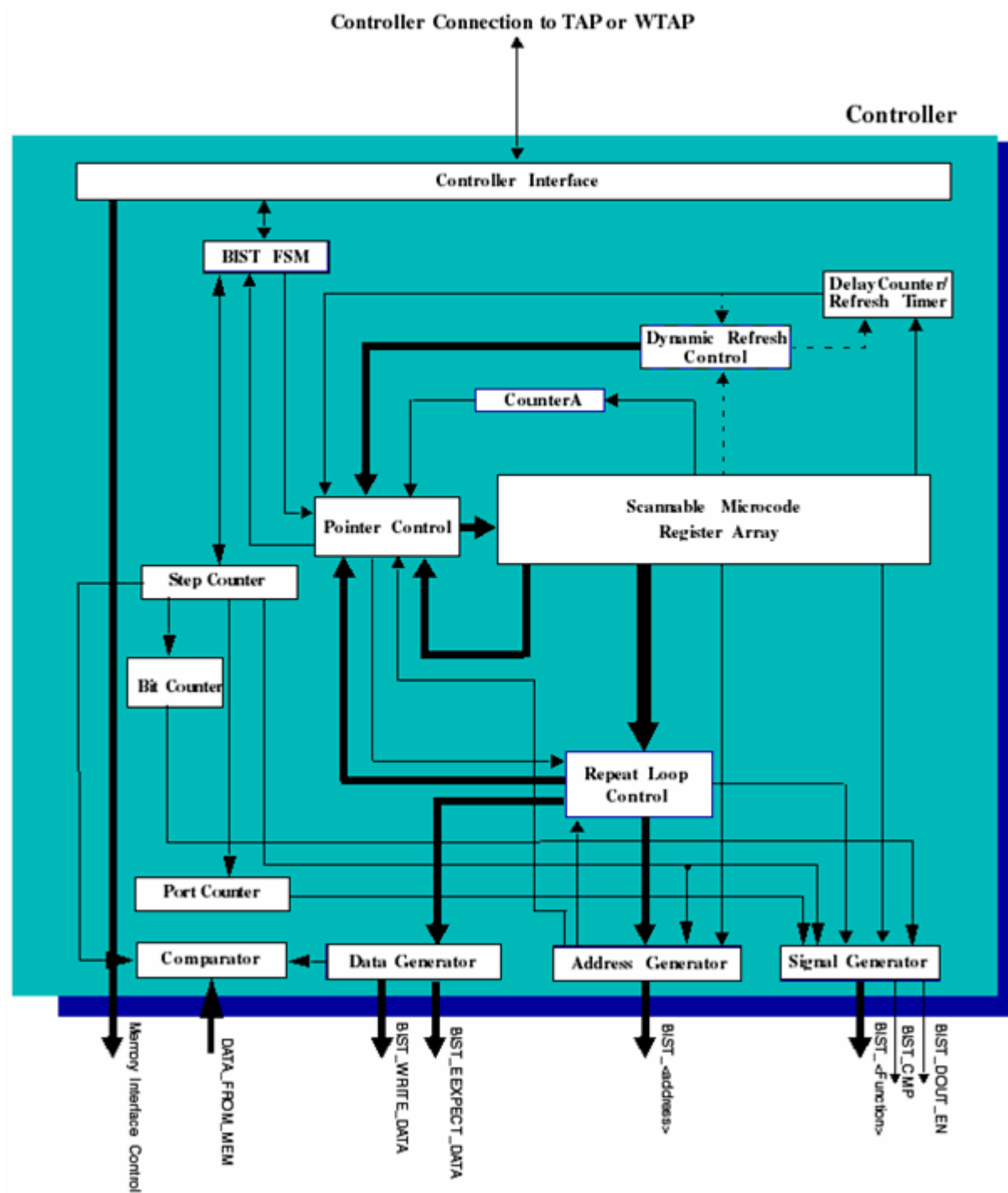
The architecture of the Programmable memory BIST controller consists of the following blocks:

- [Controller Interface Block](#)
- [BIST FSM Block](#)
- [Scannable Microcode Register Array Block](#)
- [Pointer Control Block](#)
- [Data Generator Block](#)
- [Address Generator Block](#)
- [Signal Generator Block](#)
- [Dynamic Refresh Control Block](#)
- [CounterA Block](#)
- [DelayCounter Block](#)
- [Repeat Loop Control Block](#)
- [Step Counter Block](#)
- [Port Counter](#)
- [Bit Counter](#)

- **Comparator**

Figure 6-8 illustrates the Programmable memory BIST architecture.

Figure 6-8. Programmable Controller Block Diagram



Controller Interface Block

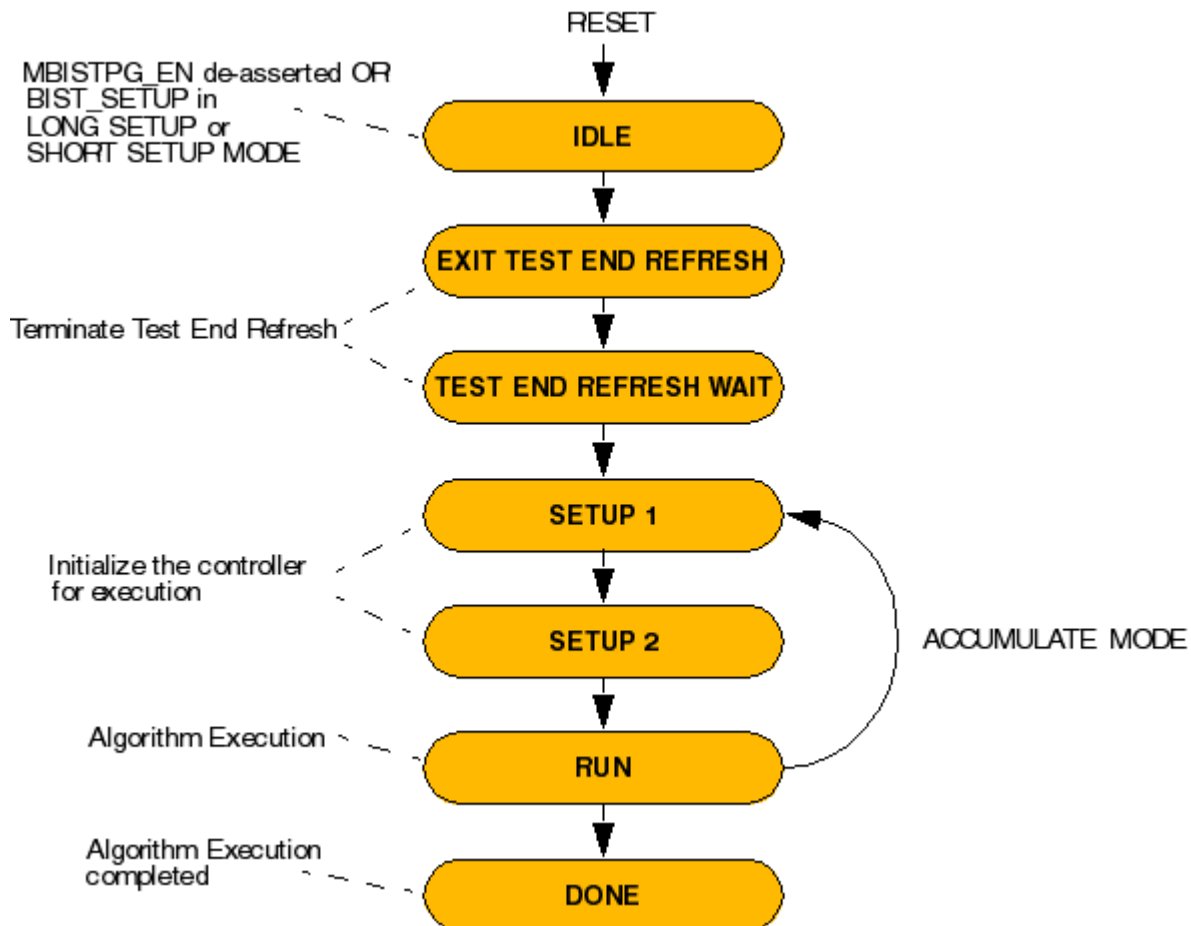
This block—Controller Interface—provides serial access through the IEEE 1149.1 TAP or WTAP. The Controller Interface performs the following tasks:

- Resynchronizes the BIST control signal to ensure a proper initialization and startup of the Programmable memory BIST controller.
- Provides a serial interface to the controller's internal configuration and diagnostic registers.

BIST FSM Block

This block—BIST Finite State Machine (FSM)—controls the initialization, setup, execution, and flags the completion of the programmed memory test. [Figure 6-9](#) illustrates the states of the BIST Finite State Machine.

Figure 6-9. BIST FSM States



Scannable Microcode Register Array Block

This block—Scannable Microcode Register Array—contains the scannable instruction registers for the algorithm to be executed. You can specify the number of instructions in the register array depending on your specific requirements and design budget. The number of instructions is specified in the ETPlanner configuration file with the [NumberOfInstructions](#) property of the ETPlanner configuration file.

The microcode instruction of the Programmable memory BIST provides parallel control of blocks, such as the Address Generator, Data Generator, Signal Generator, Repeat Loop Control, and Pointer Control block. This creates a wide but very flexible architecture for the generation of complex test algorithms. For a description of the instruction word bit fields, refer to “[ROM Interface for Programmable Controller](#).”

The instructions are addressed from the Pointer Control block. The instructions are arranged sequentially such that the first instruction for execution is always located at address zero. The second instruction is located at address one and so on.

When the `NumberOfInstructions` property is set to 0 (HardProgrammable memory BIST controller), the microcode register array will not be created. Instead a much smaller logic circuit is created to control the execution of the specified algorithm.

Pointer Control Block

This block—Pointer Control—determines the sequencing of the instructions in the Scannable Microcode Register Array.

The sequencing of instructions is determined by the currently executing instruction `NextConditions` bit field. Refer to Appendix A “[Bit Fields in Instruction Word](#).” The `NextConditions` bit field identifies the triggers to be tested. Based on the testing of these triggers, the Pointer Control determines which instruction is to be executed.

The decision tree for determining the next instruction to be executed is illustrated in [Figure 6-10](#).

Figure 6-10. Next Instruction Decision Tree

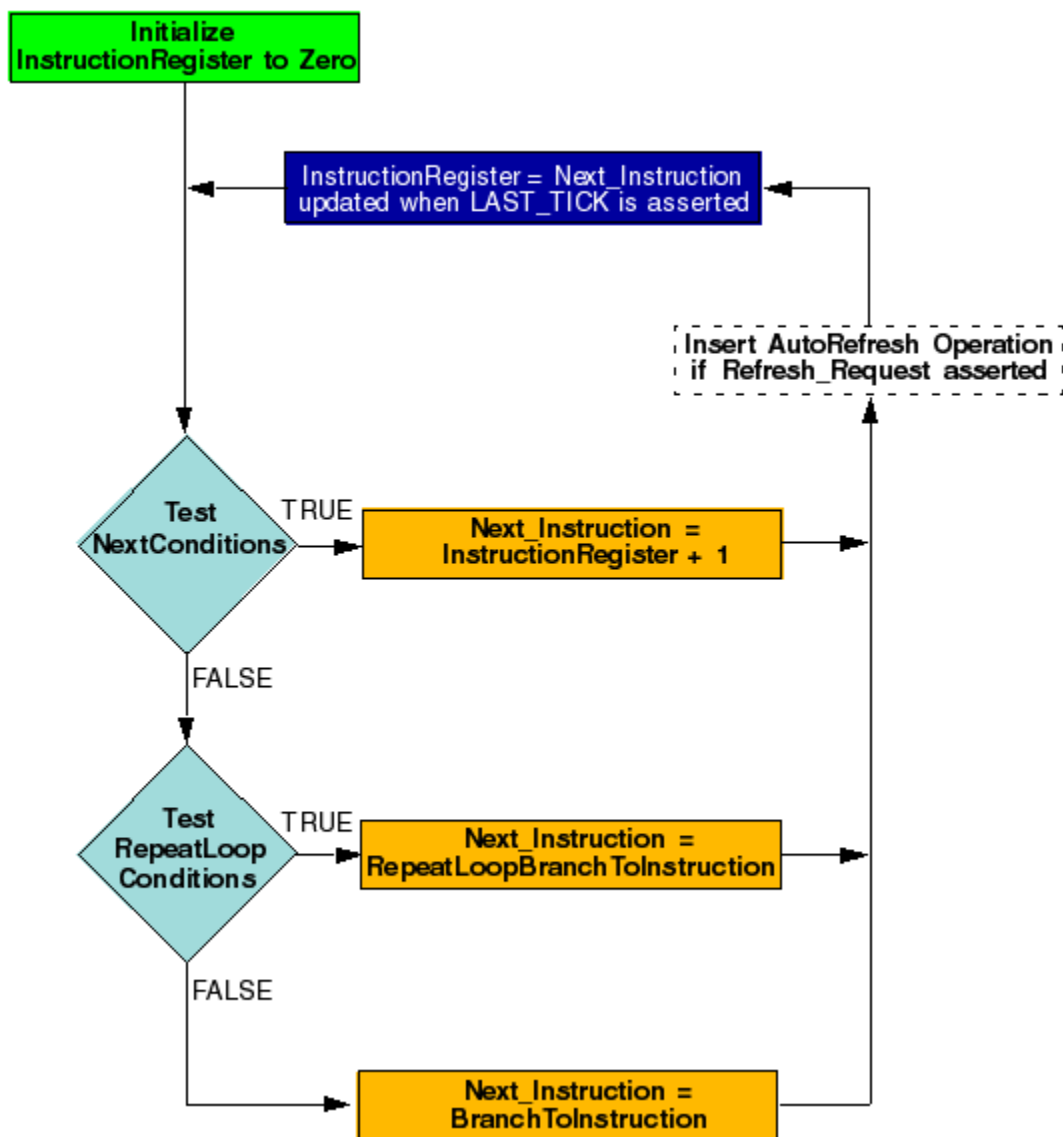


Table 6-4 summarizes what trigger is to be tested under the conditions when one of the properties in the NextConditions bit field specified to On.

Table 6-4. Triggers To Be Tested

When Next Conditions Property Specified to On	Trigger to Be Tested
RepeatLoopCmd (IncrementA, IncrementB, or IncrementBA)	RepeatLoopDone
DelayCounter_EndCount	DelayCounter_EndCount
CounterA_EndCount	CounterA_EndCount
Z_EndCount	Z_MinMax

Table 6-4. Triggers To Be Tested (cont.)

When Next Conditions Property Specified to On	Trigger to Be Tested
X1_EndCount	X1_MinMax
X0_EndCount	X0_MinMax
Y1_EndCount	Y1_MinMax
Y0_EndCount	Y0_MinMax

When the triggers are tested, and all of the triggers which are required to be tested are asserted, the result is TRUE. When the result is TRUE, the Pointer Control block selects the next sequential instruction in the Scannable Microcode Register Array for execution.

When the testing of the NextConditions is FALSE a subset of the NextConditions triggers are tested. This subset of NextConditions is called RepeatLoopConditions. The RepeatLoopConditions consists of all NextConditions except for the RepeatLoopDone trigger from the Repeat Loop Control block. When the RepeatLoopConditions are tested and the result is TRUE, the Pointer Control block selects the instruction identified by the RepeatLoopBranchToInstruction for execution.

When the testing of the RepeatLoopConditions is FALSE the Pointer Control block selects the instruction which identified by the BranchToInstruction bit field of the executing instruction as the next instruction for execution.

The Pointer Control block loads the new instruction specified for execution into the InstructionRegister when the LAST_TICK signal from the Signal Generator is asserted. The Instruction Register identifies the instruction currently executing.

Initially, when the controller is initialized for execution the first instruction located at address zero is loaded for execution.

The Pointer Control block asserts a signal LAST_STATE_DONE signal indicating that the last addressable instruction has completed execution with a TRUE test of the NextConditions.

Execution of AutoRefresh Operations for Dynamic Refresh

The Pointer Control block also inserts an AutoRefresh operation when the Refresh_Request signal is asserted and the LAST_TICK signal signifying that the Operation currently executing has completed. The next instruction for execution is not loaded until the Refresh operation has been performed, and the algorithm resumes.

Data Generator Block

This block—Data Generator—supplies write data and expect data for the memory or memories to be tested.

Data Generator Registers

The following registers are located in the Data Generator block:

- [WriteData Register](#)
- [ExpectData Register](#)
- [InvertDataWithRowBit Register](#)
- [InvertDataWithColumnBit Register](#)

WriteData Register

The WriteData register contains a user-defined pattern specified by the [DataGenerator: LoadWriteData](#) property in the Algorithm wrapper.

The number of bits in the WriteData register is defined by the ETPLanner [NumberOfDataRegisterBits](#) property.

This register changes its contents when the LAST_TICK signal is asserted from the [Signal Generator Block](#).

ExpectData Register

The ExpectData register contains a user-defined pattern specified by the [DataGenerator: LoadExpectData](#) property in the Algorithm wrapper.

The number of bits in the ExpectData Register is defined by the ETPLanner [NumberOfDataRegisterBits](#) property.

This register changes its contents when the LAST_TICK signal is asserted from the [Signal Generator Block](#).

InvertDataWithRowBit Register

The InvertDataWithRowBit register is present if one or more row address bits are specified in the [AddressCounter](#) wrapper of the memory library file.

The number of bits in the InvertDataWithRowBit register is the number of bits required to encode the number of row address bits.

This register is programmed by using the [DataGenerator: InvertDataWithRowBit](#) property in the Algorithm wrapper.

InvertDataWithColumnBit Register

The InvertDataWithColumnBit register is present if one or more column address bits are specified in the [AddressCounter](#) wrapper of the memory library file.

The number of bits in the InvertDataWithColumnBit register is the number of bits required to encode the number of column address bits.

This register is programmed by using the [DataGenerator: InvertDataWithColumnBit](#) property in the Algorithm wrapper.

Address Generator Block

This block—Address Generator block—controls the sequencing of the addresses applied to the memory or memories under test. The Address Generator block architecture provides runtime configurability and flexibility of the AddressRegister(A) and AddressRegister(B) address registers. Each address register consists of a Z, an X, and an Y component. The Z component is reserved for bank addresses, the X component is reserved for row addresses, and the Y component is reserved for column addresses. Each of the X and Y components can be further divided into two separate segments; namely an X0 and X1 segments from the X component and Y0 and Y1 segments from the Y component.

The Address Generator has the following features:

- Two address registers, namely AddressRegister(A) and AddressRegister(B), for tracking home and away cells. This is typically required for Order N^2 algorithms but is useful for storing start addresses for Order N algorithms.
- Each address register's X (Row) address may be segmented into X1 and X0 address segments:
 - Programmable number of bits in the X1 and X0 address registers
 - 2^N address stepping where N is the number of bits in the X0 row address segment
 - Can perform a test on any 2^N rows where N is the number of address bits in the X0 address segments
 - Independent increment, decrement, or hold commands to each of the X1 and X0 address segments
- Each address register's Y (Column) address may be segmented into Y1 and Y0 address segments:
 - Programmable number of bits in the Y1 and Y0 address registers
 - 2^N address stepping where N is the number of bits in the Y0 column address segment
 - Can perform a test on any 2^N rows where N is the number of address bits in the Y0 address segments
 - Independent increment, decrement, or hold commands to each of the Y1 and Y0 address segments

- Each address register may contain a Z(bank) address component.
- Rotate the A or B address registers.
- Apply XOR of A and B address registers.

Address Generator Registers

Two of each of the following registers are located in the Address Generator block, one for AddressRegister(A) and one for AddressRegister(B):

- [BankAddress Register](#)
- [RowAddress Register](#)
- [ColumnAddress Register](#)
- [ZCarryIn Register](#)
- [X1CarryIn Register](#)
- [X0CarryIn Register](#)
- [Y1CarryIn Register](#)
- [Y0CarryIn Register](#)
- [NumberX0Bits Register](#)
- [NumberY0Bits Register](#)

BankAddress Register

The BankAddress Register contains a user-defined pattern specified by the [AddressGenerator: AddressRegister: LoadBankAddress](#) property of the Algorithm wrapper.

The number of bits in the BankAddress Register is specified by the [LogicalAddressMap](#) wrapper of the memory library file.

A BankAddress Register is instantiated for both AddressRegister(A) and AddressRegister(B) if the number of bank address bits is greater than zero.

This register changes its contents when the LAST_TICK signal is asserted from the [Signal Generator Block](#).

RowAddress Register

The RowAddress Register contains a user-defined pattern specified by the [AddressGenerator: AddressRegister: LoadRowAddress](#) property of the Algorithm wrapper.

The number of bits in the RowAddress register is specified by the [LogicalAddressMap](#) wrapper of the memory library file.

A RowAddress Register is instantiated for both AddressRegister(A) and AddressRegister(B) if the number of row address bits is greater than zero.

This register changes its contents when the LAST_TICK signal is asserted from the [Signal Generator Block](#).

ColumnAddress Register

The ColumnAddress Register contains a user-defined pattern specified by the [AddressGenerator: AddressRegister: LoadColumnAddress](#) property of the Algorithm wrapper.

The number of bits in the ColumnAddress Register is specified by the [LogicalAddressMap](#) wrapper of the memory library file.

A ColumnAddress Register is instantiated for both AddressRegister(A) and AddressRegister(B) if the number of column address bits is greater than zero.

This register changes its contents when the LAST_TICK signal is asserted from the [Signal Generator Block](#).

ZCarryIn Register

The ZCarryIn Register identifies which address segments, if any, is to be used as a carry in. The ZCarryIn Register is programmed by the [AddressGenerator: AddressRegister: ZCarryIn](#) property of the Algorithm wrapper.

The number of bits in the ZCarryIn Register depends on the number of row address bits and column address bits specified by the [LogicalAddressMap](#) wrapper of the memory library file. One bit represents each of the X1, X0, Y1 and Y0 address segments.

A ZCarryIn Register is instantiated for both AddressRegister(A) and AddressRegister(B) if the number of bank address bits is greater than zero.

X1CarryIn Register

The X1CarryIn Register identifies which address segments, if any, is to be used as a carry in. The X1CarryIn Register is programmed by the [AddressGenerator: AddressRegister: X1CarryIn](#) property of the Algorithm wrapper.

The number of bits in the X1CarryIn register depends on the number of bank address bits, row address bits and column address bits specified by the [LogicalAddressMap](#) wrapper of the memory library file. One bit represents each of the Z, X0, Y1 and Y0 address segments.

A X1CarryIn Register is instantiated for both AddressRegister(A) and AddressRegister(B) if the number of row address bits is greater than zero.

X0CarryIn Register

The X0CarryIn Register identifies which address segments, if any, is to be used as a carry in. The X0CarryIn Register is programmed by the [AddressGenerator: AddressRegister: X0CarryIn](#) property of the Algorithm wrapper.

The number of bits in the X0CarryIn Register depends on the number of bank address bits, row address bits and column address bits specified by the [LogicalAddressMap](#) wrapper of the memory library file. One bit represents each of the Z, X1, Y1 and Y0 address segments.

A X0CarryIn Register is instantiated for both AddressRegister(A) and AddressRegister(B) if the smallest integer of the following is greater than zero:

- The number of row address bits minus one.
- n is the [NumberX0Bits](#) with a count range of $[0:(2n-1)]$. The row address count range for the X0 address segment must be a full binary count from all zeros to all ones.

Y1CarryIn Register

The Y1CarryIn Register identifies which address segments, if any, is to be used as a carry in. The Y1CarryIn Register is programmed by the [AddressGenerator: AddressRegister: Y1CarryIn](#) property of the Algorithm wrapper.

The number of bits in the Y1CarryIn Register depends on the number of bank address bits, row address bits and column address bits specified by the [LogicalAddressMap](#) wrapper of the memory library file. One bit represents each of the Z, X1, X0 and Y0 address segments.

A Y1CarryIn Register is instantiated for both AddressRegister(A) and AddressRegister(B) if the number of column address bits is greater than zero.

Y0CarryIn Register

The Y0CarryIn Register identifies which address segments, if any, is to be used as a carry in. The Y0CarryIn Register is programmed by the [AddressGenerator: AddressRegister: Y0CarryIn](#) property of the Algorithm wrapper.

The number of bits in the Y0CarryIn Register depends on the number of bank address bits, row address bits and column address bits specified by the [LogicalAddressMap](#) wrapper of the memory library file. One bit represents each of the Z, X1, X0 and Y1 address segments.

A Y0CarryIn Register is instantiated for both AddressRegister(A) and AddressRegister(B) if the smallest integer of the following is greater than zero:

- The number of column address bits minus one.

- n is the [NumberY0Bits](#) with a count range of $[0:(2n-1)]$. The column address count range for the Y0 address segment must be a full binary count from all zeros to all ones.

NumberX0Bits Register

The NumberX0Bits Register defines the number of bits in the X0 address segment of the X address register. The NumberX0Bits register is programmed by the [AddressGenerator: AddressRegister: NumberX0Bits](#) property of the Algorithm wrapper.

The number of bits in the NumberX0Bits Register is the number of bits required to encode the smallest integer of the following:

- The number of row address bits minus one.
- n is the [NumberX0Bits](#) with a count range of $[0:(2n-1)]$. The row address count range for the X0 address segment must be a full binary count from all zeros to all ones.

A NumberX0Bits Register is instantiated for both AddressRegister(A) and AddressRegister(B) if the number of bits in the register is greater than zero.

NumberY0Bits Register

The NumberY0Bits Register defines the number of bits in the Y0 address segment of the Y address register. The NumberY0Bits Register is programmed by the [AddressGenerator: AddressRegister: NumberY0Bits](#) property of the Algorithm wrapper.

The number of bits in the NumberY0Bits Register is the number of bits required to encode the smallest integer of the following:

- The number of column address bits minus one.
- n is the [NumberY0Bits](#) with a count range of $[0:(2n-1)]$. The column address count range for the Y0 address segment must be a full binary count from all zeros to all ones.

A NumberY0Bits Register is instantiated for both AddressRegister(A) and AddressRegister(B) if the number of bits in the register is greater than zero.

Signal Generator Block

This block—Signal Generator—is used to create the waveforms defined in the [OperationSet](#) Family wrapper.

These waveforms or operations represent transactions performed on the memory for initialization, writing, or reading. All [Operations](#) are hardwired into the Programmable memory BIST controller in groups of [OperationSet](#).

The Signal Generator also generates internal signals used by the BIST design objects. One of these internal signals is the LAST_TICK signal that is asserted on the last clock cycle of the

selected operation. When asserted, the signal `LAST_TICK` performs two functions in the controller. First, the contents of the Address or Data Registers might change as a result of the command issued to the respective blocks. Registers that are updated on the assertion of the `LAST_TICK` signal are identified. The second function of the `LAST_TICK` signal is in the [Pointer Control Block](#) which loads the next instruction to be executed.

The Signal Generator has the following capabilities:

- Controls signal pipelines
- Forces address bits to a specified value on a specified clock cycle
- Controls signals decoded with BIST address

Signal Generator Registers

The `OperationSetSelect` Registers are located in the Signal Generator block.

OperationSetSelect Register

The `OperationSetSelect` Register identifies the [OperationSet](#) from which the Operations are selected for execution. Only one `OperationSet` can be selected per algorithm. The `OperationSetSelect` register is programmed by the [TestRegisterSetup: OperationSetSelect](#) property of the Algorithm wrapper.

The number of bits in the `OperationSetSelect` Register is the number of bits required to encode the number of `OperationSets` defined for the controller.

Dynamic Refresh Control Block

This block—Dynamic Refresh Control—is instantiated only if the memory type is DRAM as specified by the [MemoryTemplate: MemoryType](#) property of the memory library file.

The Dynamic Refresh Control block provides runtime dynamic refresh as well as test end dynamic refresh of backgrounds between test steps or sub-tests.

Dynamic Refresh Control Registers

The following registers are located in the Dynamic Refresh Control block:

- `RunTimeRefreshEnable`
- `TestEndRefreshEnable`
- `TestEndRefreshInterval`

RunTimeRefreshEnable Register

The RunTimeRefreshEnable Register specifies whether the Dynamic Refresh Control block is enabled or disabled. This register is programmed with the [TestStep: Controller: RunTimeRefreshEnable](#) property of the ETVerify configuration file.

The number of bits in the RunTimeRefreshEnable Register is one.

TestEndRefreshEnable Register

The TestEndRefreshEnable Register specifies whether the Dynamic Refresh Control block is enabled or disabled. This register is programmed with the [TestStep: Controller: TestEndRefreshEnable](#) property of the ETVerify configuration file.

The number of bits in the TestEndRefreshEnable register is one.

TestEndRefreshInterval Register

The TestEndRefreshInterval Register specifies whether the refresh interval used by the Delay Counter block to generate the DelayCounter_EndCount trigger. This register is programmed with the [TestStep: Controller: TestEndRefreshInterval](#) property of the ETVerify configuration file.

The number of bits in the TestEndRefreshEnable Register is equivalent to the number of bits specified by the ETPlanner [NumberOfDelayCounterBits](#) property.

CounterA Block

This block—CounterA—is a general use counter. A counter endcount value is programmed into the CounterA_EndCount Register. The counter is instructed to increment with the CounterACmd bit field from the executing instruction. When the contents of the CounterA block is equivalent to the CounterA_EndCount register, the CounterA_EndCount trigger is generated and used by the Pointer Control block.

The CounterA block is reset to zeros in following cases:

- Prior to the execution of the algorithm.
- CounterACmd is increment and the CounterA_EndCount trigger is asserted.

CounterA_EndCounter Register

The CounterA_EndCount Register is the only register located in the CounterA block. This register contains a desired maximum count value programmed with the [TestRegisterSetup: LoadCounterA_EndCount](#) property in the Algorithm wrapper.

The number of bits in the CounterA_EndCount register is defined by the ETPlanner [NumberOfCounterABits](#) property.

This register changes its contents when the LAST_TICK signal is asserted from the [Signal Generator Block](#).

DelayCounter Block

This block—DelayCounter—can be used as a general use counter (typically to insert pauses), or as a Refresh Timer.

When used as a general purpose counter, the end count value is programmed into the DelayCounter_EndCount register. The counter is instructed to increment with the DelayCounterCmd bit field from the executing instruction. When the contents of the DelayCounter block is equivalent to the DelayCounter_EndCount register, the DelayCounter_EndCount trigger is generated and used by the Pointer Control block.

The DelayCounter is used as a Refresh Timer when the [TestStep: Controller: RunTimeRefreshEnable](#) property or the TestStep: Controller: [TestEndRefreshEnable](#) property of the ETVerify configuration file is set to On.

The DelayCounter block resets to zeros under the following conditions:

- Prior to the execution of the algorithm.
- DelayCounterCmd is increment, and the DelayCounter_EndCount trigger is asserted.

DelayCounter_EndCount Register

The DelayCounter_EndCount register is the only register located in the DelayCounter block. This register contains a desired maximum count value programmed with the [TestRegisterSetup: DelayCounterEndCount](#) property of the Algorithm wrapper.

When the DelayCounter is used as a Refresh Timer the value loaded into the DelayCounter register is a function of the BIST_CLK period and the value specified by the [TestStep: Controller: RunTimeRefreshInterval](#) property or the TestStep: Controller: [TestEndRefreshInterval](#) property of the ETVerify configuration file.

The number of bits in the DelayCounter_EndCount register is defined by the ETPlanner [NumberOfDelayCounterBits](#) property.

This register changes its contents when the LAST_TICK signal is asserted from the [Signal Generator Block](#).

When the DelayCounter is used as a Refresh Timer, the counter changes its contents on every other BIST_CLK.

Repeat Loop Control Block

This block—Repeat Loop Control—is used to re-execute a single instruction or a group of sequential instructions with the following modified instruction bit fields:

- Address segment commands:
 - ZAddressCmd
 - X1AddressCmd
 - X0AddressCmd
 - Y1AddressCmd
 - Y0AddressCmd
- WriteDataCmd
- ExpectDataCmd
- InhibitLastAddressCount
- InhibitDataCompare

The architecture of the Repeat Loop Control block consists of two repeat loops—RepeatLoop(A) and RepeatLoop(B)—with the following features or capabilities:

- Each repeat loop is capable of re-executing an instruction or sequential group of instructions up to three times.
- For each Repeat, the AddressSequence, WriteDataSequence, ExpectDataSequence, InhibitLastAddressCount, and InhibitDataCompare can be modified.
- RepeatLoop(A) and RepeatLoop(B) can be nested or used independently.
- Each of the repeat loops, RepeatLoop(A) and RepeatLoop(B), can be used only once per algorithm.

RepeatLoop Control Registers

Two of the following register sets are located in the Repeat Loop Control block—one set of registers for RepeatLoop(A) and another set of registers for RepeatLoop(B):

- [BranchToInstruction Register](#)
- [RepeatNumber Register](#)
- [Repeat1 Register](#)
- [Repeat2 Register](#)
- [Repeat3 Register](#)

BranchToInstruction Register

The BranchToInstruction register specifies the address of the branch instruction to be loaded for execution by the [Pointer Control Block](#).

The number of bits in the BranchToInstruction bit field is the number of bits required to encode on the number of instructions in the microcode as specified by the ETPlanner [NumberOfInstructions](#) property.

RepeatNumber Register

The RepeatNumber register contains the number of the [Repeat](#) wrappers specified for the [RepeatLoop](#) wrapper within the Algorithm wrapper. The number of Repeat wrappers indicates the number of re-executions of the instruction or group of instructions to be re-executed.

Repeat1 Register

The Repeat1 register contains the data in the first sequential [Repeat](#) wrapper which modifies the following instruction bit fields for all instructions in the repeated group of instructions:

- Address segment commands:
 - ZAddressCmd
 - X1AddressCmd
 - X0AddressCmd
 - Y1AddressCmd
 - Y0AddressCmd
- WriteDataCmd
- ExpectDataCmd
- InhibitLastAddressCount
- InhibitDataCompare

The modifications to the Address segment commands ZAddressCmd, X1AddressCmd, X0AddressCmd, Y1AddressCmd, Y0AddressCmd are accomplished using the [Repeat: AddressSequence](#) property of the Algorithm wrapper.

The modifications to the WriteDataCmd command are accomplished using the [Repeat: WriteDataSequence](#) property of the Algorithm wrapper.

The modifications to the ExpectDataCmd command are accomplished using the [Repeat: ExpectDataSequence](#) property within the Algorithm wrapper.

The modifications to the InhibitLastAddressCount command are accomplished using the [Repeat: InhibitLastAddressCount](#) property within the Algorithm wrapper.

The modifications to the InhibitDataCompare command are accomplished using the [Repeat: InhibitDataCompare](#) property within the Algorithm wrapper.

The bit assignments to the Repeat1 register are illustrated in [Table 6-5](#).

Table 6-5. Repeat1 Register Bit Field Assignment

Repeat1 Register	Bit Assignments
ExpectDataSequence	4
WriteDataSequence	3
AddressSequence	2
InhibitLastAddressCount	1
InhibitDataCompare	0

Repeat2 Register

The Repeat2 register is the same as the Repeat1 register. However, if specified, the Repeat2 register contains the data in the second sequential [Repeat](#) wrapper within the Algorithm wrapper if specified.

Repeat3 Register

The Repeat3 register is the same as the Repeat1 register. However, if specified, the Repeat3 register contains the data in the third sequential [Repeat](#) wrapper within the Algorithm wrapper if specified.

RepeatLoop Types

Since [RepeatLoop\(A\)](#) and [RepeatLoop\(B\)](#) can be used only once, each per algorithm only two types of loops may be created. These RepeatLoop types are as follows:

- [Independent RepeatLoop Type](#)
- [Nested RepeatLoop Type A](#)
- [Nested RepeatLoop Type B](#)

Independent RepeatLoop Type

The first RepeatLoop type that can be created is called an independent looping type. The characteristics of the independent RepeatLoop type are as follows:

- An instruction specifies only one [RepeatLoop](#).

- The group of sequential instructions from the instruction specifying the RepeatLoop to the [BranchToInstruction](#) specified for the RepeatLoop define the instructions to be repeated. Within this group of instructions the other RepeatLoop is not specified.

The independent looping type is shown in [Figure 6-11](#). Instruction2 specifies the RepeatLoop(A). The instruction(s) to be repeated is from the instruction which specifies RepeatLoop(A) to the instruction specified by RepeatLoop(A): BranchToInstruction. The group of instructions from Instruction0 to Instruction2 are to be repeated twice since two Repeat wrappers are specified for RepeatLoop(A). A second independent looping type is also specified in Instruction3. The instruction(s) to be repeated is from the instruction which specifies RepeatLoop(B) to the instruction specified by RepeatLoop(B): BranchToInstruction. The instruction Instruction3 specifies the instructions to be repeated. This instruction is to be repeated three times since three Repeat wrappers are specified for RepeatLoop(B).

Note that RepeatLoop(A) and RepeatLoop(B) can be interchanged.

The sequence of instruction execution is as follows with the Repeat Register that modifies the instructions indicated:

- Instruction0
- Instruction1
- Instruction2
 - RepeatLoop(A):BranchToInstruction:Instruction0
- Instruction0 - RepeatLoop(A):Repeat1
- Instruction1 - RepeatLoop(A):Repeat1
- Instruction2 - RepeatLoop(A):Repeat1
 - RepeatLoop(A):BranchToInstruction:Instruction0
- Instruction0 - RepeatLoop(A):Repeat2
- Instruction1 - RepeatLoop(A):Repeat2
- Instruction2 - RepeatLoop(A):Repeat2
 - RepeatLoopDone is asserted
- Instruction3
 - RepeatLoop(B):BranchToInstruction:Instruction3
- Instruction3 - RepeatLoop(B):Repeat1
 - RepeatLoop(B):BranchToInstruction:Instruction3
- Instruction3 - RepeatLoop(B):Repeat2

- RepeatLoop(B):BranchToInstruction:Instruction3
- Instruction3 - RepeatLoop(B):Repeat3
 - RepeatLoopDone is asserted

Figure 6-11. Independent RepeatLoop Type Example

```

MicroProgram {
  Instruction (Instruction0) {
    .
    .
    .
    NextConditions {
    } //end of NextConditions wrapper
  } //end of Instruction wrapper
  Instruction (Instruction1) {
    .
    .
    .
    NextConditions {
    } //end of NextConditions wrapper
  } //end of Instruction wrapper
  Instruction (Instruction2) {
    .
    .
    .
    NextConditions {
      .
      .
      .
      RepeatLoop (A) {
        BranchToInstruction: Instruction0;
        Repeat {
          .
          .
          .
        } //end of Repeat wrapper
        Repeat {
          .
          .
          .
        } //end of Repeat wrapper
      } //end of RepeatLoop(A) wrapper
    } //end of NextConditions wrapper
  } //end of Instruction wrapper
  Instruction (Instruction3) {
    .
    .
    .
    NextConditions {
      .
      .
      .
      RepeatLoop (B) {
        BranchToInstruction: Instruction3;
        Repeat {

```



```

        .
        .
        .
    } //end of Repeat wrapper
    Repeat {
        .
        .
        .
    } //end of Repeat wrapper
    Repeat {
        .
        .
        .
    } //end of Repeat wrapper
    } //end of RepeatLoop(B) wrapper
    } //end of NextConditions wrapper
    } //end of Instruction wrapper
} //end of MicroProgram wrapper

```

Nested RepeatLoop Type A

The second type of [RepeatLoop](#) that can be created is called a nested looping type. There are two sub-types of Nested RepeatLoop loops—Type A and Type B.

The Nested type A RepeatLoop has the following characteristics:

- An instruction specifies only one RepeatLoop.
- The group of sequential instructions from the instruction specifying the RepeatLoop to the BranchToInstruction specified for the RepeatLoop define the instructions to be repeated.
- The other RepeatLoop is defined in another instruction where the group of instructions specified in the first RepeatLoop are a subset of the instructions in the second RepeatLoop.

The Nested Type A looping type is shown in [Figure 6-12](#). [Instruction2](#) specifies the RepeatLoop(A). The instruction(s) to be repeated is from the instruction which specifies RepeatLoop(A) to the instruction specified by RepeatLoop(A): [BranchToInstruction](#). The group of instructions from Instruction0 to Instruction2 are to be repeated once since one [Repeat](#) wrapper is specified for RepeatLoop(A). A second RepeatLoop is also specified in Instruction3. The instruction(s) to be repeated is from the instruction which specifies RepeatLoop(B) to the instruction specified by RepeatLoop(B): BranchToInstruction. The group of instructions from Instruction0 to Instruction3 are to be repeated once since one Repeat wrapper is specified for RepeatLoop(B).

Note that RepeatLoop(A) and RepeatLoop(B) can be interchanged.

The sequence of instruction execution is as follows with the Repeat register that modifies the instructions indicated:

- Instruction0
- Instruction1
- Instruction2
 - RepeatLoop(A):BranchToInstruction:Instruction0
- Instruction0 - RepeatLoop(A):Repeat1
- Instruction1 - RepeatLoop(A):Repeat1
- Instruction2 - RepeatLoop(A):Repeat1
 - RepeatLoopDone is asserted
- Instruction3
 - RepeatLoop(B):BranchToInstruction:Instruction0
- Instruction0 - RepeatLoop(B):Repeat1
- Instruction1 - RepeatLoop(B):Repeat1
- Instruction2 - RepeatLoop(B):Repeat1
 - RepeatLoop(A):BranchToInstruction:Instruction0
- Instruction0 - RepeatLoop(B):Repeat1 - RepeatLoop(A):Repeat1
- Instruction1 - RepeatLoop(B):Repeat1 - RepeatLoop(A):Repeat1
- Instruction2 - RepeatLoop(B):Repeat1 - RepeatLoop(A):Repeat1
 - RepeatLoopDone is asserted
- Instruction3 - RepeatLoop(B):Repeat1
 - RepeatLoopDone is asserted

Figure 6-12. Nested-type A RepeatLoop Example

```

MicroProgram {
    Instruction (Instruction0) {
        .
        .
        .
        NextConditions {
        } //end of NextConditions wrapper
    } //end of Instruction wrapper
    Instruction (Instruction1) {
        .
        .
        .
        NextConditions {

```

```

    } //end of NextConditions wrapper
} //end of Instruction wrapper
Instruction (Instruction2) {
    .
    .
    .
    NextConditions {
        .
        .
        .
        RepeatLoop (A) {
            BranchToInstruction: Instruction0;
            Repeat {
                .
                .
                .
            } //end of Repeat wrapper
        } //end of RepeatLoop(A) wrapper
    } //end of NextConditions wrapper
} //end of Instruction wrapper
Instruction (Instruction3) {
    .
    .
    .
    NextConditions {
        .
        .
        .
        RepeatLoop (B) {
            BranchToInstruction: Instruction0;
            Repeat {
                .
                .
                .
            } //end of Repeat wrapper
        } //end of RepeatLoop(B) wrapper
    } //end of NextConditions wrapper
} //end of Instruction wrapper
} //end of MicroProgram wrapper

```

Nested RepeatLoop Type B

The Nested type B RepeatLoop has the following characteristics:

- An instruction specifies both RepeatLoop(A) and RepeatLoop(B).
- The loop defined by RepeatLoop(A) must always be nested in the loop defined by RepeatLoop(B). Specifically, the group of sequential instructions from the instruction specifying RepeatLoop(A) to the [RepeatLoop\(A\):BranchToInstruction](#) must be a subset of the group of sequential instructions from the instruction specifying RepeatLoop(B) to the RepeatLoop(B): BranchToInstruction.

The Nested Type looping type is shown in [Figure 6-13](#). Instruction2 specifies the RepeatLoop(A). The instruction(s) to be repeated is from the instruction which specifies RepeatLoop(A) to the instruction specified by RepeatLoop(A): BranchToInstruction. The group

of instructions from Instruction1 to Instruction2 are to be repeated once since one [Repeat](#) wrapper is specified for RepeatLoop(A). A second RepeatLoop is also specified in Instruction2. The instruction(s) to be repeated is from the instruction which specifies RepeatLoop(B) to the instruction specified by RepeatLoop(B): BranchToInstruction. The group of instructions from Instruction0 to Instruction2 are to be repeated twice since two Repeat wrappers are specified for RepeatLoop(B).

Note



RepeatLoop(A) and RepeatLoop(B) cannot be interchanged.

The sequence of instruction execution is as follows with the Repeat register that modifies the instructions indicated:

- Instruction0
- Instruction1
- Instruction2
 - RepeatLoop(A):BranchToInstruction:Instruction1
- Instruction1 - RepeatLoop(A):Repeat1
- Instruction2 - RepeatLoop(A):Repeat1
 - RepeatLoop(B): BrunchToInstruction: Instuction0
- Instruction0 - RepeatLoop(B): Repeat1
- Instruction1 - RepeatLoop(B):Repeat1
- Instruction2 - RepeatLoop(B):Repeat1
 - RepeatLoop(B): BrunchToInstruction: Instuction1
- Instruction1 - RepeatLoop(B):Repeat1 - RepeatLoop(A):Repeat1
- Instruction2 - RepeatLoop(B):Repeat1 - RepeatLoop(A):Repeat1
 - RepeatLoop(B):BranchToInstruction:Instruction0
- Instruction0 - RepeatLoop(B): Repeat2
- Instruction1 - RepeatLoop(B):Repeat2
- Instruction2 - RepeatLoop(B):Repeat2
 - RepeatLoop(A): BrunchToInstruction: Instuction1
- Instruction1 - RepeatLoop(B):Repeat2 - RepeatLoop(A):Repeat1
- Instruction2 - RepeatLoop(B):Repeat2 - RepeatLoop(A):Repeat1

- o RepeatLoopDone is asserted

Figure 6-13. Nested-type B RepeatLoop Example

```

MicroProgram {
  Instruction (Instruction0) {
    .
    .
    .
    NextConditions {
    } //end of NextConditions wrapper
  } //end of Instruction wrapper
  Instruction (Instruction1) {
    .
    .
    .
    NextConditions {
    } //end of NextConditions wrapper
  } //end of Instruction wrapper
  Instruction (Instruction2) {
    .
    .
    .
    NextConditions {
    .
    .
    .
    RepeatLoop (A) {
      BranchToInstruction: Instruction1;
      Repeat {
        .
        .
        .
      } //end of Repeat wrapper
    } //end of RepeatLoop(A) wrapper
    RepeatLoop (B) {
      BranchToInstruction: Instruction0;
      Repeat {
        .
        .
        .
      } //end of Repeat wrapper
    } //end of RepeatLoop(B) wrapper
  } //end of NextConditions wrapper
} //end of Instruction wrapper
} //end of MicroProgram wrapper

```

Step Counter Block

This module counts and controls the sequencing of memories when there are multiple steps in the configuration file.

Port Counter

This module counts and controls the sequencing of the various ports of multi-port memories.

Bit Counter

This module counts and controls the sequencing of the test data to a word-slice as specified in the configuration file.

Comparator

This module is created when shared comparators is used. It contains the comparator(s) used for checking the actual data from the memory against the expected data generated by the memory BIST controller. Note that parts of this module might reside in the memory interface when local comparators are used.

Memory Interface Design Object

The second design object of the Programmable memory BIST is the memory interface. The memory interface provides the interface between the controller design object and the memory. A memory interface design object is instantiated for each memory.

The memory interface provides such functions as:

- Expands BIST_WRITE_DATA and BIST_EXPECT_DATA to the width of the memory data bus. [Figure 6-14](#) illustrates an 8-bit BIST_WRITE_DATA bus expanded onto a 20-bit memory data bus.
- Performs logical to physical data mapping when the [TestRegisterSetup: DataPolarityEnable](#) property of the Algorithm wrapper is set to On. Refer as well to “[PhysicalDataMap](#)” of the memory library file in the *ETAssemble Tool Reference*.
- Performs logical to physical address mapping. Refer to “[PhysicalDataMap](#)” of the memory library file in the *ETAssemble Tool Reference*.
- Compares expected and data read from the memory when local comparators are used.

A block diagram of a memory interface design object is illustrated in [Figure 6-15](#).

Figure 6-14. Expanding Write and Expect Data onto the Memory Data Bus

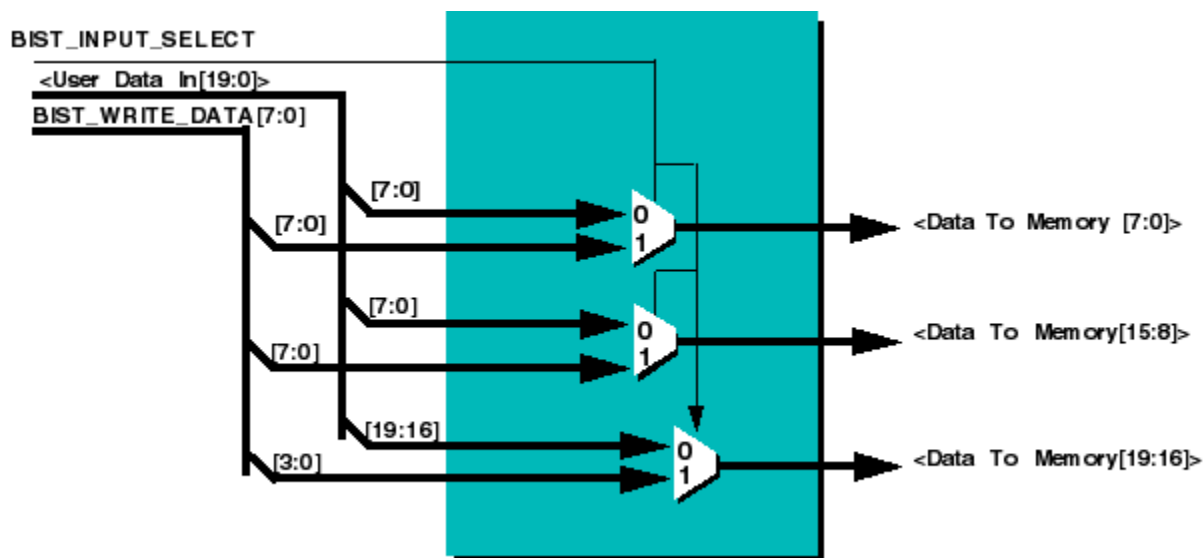
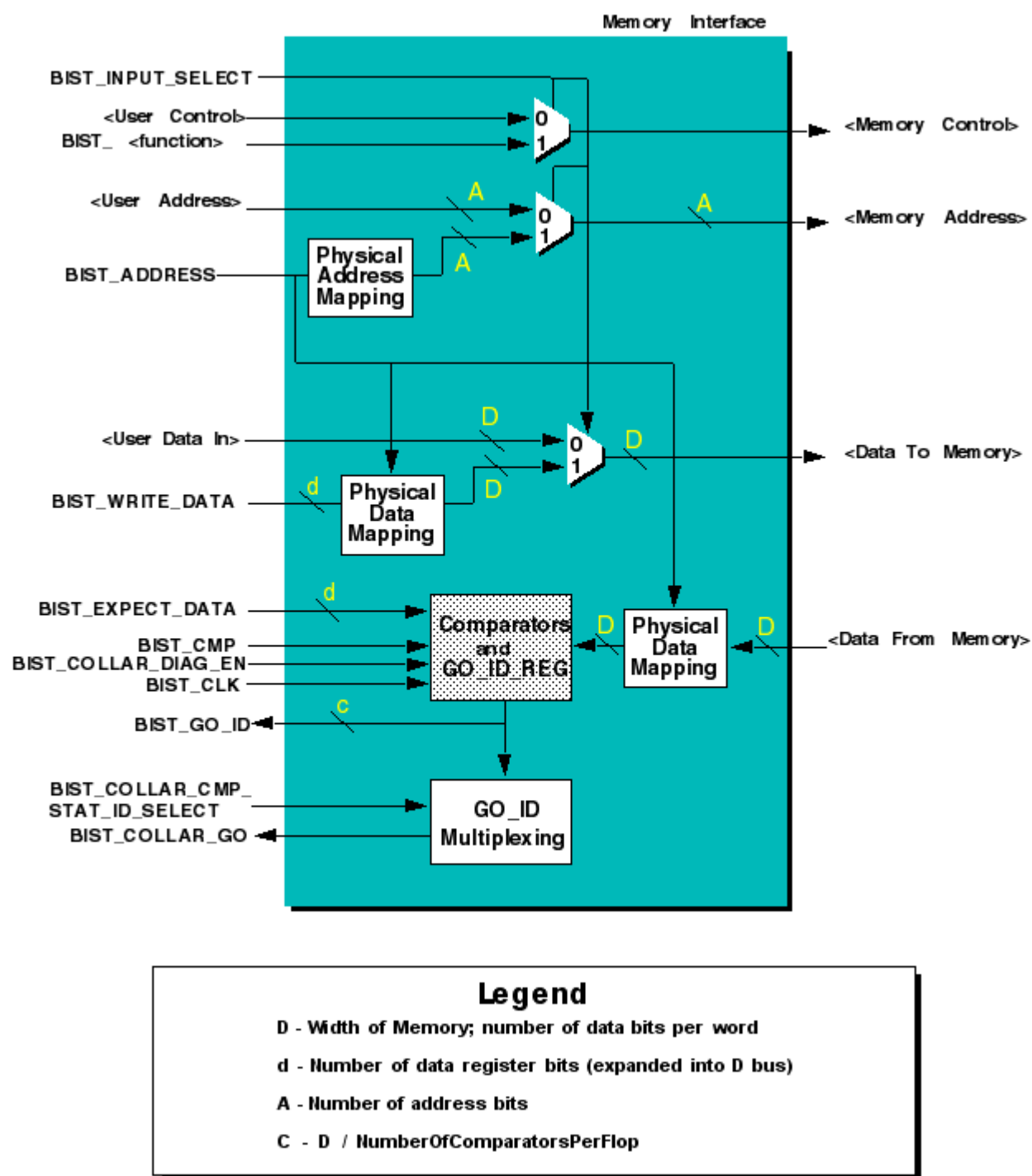


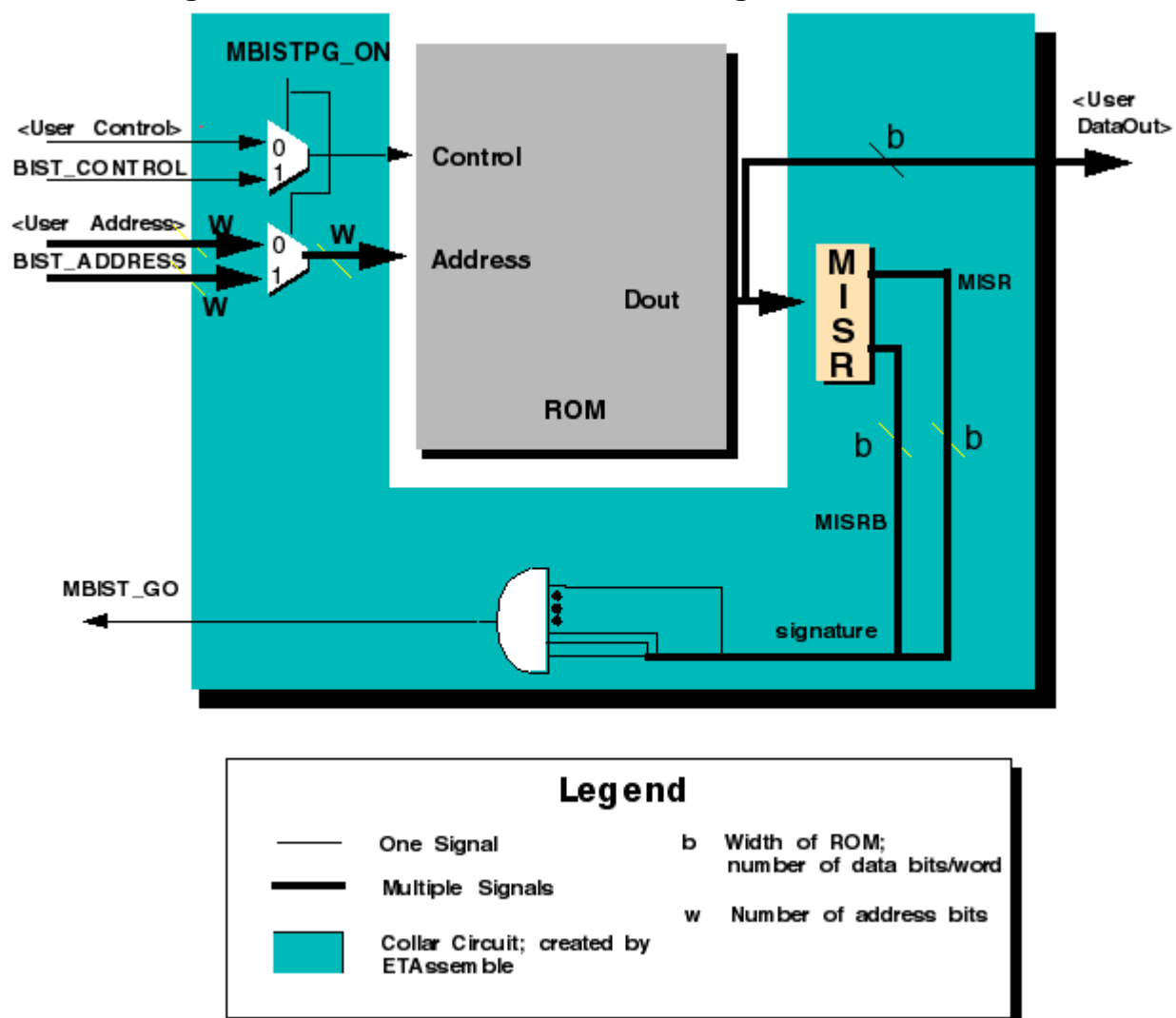
Figure 6-15. Memory Interface Block Diagram



ROM Interface for Programmable Controller

Figure 6-16 shows how a ROM is connected to the Programmable interface.

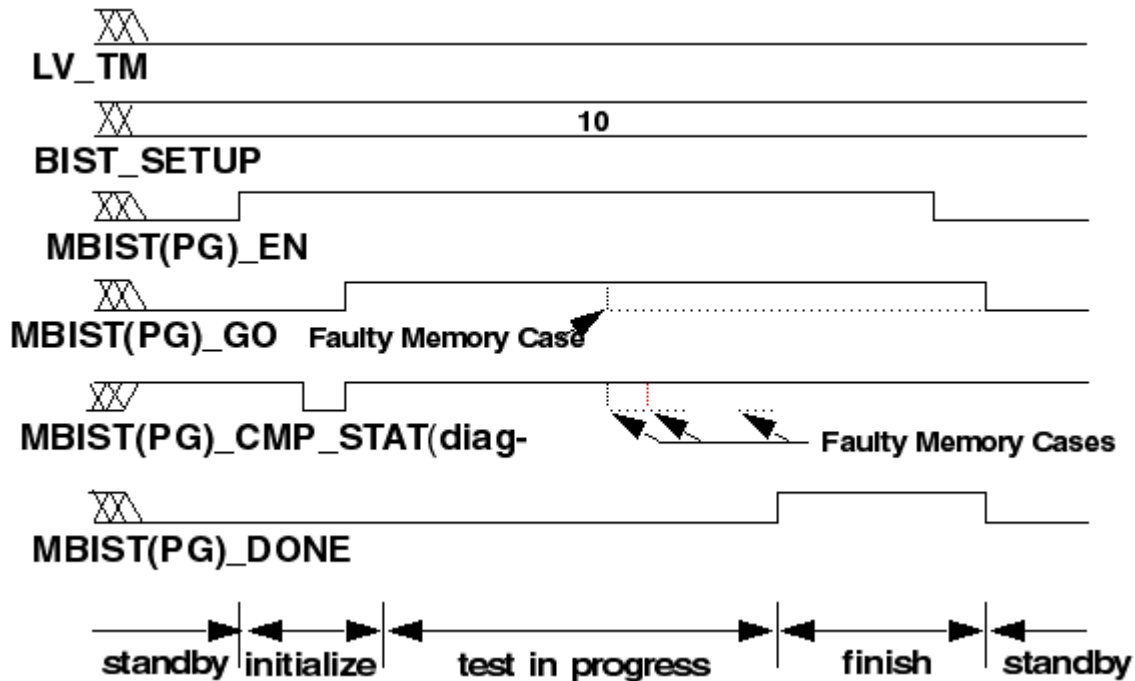
Figure 6-16. ROM Connection to the Programmable Interface



BLST Operating Protocol

The memory test is initiated when MBIST(PG)_EN is high and BIST_SETUP has the binary value 10. The first falling edge of MBIST(PG)_CMP_STAT after the rising edge of MBIST(PG)_EN marks the clock cycle before the beginning of the test. If you want to know what the algorithm is doing at any given time (For example, what was the controller trying to do when the memory failed), you can compare the clock cycle number (relative to the starting edge) to the clock cycle report in the ETAssemble log file. If an error is discovered, the MBIST(PG)_CMP_STAT signal goes low for the next cycle and then returns to high. MBIST(PG)_GO will fall and remain low for the remainder of the test. The rising edge of MBIST(PG)_DONE marks the end of the test. If MBIST(PG)_GO is still high at that time, all memories have passed the test. [Figure 6-17](#) illustrates the BIST protocol.

Figure 6-17. The BIST Operating Protocol



It is not necessary to continue the test if the MBIST(PG)_GO signal has fallen. If you are monitoring the MBIST(PG)_CMP_STAT signals, you may wish to continue to find more memory failures. You may also want to ensure that MBIST(PG)_DONE rises at the appropriate time to rule out a manufacturing defect within the memory BIST circuitry. Whether a failure occurs or not, once the MBIST(PG)_DONE rises, MBIST(PG)_EN can be de-asserted to terminate the test and return to functional mode. LV_TM, when driven high, is used to setup the collar and memory BIST controller for scan test or logic BIST.

When a TAP/WTAP controller is present on the chip, a specific instruction bit can be used to initiate the memory BIST. In addition, status bits of the TAP/WTAP's instruction register can be used to sample the MBIST(PG)_GO and MBIST(PG)_DONE signals.

Using Local Comparators

This section describes how and when comparators can be used within the memory collar/interface.

The logic used to compare embedded memory responses with expected values can be located in one of two places:

- Contained within the memory BIST controller
- Placed locally into the memory collar

Consider, for example, the controller and three collared memories depicted in [Figure 6-18](#) on page 212. The comparator logic for eRAM i and eRAM j is placed within the memory BIST controller while the comparator logic for eRAM k is placed locally in the memory's collar. There are advantages and disadvantages to both approaches, as described below.

Comparators Within the Controller

One advantage of having the comparators inside the controller is that they can be shared among memories that are tested in different controller steps; that is, memories tested sequentially. For example, in [Figure 6-18](#), eRAM i and eRAM j share a common set of comparators.

Another related advantage (not shown in [Figure 6-18](#)) is that a common set of individual comparator status signals (CMP_STAT_ID) can be routed to chip pins for bit-level diagnosis.

The main disadvantage to having the comparators placed within the controller is the required routing between the collars and the controller. For wide memories, this overhead can be significant.

Comparators Within the Memory Collar/Interface

The main advantage to local comparators is that the routing overhead between the collars/interfaces and controller when comparators are placed in the controller is essentially eliminated. As shown in [Figure 6-18](#), for eRAM k, only a single signal needs to be routed back to the controller.

The disadvantage to having comparators within a memory collar/interface is that comparators cannot be shared across memories tested in different steps. Thus, increasing the area overhead of the collar/interface.

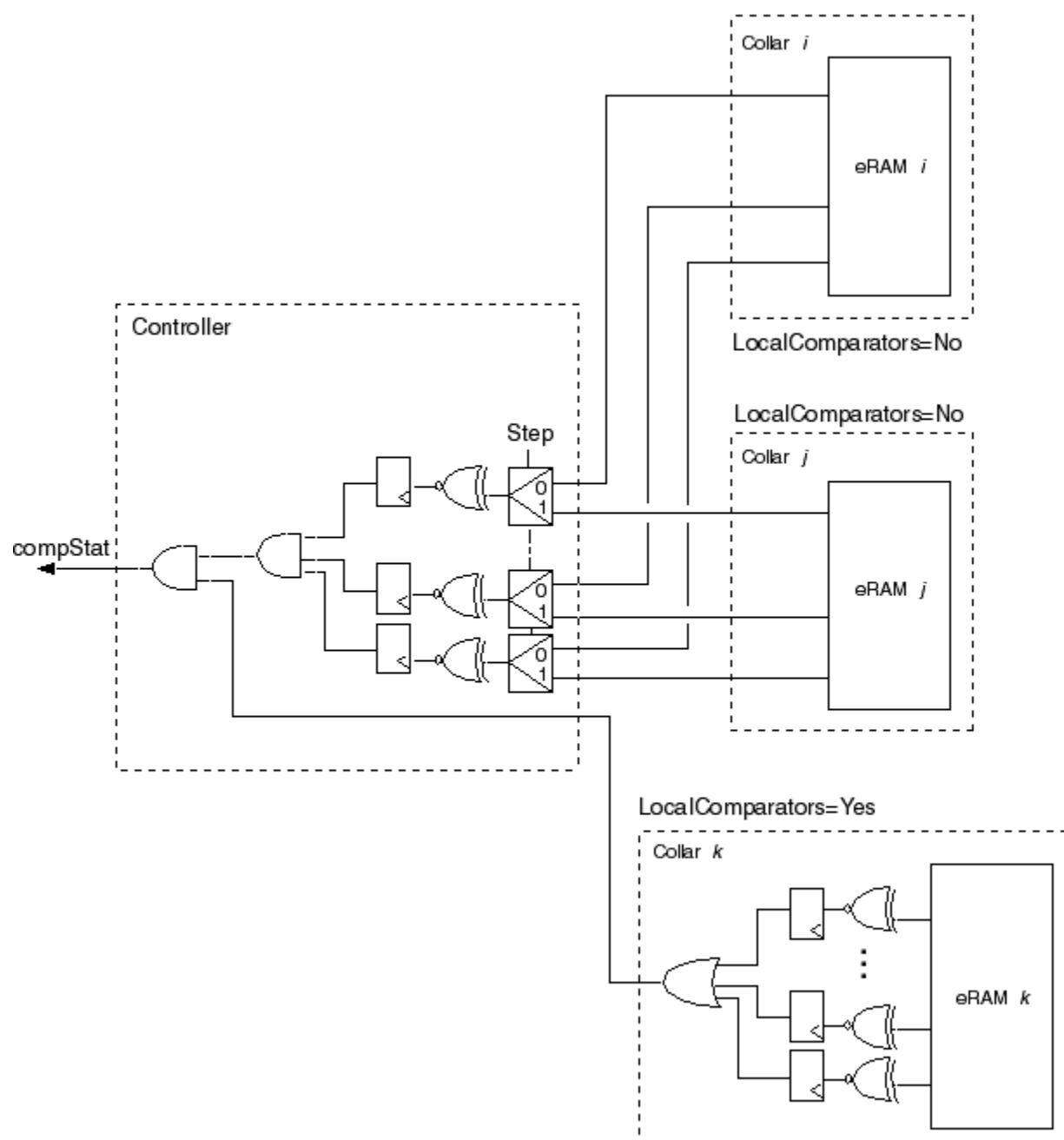
Syntax for Local Comparators

To place local comparators within a given memory collar, set the [LocalComparators](#) property to Yes in the corresponding wrapper:

- [MemBistStepOptions](#) wrapper of the [.etplan File](#)
- [MemoryBist](#) wrapper of the [.ETDefaults File](#)

[Figure 6-18](#) shows the comparator logic within the memory BIST controller and the comparator logic local to a memory's collar.

Figure 6-18. Shared Comparators and Local Comparators



Appendix A

Bit Fields in Instruction Word

This appendix describes the bit fields in a single instruction word.

This appendix covers the following topics

- [Instruction Word Bit Field Descriptions](#)
 - [OperationSelect Bit Field](#)
 - [Add_Reg_A_Equals_B Bit Field](#)
 - [Y0AddressCmd Bit Field](#)
 - [Y1AddressCmd Bit Field](#)
 - [X0AddressCmd Bit Field](#)
 - [X1AddressCmd Bit Field](#)
 - [ZAddressCmd Bit Field](#)
 - [AddressSelectCmd Bit Field](#)
 - [WriteDataCmd Bit Field](#)
 - [ExpectDataCmd Bit Field](#)
 - [RepeatLoop Bit Field](#)
 - [InhibitLastAddressCount Bit Field](#)
 - [InhibitDataCompare Bit Field](#)
 - [InhibitRefresh Bit Field](#)
 - [CounterACmd Bit Field](#)
 - [DelayCounterCmd Bit Field](#)
 - [DisableMemoriesWithoutGroupWriteEnable](#)
 - [DisableMemoriesWithoutOutputEnable](#)
 - [DisableMemoriesWithoutReadEnable](#)
 - [DisableMemoriesWithoutSelect](#)
 - [DisableMemoriesWithoutWriteEnable](#)

- [BranchToInstruction Bit Field](#)
- [NextConditions Bit Field](#)

Instruction Word Bit Field Descriptions

The scannable micro-code register array is composed of several instruction registers. As with any other micro-controller, the instruction register is divided into several fields. Each field is composed of one or more bits that have a specific function encoded into a micro-code.

The register array is made scannable to enable loading of different instructions through the TAP/WTAP. The size of the registers is determined at controller generation time based on the user input. The content of the different bit fields is calculated and shifted into the register using the ETVerify tool.

When you set [NumberOfInstructions](#) to 0, the register array is not created. Instead, smaller hardware is created to run the specific algorithm sequences that you have defined.

This section describes the bit fields in the instruction register. The purpose is to enable you to debug algorithm execution and to provide insight into the relationship of the controller architecture to the micro-code.

[Table A-1](#) provides the instruction register mapping and compositional sequence starting from instruction bit 0.

Table A-1. Instruction Register Field

# Bits in the Field	Field Description
3+	OperationSelect
2	<i>Add_Reg_A_Equals_B</i>
2	<i>Y0AddressCmd</i>
3	<i>Y1AddressCmd</i>
2	<i>X0AddressCmd</i>
3	<i>X1AddressCmd</i>
2	<i>ZAddressCmd</i>
3	<i>AddressSelectCmd</i>
4	<i>WriteDataCmd</i>
4	<i>ExpectDataCmd</i>
2	RepeatLoop
1	InhibitlastAddressCount
1	InhibitDataCompare

Table A-1. Instruction Register Field (cont.)

# Bits in the Field	Field Description
1	InhibitRefresh
1	CounterACmd
1	DelayCounterCmd
3+	BranchToInstruction
1	DisableMemoriesWithoutSelect
1	DisableMemoriesWithoutOutputEnable
1	DisableMemoriesWithoutReadEnable
1	DisableMemoriesWithoutWriteEnable
1	DisableMemoriesWithoutGroupWriteEnable
6-8	NextConditions

The number of bits in an instruction word is dependent on several factors. Each section defines the number of bits required per bit field and indicates when the bit or bit fields are present. Each of these bit fields is described in the following sections.

OperationSelect Bit Field

The OperationSelect bit field identifies the memory transaction or operation to be performed by the [Signal Generator Block](#) to the memory or memories.

The number of bits in the OperationSelect field is dependent on the number of operations specified in the memory library file's [OperationSet](#) wrapper. The operation set containing the largest number of operations determines the number of bits in the OperationSet bit field of the instruction register.

[Table A-2](#) illustrates the resulting number of bits in the OperationSelect bit field as a function of the OperationSet wrapper.

Table A-2. Number of Bits in the OperationSelect bit field

Number of <i>Operations</i> in the <i>OperationSet</i> with the most <i>Operations</i>	Number of Bits in the OperationSelect bit field
5-8	3
9-16	4
17-32	5
33-64	6

Add_Reg_A_Equals_B Bit Field

The A_Add_Reg_Equals_B bit field of the instruction word inverts the write data and/or the expect data when the value in [AddressRegister\(A\)](#) is equal to the value in [AddressRegister\(B\)](#).

The A_Add_Reg_Equals_B field is two bits wide in the instruction word and is decoded as shown in [Table A-3](#).

Table A-3. A_Add_Reg_Equals_B Bit Field Decode

Property Value	Bit Decode	Instruction Description
Off	00	Any AutoRefresh operations inserted by the Dynamic Refresh Control Block are performed normally.
Invert_WriteData	01	Inverts the Write Data applied to the memory data bus.
Invert_ExpectData	10	Inverts the Expect Data to be compared with the memory read data.
Invert_Write Data_and_ExpectData	11	Inverts both the Write Data applied to the memory data bus and the Expect Data to be compared with the memory read data.

Y0AddressCmd Bit Field

The Y0AddressCmd bit field specifies the command to be performed on the Y0 address segment of the selected Address Register. The command specified either Holds, Increments, or Decrements.

The Y0AddressCmd bit field is a two-bit field of the instruction word. This bit field is present only if more than one column address bit is specified in the [AddressCounter](#) wrapper of the [Memory Library File](#).

The two-bit command is decoded as shown in [Table A-4](#).

Table A-4. Y0AddressCmd Bit Field Decode

Property Value	Bit Decode	Instruction Description
Hold	00	Y0 address segment of the selected AddressRegister holds the current value.
Increment	10	Y0 address segment of the selected AddressRegister increments on the last tick of the executing operation.
Decrement	11	Y0 address segment of the selected AddressRegister decrements on the last tick of the executing operation.

Table A-4. Y0AddressCmd Bit Field Decode (cont.)

Property Value	Bit Decode	Instruction Description
-	01	Reserved

Y1AddressCmd Bit Field

The Y1AddressCmd bit field specifies the command to be performed on the Y1 address segment of the selected Address Register. The command specifies either Holds, Increments, Decrements, LoadMin, or LoadMax.

The Y1AddressCmd bit field is a two-bit field of the instruction word. This bit field is present only if any column address bits are specified in the [AddressCounter](#) wrapper of the [Memory Library File](#).

The two-bit command is decoded as shown in [Table A-5](#).

Table A-5. Y1AddressCmd Bit Field Decode

Property Value	Bit Decode	Instruction Description
Hold	000	Y1 address segment of the selected AddressRegister holds the current value.
Increment	010	Y1 address segment of the selected AddressRegister increments on the last tick of the executing operation.
Decrement	011	Y1 address segment of the selected AddressRegister decrements on the last tick of the executing operation.
LoadMin	100	Load Minimum Address to Y1 Segment of the selected AddressRegister.
LoadMax	101	Load Maximum address to the Y1 Segment of the selected AddressRegister
-	001, 110, 111,	Reserved

X0AddressCmd Bit Field

The X0AddressCmd bit field specifies the command to be performed on the X0 address segment of the selected Address Register. The command specifies either Holds, Increments, or Decrements.

The X0AddressCmd bit field is a two-bit field of the instruction word. This bit field is present only if more than one row address bit is specified in the [AddressCounter](#) wrapper of the [Memory Library File](#).

The two-bit command is decoded as shown in [Table A-6](#).

Table A-6. X0AddressCmd Bit Field Decode

Property Value	Bit Decode	Instruction Description
Hold	00	X0 address segment of the selected AddressRegister holds the current value.
Increment	10	X0 address segment of the selected AddressRegister increments on the last tick of the executing operation.
Decrement	11	X0 address segment of the selected AddressRegister decrements on the last tick of the executing operation.
-	01	Reserved

X1AddressCmd Bit Field

The X1AddressCmd bit field specifies the command to be performed on the X1 address segment of the selected Address Register. The command specifies either Holds, Increments, Decrements, LoadMin or LoadMax.

The X1AddressCmd bit field is a two-bit field of the instruction word. This bit field is present only if any row address bits are specified in the [AddressCounter](#) wrapper of the [Memory Library File](#)

The two-bit command is decoded as shown in [Table A-7](#).

Table A-7. X1AddressCmd Bit Field Decode

Property Value	Bit Decode	Instruction Description
Hold	000	X1 address segment of the selected AddressRegister holds the current value
Increment	010	X1 address segment of the selected AddressRegister increments on the last tick of the executing operation.
Decrement	011	X1 address segment of the selected AddressRegister decrements on the last tick of the executing operation.
LoadMin	100	Load Minimum Address to X1 Segment of the selected AddressRegister.
LoadMax	101	Load Maximum Address to X1 Segment of the selected AddressRegister.
-	001, 110, 111	Reserved

ZAddressCmd Bit Field

The ZAddressCmd bit field specifies the command to be performed on the Z address segment of the selected Address Register. The command specifies either Holds, Increments, or Decrements.

The ZAddressCmd bit field is a two-bit field of the instruction word. This bit field is present only if any bank address bits are specified in the [AddressCounter](#) wrapper of the [Memory Library File](#).

The two-bit command is decoded as shown in [Table A-8](#).

Table A-8. ZAddressCmd Bit Field Decode

Property Value	Bit Decode	Instruction Description
Hold	00	Z address segment of the selected AddressRegister holds the current value.
Increment	10	Z address segment of the selected AddressRegister increments on the last tick of the executing operation.
Decrement	11	Z address segment of the selected AddressRegister decrements on the last tick of the executing operation.
-	01	Reserved

AddressSelectCmd Bit Field

The AddressSelectCmd bit field specifies the AddressRegister in the Address Generator block to be selected. The selected AddressRegister contains the address to be applied to the memory or memories under test. The selected AddressRegister also specifies that AddressRegister is affected by the following address segment commands:

- ZAddressCmd
- X1AddressCmd
- X0AddressCmd
- Y1AddressCmd
- Y0AddressCmd

The AddressSelectCmd bit field comprises three bits in the instruction register. The three-bit command is decoded as shown in [Table A-9](#).

Table A-9. AddressSelectCmd Bit Field Programming

Property Value	Bit Decode	Instruction Description
Select_A	000	Apply AddressRegister(A) to the memory address bus and execute the address segment commands on AddressRegister(A).
Select_A_Copy_To_B	001	Apply AddressRegister(A) to the memory address bus and execute the address segment commands on AddressRegister(A). Copy the contents of AddressRegister(A) to AddressRegister(B) on the last tick of the executing operation.
Select_B	010	Apply AddressRegister(B) to the memory address bus and execute the address segment commands on AddressRegister(B).
Select_B_Copy_To_A	011	Apply AddressRegister(B) to the memory address bus and execute the address segment commands on AddressRegister(B). Copy the contents of AddressRegister(B) to AddressRegister(A) on the last tick of the executing operation.
A_XOR_B	100	Apply AddressRegister(A) XOR AddressRegister(B) to the memory address bus and execute the address segment commands on AddressRegister(A).
Select_A_RotateLeft_B	101	Apply AddressRegister(A) to the memory address bus and execute the address segment commands on AddressRegister(A). Rotate the contents of AddressRegister(B) one bit to the left on the last tick of the executing operation.
Select_B_RotateLeft_A	110	Apply AddressRegister(B) to the memory address bus and execute the address segment commands on AddressRegister(B). Rotate the contents of AddressRegister(A) one bit to the left on the last tick of the executing operation.
Select_B_RotatRight_A	111	Apply AddressRegister(B) to the memory address bus and execute the address segment commands on AddressRegister(B). Rotate the contents of AddressRegister(A) one bit to the right on the last tick of the executing operation.

WriteDataCmd Bit Field

The WriteDataCmd bit field specifies the data applied to the memory data bus to be written to the memory or memories. The WriteDataCmd field is a four-bit field in the instruction word.

The four-bit WriteDataCmd field is decoded as shown in [Table A-10](#).

Table A-10. WriteDataCmd Bit Field Decode

Property Value	Bit Decode	Instruction Description
DataReg	0000	Apply the data in the Write Data register to the memory or memories. Allow data inversion with Address bits.
InverseDataReg	0001	Apply the inverse data of the Write Data register to the memory or memories. Allow data inversion with Address bits.
Zero	0010	Apply Write Data of Zeros to the memory or memories. Allow data inversion with Address bits.
One	0011	Apply Write Data of Ones to the memory or memories. Allow data inversion with Address bits.
DataReg_Rotate	0100	Apply the data in the Write Data register to the memory or memories followed by a Rotate Left of the Write Data Register on the last tick of the executing operation. Allow data inversion with Address bits.
InverseDataReg_Rotate	0101	Apply the inverse data in the Write Data register to the memory or memories followed by a Rotate Left of the Write Data register on the last tick of the executing operation. Allow data inversion with Address bits.
DataReg_RotateWithInvert	0110	Apply the data in the Write Data register to the memory or memories followed by a Rotate Left with inverted feedback of the Write Data register on the last tick of the executing operation. Allow data inversion with Address bits.

Table A-10. WriteDataCmd Bit Field Decode (cont.)

Property Value	Bit Decode	Instruction Description
InverseDataReg_RotateWithInvert	0111	Apply the inverse data in the Write Data register to the memory or memories followed by a Rotate Left with inverted feedback of the Write Data register on the last tick of the executing operation. Allow data inversion with Address bits.
DataReg_PRDG	1100	Apply the data in the Write Data register to the memory or memories followed by the generation of a new pseudo-random data pattern on the last tick of the executing operation. Allow data inversion with Address bits.
InverseDataReg_PRDG	1101	Apply the inverse data in the Write Data register to the memory or memories followed by the generation of a new pseudo-random data pattern on the last tick of the executing operation. Allow data inversion with Address bits.
Set_DataReg	1000	Apply Write Data of Ones to the memory or memories. Do not allow data inversion with Address bits.
Reset_DataReg	1001	Apply Write Data of Zeros to the memory or memories. No not allow data inversion with Address bits.

ExpectDataCmd Bit Field

The ExpectDataCmd bit field specifies the expected data for comparison with data read from the memory or memories. The ExpectDataCmd bit field is a four-bit field in the instruction word.

The four-bit ExpectDataCmd field is decoded as shown in [Table A-11](#).

Table A-11. ExpectDataCmd Bit Field Decode

Property Value	Bit Decode	Instruction Description
DataReg	0000	Apply the data in the Expect Data register to the comparators. Allow data inversion with Address bits.
InverseDataReg	0001	Apply the inverse data of the Expect Data register to the comparators. Allow data inversion with Address bits.

Table A-11. ExpectDataCmd Bit Field Decode (cont.)

Property Value	Bit Decode	Instruction Description
Zero	0010	Apply Expect Data of Zeros to the comparators. Allow data inversion with Address bits.
One	0011	Apply Expect Data of Ones to the comparators. Allow data inversion with Address bits.
DataReg_Rotate	0100	Apply the data in the Expect Data register to the comparators followed by a Rotate Left of the Expect Data register on the last tick of the executing operation. Allow data inversion with Address bits.
InverseDataReg_Rotate	0101	Apply the inverse data in the Expect Data register to the comparators followed by a Rotate Left of the Expect Data register on the last tick of the executing operation. Allow data inversion with Address bits.
DataReg_RotateWithInvert	0110	Apply the data in the Expect Data register to the comparators followed by a Rotate Left with inverted feedback of the Expect Data register on the last tick of the executing operation. Allow data inversion with Address bits.
InverseDataReg_RotateWithInvert	0111	Apply the inverse data in the Expect Data register to the comparators followed by a Rotate Left with inverted feedback of the Expect Data register on the last tick of the executing operation. Allow data inversion with Address bits.
DataReg_PRDG	1100	Apply the data in the Expect Data register to the comparators followed by the generation of a new pseudo-random data pattern on the last tick of the executing operation. Allow data inversion with Address bits.
InverseDataReg_PRDG	1101	Apply the inverse data in the Expect Data register to the comparators followed by the generation of a new pseudo-random data pattern on the last tick of the executing operation. Allow data inversion with Address bits.

Table A-11. ExpectDataCmd Bit Field Decode (cont.)

Property Value	Bit Decode	Instruction Description
Set_DataReg	1000	Apply Expect Data of Ones to the comparators. Do not allow data inversion with Address bits.
Reset_DataReg	1001	Apply Expect Data of Zeros to the comparators. No not allow data inversion with Address bits.

RepeatLoop Bit Field

The RepeatLoop bit field of the instruction word is used to identify which repeat loop are executed in the instruction.

The RepeatLoop field is two-bits wide in the instruction word and is decoded as shown in [Table A-12](#).

Table A-12. RepeatLoop Bit Field Decode

Property Value	Bit Decode	Instruction Description
Off	00	no repeat loop is executed in this instruction.
RepeatLoop (A)	01	Repeat Loop A is executed in the instruction
RepeatLoop (B)	10	Repeat Loop B is executed in the instruction
RepeatLoop(A), RepeatLoop(B)	11	Both Repeat Loops are executed in the instruction

InhibitLastAddressCount Bit Field

The InhibitLastAddressCount bit field of the instruction word inhibits the selected address register from updating the register value on the final execution of the currently executing instruction. The next instruction loaded for execution is determined by the [Pointer Control Block](#).

The InhibitLastAddressCount bit field is one bit in the instruction word and is decoded as shown in [Table A-13](#).

Table A-13. InhibitLastAddressCount Bit Field Decode

Property Value	Bit Decode	Instruction Description
Off	0	The selected address register counts normally.

Table A-13. InhibitLastAddressCount Bit Field Decode (cont.)

Property Value	Bit Decode	Instruction Description
On	1	The selected address register is not updated on the last tick of the executing operation during the last execution of the active instruction.

InhibitDataCompare Bit Field

The InhibitDataCompare bit field of the instruction word gates the StrobeDataOut signal of the executing operation to inhibit the compare of read data from the memory and the expected data.

The InhibitDataCompare field is one-bit in the instruction word and is decoded as shown in [Table A-14](#).

Table A-14. InhibitDataCompare Bit Field Decode

Property Value	Bit Decode	Instruction Description
Off	0	The memory read data and expected data are compared normally.
On	1	The <i>StrobeDataOut</i> signal from the Signal Generator Block is gated preventing any compares between the memory read data and expect data.

InhibitRefresh Bit Field

The InhibitRefresh bit field of the instruction word inhibits any AutoRefresh operations from being inserted while the instruction is executing.

The InhibitRefresh field is one-bit in the instruction word and is decoded as shown in [Table A-15](#). The InhibitRefresh bit field is present only when the memory type specified by the MemoryTemplate: MemoryType property of the [Memory Library File](#) as DRAM.

Table A-15. InhibitRefresh Bit Field Decode

Property Value	Bit Decode	Instruction Description
Off	0	Any AutoRefresh operations inserted by the Dynamic Refresh Control Block are performed normally.
On	1	AutoRefresh operations inserted by the Dynamic Refresh Control Block are inhibited.

For programming information, refer to the [Instruction: InhibitRefresh](#) property in the Algorithm wrapper described in the [Tessent MemoryBIST User's and Reference Manual](#).

CounterACmd Bit Field

The CounterACmd bit field specifies the command issued to the [CounterA Block](#). The CounterACmd field is a one-bit field in the instruction word.

The one-bit CounterACmd field is programmed as shown in [Table A-16](#).

Table A-16. CounterACmd Bit Field Decode

Property Value	Bit Decode	Instruction Description
Hold	0	The value in the CounterA maintains its current value.
Increment	1	Increment the CounterA value by one on the last tick of the executing operation.

DelayCounterCmd Bit Field

The DelayCounterCmd bit field specifies the command issued to the Delay Counter block. The DelayCounterCmd field is a one-bit field in the instruction word. The DelayCounterCmd bit field is present if the number of bits in the Delay Counter block is greater than zero as specified by the ETPlanner [NumberOfDelayCounterBits](#) property. The [DelayCounter Block](#) is also instantiated if the NumberOfDelayCounterBits property is set to zero but the memory type specified by the MemoryTemplate: MemoryType property of the [Memory Library File](#) is DRAM.

The one-bit DelayCounterCmd field is programmed as shown in [Table A-17](#).

Table A-17. DelayCounterCmd Bit Field Decode

Property Value	Bit Decode	Instruction Description
Hold	0	The value in the DelayCounter Block maintains its current value.
Increment	1	Increment the DelayCounter value by one on the last tick of the executing operation.

DisableMemoriesWithoutGroupWriteEnable

The DisableMemoriesWithoutGroupWriteEnable bit field of the instruction word disables access to memories that do not have the *group write enable* control signal.

The DisableMemoriesWithoutGroupWriteEnable field is one bit in the instruction word and is decoded as shown in [Table A-18](#).

Table A-18. DisableMemoriesWithoutGroupWriteEnable Bit Field Decode

Property Value	Bit Decode	Instruction Description
Off	0	No special handling of memories regarding the <i>group write enable</i> control signal is necessary. The value is 0 by default.
On	1	Memories without the <i>group write enable</i> control signal will be disabled; collar enable is deasserted so that no write operation or read comparison will occur.

For programming information, refer to the [Instruction: DisableMemoriesWithoutGroupWriteEnable](#) property in the Algorithm wrapper described in the [Tessent MemoryBIST User's and Reference Manual](#).

DisableMemoriesWithoutOutputEnable

The DisableMemoriesWithoutOutputEnable bit field of the instruction word disables access to memories that do not have the *output enable* control signal.

The DisableMemoriesWithoutOutputEnable field is one bit in the instruction word and is decoded as shown in [Table A-19](#).

Table A-19. DisableMemoriesWithoutOutputEnable Bit Field Decode

Property Value	Bit Decode	Instruction Description
Off	0	No special handling of memories regarding the <i>output enable</i> control signal is necessary. The value is 0 by default.
On	1	Memories without the <i>output enable</i> control signal will be disabled; collar enable is deasserted so that no write operation or read comparison will occur.

For programming information, refer to the [Instruction: DisableMemoriesWithoutOutputEnable](#) property in the Algorithm wrapper described in the [Tessent MemoryBIST User's and Reference Manual](#).

DisableMemoriesWithoutReadEnable

The DisableMemoriesWithoutReadEnable bit field of the instruction word disables access to memories that do not have the *read enable* control signal.

The DisableMemoriesWithoutReadEnable field is one bit in the instruction word and is decoded as shown in [Table A-20](#).

Table A-20. DisableMemoriesWithoutReadEnable Bit Field Decode

Property Value	Bit Decode	Instruction Description
Off	0	No special handling of memories regarding the <i>read enable</i> control signal is necessary. The value is 0 by default.
On	1	Memories without the <i>read enable</i> control signal will be disabled; collar enable is deasserted so that no write operation or read comparison will occur.

For programming information, refer to the [Instruction: DisableMemoriesWithoutReadEnable](#) property in the Algorithm wrapper described in the [Tessent MemoryBIST User's and Reference Manual](#).

DisableMemoriesWithoutSelect

The DisableMemoriesWithoutSelect bit field of the instruction word disables access to memories that do not have the *select* control signal.

The DisableMemoriesWithoutSelect field is one bit in the instruction word and is decoded as shown in [Table A-21](#).

Table A-21. DisableMemoriesWithoutSelect Bit Field Decode

Property Value	Bit Decode	Instruction Description
Off	0	No special handling of memories regarding the <i>select</i> control signal is necessary. The value is 0 by default.
On	1	Memories without the <i>select</i> control signal will be disabled; collar enable is deasserted so that no write operation or read comparison will occur.

For programming information, refer to the [Instruction:DisableMemoriesWithoutSelect](#) property in the Algorithm wrapper described in the [Tessent MemoryBIST User's and Reference Manual](#).

DisableMemoriesWithoutWriteEnable

The DisableMemoriesWithoutWriteEnable bit field of the instruction word disables access to memories that do not have the *write enable* control signal.

The DisableMemoriesWithoutWriteEnable field is one bit in the instruction word and is decoded as shown in [Table A-22](#).

Table A-22. DisableMemoriesWithoutWriteEnable Bit Field Decode

Property Value	Bit Decode	Instruction Description
Off	0	No special handling of memories regarding the <i>write enable</i> control signal is necessary. The value is 0 by default.
On	1	Memories without the <i>write enable</i> control signal will be disabled; collar enable is deasserted so that no write operation or read comparison will occur.

For programming information, refer to the [Instruction: DisableMemoriesWithoutWriteEnable](#) property in the Algorithm wrapper described in the [Tessent MemoryBIST User's and Reference Manual](#).

BranchToInstruction Bit Field

The BranchToInstruction bit field of the instruction register specifies the address of the branch instruction to be loaded for execution by the Pointer Control block.

The number of bits in the BranchToInstruction bit field depends on the number of instructions in the microcode as specified by the ETPlanner [NumberOfInstructions](#) property.

[Table A-23](#) illustrates the resulting number of bits in the BranchToInstruction bit field as a function of the MemBistControllerOptions: NumberOfInstructions property.

Table A-23. Number of Bits in the BranchToInstruction bit field

Range of NumberOfInstructions specified	Number of Bits in the BranchToInstruction bit field
5-8	3
9-16	4
17-32	5
33-64	6

NextConditions Bit Field

The NextConditions bit field of the instruction register defines the set of conditions used by the Pointer Control block to determine the next instruction to be executed.

For each instruction, the value loaded into the NextConditions bit field of the instruction register is specified using the properties of the [Algorithm: Instruction: NextConditions](#) wrapper.

The bit assignments of the NextConditions field are illustrated in [Table A-24](#). Descriptions of the properties of the NextConditions field follow the table.

Table A-24. NextConditions Bit Field Assignment

Next Conditions	Bit Assignments
RepeatLoopCmd	8 7
DelayCounterEndCount	6
CounterAEndCount	5
Z_EndCount	4
X1_EndCount	3
X0_EndCount	2
Y1_EndCount	1
Y0_EndCount	0

Y0_EndCount

When the [NextConditions: Y0_EndCount](#) property of the [Algorithm](#) wrapper is programmed as On, the Y0_EndCount trigger from the [Address Generator Block](#) is to be tested. The Y0_EndCount trigger from the Address Generator block is asserted in one of the following circumstances:

- When the Y0 address segment is incrementing, and the segment has reached the count range maximum.
- When the Y0 address segment is decremented, and the segment has reached the count range minimum.

This bit is not present in the NextConditions bit field if one or zero column address bits are specified in the [AddressCounter](#) wrapper in the memory library file.

Y1_EndCount

When the [NextConditions: Y1_EndCount](#) property of the [Algorithm](#) wrapper is programmed as On, the Y1_EndCount trigger from the [Address Generator Block](#) is to be tested. The Y1_EndCount trigger from the Address Generator block is asserted in one of the following circumstances:

- When the Y1 address segment is incrementing and the segment has reached the count range maximum.
- When the Y1 address segment is decremented and the segment has reached the count range minimum.

This bit is not present in the NextConditions bit field when no column address bits are specified in the [AddressCounter](#) wrapper in the memory library file.

X0_EndCount

When the [NextConditions: X0_EndCount](#) property of the [Algorithm](#) wrapper is programmed as On, the X0_EndCount trigger from the [Address Generator Block](#) is to be tested. The X0_EndCount trigger from the Address Generator block is asserted in one of the following circumstances:

- When the X0 address segment is incrementing, and the segment has reached the count range maximum.
- When the X0 address segment is decremented, and the segment has reached the count range minimum.

This bit is not present in the NextConditions bit field when one or zero row address bits are specified in the [AddressCounter](#) wrapper in the memory library file.

X1_EndCount

When the [NextConditions: X1_EndCount](#) property of the [Algorithm](#) wrapper is programmed as On, the X1_EndCount trigger from the [Address Generator Block](#) is to be tested. The X1_EndCount trigger from the Address Generator block is asserted in one of the following circumstances:

- When the X1 address segment is incrementing, and the segment has reached the count range maximum.
- When the X1 address segment is decremented, and the segment has reached the count range minimum.

This bit is not present in the NextConditions bit field when no row address bits are specified in the [AddressCounter](#) wrapper in the memory library file.

Z_EndCount

When the [NextConditions: Z_EndCount](#) property of the [Algorithm](#) wrapper is programmed as On, the Z_EndCount trigger from the [Address Generator Block](#) is to be tested. The Z_EndCount trigger from the [Address Generator Block](#) is asserted in one of the following circumstances:

- When the Z address segment is incrementing, and the segment has reached the count range maximum.
- When the Z address segment is decremented, and the segment has reached the count range minimum.

This bit is not present in the NextConditions bit field if no bank address bits are specified in the [AddressCounter](#) wrapper in the memory library file.

CounterAEndCount

When the [NextConditions: CounterAEndCount](#) property of the of the [Algorithm](#) wrapper is programmed as On, the CounterAEndCount trigger from the [CounterA Block](#) is to be tested. The CounterAEndCount trigger from the Counter A block is asserted when the Counter A value is equivalent to the value specified for the [TestRegisterSetup: LoadCounterA_EndCount](#) property within the [Algorithm](#) wrapper.

This bit is always present in the NextConditions bit field.

DelayCounterEndCount

When the [NextConditions: DelayCounterEndCount](#) property of the [Algorithm](#) wrapper is programmed as On, the DelayCounterEndCount trigger from the [DelayCounter Block](#) /Refresh Timer block is to be tested. The DelayCounterEndCount trigger is asserted when the Delay Counter/Refresh Timer value is equivalent to the value specified for the [TestRegisterSetup: LoadDelayCounter_EndCount](#) property within the [Algorithm](#) wrapper.

This bit is not present in the NextConditions bit field when zero delay counter bits are specified in the ETPlanner [NumberOfDelayCounterBits](#) property.

RepeatLoopCmd

The RepeatLoopCmd comprises two bits in the NextConditions field. This two-bit command is decoded as illustrated in [Table A-25](#).

Table A-25. RepeatLoopCmd Decode

RepeatLoop Wrapper(s) Specified	Bit Decode	Description
No Repeat Loop wrappers are specified	00	Idle
Only RepeatLoop(A) wrapper is specified	01	Increment Loop Counter A
Only RepeatLoop(B) wrapper is specified	10	Increment Loop Counter B
Both RepeatLoop(A) and RepeatLoop(B) wrappers are specified	11	Increment Loop Counter BA

When the RepeatLoopCmd performs an increment of any of the Repeat Loop counters, the RepeatLoopDone trigger from the [Repeat Loop Control Block](#) is tested.

The RepeatLoopCmd bits are always present in the NextConditions bit field.

There are several ways to get help when setting up and using Tessent® software tools. Depending on your need, help is available from documentation, online command help, and Mentor Graphics Support.

Documentation

A comprehensive set of reference manuals, user guides, and release notes is available in two formats:

- HTML for searching and viewing online
- PDF for searching, viewing online, and printing

The documentation is available from each software tool and online at:

<http://supportnet.mentor.com>

For more information on setting up and using Tessent documentation, see the “[Using Tessent Documentation](#)” chapter in the *Managing Mentor Graphics Tessent Software* manual.

Mentor Graphics Support

Mentor Graphics software support includes software enhancements, access to comprehensive online services with SupportNet, and the optional On-Site Mentoring service. For details, see:

<http://supportnet.mentor.com/about/>

If you have questions about a software release, you can log in to SupportNet and search thousands of technical solutions, view documentation, or open a Service Request online:

<http://supportnet.mentor.com>

If your site is under current support and you do not have a SupportNet login, you can register for SupportNet by filling out a short form here:

<http://supportnet.mentor.com/user/register.cfm>

All customer support contact information is available here:

<http://supportnet.mentor.com/contacts/supportcenters/index.cfm>

Third-Party Information

For information about third-party software included with this release of Tessent products, refer to the [*Third-Party Software for Tessent Products*](#).



End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/eula

IMPORTANT INFORMATION

USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. ORDERS, FEES AND PAYMENT.

- 1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement ("Order(s)"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order or presented via any electronic portal or other automated order management system will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.
- 1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax, consumption tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.
- 1.3. All Products are delivered FCA factory (Incoterms 2010), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 5.2); (b) for Customer's internal business purposes; (c) for the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer provides any feedback or requests any change or enhancement to Products,

whether in the course of receiving support or consulting services, evaluating Products, performing beta testing or otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

4. **BETA CODE.**

- 4.1. Portions or all of certain Software may contain code for experimental testing and evaluation (which may be either alpha or beta, collectively "Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.
- 4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.
- 4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. **RESTRICTIONS ON USE.**

- 5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use Products except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer becomes aware of such unauthorized disclosure or use. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any source code from Software. Log files, data files, rule files and script files generated by or for the Software (collectively "Files"), including without limitation files containing Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") which are Mentor Graphics' proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Software or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Software, or disclose to any third party the results of, or information pertaining to, any benchmark.
- 5.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer's employees or on-site contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.
- 5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms

of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.

5.4. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer with updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics' then current End-User Support Terms located at <http://supportnet.mentor.com/about/legal/>.

7. **LIMITED WARRANTY.**

7.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification, improper installation or Customer is not in compliance with this Agreement. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

7.2. THE WARRANTIES SET FORTH IN THIS SECTION 7 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

8. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 8 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

9. **HAZARDOUS APPLICATIONS.** CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING ITS PRODUCTS USED IN APPLICATIONS WHERE THE FAILURE OR INACCURACY OF ITS PRODUCTS MIGHT RESULT IN DEATH OR PERSONAL INJURY ("HAZARDOUS APPLICATIONS"). EXCEPT TO THE EXTENT THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. **INFRINGEMENT.**

- 11.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to such action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

- 11.2. If a claim is made under Subsection 11.1 Mentor Graphics may, at its option and expense: (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.
- 11.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.
- 11.4. THIS SECTION 11 IS SUBJECT TO SECTION 8 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, FOR DEFENSE, SETTLEMENT AND DAMAGES, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.

12. TERMINATION AND EFFECT OF TERMINATION.

- 12.1. If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.
- 12.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.
13. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and United States ("U.S.") government agencies, which prohibit export, re-export or diversion of certain products, information about the products, and direct or indirect products thereof, to certain countries and certain persons. Customer agrees that it will not export or re-export Products in any manner without first obtaining all necessary approval from appropriate local and U.S. government agencies. If Customer wishes to disclose any information to Mentor Graphics that is subject to any U.S. or other applicable export restrictions, including without limitation the U.S. International Traffic in Arms Regulations (ITAR) or special controls under the Export Administration Regulations (EAR), Customer will notify Mentor Graphics personnel, in advance of each instance of disclosure, that such information is subject to such export restrictions.
14. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. The parties agree that all Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to U.S. FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. government or a U.S. government subcontractor is subject solely to the terms and conditions set forth in this Agreement, which shall supersede any conflicting terms or conditions in any government order document, except for provisions which are contrary to applicable mandatory federal laws.
15. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.
16. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FlexNet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 16 shall survive the termination of this Agreement.
17. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the U.S. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, U.S., if Customer is located in North or South America, and the laws of Ireland if

Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. Nothing in this section shall restrict Mentor Graphics' right to bring an action (including for example a motion for injunctive relief) against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

18. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.
19. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing, signed by an authorized representative of each party. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 130502, Part No. 255853