

Agilent E2927A Opt. 320 C-API/PPR

## **Reference**



**Agilent Technologies**

## Important Notice

This document contains propriety information that is protected by copyright. All rights are reserved. Neither the documentation nor software may be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of Agilent Technologies.

© Copyright 1999, 2000 by:  
Agilent Technologies  
Herrenberger Straße 130  
D-71034 Böblingen  
Germany

The information in this manual is subject to change without notice. Agilent Technologies makes no warranty of any kind with regard to this manual, including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

Agilent Technologies shall not be liable for errors contained herein or direct, indirect, special, incidental, or consequential damages in connection with the furnishing, performance, or use of this manual.

Brand or product names are trademarks or registered trademarks of their respective companies or organizations.

Authors: Stephan Greisinger and Anja Schauer, t3 medien GmbH

# Contents

|   |    |
|---|----|
| Introduction                            | 13 |
| Conventions                             | 14 |
| Sections of the C-API Reference         | 15 |
| General Functions                       | 17 |
| Connection and Initialization Functions | 17 |
| BestClose                               | 18 |
| BestDevIdentifierGet                    | 19 |
| BestOpen                                | 21 |
| BestPing                                | 22 |
| BestRS232BaudRateSet                    | 22 |
| Administration Functions                | 23 |
| BestAllResourceUnlock                   | 24 |
| BestCapabilityCheck                     | 24 |
| BestCapabilityRead                      | 26 |
| BestResourceIsLocked                    | 27 |
| BestResourceLock                        | 28 |
| BestResourceUnlock                      | 29 |
| BestSystemInfoGet                       | 30 |
| BestVersionGet                          | 31 |
| Power Up and Reset Control Functions    | 32 |
| BestAllPropDefaultLoad                  | 33 |
| BestAllPropLoad                         | 34 |
| BestAllPropStore                        | 34 |
| BestBoardPropGet                        | 35 |
| BestBoardPropSet                        | 35 |
| BestBoardReset                          | 36 |
| BestPowerUpPropGet                      | 36 |
| BestPowerUpPropSet                      | 37 |
| BestSMReset                             | 37 |
| Card Status Functions                   | 38 |
| BestStatusRegClear                      | 38 |
| BestStatusRegGet                        | 39 |

|   |           |
|---|-----------|
| <b>PCI Analyzer Functions</b>                   | <b>41</b> |
| <b>Protocol Observer Functions</b>              | <b>42</b> |
| BestObsBitPositionFind                          | 43        |
| BestObsErrResultGet                             | 44        |
| BestObsErrStringGet                             | 45        |
| BestObsMaskGet                                  | 46        |
| BestObsMaskSet                                  | 47        |
| BestObsPropDefaultSet                           | 48        |
| BestObsRuleErrTypeGet                           | 48        |
| BestObsStatusClear                              | 49        |
| BestObsStatusGet                                | 49        |
| <b>Timing Check Functions</b>                   | <b>50</b> |
| BestTimCheckDefaultSet                          | 51        |
| BestTimCheckGenPropGet                          | 52        |
| BestTimCheckGenPropSet                          | 53        |
| BestTimCheckMaskGet                             | 54        |
| BestTimCheckMaskSet                             | 55        |
| BestTimCheckProg                                | 56        |
| BestTimCheckPropGet                             | 57        |
| BestTimCheckPropSet                             | 58        |
| BestTimCheckRead                                | 59        |
| BestTimCheckResultGet                           | 59        |
| BestTimCheckStatusClear                         | 60        |
| BestTimCheckStatusGet                           | 61        |
| <b>Pattern Term Function</b>                    | <b>62</b> |
| BestPattSet                                     | 62        |
| <b>Trace Memory Trigger Sequencer Functions</b> | <b>67</b> |
| BestTrigSeqGenPropDefaultSet                    | 68        |
| BestTrigSeqGenPropGet                           | 68        |
| BestTrigSeqGenPropSet                           | 69        |
| BestTrigSeqProg                                 | 69        |
| BestTrigSeqPropDefaultSet                       | 70        |
| BestTrigSeqTranCondPropSet                      | 71        |
| BestTrigSeqTranPropDefaultSet                   | 73        |
| BestTrigSeqTranPropSet                          | 74        |
| <b>Trace Memory Functions</b>                   | <b>75</b> |
| BestAnalyzerRun                                 | 75        |

|                                      |           |
|--------------------------------------|-----------|
| BestAnalyzerStop                     | 76        |
| BestTraceBitPosGet                   | 77        |
| BestTraceBytePerLineGet              | 78        |
| BestTraceDataGet                     | 79        |
| BestTracePattPropSet                 | 80        |
| BestTracePropSet                     | 81        |
| BestTraceRun                         | 82        |
| BestTraceStatusGet                   | 82        |
| BestTraceStop                        | 83        |
| <b>Performance Measure Functions</b> | <b>84</b> |
| BestPerfCtrRead                      | 85        |
| BestPerfGenPropDefaultSet            | 86        |
| BestPerfGenPropGet                   | 87        |
| BestPerfGenPropSet                   | 88        |
| BestPerfRun                          | 89        |
| BestPerfSeqProg                      | 89        |
| BestPerfSeqPropDefaultSet            | 90        |
| BestPerfSeqTranCondPropSet           | 91        |
| BestPerfSeqTranPropDefaultSet        | 92        |
| BestPerfSeqTranPropSet               | 93        |
| BestPerfStatusGet                    | 94        |
| BestPerfStop                         | 95        |
| BestPerfUpdate                       | 95        |
| <b>PCI Exerciser Functions</b>       | <b>97</b> |
| <b>Exerciser Generic Functions</b>   | <b>98</b> |
| BestExerciserGenPropGet              | 99        |
| BestExerciserGenPropSet              | 100       |
| BestMasterBlockPageRun               | 101       |
| BestMasterBlockRun                   | 102       |
| BestMasterGenPropDefaultSet          | 103       |
| BestMasterGenPropGet                 | 104       |
| BestMasterGenPropSet                 | 105       |
| BestMasterCondStartPattSet           | 106       |
| BestMasterStop                       | 106       |
| BestTargetGenPropDefaultSet          | 107       |
| BestTargetGenPropGet                 | 107       |
| BestTargetGenPropSet                 | 108       |

|  |            |
|--|------------|
| <b>Master Programming Functions</b>              | <b>109</b> |
| BestMasterAttrGroupLineProg                      | 111        |
| BestMasterAttrGroupLineRead                      | 112        |
| BestMasterAttrLineProg                           | 113        |
| BestMasterAttrLineRead                           | 114        |
| BestMasterAttrPageInit                           | 115        |
| BestMasterAttrPropDefaultSet                     | 116        |
| BestMasterAttrPropGet                            | 116        |
| BestMasterAttrPropSet                            | 117        |
| BestMasterBlockEndProg                           | 118        |
| BestMasterBlockLineProg                          | 119        |
| BestMasterBlockLineRead                          | 120        |
| BestMasterBlockPageInit                          | 121        |
| BestMasterBlockPropDefaultSet                    | 122        |
| BestMasterBlockPropGet                           | 122        |
| BestMasterBlockPropSet                           | 123        |
| BestMasterByteEnableProg                         | 124        |
| BestMasterByteEnableRead                         | 124        |
| <b>Target Programming Functions</b>              | <b>125</b> |
| BestTargetAttrGroupLineProg                      | 127        |
| BestTargetAttrGroupLineRead                      | 128        |
| BestTargetAttrLineProg                           | 129        |
| BestTargetAttrLineRead                           | 130        |
| BestTargetAttrPageInit                           | 131        |
| BestTargetAttrPageSelect                         | 132        |
| BestTargetAttrPropDefaultSet                     | 132        |
| BestTargetAttrPropGet                            | 133        |
| BestTargetAttrPropSet                            | 134        |
| BestTargetDecoderPowerUpProg                     | 135        |
| BestTargetDecoderPowerUpRead                     | 136        |
| BestTargetDecoderProg                            | 137        |
| BestTargetDecoderPropDefaultSet                  | 138        |
| BestTargetDecoderPropGet                         | 139        |
| BestTargetDecoderPropSet                         | 140        |
| BestTargetDecoderRead                            | 141        |
| <b>Configuration Space Programming Functions</b> | <b>142</b> |
| BestConfRegGet                                   | 143        |
| BestConfRegSet                                   | 144        |

|                                     |     |
|-------------------------------------|-----|
| BestConfRegMaskGet                  | 145 |
| BestConfRegMaskSet                  | 146 |
| BestConfigScan                      | 147 |
| BestConfigScanPrint                 | 147 |
| BestPCIConfigCheck                  | 148 |
| Expansion ROM Programming Functions | 148 |
| BestExpRomByteRead                  | 149 |
| BestExpRomByteWrite                 | 150 |
| Data Memory Functions               | 150 |
| BestDataMemInit                     | 151 |
| BestDataMemRead                     | 151 |
| BestDataMemWrite                    | 152 |
| Host Access Functions               | 153 |
| BestHostPCIRegGet                   | 154 |
| BestHostPCIRegSet                   | 155 |
| BestHostSysMemAccessPrepare         | 156 |
| BestHostSysMemDump64                | 157 |
| BestHostSysMemFill64                | 159 |
| Interrupt Generation Function       | 161 |
| BestInterruptGenerate               | 161 |
| Built-In Test Functions             | 162 |
| BestTestPropDefaultSet              | 162 |
| BestTestPropSet                     | 163 |
| BestTestProtErrDetect               | 164 |
| BestTestResultDump                  | 165 |
| BestTestRun                         | 166 |

|  |            |
|--|------------|
| <b>Interface Control Functions</b>                 | <b>169</b> |
| <b>CPU Port Programming Functions</b>              | <b>169</b> |
| BestCPUportWordBlockRead                           | 170        |
| BestCPUportWordBlockWrite                          | 171        |
| BestCPUportIntrClear                               | 172        |
| BestCPUportIntrStatusGet                           | 172        |
| BestCPUportPropSet                                 | 173        |
| BestCPUportRead                                    | 174        |
| BestCPUportRST                                     | 175        |
| BestCPUportWrite                                   | 176        |
| <b>Static I/O Port Programming Functions</b>       | <b>177</b> |
| BestStaticPinWrite                                 | 177        |
| BestStaticPropGet                                  | 178        |
| BestStaticPropSet                                  | 179        |
| BestStaticRead                                     | 180        |
| BestStaticWrite                                    | 180        |
| <b>Trigger I/O Sequencer Programming Functions</b> | <b>181</b> |
| BestTrigIOGenPropDefaultSet                        | 182        |
| BestTrigIOGenPropGet                               | 182        |
| BestTrigIOGenPropSet                               | 183        |
| BestTrigIOSeqProg                                  | 183        |
| BestTrigIORun                                      | 184        |
| BestTrigIOSeqPropDefaultSet                        | 184        |
| BestTrigIOSeqTranCondPropSet                       | 185        |
| BestTrigIOSeqTranPropDefaultSet                    | 186        |
| BestTrigIOSeqTranPropSet                           | 187        |
| BestTrigIOStop                                     | 187        |
| <b>Display Functions</b>                           | <b>188</b> |
| BestDisplayPropSet                                 | 188        |
| BestDisplayWrite                                   | 189        |
| <b>Mailbox Functions</b>                           | <b>189</b> |
| BestMailboxReceiveRegRead                          | 190        |
| BestMailboxSendRegWrite                            | 191        |
| BestPCICfgMailboxReceiveRegRead                    | 192        |
| BestPCICfgMailboxSendRegWrite                      | 193        |
| <b>Power Management Event Functions</b>            | <b>194</b> |
| BestPMERead  | 194        |



|   |     |
|---|-----|
| BestPMEWrite  | 195 |
| <b>Protocol Permutator and Randomizer Functions</b> | 197 |
| <b>PPR Administrating Functions</b>                 | 197 |
| BestPprDelete                                       | 198 |
| BestPprGenPropDefaultSet                            | 198 |
| BestPprGenPropGet                                   | 199 |
| BestPprGenPropSet                                   | 200 |
| BestPprInit   | 201 |
| <b>Master Block Permutation Functions</b>           | 202 |
| BestPprBlockCoverageGet                             | 203 |
| BestPprBlockGenerate                                | 204 |
| BestPprBlockInit                                    | 204 |
| BestPprBlockPermPropDefaultSet                      | 205 |
| BestPprBlockPermPropGet                             | 205 |
| BestPprBlockPermPropSet                             | 206 |
| BestPprBlockResultGet                               | 207 |
| BestPprBlockVariationDefaultSet                     | 207 |
| BestPprBlockVariationGet                            | 208 |
| BestPprBlockVariationSet                            | 209 |
| <b>Master Attribute Permutation Functions</b>       | 210 |
| BestPprMAttrCoverageGet                             | 211 |
| BestPprMAttrGenerate                                | 212 |
| BestPprMAttrInit                                    | 212 |
| BestPprMAttrPermPropDefaultSet                      | 213 |
| BestPprMAttrPermPropGet                             | 213 |
| BestPprMAttrPermPropSet                             | 214 |
| BestPprMAttrResultGet                               | 215 |
| BestPprMAttrVariationDefaultSet                     | 215 |
| BestPprMAttrVariationGet                            | 216 |
| BestPprMAttrVariationSet                            | 217 |
| <b>PPR Report Functions</b>                         | 218 |
| BestPprReportDelete                                 | 218 |
| BestPprReportFile                                   | 219 |
| BestPprReportPropDefaultSet                         | 219 |
| BestPprReportPropGet                                | 220 |
| BestPprReportPropSet                                | 221 |
| BestPprReportWrite                                  | 222 |

|  |     |
|--|-----|
| Target Attribute Permutation Functions | 224 |
| BestPprTAttrCoverageGet                | 225 |
| BestPprTAttrGenerate                   | 226 |
| BestPprTAttrInit                       | 226 |
| BestPprTAttrPermPropDefaultSet         | 227 |
| BestPprTAttrPermPropGet                | 227 |
| BestPprTAttrPermPropSet                | 228 |
| BestPprTAttrResultGet                  | 229 |
| BestPprTAttrVariationDefaultSet        | 229 |
| BestPprTAttrVariationGet               | 230 |
| BestPprTAttrVariationSet               | 231 |
| Error Handling                         | 233 |
| Error Functions                        | 233 |
| BestLastErrorGet                       | 234 |
| BestLastErrorStringGet                 | 234 |
| BestErrorStringGet                     | 235 |
| Type Definitions                       | 237 |
| b_addrspacetype                        | 237 |
| b_blkproptype                          | 238 |
| b_boardproptype                        | 241 |
| b_cpuproptype                          | 242 |
| b_decodertype                          | 243 |
| b_decproptype                          | 243 |
| Decoding Properties                    | 244 |
| Info Properties                        | 248 |
| Resource Properties                    | 249 |
| b_errtype                              | 250 |
| Software Errors                        | 250 |
| Firmware Errors                        | 253 |
| b_exercisergenproptype                 | 256 |
| b_mastergenproptype                    | 257 |
| b_mattrgrouptype                       | 259 |
| b_mattrproptype                        | 260 |
| Address Phase Attributes (Master)      | 260 |

|                                   |     |
|-----------------------------------|-----|
| Data Phase Attributes (Master)    | 263 |
| Control Attributes (Master)       | 265 |
| b_obsruletype                     | 266 |
| b_obsstatustype                   | 267 |
| b_perfgenproptype                 | 268 |
| b_perfseqtrancondproptype         | 268 |
| b_perfseqtranproptype             | 269 |
| b_porttype                        | 270 |
| b_puproptype                      | 271 |
| b_resourcetype                    | 272 |
| b_signaltype (for Timing Check)   | 273 |
| b_signaltype (List of Signals)    | 274 |
| Signal Types                      | 275 |
| Bus Signals                       | 276 |
| Bus States                        | 277 |
| Static I/O Signals                | 280 |
| Transaction Attributes            | 280 |
| Markers                           | 282 |
| Exerciser Signals                 | 283 |
| Internal Counters                 | 283 |
| Checks                            | 283 |
| Gap Information                   | 284 |
| b_sizetype                        | 284 |
| b_staticproptype                  | 285 |
| b_systeminfotype                  | 285 |
| b_targetgenproptype               | 286 |
| b_tattrgrouptype                  | 287 |
| b_tattrproptype                   | 288 |
| Address Phase Attributes (Target) | 288 |
| Data Phase Attributes (Target)    | 289 |
| Control Attributes (Target)       | 291 |
| b_tcgenproptype                   | 291 |
| b_tcproptype                      | 292 |
| b_tcstatustype                    | 292 |
| b_testproptype                    | 293 |

|  |     |
|--|-----|
| b_tracepattproptype                    | 294 |
| b_traceproptype                        | 295 |
| b_tracestatustype                      | 296 |
| b_trigioseqgenproptype                 | 297 |
| b_trigioseqtrancondproptype            | 298 |
| b_trigioseqtranproptype                | 299 |
| b_trigseqgenproptype                   | 299 |
| b_trigseqtrancondproptype              | 300 |
| b_trigseqtranproptype                  | 300 |
| b_versionproptype                      | 301 |
| bppr_algorithmtype                     | 302 |
| bppr_algorithmtype: B_BLK_CMDS Details | 302 |
| bppr_blkpermproptype                   | 304 |
| bppr_blkresultparamtype                | 306 |
| bppr_blkvarparamtype                   | 307 |
| Value Lists                            | 308 |
| bppr_genproptype                       | 310 |
| bppr_mattrpermproptype                 | 311 |
| bppr_mattrresultparamtype              | 312 |
| bppr_reportproptype                    | 313 |
| bppr_tattrpermproptype                 | 314 |
| bppr_tattrresultparamtype              | 315 |

# Introduction

The C\_API Reference describes all C functions, types and definitions of the application programming interface of the Agilent PCI testcard. It also provides the commands and abbreviations to be used in the command line interface (CLI) of the graphical user interface.

To develop C programs or to use the command line interface, you should have good background knowledge of the Agilent PCI testcard and the programming models.

This is provided in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

After you have read this Guides, you will be ready to use the C-API functions/CLI commands.

The documentation of C functions needs a consistent approach to name constants, types, and functions as shown in “*Conventions*” on page 14.

For information about the structure of the reference and the structure within individual sections, see “*Sections of the C-API Reference*” on page 15.

# Conventions

**Programming Interfaces** The Agilent PCI testcard provides two programming interfaces:

- The application programming interface (C-API) allows control of the software by means of C function calls.
- The Command Line Interface (CLI) provides a means of using the function calls directly from the command line of the graphical user interface (GUI). No C compiler is needed. This interface is used for simple interactive testing.

The functions are described with their syntax, return value and parameters. The CLI equivalent is specified for each function. For typing convenience, an abbreviated form of each CLI command and its associated parameters is used.

**Naming Conventions** The following conventions are used to name constants, types, and functions:

- Naming of Constants:

Constants are written in capital letters. Each name begins with “B\_”.

**Example:** B\_RESLOCK\_EXERCISER.

**NOTE**

Constants which are used exclusively in Protocol Permutator and Randomizer functions begin with “BPPR\_”.

**Example:** BPPR\_GEN\_BUSSPEED

- Naming of Types:

Type names are written in lower case letters. Each name begins with “b\_” and ends with “type”.

**Example:** b\_resourcetype

**NOTE**

Types which are used exclusive in Protocol Permutator and Randomizer functions begin with “bppr\_”.

**Example:** bppr\_genproptype

- Naming of Functions:

Function names are written in lower case letters and capital letters. Each name begins with “Best”. The action is indicated by the last words in the call.

**Example:** BestResourceLock

**NOTE**

Protocol Permutator and Randomizer functions begin with “BestPpr”.

**Example:** BestPprGenPropSet

## Sections of the C-API Reference

The C-API Reference is divided into the following sections:

- “General Functions” on page 17
- “PCI Analyzer Functions” on page 41
- “PCI Exerciser Functions” on page 97
- “Interface Control Functions” on page 169
- “Protocol Permutator and Randomizer Functions” on page 197
- “Error Handling” on page 233
- “Type Definitions” on page 237

All sections except the type definition section are again thematically grouped. Every group starts with an overview of the functions in logical order, followed by the full description of the functions in alphabetical order.

The type definition section is not thematically grouped. All available type definitions are listed in alphabetical order.





# General Functions

The general functions are divided into the following sections:

- “*Connection and Initialization Functions*” on page 17
- “*Administration Functions*” on page 23
- “*Power Up and Reset Control Functions*” on page 32
- “*Card Status Functions*” on page 38

## Connection and Initialization Functions

The following functions are used for connection and initialization purposes:

| Function                                 | Result  |
|--|---|
| <i>“BestDevIdentifierGet” on page 19</i> | Returns the identifier of a PCI device.             |
| <i>“BestOpen” on page 21</i>             | Opens a connection to the testcard.                 |
| <i>“BestClose” on page 18</i>            | Closes the connection to the testcard.              |
| <i>“BestPing” on page 22</i>             | Checks a connection to the testcard.                |
| <i>“BestRS232BaudRateSet” on page 22</i> | Sets the baud rate if the serial interface is used. |

How to use the functions is described in “*Connection and Initialization*” in the *Agilent E2927A Opt. 320 C-API/PPR Programmer’s Guide*.

## BestClose

**Call** `b_errtype BestClose( b_handletype handle );`

**Description** Closes the session and frees any allocated memory. If the serial port has been used, it also resets the baud rate to 9600.

**CLI Equivalent** No CLI equivalent. Closing the CLI window executes BestClose automatically.

**CLI Abbreviation** No CLI abbreviation.

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestOpen*” on page 21

## BestDevIdentifierGet

**Call** `b_errtype BestDevIdentifierGet (`  
     `b_int32 vendor_id,`  
     `b_int32 device_id,`  
     `b_int32 subsys_id,`  
     `b_int32 *devid );`

**Description** Used if the PCI port is applied for in-system analysis (when the software is running on the system under test). The function returns the device identifier of the testcard on the PCI bus. The returned device identifier can be used as port number for the function “*BestOpen*” on page 21.

If multiple cards are plugged into the system, the number stored in the subsystem id register in the configuration space can be used to distinguish between different testcards.

**NOTE** This function can only be used in systems with standard PCI BIOS. For other systems, you must build the device identifier on your own. Then you must follow the rules shown in “*Device Identifier Format*” on page 20.

**CLI Equivalent** `BestDevIdentifierGet vendor_id=<vendor_id> device_id=<device_id>`  
`subsys_id=<subsys_id>`

**CLI Abbreviation** `diget vendor=<vendor_id> dev=<device_id> subsys=<subsys_id>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **vendor\_id** Vendor ID of the testcard (default = 103C\h for Agilent).

**device\_id** Device ID (vendor dependent) for the testcard (for example, 2940\h).

**number** Index to distinguish between different testcards in one system. The first card has index “0”, the second “1”, and so forth.

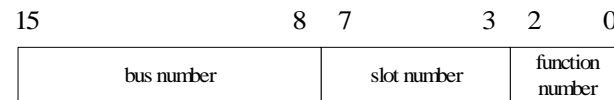
**Output Parameters** **devid** Device identifier within the system. This is the device number used to access the card’s configuration space (for example, in “*BestOpen*” on page 21).

**See also** –

## Device Identifier Format

Usually you will not need the device identifier format, because you only need to pass the device identifier returned by `BestDevIdentifierGet` to the `BestOpen` call.

The function co-operates only with standard PCI BIOS. If you use another BIOS or run a system without BIOS, you can build the device identifier yourself by observing the following format rules:



Used to access functions of multi-function PCI devices. For single-function PCI devices, the function number is 0.

## BestOpen

**Call** `b_errtype BestOpen(  
           b_handletype *handle,  
           b_porttype port,  
           b_int32 portnum );`

**Description** Opens and checks the connection to the PCI testcard and initializes the internal structures and variables for the control port.

This function must be called before calling any other function. It returns a handle for the session, which is used by all subsequent C-API functions. The handle identifies each testcard and declares the control port for the session (for example, the Fast Host Interface).

You can open multiple sessions for one testcard (for example, one Fast Host Interface session and one PCI session).

**NOTE** You cannot open multiple sessions for the same port simultaneously.

**CLI Equivalent** No CLI equivalent.

The function is automatically called when you open the CLI window. Port and port name are then constant during the entire CLI session.

**CLI Abbreviation** No CLI abbreviation.

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **port** Type of control port (for example, Fast Host Interface); see “*b\_porttype*” on page 270.

**portnum** Control port; see “*b\_porttype*” on page 270.

**Output Parameters** **handle** Handle to identify the session (comparable to a file handle).

**See also** “*BestClose*” on page 18  
 “*BestDevIdentifierGet*” on page 19

## BestPing

|                         |   |
|-------------------------|---|
| <b>Call</b>             | <code>b_errtype BestPing( b_handletype handle );</code>   |
| <b>Description</b>      | Checks the connection to the card. If the connection is ok, both card LEDs will flash simultaneously. |
| <b>CLI Equivalent</b>   | BestPing  |
| <b>CLI Abbreviation</b> | ping  |
| <b>Return Value</b>     | Error code; see “ <i>b_errtype</i> ” on page 250.   |
| <b>Input Parameters</b> | <b>handle</b> Handle that identifies the session and the connection used by the session.              |
| <b>See also</b>         | –   |

## BestRS232BaudRateSet

|                         |  |
|-------------------------|--|
| <b>Call</b>             | <code>b_errtype BestRS232BaudRateSet (         b_handletype handle,         b_int32 baudrate );</code> |
| <b>Description</b>      | Changes the baud rate.   |
| <b>CLI Equivalent</b>   | BestRS232BaudRateSet baudrate=<baudrate>   |
| <b>CLI Abbreviation</b> | brset baud=<baudrate>  |
| <b>Return Value</b>     | Error code; see “ <i>b_errtype</i> ” on page 250.  |
| <b>Input Parameters</b> | <b>handle</b> Handle to identify the session.  |
|                         | <b>baudrate</b> Baud rate value; see table below.  |
| <b>See also</b>         | –  |

## Baud Rate Values

| Value (CLI Abbreviation) | Baud Rate  |
|--------------------------|------------|
| B_BD_9600<br>(9600)      | 9600 baud  |
| B_BD_19200<br>(19200)    | 19200 baud |
| B_BD_38400<br>(38400)    | 38400 baud |
| B_BD_57600<br>(57600)    | 57600 baud |

## Administration Functions

The following functions are used to get information about the system under test and to lock and unlock resources:

| Function                                  | Result   |
|---|--|
| <i>"BestVersionGet" on page 31</i>        | Checks the version of a component of the card. |
| <i>"BestCapabilityCheck" on page 24</i>   | Checks if a card capability is enabled.        |
| <i>"BestCapabilityRead" on page 26</i>    | Reads the OR-combined capability code.         |
| <i>"BestSystemInfoGet" on page 30</i>     | Reads out PCI system information.              |
| <i>"BestResourceLock" on page 28</i>      | Locks a resource.                              |
| <i>"BestResourceIsLocked" on page 27</i>  | Checks whether a resource is locked.           |
| <i>"BestResourceUnlock" on page 29</i>    | Unlocks a resource.                            |
| <i>"BestAllResourceUnlock" on page 24</i> | Unlocks all resources.                         |

How to use the functions is described in "Administration" in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestAllResourceUnlock

**Call** `b_errtype BestAllResourceUnlock( b_hndletype handle );`

**Description** Unlocks all locked resources (resets the internal counter incremented by each `BestResourceLock` call). This function can be used to re-establish the communication if the program hangs.

**CLI Equivalent** `BestAllResourceUnlock`

**CLI Abbreviation** `allresunlock`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestResourceUnlock*” on page 29  
 “*BestResourceIsLocked*” on page 27  
 “*BestResourceLock*” on page 28

## BestCapabilityCheck

**Call** `b_errtype BestCapabilityCheck(  
                   b_hndletype handle,  
                   b_int32 capa_code );`

**Description** Checks whether or not a capability of the testcard is enabled.

**CLI Equivalent** `BestCapabilityCheck capa_code=<capa_code>`

**CLI Abbreviation** `capachk code=<capa_code>`

**Return Value** **B\_E\_OK** – all checked capabilities are enabled.

**B\_E\_NO\_CAPABILITY** – one or more capabilities are not enabled.

**Input Parameters** **handle** Handle to identify the session.

**capa\_code** Value of OR-combined capability codes; see “*Capability Code Values*” on page 25.

**See also** “*BestCapabilityRead*” on page 26



## Capability Code Values

| Capability Code Value (CLI Abbreviation)  | Check for   |
|---|---|
| B_CAPABILITY_ALL<br>(capaall, 0xFFFF F800)  | all capabilities  |
| B_CAPABILITY_NO<br>(capano, 0)  | no capability   |
| B_CAPABILITY_64_BIT<br>(capa64, 0x0010 0000\h)  | 64-bit capability   |
| B_CAPABILITY_66_MHZ_AN<br>(capa66an)  | 66-MHz-capability for the Analyzer  |
| B_CAPABILITY_66_MHZ_EX<br>(capa66ex)  | 66-MHz-capability for the Exerciser   |
| B_CAPABILITY_ANALYZER<br>(capaan, 0x0002 0000\h)  | the Analyzer capability   |
| B_CAPABILITY_EXERCISER<br>(capaex, 0x0004 0000\h)   | the Exerciser capability  |
| B_CAPABILITY_HOSTINT<br>(capahint, 0x0008 0000\h)   | host interface capability (required for CPU port, static I/O and host access functions) |
| B_CAPABILITY_TRACEDEPTH_NONE<br>(0x0000 0000\h)<br><br>B_CAPABILITY_TRACEDEPTH_32k<br>(0x0000 0100\h)<br><br>B_CAPABILITY_TRACEDEPTH_64k<br>(0x0000 0200\h)<br><br>B_CAPABILITY_TRACEDEPTH_128k<br>(0x0000 0300\h)<br><br>B_CAPABILITY_TRACEDEPTH_256k<br>(0x0000 0400\h)<br><br>B_CAPABILITY_TRACEDEPTH_512k<br>(0x0000 0500\h)<br><br>B_CAPABILITY_TRACEDEPTH_1M<br>(0x0000 0600\h)<br><br>B_CAPABILITY_TRACEDEPTH_2M<br>(0x0000 0700\h)<br><br>B_CAPABILITY_TRACEDEPTH_4M<br>(0x0000 0800\h) | trace memory capabilities   |

| Capability Code Value (CLI Abbreviation)     | Check for                      |
|--|--------------------------------|
| B_CAPABILITY_PERFSEQ_NONE<br>(0x0000 0000\h) | available performance measures |
| B_CAPABILITY_PERFSEQ_1<br>(0x0000 1000\h)    |                                |
| B_CAPABILITY_PERFSEQ_2<br>(0x0000 2000\h)    |                                |
| B_CAPABILITY_PERFSEQ_3<br>(0x0000 3000\h)    |                                |
| B_CAPABILITY_PERFSEQ_4<br>(0x0000 4000\h)    |                                |
| B_CAPABILITY_PERFSEQ_5<br>(0x0000 5000\h)    |                                |
| B_CAPABILITY_PERFSEQ_6<br>(0x0000 6000\h)    |                                |
| B_CAPABILITY_PERFSEQ_8<br>(0x0000 8000\h)    |                                |

## BestCapabilityRead

**Call** `b_errtype BestCapabilityRead(  
          b_handletype  handle,  
          b_int32      *capa_code );`

**Description** Reads the OR-combined capability codes.

**CLI Equivalent** BestCapabilityRead

**CLI Abbreviation** caparead

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**Output Parameters** **capa\_code** Value of the OR-combined capability codes; see “*Capability Code Values*” on page 25.

**See also** “*BestCapabilityCheck*” on page 24

## BestResourceIsLocked

**Call**     `b_errtype BestResourceIsLocked(  
                  b_handletype     handle,  
                  b_resourcetype    resource,  
                  b_int32           *lock_count  
                  b_porttype        *lock_port );`

**Description**     Checks whether a resource has been locked by which port and how many times in reference to the current session.

**CLI Equivalent**     `BestResourceIsLocked resource=<resource>`

**CLI Abbreviation**     `resislocked res=<resource>`

**Return Value**     Error code; see “*b\_errtype*” on page 250.

**Input Parameters**     **handle**     Handle to identify the session.

**resource**     Resource to be checked; see “*b\_resourcetype*” on page 272.

**Output Parameters**     **lock\_count**     Number of calls that have already locked the resource. If the resource is unlocked, the value is 0.

**lock\_port**     Port that currently locks the resource; see “*b\_porttype*” on page 270. If B\_PORT\_CURRENT is returned, the resource is locked by the current session.

**See also**     “*BestResourceUnlock*” on page 29  
                  “*BestAllResourceUnlock*” on page 24  
                  “*BestResourceLock*” on page 28  
                  “*BestPing*” on page 22

## BestResourceLock

**Call**     `b_errtype BestResourceLock(  
                  b_handletype    handle,  
                  b_resourcetype   resource );`

**Description**    Locks a resource.

This function fails if the resource is already locked by another port. The function does not fail if it is locked by the same port. Each time one resource is locked, an internal counter is incremented. This counter can be queried using “*BestResourceIsLocked*” on page 27.

**CLI Equivalent**    `BestResourceLock resource=<resource>`

**CLI Abbreviation**    `reslock res=<resource>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**handle**     Handle to identify the session.

**resource**     Resource to be locked; see “*b\_resourcetype*” on page 272.

**See also**     “*BestResourceUnlock*” on page 29  
              “*BestAllResourceUnlock*” on page 24

## BestResourceUnlock

**Call** `b_errtype BestResourceUnlock(  
    b_handletype handle,  
    b_resourcetype resource );`

**Description** Unlocks a resource gradually.

The internal counter is decremented. If the counter is at 0, the resource is unlocked.

**CLI Equivalent** `BestResourceUnlock resource=<resource>`

**CLI Abbreviation** `resunlock res=<resource>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**resource** Resource to be unlocked; see “*b\_resourcetype*” on page 272.

**See also** “*BestResourceIsLocked*” on page 27  
“*BestResourceLock*” on page 28  
“*BestAllResourceUnlock*” on page 24

## BestSystemInfoGet

**Call** `b_errtype BestSystemInfoGet (`  
           `b_handletype      handle,`  
           `b_systeminfotype  infoprop,`  
           `b_int32              *value );`

**Description** Reads information like bus width or bus speed on the PCI system under test.

**CLI Equivalent** `BestSystemInfoGet infoprop=<infoprop>`

**CLI Abbreviation** `siget prop=<infoprop>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**infoprop** Information property to be read; see “*b\_systeminfotype*” on page 285.

**Output Parameters** **value** Value of the property; see “*b\_systeminfotype*” on page 285.

**See also** “*BestVersionGet*” on page 31

## BestVersionGet

**Call**     `b_errtype BestVersionGet (`  
                  `b_handletype        handle,`  
                  `b_versionproptype versionprop,`  
                  `b_charptrtype        *string );`

**Description**     Reads the version/date information stored on the EEPROMs of the testcards. This information is needed to check consistency between the firmware/HW and the C-API code.

**CLI Equivalent**     `BestVersionGet versionprop=<versionprop>`

**CLI Abbreviation**     `vget prop=<versionprop>`

**Return Value**     Error code; see “*b\_errtype*” on page 250.

**Input Parameters**     **handle**     Handle to identify the session.

**versionprop**     Version property to be read; see “*b\_versionproptype*” on page 301.

**Output Parameters**     **string**     Version or date string. This string is statically allocated and is overwritten each time this function is called.

**See also**     “*BestSystemInfoGet*” on page 30

# Power Up and Reset Control Functions

The following functions are used to control power up and reset:

| Function                                   | Result                                      |
|--|---|
| <i>"BestPowerUpPropSet" on page 37</i>     | Sets a power up property.                   |
| <i>"BestPowerUpPropGet" on page 36</i>     | Gets a power up property.                   |
| <i>"BestAllPropStore" on page 34</i>       | Stores current settings as user defaults.   |
| <i>"BestAllPropLoad" on page 34</i>        | Loads user defaults as current settings.    |
| <i>"BestAllPropDefaultLoad" on page 33</i> | Loads factory defaults as current settings. |
| <i>"BestSMReset" on page 37</i>            | Resets all state machines.                  |
| <i>"BestBoardReset" on page 36</i>         | Resets the testcard.                        |
| <i>"BestBoardPropSet" on page 35</i>       | Sets a testcard property.                   |
| <i>"BestBoardPropGet" on page 35</i>       | Gets a testcard property.                   |

How to use the functions is described in *"Power-Up and Reset Control"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.



## BestAllPropDefaultLoad

**Call** `b_errtype BestAllPropDefaultLoad( b_handletype handle );`

**Description** Loads the factory defaults as current settings.

**NOTE** This function affects the current decoder settings and may not to be called while the card is under operation. First power off the PCI system under test or unplug the card.

Properties that affect settings of configuration space header registers will be altered only if the power up property B\_PU\_CONFRESTORE is set to 0. This property can be set with “*BestPowerUpPropSet*” on page 37.

**CLI Equivalent** `BestAllPropDefaultLoad`

**CLI Abbreviation** `aprpdefload`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestAllPropLoad*” on page 34  
“*BestPowerUpPropGet*” on page 36  
“*b\_puproptype*” on page 271

## BestAllPropLoad

**Call** `b_errtype BestAllPropLoad( b_handletype handle );`

**Description** Loads the user defaults as current settings.

**NOTE** Properties that affect settings of configuration space header registers will be altered only if the power up property `B_PU_CONFRESTORE` is set to 0. This property can be set with *“BestPowerUpPropSet” on page 37*.

**CLI Equivalent** `BestAllPropLoad`

**CLI Abbreviation** `aprpload`

**Return Value** Error code; see *“b\_errtype” on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**See also** *“BestAllPropDefaultLoad” on page 33*  
*“BestPowerUpPropGet” on page 36*  
*“b\_puproptype” on page 271*

## BestAllPropStore

**Call** `b_errtype BestAllPropStore( b_handletype handle );`

**Description** Stores the current settings of the testcard as user defaults. The settings are then used after power up or after calling *“BestAllPropLoad” on page 34*.

**CLI Equivalent** `BestAllPropStore`

**CLI Abbreviation** `aprpstore`

**Return Value** Error code; see *“b\_errtype” on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**See also** –

## BestBoardPropGet

**Call** `b_errtype BestBoardPropGet (`  
     `b_handletype     handle,`  
     `b_boardproptype boardprop,`  
     `b_int32           *value );`

**Description** Reads the testcard properties. They determine the testcard mode for PCI resets (RST# mode) and error reporting.

**CLI Equivalent** `BestBoardPropGet boardprop=<boardprop>`

**CLI Abbreviation** `bdprpget prop=<boardprop>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**boardprop** Property to get; see “*b\_boardproptype*” on page 241.

**Output Parameters** **value** Value of the property; see “*b\_boardproptype*” on page 241.

**See also** “*BestPowerUpPropGet*” on page 36

## BestBoardPropSet

**Call** `b_errtype BestBoardPropSet (`  
     `b_handletype     handle,`  
     `b_boardproptype boardprop,`  
     `b_int32           value );`

**Description** Sets the testcard properties. They determine the testcard mode for PCI resets (RST# mode) and error reporting.

**CLI Equivalent** `BestBoardPropSet boardprop=<boardprop> value=<value>`

**CLI Abbreviation** `bdprpset prop=<boardprop> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**boardprop** Property to be set; see “*b\_boardproptype*” on page 241.

**value** Value to which the property is set; see “*b\_boardproptype*” on page 241.

**See also** “*BestPowerUpPropSet*” on page 37

## BestBoardReset

**Call** `b_errtype BestBoardReset( b_handletype handle );`

**Description** Performs a testcard reset. This is equivalent to re-powering the testcard. However, decoder and configuration space remain unchanged.

**CLI Equivalent** `BestBoardReset`

**CLI Abbreviation** `bdreset`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestBoardReset*” on page 36  
“*BestSMReset*” on page 37

## BestPowerUpPropGet

**Call** `b_errtype BestPowerUpPropGet (
 b_handletype handle,
 b_puproptype pu_prop,
 b_int32 *value );`

**Description** Reads the current value of a power up property.

**CLI Equivalent** `BestPowerUpPropGet pu_prop=<pu_prop> value=<value>`

**CLI Abbreviation** `puprpget prop=<pu_prop> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**pu\_prop** Specifies the power up property to be read; see “*b\_puproptype*” on page 271.

**Output Parameters** **value** Value of the property; see “*b\_puproptype*” on page 271.

**See also** “*BestPowerUpPropSet*” on page 37  
“*BestBoardPropGet*” on page 35

## BestPowerUpPropSet

**Call** `b_errtype BestPowerUpPropSet (`  
           `b_handletype handle,`  
           `b_puproptype pu_prop,`  
           `b_int32 value );`

**Description** Sets a power up property.

**CLI Equivalent** `BestPowerUpPropSet pu_prop=<pu_prop> value=<value>`

**CLI Abbreviation** `puprpset prop=<pu_prop> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**pu\_prop** Specifies the power up property to be set; see “*b\_puproptype*” on page 271.

**value** Value to which the property is set; see “*b\_puproptype*” on page 271.

**See also** “*BestPowerUpPropGet*” on page 36  
 “*BestBoardPropSet*” on page 35

## BestSMReset

**Call** `b_errtype BestSMReset ( b_handletype handle );`

**Description** Resets the state machines of master, target, and analyzer. It ignores bus transactions that may currently be in progress. Therefore, it should be used only in the case of an error.

**CLI Equivalent** `BestSMReset`

**CLI Abbreviation** `smreset`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestBoardReset*” on page 36

# Card Status Functions

The following functions are used to access the testcard's status register:

| Function                               | Result   |
|--|--|
| <i>"BestStatusRegGet" on page 39</i>   | Reads the testcard's status register.          |
| <i>"BestStatusRegClear" on page 38</i> | Clears bits in the testcard's status register. |

How to use the functions is described in *"Card Status Register Access"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestStatusRegClear

**Call** `b_errtype BestStatusRegClear(  
    b_handletype handle,  
    b_int32 clearpattern );`

**Description** Clears bits in the testcard status register. The bits to be cleared are specified by a 1 in the bit mask (clear pattern).

**CLI Equivalent** `BestStatusRegClear clearpattern=<clearpattern>`

**CLI Abbreviation** `sregclear clear=<clearpattern>`

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**clearpattern** Pattern specifying the bits to be cleared.

- 1 – clears the bit
- 0 – the bit remains unchanged

For example: `11111111\b` clears all bits.

**See also** *"BestStatusRegGet" on page 39*

## BestStatusRegGet

**Call**     `b_errtype BestStatusRegGet (`  
                 `b_handletype    handle,`  
                 `b_int32            *value );`

**Description**   Reads the testcard status register contents.

**NOTE**        This is not the status register of the configuration space header.

**CLI Equivalent**   `BestStatusRegGet`

**CLI Abbreviation**   `sregget`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**   **handle**    Handle to identify the session.

**Output Parameters**   **value**    Contents of the status register; see “*Testcard Status Register*” on page 40.

**See also**        “*BestStatusRegClear*” on page 38

## Testcard Status Register

Type RO = read-only

Type RC = read/clear

| Bit     | Type | Default | Description        | Value  |
|---------|------|---------|--------------------|--|
| [0]     | RO   | 0       | Master Run Bit     | 1= master active                                       |
| [1]     | RO   | 0       | Target Active Bit  | 1= target active                                       |
| [2]     | RO   | 0       | Observer Run Bit   | Always 1   |
| [3]     | RO   | 0       | Trace Run Bit      | 1= trace memory in run mode                            |
| [4]     | RO   | 0       | Protocol Error     | 1= protocol error detected                             |
| [5]     | RC   | 0       | Data Compare Error | 1= data compare error detected                         |
| [6]     | RC   | 0       | Functional Error   | 1= functional error during command                     |
| [7]     | RC   | 0       | Block aborted.     | 1= at least one block has not been executed completely |
| [8]     | RC   | 0       | INTA asserted      | 1= asserted  |
| [9]     | RC   | 0       | INTB asserted      | 1= asserted  |
| [10]    | RC   | 0       | INTC asserted      | 1= asserted  |
| [11]    | RC   | 0       | INTD asserted      | 1= asserted  |
| [12]    | RO   | 0       | Test run failed    | 1= test run has failed                                 |
| [13]    | RO   | 0       | Reserved           |  |
| [14]    | RC   | 0       | Self Traffic       | 1=master has accessed its own target                   |
| [15]    | RO   | 0       | Reserved           |  |
| [16]    | RC   | 0       | PCI Reset          | 1=PCI Reset has occurred                               |
| [17:31] | RO   | 0       | Reserved           |  |



# PCI Analyzer Functions

The PCI Analyzer Functions are divided into the following sections:

- *“Protocol Observer Functions” on page 42*
- *“Timing Check Functions” on page 50*
- *“Pattern Term Function” on page 62*
- *“Trace Memory Trigger Sequencer Functions” on page 67*
- *“Trace Memory Functions” on page 75*
- *“Performance Measure Functions” on page 84*

# Protocol Observer Functions

The following functions are used for the protocol observer:

| Function                                   | Result  |
|--|---|
| <i>"BestObsMaskSet" on page 47</i>         | Sets an individual error mask bit.                                |
| <i>"BestObsMaskGet" on page 46</i>         | Reads an individual error mask bit.                               |
| <i>"BestObsPropDefaultSet" on page 48</i>  | Sets all observer properties to their default values.             |
| <i>"BestObsStatusGet" on page 49</i>       | Reads the observer registers.                                     |
| <i>"BestObsErrResultGet" on page 44</i>    | Returns all errors found in the error registers in a text string. |
| <i>"BestObsBitPositionFind" on page 43</i> | Finds a set bit in the error registers.                           |
| <i>"BestObsErrStringGet" on page 45</i>    | Returns the protocol rule text for a specific error.              |
| <i>"BestObsRuleErrTypeGet" on page 48</i>  | Returns the protocol rule identifier for a specific error.        |
| <i>"BestObsStatusClear" on page 49</i>     | Clears the current status of the observer.                        |

How to use the functions is described in *"Protocol Observer Programming"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestObsBitPositionFind

**Call**    `b_errtype BestObsBitPositionFind(  
                   b_handletype    handle,  
                   b_int32        *errstat,  
                   b_int32        *errstat2,  
                   b_int32        *bitposition );`

**Description**    Returns the position of a bit set in the register pair with each call and resets this bit in the register pair. The register pair values can be requested with “*BestObsStatusGet*” on page 49 and the bit position converted into a rule using “*BestObsRuleErrTypeGet*” on page 48.

**CLI Equivalent**    No CLI equivalent.

**CLI Abbreviation**    No CLI abbreviation.

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**errstat, errstat2**    Error status register pair; see “*b\_obsstatustype*” on page 267. This input parameters affect the output.

**bitposition**    Position of the bit in the register pair. If no bit is set, the function returns 0xFFFFFFFF.

**See also**    –

## BestObsErrResultGet

**Call**    `b_errtype BestObsErrResultGet (`  
              `b_handletype    handle,`  
              `b_int32            errstat,`  
              `b_int32            errstat2,`  
              `b_charptrtype    *errstring );`

**Description**    Returns a text string containing all errors found in the specified register pair. The register pair values can be requested with “*BestObsStatusGet*” on page 49.

**CLI Equivalent**    `BestObsErrResultGet errstat=<errstat> errstat2=<errstat2>`

**CLI Abbreviation**    `oerresultget errstat=<errstat> errstat2=<errstat2>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**errstat, errstat2**    Error register pair; see “*b\_obsstatustype*” on page 267.

**Output Parameters**    **errstring**    Text string.

**See also**    –

## BestObsErrStringGet

**Call** `b_errtype BestObsErrStringGet (`  
           `b_handletype     handle,`  
           `b_int32         bitposition,`  
           `b_charptrtype   *errstring );`

**Description** Returns the text string specified by the bit position of a rule violation, returned from “*BestObsStatusGet*” on page 49.

**Example:**

If a rule 2 violation is indicated by one of the error registers and you call this function with this value, the function will return “IRDY# must not be asserted on the same clock edge that FRAME# is asserted, but one or more clocks later”.

**CLI Equivalent** `BestObsErrStringGet bitposition=<bitposition>`

**CLI Abbreviation** `oestrget pos=<bitposition>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**bitposition** Bit position that specifies the rule.

**Output Parameters** **errstring** Text string describing the rule.

**See also** “*b\_obsruletype*” on page 266

## BestObsMaskGet

**Call**     `b_errtype BestObsMaskGet (  
                  b_handletype  handle,  
                  b_obsruletype  obsrule,  
                  b_int32       *value );`

**Description**     Reads the mask bit of the specified rule.

**CLI Equivalent**     `BestObsMaskGet obsrule=<obsrule>`

**CLI Abbreviation**     `omget rule=<obsrule>`

**Return Value**     Error code; see “*b\_errtype*” on page 250.

**Input Parameters**     **handle**     Handle to identify the session.

**obsrule**     Protocol rule identifier; see “*b\_obsruletype*” on page 266.

**Output Parameters**     **value**     Value of the mask bit:

- 0 (default) – rule not masked (enabled)
- 1 – rule masked (disabled)

**See also**     “*BestObsMaskSet*” on page 47

## BestObsMaskSet

**Call** `b_errtype BestObsMaskSet (`  
     `b_handletype handle,`  
     `b_obsruletype obsrule,`  
     `b_int32 value );`

**Description** Masks protocol rules, so that their violations are not indicated in the observer's first error register.

**NOTE** Violations of masked rules are not totally ignored: bit 2 of the observer status register is set and the rule violation is indicated in the accumulated error register. For status and error registers, see "*b\_obsstatustype*" on page 267.

**CLI Equivalent** `BestObsMaskSet obsrule=<obsrule> value=<value>`

**CLI Abbreviation** `omset rule=<obsrule> val=<value>`

**Return Value** Error code; see "*b\_errtype*" on page 250.

**Input Parameters** **handle** Handle to identify the session.

**obsrule** Protocol rule to be masked; see "*b\_obsruletype*" on page 266.

**value** Value to be entered in the mask:

- 0 (default) – rule not masked (enabled)
- 1 – rule masked (disabled)

**See also** "*BestObsMaskGet*" on page 46

## BestObsPropDefaultSet

**Call** `b_errtype BestObsPropDefaultSet( b_handletype handle );`

**Description** Sets all mask bits to 0, so that all protocol rules will be observed, except SEM\_8 and SEM\_9, which are masked by default.

**CLI Equivalent** `BestObsPropDefaultSet`

**CLI Abbreviation** `oprpdefset`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestObsMaskSet*” on page 47  
“*b\_obsruletype*” on page 266

## BestObsRuleErrTypeGet

**Call** `b_errtype BestObsRuleErrTypeGet(  
    b_handletype handle  
    b_int32 bitposition,  
    b_obsruletype *obsrule );`

**Description** Returns the rule identifier specified by the given bit position.

### Example:

If a rule 2 violation is indicated by one of the error registers and you call this function with this value, the function will return the numeric constant of B\_R\_IRDY\_0. The numeric constants can be used as parameters in other function calls (for example in “*BestObsMaskGet*” on page 46).

**CLI Equivalent** `BestObsRuleErrTypeGet bitposition=<bitposition>`

**CLI Abbreviation** `osruleget pos=<bitposition>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**bitposition** Bit position specifying the rule in one of the error registers.

**Output Parameters** **obsrule** Protocol rule identifier; see “*b\_obsruletype*” on page 266.

**See also** –



## BestObsStatusClear

**Call** `b_errtype BestObsStatusClear( b_handletype handle );`

**Description** Clears the observer status register, and the first and accumulated error registers; see “*b\_obsstatustype*” on page 267.

**CLI Equivalent** `BestObsStatusClear`

**CLI Abbreviation** `osclear`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestObsStatusGet*” on page 49

## BestObsStatusGet

**Call** `b_errtype BestObsStatusGet(  
                   b_handletype handle,  
                   b_obsstatustype obsstatus,  
                   b_int32 *value );`

**Description** Reads the observer registers. Use this function to determine

- the first error that has occurred during a run,
- all errors that have occurred during a run, or
- the observer status (running/stopped, errors/no errors).

To convert the value passed back into a meaningful text string, use function “*BestObsErrStringGet*” on page 45.

To reset the registers, use the function “*BestObsStatusClear*” on page 49.

**CLI Equivalent** `BestObsStatusGet obsstatus=<obsstatus>`

**CLI Abbreviation** `osget stat=<obsstatus>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**obsstatus** Observer register to be read; see “*b\_obsstatustype*” on page 267.

**Output Parameters** **value** Value of the queried register.

**See also** –

# Timing Check Functions

**NOTE** At present the timing check is only available for 33 MHz PCI busses.

The following functions are used to program the timing check:

| Function                                    | Result  |
|---|---|
| <i>"BestTimCheckGenPropSet" on page 53</i>  | Sets a generic timing check property.   |
| <i>"BestTimCheckGenPropGet" on page 52</i>  | Reads a generic timing check property.  |
| <i>"BestTimCheckMaskSet" on page 55</i>     | Sets a signal bit mask.   |
| <i>"BestTimCheckMaskGet" on page 54</i>     | Reads a signal bit mask.  |
| <i>"BestTimCheckDefaultSet" on page 51</i>  | Sets signal bit masks and timing check properties to default values.          |
| <i>"BestTimCheckStatusGet" on page 61</i>   | Reads the current status of the timing check.                                 |
| <i>"BestTimCheckStatusClear" on page 60</i> | Clears the status registers of the timing check.                              |
| <i>"BestTimCheckPropSet" on page 58</i>     | Sets setup time and hold time values in the preparation register.             |
| <i>"BestTimCheckPropGet" on page 57</i>     | Reads setup time and hold time values from the preparation register.          |
| <i>"BestTimCheckRead" on page 59</i>        | Copies the timing check properties from the card to the preparation register. |
| <i>"BestTimCheckProg" on page 56</i>        | Copies the timing check properties from the preparation register to the card. |
| <i>"BestTimCheckResultGet" on page 59</i>   | Returns the current results of the timing check in a text string.             |

How to use the functions is described in *"Timing Check Programming"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestTimCheckDefaultSet

**Call** `b_errtype BestTimCheckDefaultSet( b_handletype handle );`

**Description** Sets the following properties to default values:

- signal bit mask
- timing check properties
- generic timing check properties

**CLI Equivalent** `BestTimCheckDefaultSet`

**CLI Abbreviation** `tcdefset`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestTimCheckMaskSet*” on page 55  
“*BestTimCheckPropSet*” on page 58  
“*BestTimCheckGenPropSet*” on page 53  
“*b\_tcproptype*” on page 292  
“*b\_tcgengenproptype*” on page 291

## BestTimCheckGenPropGet

**Call**     `b_errtype BestTimCheckGenPropGet (`  
                  `b_handletype        handle,`  
                  `b_tcgenproptype    tcgenprop,`  
                  `b_int32                *value );`

**Description**     Reads a generic timing check property.

The generic timing check property determines whether the PCI Specification values of setup and hold time or the values in the preparation register are used.

**CLI Equivalent**     `BestTimCheckGenPropGet tcgenprop=<tcgenprop>`

**CLI Abbreviation**     `tcgprpget prop=<tcgenprop>`

**Return Value**     Error code; see “*b\_errtype*” on page 250.

**Input Parameters**     **handle**     Handle to identify the session.

**tcgenprop**     Generic property to be read; see “*b\_tcgenproptype*” on page 291.

**Output Parameters**     **value**     Value of the generic property; see “*b\_tcgenproptype*” on page 291.

**See also**     “*BestTimCheckDefaultSet*” on page 51  
                  “*BestTimCheckGenPropSet*” on page 53

## BestTimCheckGenPropSet

**Call**    `b_errtype BestTimCheckGenPropSet (`  
                  `b_handletype        handle,`  
                  `b_tcgenproptype    tcgenprop,`  
                  `b_int32                value );`

**Description**    Sets a generic timing check property.

The generic timing check property determines whether the PCI Specification values of setup and hold time or the values in the preparation register are used.

**CLI Equivalent**    `BestTimCheckGenPropSet tcgenprop=<tcgenprop> value=<value>`

**CLI Abbreviation**    `tcgprpset prop=<tcgenprop> val=<value>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**tcgenprop**    Generic property to be set; see “*b\_tcgenproptype*” on page 291.

**value**    Value of the generic property; see “*b\_tcgenproptype*” on page 291.

**See also**    “*BestTimCheckDefaultSet*” on page 51  
               “*BestTimCheckGenPropGet*” on page 52

## BestTimCheckMaskGet

**Call**

```
b_errtype BestTimCheckMaskGet (
    b_handletype  handle,
    b_signaltype  signal,
    b_int32       *value );
```

**Description** Reads the bit mask of a signal.

**CLI Equivalent** BestTimCheckMaskGet signal=<signal>

**CLI Abbreviation** tcmget sig=<signal>

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**signal** Signal of which the bit mask is to be read; see “*b\_signaltype* (for Timing Check)” on page 273.

**Output Parameters** **value** Bit mask of each signal. The signal lengths can range from 1 to 32 bits (for example, signal AD32). The value contains valid mask bits matching the length of the signal.

Value of the mask bit:

- 0 (default) – not masked (enabled)
- 1 – masked (disabled)

**See also** “*BestTimCheckDefaultSet*” on page 51  
 “*BestTimCheckMaskSet*” on page 55

## BestTimCheckMaskSet

**Call** `b_errtype BestTimCheckMaskSet (`  
     `b_handletype handle,`  
     `b_signaltype signal,`  
     `b_int32 value );`

**Description** Sets the bit mask for a signal.

A masked signal can no longer be used as a trigger, but the violation will be shown in the timing check report.

**CLI Equivalent** `BestTimCheckMaskSet signal=<signal> value=<value>`

**CLI Abbreviation** `tcmset sig=<signal> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**signal** Signal to be masked; see “*b\_signaltype (for Timing Check)*” on page 273.

**value** Bit mask of each signal. The signal lengths can range from 1 to 32 bits (for example, signal AD32). The value must contain valid mask bits matching the length of the signal.

Value of the mask bit:

- 0 (default) – not masked (enabled)
- 1 – masked (disabled)

**See also** “*BestTimCheckDefaultSet*” on page 51  
 “*BestTimCheckMaskGet*” on page 54

## BestTimCheckProg

**Call** `b_errtype BestTimCheckProg( b_handletype handle );`

**Description** Writes the timing check properties from the preparation register to the testcard. This function also clears the timing check status register and performs a consistency check on the preparation register contents.

Before calling this function, use “*BestTimCheckPropSet*” on page 58 to program the preparation register.

**NOTE** If you want to program your own setup time and hold time values from the preparation register instead of the time values of the PCI Specification, you first need to set the generic timing check property B\_TCGEN\_SPEC to 0. To set this property, use “*BestTimCheckGenPropSet*” on page 53.

**CLI Equivalent** BestTimCheckProg

**CLI Abbreviation** tcprog

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*b\_tcgenproptype*” on page 291  
 “*b\_tcproptype*” on page 292  
 “*b\_tcstatustype*” on page 292  
 “*BestTimCheckRead*” on page 59



## BestTimCheckPropGet

**Call**     `b_errtype BestTimCheckPropGet (`  
                  `b_handletype    handle,`  
                  `b_tcproptype    tcprop,`  
                  `b_int32            *value );`

**Description**     Reads the setup time and the hold time values from the preparation register.

To get current values, first use “*BestTimCheckRead*” on page 59 to read the timing check properties from the testcard to the preparation register.

**CLI Equivalent**     `BestTimCheckPropGet tcprop=<tcprop>`

**CLI Abbreviation**   `tcprpget prop=<tcprop>`

**Return Value**     Error code; see “*b\_errtype*” on page 250.

**Input Parameters**   **handle**     Handle to identify the session.

**tcprop**     Property to be read; see “*b\_tcproptype*” on page 292.

**Output Parameters**   **value**     Value of the property; see “*b\_tcproptype*” on page 292.

**See also**     “*BestTimCheckDefaultSet*” on page 51  
                  “*BestTimCheckPropSet*” on page 58

## BestTimCheckPropSet

**Call** `b_errtype BestTimCheckPropSet (`  
     `b_handletype    handle,`  
     `b_tcproptype    tcprop,`  
     `b_int32          value );`

**Description** Defines the setup time and the hold time values in the preparation register.

To write these values to the testcard, use “*BestTimCheckProg*” on page 56.

**NOTE** Setting these values takes effect only if the property B\_TCGEN\_SPEC is set to 0. To set this property, use “*BestTimCheckGenPropSet*” on page 53.

**CLI Equivalent** `BestTimCheckPropSet tcprop=<tcprop> value=<value>`

**CLI Abbreviation** `tcprpset prop=<tcprop> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**tcprop** Property to be set; see “*b\_tcproptype*” on page 292 .

**value** Value of the property; see “*b\_tcproptype*” on page 292.

**See also** “*BestTimCheckDefaultSet*” on page 51  
 “*BestTimCheckPropGet*” on page 57

## BestTimCheckRead

**Call** `b_errtype BestTimCheckRead( b_handletype handle );`

**Description** Reads the timing check properties from the testcard to the preparation register.

To read the timing check properties from the preparation register, use “*BestTimCheckPropGet*” on page 57.

**CLI Equivalent** `BestTimCheckRead`

**CLI Abbreviation** `tcread`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*b\_tcproptype*” on page 292  
“*BestTimCheckProg*” on page 56

## BestTimCheckResultGet

**Call** `b_errtype BestTimCheckResultGet(  
          b_handletype handle,  
          b_charptrtype *errortext );`

**Description** Returns a text string containing the violated signals.

**NOTE** Before using this function, ensure that the timing check can be performed properly under the current conditions. Use “*BestTimCheckStatusGet*” on page 61 to query B\_TC\_ERROR of property B\_TC\_TCSTAT. If the error flag is set, the returned text string may contain wrong information.

**CLI Equivalent** `BestTimCheckResultGet`

**CLI Abbreviation** `tcresultget`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**Output Parameters** **errortext** Text string.

**See also** “*b\_tcstatustype*” on page 292

## BestTimCheckStatusClear

**Call** `b_errtype BestTimCheckStatusClear( b_handletype handle );`

**Description** Clears the status register contents (timing check result, violation flag) of the timing check.

**NOTE** The flag indicating instable frequency/calibration error B\_TC\_ERROR remains uncleared. To clear this bit, first eliminate the error. Then execute “*BestTimCheckProg*” on page 56. The flag will be cleared, if the function has been executed successfully.

**CLI Equivalent** `BestTimCheckStatusClear`

**CLI Abbreviation** `tcsclear`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestTimCheckStatusGet*” on page 61  
“*b\_tcstatustype*” on page 292

## BestTimCheckStatusGet

**Call** `b_errtype BestTimCheckStatusGet (`  
     `b_handletype       handle,`  
     `b_tcstatustype     status,`  
     `b_int32            *value );`

**Description** Reads the status register of the timing check.

Use this function to determine whether the timing check can be performed properly under the current conditions and whether signals have been violated.

The timing check cannot be performed properly if

- the card is not calibrated,
- the bus frequency has changed since the last successful execution of “*BestTimCheckProg*” on page 56,
- the bus frequency is instable.

**NOTE** Always use this function before getting timing check results with “*BestTimCheckResultGet*” on page 59.

**CLI Equivalent** `BestTimCheckStatusGet tcstatus=<status>`

**CLI Abbreviation** `tcsget stat=<status>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**status** Timing check register to be read; see “*b\_tcstatustype*” on page 292.

**Output Parameters** **value** Value of the register; see “*b\_tcstatustype*” on page 292.

**See also** –

# Pattern Term Function

The following function is used for the pattern terms:

| Function                        | Result                   |
|---------------------------------|--------------------------|
| <i>"BestPattSet" on page 62</i> | Programs a pattern term. |

How to use the functions is described in *"Programming the Pattern Terms"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestPattSet

**Call** `b_errtype BestPattSet(`  
     `b_handletype handle,`  
     `b_int32 pattern_ident,`  
     `b_charptrtype pattern );`

**Description** Specifies a pattern term. This term can then be used in the condition strings of a sequencer description table.

**CLI Equivalent** `BestPattSet pattern_ident=<pattern_ident> pattern=<pattern>`

**CLI Abbreviation** `pattset pid=<pattern_ident> patt=<pattern>`

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**pattern\_ident** Pattern term identifier; see *"Pattern Term Identifiers" on page 63*.

**pattern** String containing the logical expression for the condition. The string must be set in quotation marks.

To build pattern terms see

- *"Syntax Diagrams for Numbers and 10XNumbers" on page 64*
- *"Standard Pattern Term Operators" on page 64* and *"Standard Pattern Term Syntax Diagram" on page 65*
- *"Transitional Pattern Term Operators" on page 66* and *"Transitional Pattern Term Syntax Diagram" on page 66*

For specifiable signals, see *"b\_signaltype (List of Signals)" on page 274*.

**See also** –

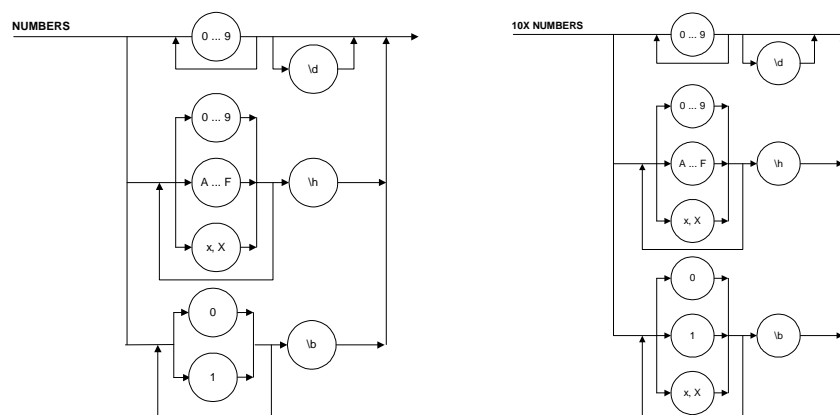
## Pattern Term Identifiers

These identifiers are used to select the pattern term to be programmed when using the function *BestPattSet* on page 62.

| Identifier (CLI Abbreviation) | Description  |
|-------------------------------|--|
| B_PATT_TERM_0 (pt0)           | Both transitional and standard pattern term (to specify this, use <i>BestTracePropSet</i> on page 81).<br>Used for trace memory triggering only. |
| B_PATT_TERM_1 (pt1)           | Standard pattern terms.  |
| ...                           | Used for trace memory triggering, performance analysis, and master conditional start.  |
| B_PATT_TERM_23 (pt23)         |  |

## Syntax Diagrams for Numbers and 10XNumbers

To program pattern terms, hexadecimal, decimal and binary numbers can be used. Signals of type “10X” additionally allow the use of “Don’t Cares” (= x or X). This is expressed in the following syntax diagrams:



These numbers can be used when building logical expressions to program a standard or transitional pattern term.

## Standard Pattern Term Operators

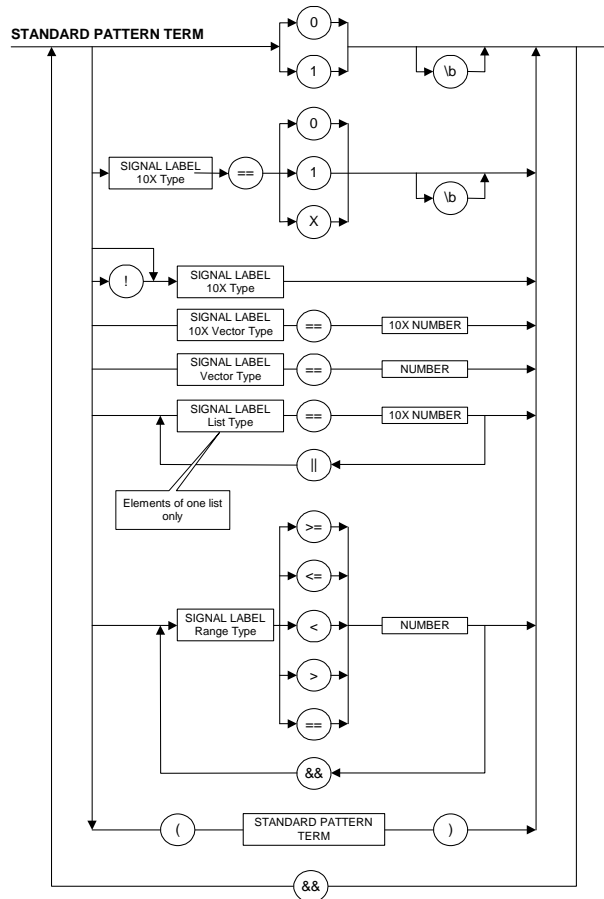
The following table shows the operators that can be used in standard pattern terms to combine the signal labels. The operators appear in the order of their priority:

| Operator | Operation            | Applicable      |
|----------|----------------------|-----------------|
| !        | negation             | for all signals |
| ==       | compare for equality | for all signals |
| &&       | logical AND          | for all signals |

Signals of “10X vector type” can be queried bitwise; see the examples in “*Standard Pattern Term Syntax Diagram*” on page 65.



## Standard Pattern Term Syntax Diagram



### Examples

- "b\_state==3\h && AD32==b8xxx\h"

Detects address phases and addresses in video memory range.

- "FRAME && (b\_state==2 || b\_state==3) && (AD32==b8xxx\h)"

## Transitional Pattern Term Operators

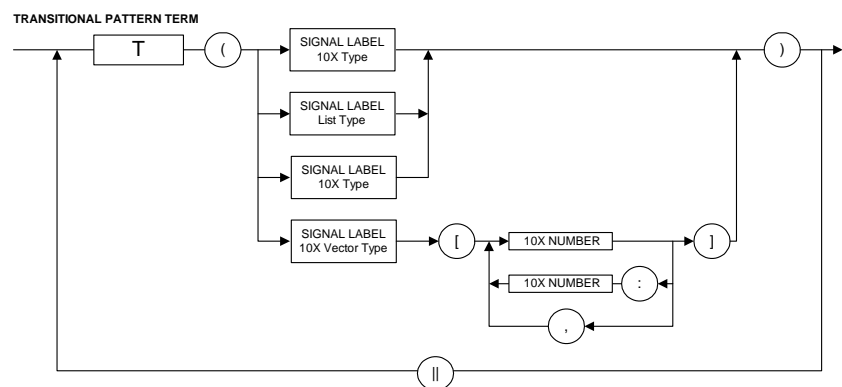
To build the transitional pattern term, use the transition function  $T()$  to determine whether a signal changes its state. Only  $pt0$  can be used as transitional pattern term.

Any signal from the “Signal” column of the table in *b\_signaltype (List of Signals)* may be specified as an argument for this function. Signals of “10X vector type” can be queried bitwise. See the examples in “*Transitional Pattern Term Syntax Diagram*” on page 66.

If multiple signals are to be queried, they can be combined via logical OR:

| Operator | Operation  | Applicable      |
|----------|------------|-----------------|
|          | logical OR | for all signals |

## Transitional Pattern Term Syntax Diagram



### Examples:

- `"T(GNT) || T(REQ)"`  
Detects transitions of the GNT# or REQ# lines.
- `"T(AD32[31:17, 4:2])"`  
Detects toggling signals on address/data lines 2 to 4 and 17 to 31.

# Trace Memory Trigger Sequencer Functions

The following functions are used to program the trace memory trigger sequencer:

| Function  | Result   |
|---|--|
| <i>"BestTrigSeqGenPropDefaultSet" on page 68</i>  | Sets the preload value of the feedback counter to the default value.                             |
| <i>"BestTrigSeqGenPropGet" on page 68</i>         | Reads the preload value of the feedback counter.   |
| <i>"BestTrigSeqGenPropSet" on page 69</i>         | Sets the preload value of the feedback counter.  |
| <i>"BestTrigSeqPropDefaultSet" on page 70</i>     | Sets all properties in the trigger sequencer description table to default values.                |
| <i>"BestTrigSeqTranPropDefaultSet" on page 73</i> | Sets all properties of a transient in the trigger sequencer description table to default values. |
| <i>"BestTrigSeqTranPropSet" on page 74</i>        | Sets a numeric transition property ("state" or "next state").                                    |
| <i>"BestTrigSeqTranCondPropSet" on page 71</i>    | Sets a condition in the trigger sequencer description table.                                     |
| <i>"BestTrigSeqProg" on page 69</i>               | Writes the sequencer description table to sequencer memory.                                      |

How to use the functions is described in *"Sequencer Programming"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestTrigSeqGenPropDefaultSet

**Call** `b_errtype BestTrigSeqGenPropDefaultSet( b_handletype handle );`

**Description** Sets the value of the generic trigger sequencer property to 0. This property determines the preload value of a trace memory trigger sequencer feedback counter.

**CLI Equivalent** `BestTrigSeqGenPropDefaultSet`

**CLI Abbreviation** `tsgenprpdefset`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestTrigSeqGenPropGet*” on page 68  
 “*BestTrigSeqGenPropSet*” on page 69  
 “*b\_trigseqgenproptype*” on page 299

## BestTrigSeqGenPropGet

**Call** `b_errtype BestTrigSeqGenPropGet (
 b_handletype handle,
 b_trigseqgenproptype trigseqgenprop,
 b_int32 *value );`

**Description** Reads the value of the generic trigger sequencer property. This property determines the preload value of the trace memory trigger sequencer feedback counter.

**CLI Equivalent** `BestTrigSeqGenPropGet trigseqgenprop=<trigseqgenprop>`

**CLI Abbreviation** `tsgenprpget prop=<trigseqgenprop>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**trigseqgenprop** Property to be read; see “*b\_trigseqgenproptype*” on page 299.

**Output Parameters** **value** Value of the property; see “*b\_trigseqgenproptype*” on page 299.

**See also** “*BestTrigSeqGenPropDefaultSet*” on page 68  
 “*BestTrigSeqGenPropSet*” on page 69

## BestTrigSeqGenPropSet

**Call** `b_errtype BestTrigSeqGenPropSet (`  
           `b_handletype           handle,`  
           `b_trigseqgenproptype   trigseqgenprop,`  
           `b_int32                   value );`

**Description** Sets the value of the generic trigger sequencer property. This property determines the preload value of the trace memory trigger sequencer feedback counter.

**CLI Equivalent** `BestTrigSeqGenPropSet trigseqgenprop=<trigseqgenprop> value=<value>`

**CLI Abbreviation** `tsgenprpset prop=<trigseqgenprop> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**trigseqgenprop** Property to be set; see “*b\_trigseqgenproptype*” on page 299.

**value** Value the property is set to; see “*b\_trigseqgenproptype*” on page 299.

**See also** “*BestTrigSeqGenPropDefaultSet*” on page 68  
 “*BestTrigSeqGenPropGet*” on page 68

## BestTrigSeqProg

**Call** `b_errtype BestTrigSeqProg( b_handletype handle );`

**Description** Writes the information stored in the trigger sequencer description table to the sequencer memory.

This function also checks whether the transition conditions of one state are consistent. If they are not, the function returns an error.

**CLI Equivalent** `BestTrigSeqProg`

**CLI Abbreviation** `tsprog`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** –

## BestTrigSeqPropDefaultSet

**Call** `b_errtype BestTrigSeqPropDefaultSet( b_handletype handle );`

**Description** Initializes the trace memory trigger sequencer description table and sets all properties to default values.

**CLI Equivalent** `BestTrigSeqPropDefaultSet`

**CLI Abbreviation** `tsprpdefset`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestTrigSeqTranPropDefaultSet*” on page 73

## BestTrigSeqTranCondPropSet

**Call** `b_errtype BestTrigSeqTranCondPropSet (`  
     `b_handletype                    handle,`  
     `b_int32                          transient,`  
     `b_trigseqtrancondproptype      trigseqtrancondprop,`  
     `b_charptrtype                  condition );`

**Description** Sets a condition in the trace memory trigger sequencer description table.

The condition property can be the transition condition, trigger condition, storage qualifier condition, or conditions to decrement and preload the feedback counter.

**CLI Equivalent** `BestTrigSeqTranCondPropSet transient=<transient>`  
`trigseqtrancondprop=<trigseqtrancondprop> condition=<condition>`

**CLI Abbreviation** `tstrancprpset tran=<transient> prop=<trigseqtrancondprop>`  
`con=<condition>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**transient** Transient number (0 ... 255).

**trigseqtrancondprop** Condition property to be set; see “*b\_trigseqtrancondproptype*” on page 300.

**condition** Condition string. The string must be set in quotation marks. See “*Conditions Reference*” on page 72.

**See also** “*BestTrigSeqTranPropDefaultSet*” on page 73  
 “*BestTrigSeqTranPropSet*” on page 74

## Conditions Reference

Conditions are used in the sequencer description tables.

A condition is either true (1) or false (0) and controls the behavior of the specified sequencer. Conditions are specified by means of a logical expression (condition string).

These condition strings can consist of:

- Pattern term identifiers (pt0 ... pt23); see *“Pattern Term Identifiers” on page 63*.
- The terminal count of the sequencer feedback counter (tc).
- Logical operators (logical AND, logical OR, ...).
- True or false settings (“1” or “0”).

### Example:

- `"( !pt0 || pt1 || pt2 ) && tc"`

Valid condition string to program the trigger sequencer.

### Pattern Identifiers for Sequencers

The following table shows the pattern identifiers that can be used in condition strings for the different sequencers:

| Identifier   | Description  |
|--------------|--|
| pt0          | Used as a transitional or standard pattern term (see <i>“b_traceproptype” on page 295</i> ). |
| pt1 ... pt23 | Used as standard pattern terms only.   |
| tc           | Terminal count of its feedback counter.  |
| tcc          | Terminal count of trigger IO feedback counter C  |
| tcd          | Terminal count of trigger IO feedback counter D  |

### Logical Operators

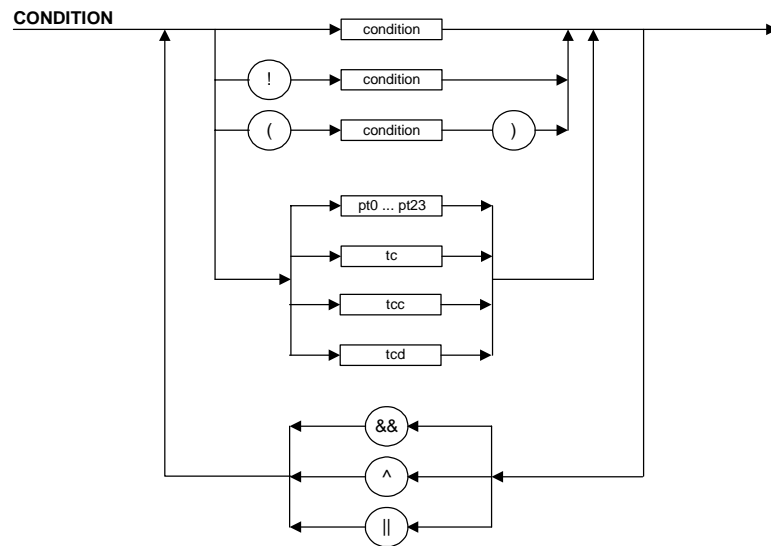
The following table shows the logical operators in the order of their priority:

| Operator | Operation   |
|----------|-------------|
| !        | negation    |
| &&       | logical AND |
| ^        | logical XOR |
|          | logical OR  |

“(“ and “)” can be used to override priorities.



**Condition Syntax Diagram** The syntax of the condition strings is expressed in the following diagram:



## BestTrigSeqTranPropDefaultSet

**Call** `b_errtype BestTrigSeqTranPropDefaultSet (`  
           `b_handletype handle,`  
           `b_int32 transient );`

**Description** Sets all properties of a transient in the trace memory trigger sequencer description table to default values.

For a description of properties and default values, refer to “*b\_trigseqtranproptype*” on page 300 and “*b\_trigseqtrancondproptype*” on page 300.

**CLI Equivalent** `BestTrigSeqTranPropDefaultSet transient=<transient>`

**CLI Abbreviation** `tstranprpdefset tran=<transient>`

**Return Value** Error code; see “*b\_errtype*” on page 250

**Input Parameters** **handle** Handle to identify the session.

**transient** Transient number (0 ... 255).

**See also** “*BestTrigSeqTranPropSet*” on page 74  
 “*BestTrigSeqTranCondPropSet*” on page 71

## BestTrigSeqTranPropSet

**Call** `b_errtype BestTrigSeqTranPropSet (`  
     `b_handletype               handle,`  
     `b_int32                    transient,`  
     `b_trigseqtranproptype   trigseqtranprop,`  
     `b_int32                    value );`

**Description** Sets a numeric transition property (“state” or “next state”) in the trace memory trigger sequencer description table.

**CLI Equivalent** `BestTrigSeqTranPropSet transient=<transient>`  
`trigseqtranprop=<trigseqtranprop> value=<value>`

**CLI Abbreviation** `tstranprpset tran=<transient> prop=<trigseqtranprop> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**transient** Transient number (0 ... 255).

**trigseqtranprop** Property to be set; see “*b\_trigseqtranproptype*” on page 300.

**value** Value the property is set to, “*b\_trigseqtranproptype*” on page 300.

**See also** “*BestTrigSeqTranPropDefaultSet*” on page 73  
 “*BestTrigSeqTranCondPropSet*” on page 71

# Trace Memory Functions

The following functions are used for the trace memory:

| Function                                    | Result   |
|---|--|
| <i>"BestAnalyzerRun" on page 75</i>         | Starts the PCI analyzer trace memory.  |
| <i>"BestAnalyzerStop" on page 76</i>        | Stops the PCI analyzer trace memory.   |
| <i>"BestTracePropSet" on page 81</i>        | Sets a property for the trace memory (for compatibility reasons).  |
| <i>"BestTraceRun" on page 82</i>            | Enables trace memory.  |
| <i>"BestTraceStop" on page 83</i>           | Stops the current trace run.   |
| <i>"BestTraceDataGet" on page 79</i>        | Loads trace memory lines from the testcard.  |
| <i>"BestTraceStatusGet" on page 82</i>      | Reads the trace status register, the line number of the trigger event, and the number of lines captured. |
| <i>"BestTraceBitPosGet" on page 77</i>      | Returns the position and length of a signal in a trace memory line.                                      |
| <i>"BestTraceBytePerLineGet" on page 78</i> | Returns the number of bytes of a trace memory line.  |
| <i>"BestTracePattPropSet" on page 80</i>    | Sets compare patterns for trace memory control.  |

How to use the functions is described in *"Trace Memory Programming"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestAnalyzerRun

**Call** `b_errtype BestAnalyzerRun( b_handletype handle );`

**Description** Starts the PCI analyzer trace memory. This function is equivalent to *"BestTraceRun" on page 82*.

**CLI Equivalent** `BestAnalyzerRun`

**CLI Abbreviation** `arun`

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**See also** *"BestAnalyzerStop" on page 76*

## BestAnalyzerStop

**Call** `b_errtype BestAnalyzerStop( b_handletype handle );`

**Description** Stops the PCI analyzer (protocol observer and trace memory). The current run status information and trace memory content are not affected.

The trace memory contains 100% pre-trigger history and is ready to be uploaded.

This function is equivalent to “*BestTraceStop*” on page 83.

**CLI Equivalent** `BestAnalyzerStop`

**CLI Abbreviation** `astop`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestAnalyzerRun*” on page 75

## BestTraceBitPosGet

**Call** `b_errtype BestTraceBitPosGet (`  
     `b_handletype handle,`  
     `b_signaltype signal,`  
     `b_int32 *position,`  
     `b_int32 *length );`

**Description** Returns the position and length of the specified signal in trace memory lines.

**CLI Equivalent** `BestTraceBitPosGet signal=<signal>`

**CLI Abbreviation** `trcbtposget signal=<signal>`

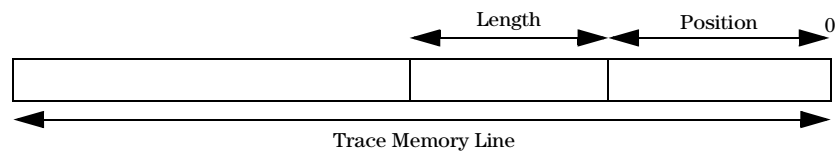
**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**signal** Signal of which the position and length are to be determined; see “*b\_signaltype (List of Signals)*” on page 274.

**Output Parameters** **position** Bit position of the specified signal within a trace memory line. This value is the offset (in bits) from the least significant bit to the first bit of the specified signal.

**length** Length in bits of the signal data. This is the width of the signal.



**See also** “*BestTraceBytePerLineGet*” on page 78

## BestTraceBytePerLineGet

**Call**     `b_errtype BestTraceBytePerLineGet(  
                  b_handletype    handle,  
                  b_int32        *bytes_per_line );`

**Description**    Returns the number of bytes in a trace memory line. This value depends on the Agilent E2920 hardware actually used.

**CLI Equivalent**    `BestTraceBytePerLineGet`

**CLI Abbreviation**    `trcbtplget`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**     Handle to identify the session.

**Output Parameters**    **bytes\_per\_line**    Number of bytes per line of trace data.

**See also**        “*BestTraceBitPosGet*” on page 77

## BestTraceDataGet

**Call**    `b_errtype BestTraceDataGet (`  
                  `b_handletype    handle,`  
                  `b_int32        startline,`  
                  `b_int32        n_of_lines,`  
                  `b_int32        *data );`

**Description**    Loads trace memory lines from the testcard.

**CLI Equivalent**    `BestTraceDataGet startline=<startline> n_of_lines=<n_of_lines>`

**CLI Abbreviation**    `trcdget start=<startline> n_of_lines=<n_of_lines>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**startline**    First line to be loaded from trace memory.

**n\_of\_lines**    Number of lines to be loaded from trace memory.

**Output Parameters**    **data**    Array of 32-bit values (dwords) containing the data read from trace memory.

The required size of this array can be calculated in the following way:

- Size of the array in bytes =  
                  number of lines (n\_of\_lines) × number of bytes per line;  
                  (the number of bytes per line can be requested with  
                  “*BestTraceBytePerLineGet*” on page 78)
- Size of the array in dwords =  
                  size of the array in bytes / 4

**See also**    –

## BestTracePattPropSet

**Call**     `b_errtype BestTracePattPropSet (`  
                  `b_handletype          handle,`  
                  `b_tracepattproptype  tracepattprop,`  
                  `b_charptrtype          pattern );`

**Description**     Sets compare patterns for trace memory control.

**CLI Equivalent**     `BestTracePattPropSet tracepattprop=<tracepattprop>`  
                          `pattern=<pattern>`

**CLI Abbreviation**     `trcprrpset prop=<tracepattprop> patt=<pattern>`

**Return Value**     Error code; see “*b\_errtype*” on page 250.

**Input Parameters**     **handle**     Handle to identify the session.

**tracepattprop**     Property to be set; see “*b\_tracepattproptype*” on page 294.

**pattern**     Logical expression for the condition; see “*b\_tracepattproptype*” on page 294. The string must be set in quotation marks.

**See also**     –



## BestTracePropSet

**Call**     `b_errtype BestTracePropSet (`  
             `b_handletype     handle,`  
             `b_traceproptype    traceprop,`  
             `b_int32             value );`

**Description**     Sets a property for the trace memory.

**CLI Equivalent**     `BestTracePropSet traceprop=<traceprop> value=<value>`

**CLI Abbreviation**   `trcprpset prop=<traceprop> val=<value>`

**Return Value**     Error code; see “*b\_errtype*” on page 250.

**Input Parameters**   **handle**     Handle to identify the session.

**traceprop**     Property to be set; see “*b\_traceproptype*” on page 295.

**value**     Value the property is set to; see “*b\_traceproptype*” on page 295.

**See also**     –

## BestTraceRun

**Call** `b_errtype BestTraceRun( b_handletype handle );`

**Description** Enables the trace memory. Data is acquired according to the trace memory trigger sequencer. This function is equivalent to “*BestAnalyzerRun*” on page 75.

**CLI Equivalent** `BestTraceRun`

**CLI Abbreviation** `trcrun`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestTraceStop*” on page 83

## BestTraceStatusGet

**Call** `b_errtype BestTraceStatusGet (
 b_handletype handle,
 b_tracestatustype tracestatus
 b_int32 *status );`

**Description** Reads the following data from the trace memory:

- trace status register,
- line number of the trigger event, or
- number of lines captured.

It is intended to be used to control the trace run.

**CLI Equivalent** `BestTraceStatusGet tracestatus=<tracestatus>`

**CLI Abbreviation** `tsget stat=<tracestatus>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**tracestatus** Status to be read; see “*b\_tracestatustype*” on page 296.

**Output Parameters** **status** Value of the queried status; see “*b\_tracestatustype*” on page 296.

**See also** –

## BestTraceStop

**Call** `b_errtype BestTraceStop( b_handletype handle );`

**Description** Stops the current trace run. The current run status information is not affected.

The trace memory contains 100% pre-trigger history and is ready to be uploaded.

This function is equivalent to “*BestAnalyzerStop*” on page 76.

**CLI Equivalent** `BestTraceStop`

**CLI Abbreviation** `trcstop`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestTraceRun*” on page 82

# Performance Measure Functions

**Setup and Programming** The functions of this category are used to set up and program the performance measures and the corresponding sequencers.

| Function  | Result   |
|---|--|
| <i>"BestPerfGenPropDefaultSet" on page 86</i>     | Sets the generic performance properties to default values.                           |
| <i>"BestPerfGenPropSet" on page 88</i>            | Sets the value of a generic performance measure property.                            |
| <i>"BestPerfGenPropGet" on page 87</i>            | Reads the value of a generic performance measure property.                           |
| <i>"BestPerfSeqPropDefaultSet" on page 90</i>     | Initializes a sequencer description table of a performance measure.                  |
| <i>"BestPerfSeqTranPropDefaultSet" on page 92</i> | Sets all properties of a transient to default values.                                |
| <i>"BestPerfSeqTranPropSet" on page 93</i>        | Sets a numeric transition property ("state" or "next state").                        |
| <i>"BestPerfSeqTranCondPropSet" on page 91</i>    | Sets a condition in the sequencer description table of a performance measure.        |
| <i>"BestPerfSeqProg" on page 89</i>               | Writes the sequencer description table of a performance measure to sequencer memory. |

**Running and Evaluating** The following functions are used to run and evaluate the measurement.

| Function                              | Result   |
|---------------------------------------|--|
| <i>"BestPerfRun" on page 89</i>       | Starts all counters.                             |
| <i>"BestPerfStatusGet" on page 94</i> | Reads the status of the selected measure.        |
| <i>"BestPerfStop" on page 95</i>      | Stops all counters.                              |
| <i>"BestPerfUpdate" on page 95</i>    | Captures the values of all performance counters. |
| <i>"BestPerfCtrRead" on page 85</i>   | Reads counter values.                            |

How to use the functions is described in *"Performance Measurement Programming"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestPerfCtrRead

**Call** `b_errtype BestPerfCtrRead(  
     b_handletype handle,  
     b_int32 measure,  
     b_int32 counter_id,  
     b_int32 *value );`

**Description** Reads the counter values of a performance measure.

The counter values must have been previously updated with “*BestPerfUpdate*” on page 95.

**CLI Equivalent** `BestPerfCtrRead measure=<measure> counter_id=<counter_id>`

**CLI Abbreviation** `pctrread meas=<measure> cid=<counter_id>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**measure** Value to identify the measure; see “*Measure Identifier*” on page 85.

**counter\_id** Value to identify the counter; see “*Counter Identifier*” on page 86.

**Output Parameters** **value** Value of the queried counter.

**See also** –

## Measure Identifier

| Measure Identifier (CLI Abbreviation) | Description                                      |
|---------------------------------------|--|
| B_PERFMEAS_0 (meas0, 0)               | Selects one of the performance measures 0 ... 7. |
| ...                                   |  |
| B_PERFMEAS_7 (meas7, 7)               |  |

## Counter Identifier

| Counter Identifier (CLI Abbreviation)         | Description  |
|---|--|
| B_PERFCTR_A (ctra)<br>B_PERFCTR_A_HI (ctrahi) | Nominator counter value (lower and upper 32 bits).           |
| B_PERFCTR_B (ctrb)<br>B_PERFCTR_B_HI (ctrbhi) | Denominator counter value (lower and upper 32 bits).         |
| B_PERFCTR_C (ctrc)<br>B_PERFCTR_C_HI (ctrchi) | Feedback counter C value (lower and upper 32 bits).          |
| B_REFCTR (refc)<br>B_REFCTR_HI (refchi)       | PCI clock reference counter value (lower and upper 32 bits). |

## BestPerfGenPropDefaultSet

**Call** `b_errtype BestPerfGenPropDefaultSet (`  
           `b_handletype    handle,`  
           `b_int32         measure );`

**Description** Sets the generic performance properties of a measure to default values. These properties are used to

- determine the mode used to increment the counter A,
- set the preload value of the feedback counter.

For generic properties and default values, refer to “*b\_perfgenproptype*” on page 268.

**CLI Equivalent** `BestPerfGenPropDefaultSet measure=<measure>`

**CLI Abbreviation** `pgenprpdefset meas=<measure>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**measure** Value to identify the measure; see “*Measure Identifier*” on page 85.

**See also** “*BestPerfGenPropGet*” on page 87  
 “*BestPerfGenPropSet*” on page 88

## BestPerfGenPropGet

**Call** `b_errtype BestPerfGenPropGet (`  
     `b_handletype           handle,`  
     `b_int32                measure,`  
     `b_perfgenproptype     perfgenprop,`  
     `b_int32                *value );`

**Description** Reads the value of a generic performance property of a measure. These properties are used to

- determine the mode used to increment the nominator counter,
- set the preload value of the feedback counter.

**CLI Equivalent** `BestPerfGenPropGet measure=<measure> perfgenprop=<perfgenprop>`

**CLI Abbreviation** `pgenprpget meas=<measure> prop=<perfgenprop>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**measure** Value to identify the measure; see “*Measure Identifier*” on page 85.

**perfgenprop** Property to be queried; see “*b\_perfgenproptype*” on page 268.

**Output Parameters** **value** Value of the queried property; see “*b\_perfgenproptype*” on page 268.

**See also** “*BestPerfGenPropDefaultSet*” on page 86  
 “*BestPerfGenPropSet*” on page 88

## BestPerfGenPropSet

**Call** `b_errtype BestPerfGenPropSet (`  
     `b_handletype           handle,`  
     `b_int32                measure,`  
     `b_perfgenproptype   perfgenprop,`  
     `b_int_32             value );`

**Description** Sets the value of a generic performance property of a measure. These properties are used to

- determine the mode used to increment the nominator counter,
- set the preload value of the feedback counter.

**CLI Equivalent** `BestPerfGenPropSet measure=<measure> perfgenprop=<perfgenprop>`  
`value=<value>`

**CLI Abbreviation** `pgenprpset meas=<measure> prop=<perfgenprop> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**measure** Value to identify the measure; see “*Measure Identifier*” on page 85.

**perfgenprop** Property to be set; see “*b\_perfgenproptype*” on page 268.

**value** Value to which the property is set; see “*b\_perfgenproptype*” on page 268.

**See also** “*BestPerfGenPropDefaultSet*” on page 86  
 “*BestPerfGenPropGet*” on page 87



## BestPerfRun

**Call** `b_errtype BestPerfRun( b_handletype handle );`

**Description** Starts all performance counters (and the trigger I/O sequencer).

**CLI Equivalent** `BestPerfRun`

**CLI Abbreviation** `prun`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestPerfStop*” on page 95  
“*BestTrigIORun*” on page 184

## BestPerfSeqProg

**Call** `b_errtype BestPerfSeqProg(  
          b_handletype handle,  
          b_int32      measure );`

**Description** Writes the information stored in the description table of the performance measure sequencer to the sequencer memory of the selected measure.

This function also checks whether the transition conditions of one state are consistent. If they are not, the function returns an error.

**CLI Equivalent** `BestPerfSeqProg measure=<measure>`

**CLI Abbreviation** `psprog meas=<measure>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**measure** Value to identify the measure; see “*Measure Identifier*” on page 85.

**See also** —

## BestPerfSeqPropDefaultSet

**Call**     `b_errtype BestPerfSeqPropDefaultSet (`  
                  `b_handletype    handle,`  
                  `b_int32        measure );`

**Description**     Initializes the sequencer description table of the selected measure.

For a description of properties and default values, refer to “*b\_perfseqtranproptype*” on page 269 and “*b\_perfseqtrancondproptype*” on page 268.

**CLI Equivalent**     `BestPerfSeqPropDefaultSet measure=<measure>`

**CLI Abbreviation**     `psprpdefset meas=<measure>`

**Return Value**     Error code; see “*b\_errtype*” on page 250.

**Input Parameters**     **handle**     Handle to identify the session.

**measure**     Value to identify the measure; see “*Measure Identifier*” on page 85.

**See also**     “*BestPerfSeqTranPropDefaultSet*” on page 92

## BestPerfSeqTranCondPropSet

**Call** `b_errtype BestPerfSeqTranCondPropSet (`  
     `b_handletype                      handle,`  
     `b_int32                            measure,`  
     `b_int32                            transient,`  
     `b_perfseqtrancondproptype   perfseqtrancondprop,`  
     `b_charptrtype                    condition);`

**Description** Sets a condition in the sequencer description table.

The conditions are transition condition, conditions to increment nominator or denominator counter, and conditions to decrement or preload the feedback counter.

**CLI Equivalent** `BestPerfSeqTranCondPropSet measure=<measure> transient=<transient>`  
`perfseqtrancondprop=<perfseqtrancondprop> condition=<condition>`

**CLI Abbreviation** `pstrancprpset meas=<measure> tran=<transient>`  
`prop=<perfseqtrancondprop> con=<condition>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**measure** Value to identify the measure; see “*Measure Identifier*” on page 85.

**transient** Transient number (0 ... 255).

**perfseqtrancondprop** Property to be set; see “*b\_perfseqtrancondproptype*” on page 268.

**condition** Condition string to which the property is set. The string must be set in quotation marks. See “*Conditions Reference*” on page 72.

**See also** “*BestPerfSeqTranPropDefaultSet*” on page 92  
 “*BestPerfSeqTranPropSet*” on page 93

## BestPerfSeqTranPropDefaultSet

**Call** `b_errtype BestPerfSeqTranPropDefaultSet (`  
           `b_handletype  handle,`  
           `b_int32      measure,`  
           `b_int32      transient );`

**Description** Sets all properties of a transient in the sequencer description table to default values.

For a description of properties and default values, refer to “*b\_perfseqtranproptype*” on page 269 and “*b\_perfseqtrancondproptype*” on page 268.

**CLI Equivalent** `BestPerfSeqTranPropDefaultSet measure=<measure>`  
                   `transient=<transient>`

**CLI Abbreviation** `pstranprpdefset meas=<measure> tran=<transient>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**measure** Value to identify the measure; see “*Measure Identifier*” on page 85.

**transient** Transient number (0 ... 255).

**See also** “*BestPerfSeqTranCondPropSet*” on page 91  
           “*BestPerfSeqTranPropSet*” on page 93  
           “*BestTrigSeqPropDefaultSet*” on page 70

## BestPerfSeqTranPropSet

**Call** `b_errtype BestPerfSeqTranPropSet (`  
     `b_handletype                    handle,`  
     `b_int32                        measure,`  
     `b_int32                        transient,`  
     `b_perfseqtranproptype        perfseqtranprop,`  
     `b_int32                        value );`

**Description** Sets a numeric transition property (“state” or “next state”) in the sequencer description table of the selected measure.

**CLI Equivalent** `BestPerfSeqTranPropSet measure=<measure> transient=<transient>`  
`perfseqtranprop=<perfseqtranprop> value=<value>`

**CLI Abbreviation** `pstranprpset meas=<measure> tran=<transient> prop=<perfseqtranprop>`  
`val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**measure** Value to identify the measure; see “*Measure Identifier*” on page 85.

**transient** Transient number (0 ... 255).

**perfseqtranprop** Property to be set; see “*b\_perfseqtranproptype*” on page 269.

**value** Value to which the property is set.

**See also** “*BestPerfSeqTranCondPropSet*” on page 91  
 “*BestPerfSeqTranPropDefaultSet*” on page 92

## BestPerfStatusGet

**Call** `b_errtype BestPerfStatusGet (`  
     `b_handletype handle,`  
     `b_int32 measure,`  
     `b_int32 *value );`

**Description** Queries the status of the selected measure.

**CLI Equivalent** `BestPerfStatusGet measure=<measure>`

**CLI Abbreviation** `psget meas=<measure>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**measure** Value to identify the measure; see “*Measure Identifier*” on page 85.

**Output Parameters** **value** Value of the performance status register; see table below.

**See also** –

## Performance Status Register

The following table shows the meanings of the single bits of the performance status register.

| Bit      | Meaning                                  |
|----------|--|
| 0        | 1 = counter A overflow                   |
| 1        | 1 = counter B overflow                   |
| 2        | 1 = PCI clock reference counter overflow |
| 3        | 1 = measure started                      |
| 4 ... 31 | Not used.                                |

## BestPerfStop

**Call** `b_errtype BestPerfStop( b_handletype handle );`

**Description** Stops all performance counters (and the trigger I/O sequencer). Note that there is no need to stop the counters to get their current values.

**CLI Equivalent** `BestPerfStop`

**CLI Abbreviation** `pstop`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestPerfRun*” on page 89  
“*BestTrigIOStop*” on page 187

## BestPerfUpdate

**Call** `b_errtype BestPerfUpdate( b_handletype handle );`

**Description** Captures the values of all performance counters at the same PCI clock cycle. The counters are then reset and restarted.

The captured values can be read with “*BestPerfCtrRead*” on page 85.

**CLI Equivalent** `BestPerfUpdate`

**CLI Abbreviation** `pupdate`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** —





# PCI Exerciser Functions

The PCI Exerciser functions are divided into the following sections:

- *“Exerciser Generic Functions” on page 98*
- *“Master Programming Functions” on page 109*
- *“Target Programming Functions” on page 125*
- *“Configuration Space Programming Functions” on page 142*
- *“Expansion ROM Programming Functions” on page 148*
- *“Data Memory Functions” on page 150*
- *“Host Access Functions” on page 153*
- *“Interrupt Generation Function” on page 161*
- *“Built-In Test Functions” on page 162*

NOTE All Exerciser functions are only available with Opt. 300 of the Agilent E2920 series.

# Exerciser Generic Functions

The following functions are used to prepare for exerciser programming:

| Function   | Result  |
|--|---|
| <i>"BestExerciserGenPropSet" on page 100</i>     | Sets the type of attribute memory organization.                             |
| <i>"BestExerciserGenPropGet" on page 99</i>      | Reads the type of attribute memory organization.                            |
| <i>"BestMasterBlockRun" on page 102</i>          | Runs one block with the transaction properties in the preparation register. |
| <i>"BestMasterBlockPageRun" on page 101</i>      | Runs a block page of the master block transfer memory.                      |
| <i>"BestMasterStop" on page 106</i>              | Stops the master.   |
| <i>"BestMasterGenPropDefaultSet" on page 103</i> | Sets all master generic properties to default values.                       |
| <i>"BestMasterGenPropSet" on page 105</i>        | Sets value of a master generic property.                                    |
| <i>"BestMasterCondStartPattSet" on page 106</i>  | Sets a master run condition.  |
| <i>"BestMasterGenPropGet" on page 104</i>        | Reads value of a master generic property.                                   |
| <i>"BestTargetGenPropDefaultSet" on page 107</i> | Sets all target generic properties to default values.                       |
| <i>"BestTargetGenPropSet" on page 108</i>        | Sets value of a target generic property.                                    |
| <i>"BestTargetGenPropGet" on page 107</i>        | Reads value of a target generic property.                                   |

How to use the functions is described in *"Programming the Exerciser"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestExerciserGenPropGet

**Call**    `b_errtype BestExerciserGenPropGet (`  
                  `b_handletype                  handle,`  
                  `b_exercisergenproptype  exeprop,`  
                  `b_int32                      *value );`

**Description**    Reads the generic exerciser property. The generic exerciser property determines the type of attribute memory organization.

**CLI Equivalent**    `BestExerciserGenPropGet exeprop=<exeprop>`

**CLI Abbreviation**    `egprpget prop=<exeprop>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**exeprop**    Exerciser property to be read; see “*b\_exercisergenproptype*” on page 256.

**Output Parameters**    **value**    Value of the exerciser property; see “*b\_exercisergenproptype*” on page 256.

**See also**    “*BestExerciserGenPropSet*” on page 100

## BestExerciserGenPropSet

**Call**    `b_errtype BestExerciserGenPropSet (`  
                  `b_handletype          handle,`  
                  `b_exercisergenproptype  exeprop,`  
                  `b_int32                  value );`

**Description**    Sets the generic exerciser property. The generic exerciser property determines the type of attribute memory organization.

To make the setting persistent, use “*BestAllPropStore*” on page 34.

**CLI Equivalent**    `BestExerciserGenPropSet exeprop=<exeprop> value=<value>`

**CLI Abbreviation**    `egprpset prop=<exeprop> val=<value>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**exeprop**    Exerciser property to be set; see “*b\_exercisergenproptype*” on page 256.

**value**    Value to which the exerciser property is set; see “*b\_exercisergenproptype*” on page 256.

**See also**    “*BestExerciserGenPropGet*” on page 99

## BestMasterBlockPageRun

**Call** `b_errtype BestMasterBlockPageRun(  
           b_handletype  handle,  
           b_int32      page_num );`

**Description** Runs a block page in the master block transfer memory.

The function returns as soon as the block transfer has started. To ensure that a page run has been completed, poll the status of the master with “*BestStatusRegGet*” on page 39 until the master is inactive.

Execution stops if a master abort occurs and a compare error is generated if the compare flag is set.

The master enable bit in the testcard’s configuration space must be set to 1 before a master run is started. This bit can be set with “*BestMasterGenPropSet*” on page 105.

**NOTE** BestMasterBlockPageRun fails if you call this function while a transaction is running.

**CLI Equivalent** `BestMasterBlockPageRun page_num=<page_num>`

**CLI Abbreviation** `mbpgrun page=<page_num>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**page\_num** Number of the block page that is to be executed (0 ... 15, page 0 contains default values). All blocks defined in this page are executed consecutively.

**See also** “*BestMasterBlockRun*” on page 102

## BestMasterBlockRun

**Call** `b_errtype BestMasterBlockRun( b_hndletype handle );`

**Description** Runs one PCI block specified by the current settings in the preparation register of the master block transfer memory.

This function returns as soon as the block transfer has started. To ensure that a block run has been completed, poll the status of the master using “*BestStatusRegGet*” on page 39 until the master is inactive.

Execution stops if a master abort occurs and a compare error is generated if the compare flag is set.

The master enable bit in the testcard’s configuration space must be set to 1 before a master run is started. This bit can be set with “*BestMasterGenPropSet*” on page 105.

**NOTE** BestMasterBlockRun fails if you call this function while a transaction is running.

**CLI Equivalent** BestMasterBlockRun

**CLI Abbreviation** mbrun

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestMasterBlockPageRun*” on page 101

## BestMasterGenPropDefaultSet

**Call** `b_errtype BestMasterGenPropDefaultSet( b_handletype handle );`

**Description** Sets all master generic properties to their default values. For a list of these properties, see “*b\_mastergenproptype*” on page 257.

**NOTE** BestMasterGenPropDefaultSet fails if you call this function while a transaction is running.

Properties that affect settings of configuration space header registers will be altered only if the power up property B\_PU\_CONFRESTORE is set to 0. This property can be set with “*BestPowerUpPropSet*” on page 37.

**CLI Equivalent** BestMasterGenPropDefaultSet

**CLI Abbreviations** mgprpdefset

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestMasterGenPropSet*” on page 105  
“*BestMasterGenPropGet*” on page 104

## BestMasterGenPropGet

**Call** `b_errtype BestMasterGenPropGet (`  
           `b_handletype        handle,`  
           `b_mastergenproptype  mastergenprop,`  
           `b_int32                *value );`

**Description** Reads the value of a generic master property.

**NOTE** BestMasterGenPropGet fails if you call this function while a transaction is running.

**CLI Equivalent** `BestMasterGenPropGet mastergenprop=<mastergenprop>`

**CLI Abbreviation** `mgprpget prop=<mastergenprop>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**mastergenprop** Property to be read; see “*b\_mastergenproptype*” on page 257.

**Output Parameters** **value** Property value; see “*b\_mastergenproptype*” on page 257.

**See also** “*BestMasterGenPropDefaultSet*” on page 103  
 “*BestMasterGenPropSet*” on page 105



## BestMasterGenPropSet

**Call** `b_errtype BestMasterGenPropSet (`  
           `b_handletype                handle,`  
           `b_mastergenproptype      mastergenprop,`  
           `b_int32                   value );`

**Description** Sets the value of a generic master property. Generic master properties are valid during a master run.

**NOTE** BestMasterGenPropSet fails if you call this function while a transaction is running.

**CLI Equivalent** `BestMasterGenPropSet mastergenprop=<mastergenprop> value=<value>`

**CLI Abbreviations** `mgprpset prop=<mastergenprop> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**mastergenprop** Property to be set; see “*b\_mastergenproptype*” on page 257.

**value** Value to which the property is set; see “*b\_mastergenproptype*” on page 257.

**See also** “*BestMasterGenPropDefaultSet*” on page 103  
 “*BestMasterGenPropGet*” on page 104

## BestMasterCondStartPattSet

**Call** `b_errtype BestMasterCondStartPattSet (`  
           `b_handletype     handle,`  
           `b_charptrtype   pattern );`

**Description** Sets condition for the start of the master. This allows you to conditionally start the master based on a specific bus event. The bus events (bus signals) can constitute the master start condition.

For a list of the available bus signals, refer to the “*b\_signaltype (List of Signals)*” on page 274.

**NOTE** BestMasterCondStartPattSet fails if you call this function while a transaction is running.

**CLI Equivalent** `BestMasterCondStartPattSet pattern=<pattern>`

**CLI Abbreviation** `mcpset patt=<pattern>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**pattern** The pattern string that defines the compare pattern. Please refer to “*BestPattSet*” on page 62 for details on pattern term programming and pattern syntax.

**See also** –

## BestMasterStop

**Call** `b_errtype BestMasterStop( b_handletype handle );`

**Description** Stops the master after completing the current data transfer.

**CLI Equivalent** `BestMasterStop`

**CLI Abbreviation** `mstop`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** –

## BestTargetGenPropDefaultSet

**Call** `b_errtype BestTargetGenPropDefaultSet ( b_handletype handle );`

**Description** Sets the target generic properties to their default values. For a list of these properties, see “*b\_targetgenproptype*” on page 286.

**NOTE** Properties that affect settings of configuration space header registers will be altered only if the power up property B\_PU\_CONFRESTORE is set to 0. This property can be set with “*BestPowerUpPropSet*” on page 37.

**CLI Equivalent** `BestTargetGenPropDefaultSet`

**CLI Abbreviation** `tgprpdefset`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestTargetGenPropSet*” on page 108  
“*BestTargetGenPropGet*” on page 107

## BestTargetGenPropGet

**Call** `b_errtype BestTargetGenPropGet (
 b_handletype handle,
 b_targetgenproptype targetgenprop,
 b_int32 *value );`

**Description** Reads a generic target property.

**CLI Equivalent** `BestTargetGenPropGet targetgenprop=<targetgenprop>`

**CLI Abbreviation** `tgprpget prop=<targetgenprop>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**targetgenprop** Property to be read; see “*b\_targetgenproptype*” on page 286.

**Output Parameters** **value** Property value; see “*b\_targetgenproptype*” on page 286.

**See also** “*BestTargetGenPropDefaultSet*” on page 107  
“*BestTargetGenPropSet*” on page 108

## BestTargetGenPropSet

**Call** `b_errtype BestTargetGenPropSet (`  
           `b_handletype        handle,`  
           `b_targetgenproptype  targetgenprop,`  
           `b_int32                value );`

**Description** Sets a generic target property.

**CLI Equivalent** `BestTargetGenPropSet targetgenprop=<targetgenprop> value=<value>`

**CLI Abbreviation** `tgprpset prop=<targetgenprop> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**targetgenprop** Property to be set; see “*b\_targetgenproptype*” on page 286.

**value** Value to which the property is set; see “*b\_targetgenproptype*” on page 286.

**See also** “*BestTargetGenPropDefaultSet*” on page 107  
 “*BestTargetGenPropGet*” on page 107

# Master Programming Functions

**Master Block Transfer Pages** The following functions are used to program the master block transfer pages:

| Function   | Result  |
|--|---|
| <i>"BestMasterBlockPageInit" on page 121</i>       | Initializes a master block transfer memory page.                          |
| <i>"BestMasterBlockPropDefaultSet" on page 122</i> | Sets the master block transfer memory's preparation register to defaults. |
| <i>"BestMasterBlockPropSet" on page 123</i>        | Sets a master block transaction property in the preparation register.     |
| <i>"BestMasterBlockPropGet" on page 122</i>        | Reads a master block transaction property from the preparation register.  |
| <i>"BestMasterBlockLineProg" on page 119</i>       | Programs the preparation register to a master block transfer memory line. |
| <i>"BestMasterBlockLineRead" on page 120</i>       | Reads a master block transfer memory line to the preparation register.    |
| <i>"BestMasterBlockEndProg" on page 118</i>        | Programs an "end of page" (EOP) to a master block transfer memory line.   |
| <i>"BestMasterBlockPageInit" on page 121</i>       | Initializes a master block transfer memory page.                          |

**Master Attribute Pages** The following functions are used to program master attribute pages:

| Function  | Result  |
|---|---|
| <i>"BestMasterAttrPageInit" on page 115</i>       | Initializes a master attribute memory page.   |
| <i>"BestMasterAttrPropDefaultSet" on page 116</i> | Sets the master attribute memory's preparation register to defaults.                  |
| <i>"BestMasterAttrPropSet" on page 117</i>        | Sets a master attribute property in the preparation register.                         |
| <i>"BestMasterAttrPropGet" on page 116</i>        | Reads a master attribute property from the preparation register.                      |
| <i>"BestMasterAttrLineProg" on page 113</i>       | Programs the master attribute memory's preparation register to a memory line.         |
| <i>"BestMasterAttrLineRead" on page 114</i>       | Reads a master attribute memory line to the preparation register.                     |
| <i>"BestMasterAttrGroupLineProg" on page 111</i>  | Programs a group of master attributes from the preparation register to a memory line. |
| <i>"BestMasterAttrGroupLineRead" on page 112</i>  | Reads a group of master attributes from a memory line to the preparation register.    |

**Byte Enable Memory** The following functions are used to program the byte enable memory:

| Function                                      | Result                                     |
|---|--|
| <i>"BestMasterByteEnableProg" on page 124</i> | Programs a line in the byte enable memory. |
| <i>"BestMasterByteEnableRead" on page 124</i> | Reads a line from the byte enable memory.  |

How to use the functions is described in *"Programming the Exerciser as a Master Device"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestMasterAttrGroupLineProg

**Call** `b_errtype BestMasterAttrGroupLineProg(  
     b_handletype      handle,  
     b_mattrgrouptype   group,  
     b_int32           page_num,  
     b_int32           offset );`

**Description** Writes the attributes of a group from the preparation register to a line in the attribute memory.

After you have programmed all memory lines within a group, set the individual loop bit B\_M\_DOLOOP with “*BestMasterAttrPropSet*” on page 117 to indicate the last line of the current group.

**NOTE** Within a page you can use either group or non-group memory programming functions, but not both together.

**CLI Equivalent** `BestMasterAttrGroupLineProg group=<group> page_num=<page_num>  
 offset=<offset>`

**CLI Abbreviation** `magrplprog grp=<group> page=<page_num> offs=<offset>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**group** Identifier of the group; see “*b\_mattrgrouptype*” on page 259.

**page\_num** Number of the page on which the line is to be programmed.

Valid page numbers are:

- 1 ... 63 if page size is 4
- 1 ... 7 if page size is 32

The page size is set by the generic exerciser property B\_EGEN\_ATTRPAGESIZE with “*BestExerciserGenPropSet*” on page 100.

**offset** Number of the line to be programmed. Line counting starts from 0.

**NOTE** You can program more lines than are available on a page. This results in concatenated pages.

**See also** “*BestMasterAttrPageInit*” on page 115  
 “*BestMasterAttrGroupLineRead*” on page 112  
 “*BestMasterAttrLineProg*” on page 113

## BestMasterAttrGroupLineRead

**Call** `b_errtype BestMasterAttrGroupLineRead(  
           b_handletype        handle,  
           b_mattrgrouptype   group,  
           b_int32            page_num,  
           b_int32            offset );`

**Description** Reads the attributes of a line within a group in the attribute memory to the preparation register.

**CLI Equivalent** `BestMasterAttrGroupLineRead group=<group> page_num=<page_num>  
 offset=<offset>`

**CLI Abbreviation** `magrplread grp=<group> page=<page_num> offs=<offset>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**group** Identifier of the group; see “*b\_mattrgrouptype*” on page 259.

**page\_num** Number of the page on which the line is to be read.

**offset** Number of the line to be read. Line counting starts from 0.

**See also** “*BestMasterAttrPageInit*” on page 115  
 “*BestMasterAttrGroupLineProg*” on page 111  
 “*BestMasterAttrLineRead*” on page 114



## BestMasterAttrLineProg

**Call** `b_errtype BestMasterAttrLineProg(  
           b_handletype  handle,  
           b_int32      page_num,  
           b_int32      offset );`

**Description** Writes the contents of the preparation register to a line in the attribute memory.

After you have programmed all memory lines, set the general loop bit `B_M_DOLOOP` with “*BestMasterAttrPropSet*” on page 117 to indicate the last line in the attribute page.

**NOTE** Within a page you can use either group or non-group memory programming functions, but not both together.

**CLI Equivalent** `BestMasterAttrLineProg page_num=<page_num> offset=<offset>`

**CLI Abbreviation** `malprog page=<page_num> offs=<offset>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**page\_num** Number of the page on which the line is to be programmed.

Valid page numbers are:

- 1 ... 63 if page size is 4
- 1 ... 7 if page size is 32

The page size is set by the generic exerciser property `B_EGEN_ATTRPAGESIZE` with “*BestExerciserGenPropSet*” on page 100.

**offset** Number of the line to be programmed. Line counting starts from 0.

**NOTE** You can program more lines than are available on a page. This results in concatenated pages.

**See also** “*BestMasterAttrPageInit*” on page 115  
 “*BestMasterAttrLineRead*” on page 114  
 “*BestMasterAttrGroupLineProg*” on page 111

## BestMasterAttrLineRead

**Call**     `b_errtype BestMasterAttrLineRead(  
                   b_handletype    handle,  
                   b_int32        page_num,  
                   b_int32        offset );`

**Description**   Reads the contents of an attribute memory's line to the preparation register.

**CLI Equivalent**   `BestMasterAttrLineRead page_num=<page_num> offset=<offset>`

**CLI Abbreviation**   `malread page=<page_num> offs=<offset>`

**Return Value**     Error code; see “*b\_errtype*” on page 250.

**Input Parameters**   **handle**     Handle to identify the session.

**page\_num**     Number of the page on which the line is to be read.

**offset**     Number of the line to be read. Line counting starts from 0.

**See also**     “*BestMasterAttrPageInit*” on page 115  
                   “*BestMasterAttrLineProg*” on page 113  
                   “*BestMasterAttrGroupLineRead*” on page 112

## BestMasterAttrPageInit

**Call** `b_errtype BestMasterAttrPageInit (`  
           `b_handletype  handle,`  
           `b_int32       page_num );`

**Description** Initializes a master attribute memory page (and all pages concatenated to this page).

This function must be called once before an attribute page can be programmed or read.

**NOTE** BestMasterAttrPageInit fails if you call this function while a transaction is running.

**CLI Equivalent** `BestMasterAttrPageInit page_num=<page_num>`

**CLI Abbreviation** `mapginit page=<page_num>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**page\_num** Number of the memory page to be initialized.

Valid page numbers are:

- 1 ... 63 if page size is 4
- 1 ... 7 if page size is 32

The page size is set by the generic exerciser property `B_EGEN_ATTRPAGESIZE` with “*BestExerciserGenPropSet*” on page 100.

**See also** “*BestMasterAttrGroupLineProg*” on page 111  
 “*BestMasterAttrGroupLineRead*” on page 112  
 “*BestMasterAttrLineProg*” on page 113  
 “*BestMasterAttrLineRead*” on page 114

## BestMasterAttrPropDefaultSet

**Call** `b_errtype BestMasterAttrPropDefaultSet( b_handletype handle );`

**Description** Sets the preparation register of the master attribute memory to default values. For a description of attributes and default values, see “*b\_mattrproptype*” on page 260.

**CLI Equivalent** `BestMasterAttrPropDefaultSet`

**CLI Abbreviation** `maprpdefset`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestMasterAttrPropSet*” on page 117  
“*BestMasterAttrPropGet*” on page 116

## BestMasterAttrPropGet

**Call** `b_errtype BestMasterAttrPropGet (
 b_handletype handle,
 b_mattrproptype mattrprop,
 b_int32 *value );`

**Description** Reads a master attribute from the preparation register.

**CLI Equivalent** `BestMasterAttrPropGet mattrprop=<mattrprop>`

**CLI Abbreviation** `maprpget prop=<mattrprop>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**mattrprop** Master attribute to be read; see “*b\_mattrproptype*” on page 260.

**Output Parameters** **value** Value of the attribute; see “*b\_mattrproptype*” on page 260.

**See also** “*BestMasterAttrPropDefaultSet*” on page 116  
“*BestMasterAttrPropSet*” on page 117

## BestMasterAttrPropSet

**Call** `b_errtype BestMasterAttrPropSet (`  
     `b_handletype        handle,`  
     `b_mattrproptype    mattrprop,`  
     `b_int32             value );`

**Description** Sets a master attribute in the preparation register.

After you have set all attributes to the required values in the register, you can program the complete register to a memory line with “*BestMasterAttrLineProg*” on page 113.

**CLI Equivalent** `BestMasterAttrPropSet mattrprop=<mattrprop> value=<value>`

**CLI Abbreviation** `maprpset prop=<mattrprop> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**mattrprop** Master attribute to be set; see “*b\_mattrproptype*” on page 260.

**value** Value to which the attribute is set; see “*b\_mattrproptype*” on page 260.

**See also** “*BestMasterAttrPropDefaultSet*” on page 116  
 “*BestMasterAttrPropGet*” on page 116

## BestMasterBlockEndProg

**Call**     `b_errtype BestMasterBlockEndProg(  
                   b_handletype    handle,  
                   b_int32        page_num,  
                   b_int32        offset );`

**Description**    Programs the “end of page” (EOP) line in the master block transfer memory. This function should conclude memory programming.

**NOTE**            After initialization, the last line of a page is pre-programmed with EOP.

**CLI Equivalent**    `BestMasterBlockEndProg page_num=<page_num> offset=<offset>`

**CLI Abbreviation**    `mbeprog page=<page_num> offs=<offset>`

**Return Value**      Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**      Handle to identify the session.

**page\_num**      Number of the block transfer memory page.

**offset**      Number of the line to be programmed. Line counting starts from 0.

**See also**            “*BestMasterBlockLineProg*” on page 119

## BestMasterBlockLineProg

**Call**    `b_errtype BestMasterBlockLineProg(  
                   b_handletype    handle,  
                   b_int32        page_num  
                   b_int32        offset );`

**Description**    Writes the preparation register to a line of a master block transfer memory page.

After you have programmed all required memory lines, program the “end of page” to your last memory line with “*BestMasterBlockEndProg*” on page 118 (this is not necessary if the last line is physically the end of the page).

**NOTE**    The last line in the memory cannot be programmed, it can only contain an EOP.

**CLI Equivalent**    `BestMasterBlockLineProg page_num=<page_num> offset=<offset>`

**CLI Abbreviation**    `mblprog page=<page_num> offs=<offset>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**page\_num**    Number of the block transfer memory page (1 ... 15, page 0 ist read-only).

Pages can be unavailable if they are part of a sequence of concatenated pages (if the physical page size is overridden).

**offset**    Number of the line to be programmed. Line counting starts from 0.

**See also**    “*BestMasterBlockPageInit*” on page 121  
               “*BestMasterBlockLineRead*” on page 120

## BestMasterBlockLineRead

**Call**    `b_errtype BestMasterBlockLineRead(  
           b_handletype    handle,  
           b_int32        page_num,  
           b_int32        offset,  
           b_int32        *eop );`

**Description**    Reads the properties of one block (that is: one line of the master block transfer memory) to the preparation register, if this line is not an EOP line.

If the line is an EOP line, the preparation register remains unchanged and the EOP flag (output parameter) is set to 1.

**CLI Equivalent**    `BestMasterBlockLineRead page_num=<page_num> offset=<offset>`

**CLI Abbreviation**    `mbread page=<page_num> offs=<offset>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**page\_num**    Number of the block transfer memory page.

**offset**    Number of the line to be read. Line counting starts from 0.

**Output Parameters**    **eop**    EOP flag:

- 0 – the read line is not an EOP line.
- 1 – the read line is an EOP line.

**See also**    “*BestMasterBlockLineProg*” on page 119



## BestMasterBlockPageInit

**Call**     `b_errtype BestMasterBlockPageInit(  
                  b_handletype    handle,  
                  b_int32        page_num );`

**Description**    Initializes a master block transfer page and sets the current block pointer to the beginning of the page. This function must be called once before a page can be programmed.

If this function is applied to a concatenated page, all concatenated pages—previous and next—will also be initialized.

**NOTE**     MasterBlockPageInit fails if you call this function while a transaction is running.

**CLI Equivalent**    `BestMasterBlockPageInit page_num=<page_num>`

**CLI Abbreviation**   `mbpginit page=<page_num>`

**Return Value**     Error code; see “*b\_errtype*” on page 250.

**Input Parameters**   **handle**     Handle to identify the session.

**page\_num**     Number of the page to be initialized (1 ... 15, page 0 cannot be overwritten).

**See also**     “*BestMasterBlockLineProg*” on page 119

## BestMasterBlockPropDefaultSet

**Call** `b_errtype BestMasterBlockPropDefaultSet ( b_handletype handle );`

**Description** Sets the preparation register of the block transfer memory to default values.

For a description of properties and default values, see “*b\_blkproptype*” on page 238.

**CLI Equivalent** `BestMasterBlockPropDefaultSet`

**CLI Abbreviation** `mbprpdefset`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestMasterBlockPropSet*” on page 123  
“*BestMasterBlockPropGet*” on page 122

## BestMasterBlockPropGet

**Call** `b_errtype BestMasterBlockPropGet (
 b_handletype handle,
 b_blkproptype blk_prop,
 b_int32 *value );`

**Description** Reads a master block transaction property from the preparation register.

**CLI Equivalent** `BestMasterBlockPropGet blk_prop=<blk_prop>`

**CLI Abbreviation** `mbprpget prop=<blk_prop>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**blk\_prop** Block transaction property to be read; see “*b\_blkproptype*” on page 238.

**Output Parameters** **value** Value of the property; see “*b\_blkproptype*” on page 238.

**See also** “*BestMasterBlockPropDefaultSet*” on page 122  
“*BestMasterBlockPropSet*” on page 123

## BestMasterBlockPropSet

**Call**    `b_errtype BestMasterBlockPropSet (`  
                  `b_handletype     handle,`  
                  `b_blkproptype   blk_prop,`  
                  `b_int32            value );`

**Description**    Sets a master block transaction property in the preparation register.

After you have set all properties to the required values in the register, the complete register can be programmed to a memory line with *“BestMasterBlockLineProg”* on page 119.

**CLI Equivalent**    `BestMasterBlockPropSet blk_prop=<blk_prop> value=<value>`

**CLI Abbreviation**    `mbprpset prop=<blk_prop> val=<value>`

**Return Value**    Error code; see *“b\_errtype”* on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**blk\_prop**    Property to be set; see *“b\_blkproptype”* on page 238.

**value**    Value to which the property is set; see *“b\_blkproptype”* on page 238.

**See also**    *“BestMasterBlockPropDefaultSet”* on page 122  
               *“BestMasterBlockPropGet”* on page 122

## BestMasterByteEnableProg

**Call** `b_errtype BestMasterByteEnableProg(  
           b_handletype  handle,  
           b_int32      offset,  
           b_int32      value );`

**Description** Programs the byte enables for one transaction to the byte enable memory.

**CLI Equivalent** `BestMasterByteEnableProg offset=<offset> value=<value>`

**CLI Abbreviation** `mbytenprog offs=<offset> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**offset** Byte enable memory line. Programmable lines are 16 ... 255 (lines 0 ... 15 are read-only for compatibility reasons).

**value** Value (0 ... 255) to be programmed to the byte enable memory line.

**See also** “*BestMasterByteEnableRead*” on page 124

## BestMasterByteEnableRead

**Call** `b_errtype BestMasterByteEnableRead(  
           b_handletype  handle,  
           b_int32      offset,  
           b_int32      *value );`

**Description** Reads one line from byte enable memory (byte enables for one transfer).

**CLI Equivalent** `BestMasterByteEnableRead offset=<offset>`

**CLI Abbreviation** `mbytenread offs=<offset>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**offset** Byte enable memory line to be read (0 ... 255).

**Output Parameters** **value** Value programmed in the referring byte enable memory line.

**See also** “*BestMasterByteEnableProg*” on page 124

# Target Programming Functions

The functions described in this section are used to program the exerciser as a target.

**Decoder Properties** The following functions are used to program the decoder properties:

| Function  | Result  |
|---|---|
| <i>"BestTargetDecoderPropSet" on page 140</i>     | Sets a decoder property in the preparation register.  |
| <i>"BestTargetDecoderPropGet" on page 139</i>     | Reads a decoder property from the preparation register.   |
| <i>"BestTargetDecoderProg" on page 137</i>        | Programs a decoder with properties as set in the preparation register.                            |
| <i>"BestTargetDecoderRead" on page 141</i>        | Reads the properties of a decoder to the preparation register.                                    |
| <i>"BestTargetDecoderPowerUpProg" on page 135</i> | Programs the power up properties of a decoder with properties as set in the preparation register. |
| <i>"BestTargetDecoderPowerUpRead" on page 136</i> | Reads the power up properties of a decoder to the preparation register.                           |

**Target Attribute Memory** The following functions are used to program the target attribute memory:

| Function  | Result  |
|---|---|
| <i>"BestTargetAttrPageInit" on page 131</i>       | Initializes a target attribute memory page.   |
| <i>"BestTargetAttrPropDefaultSet" on page 132</i> | Sets the target attribute memory's preparation register to the default values.        |
| <i>"BestTargetAttrPropSet" on page 134</i>        | Sets a target attribute property in the preparation register.                         |
| <i>"BestTargetAttrPropGet" on page 133</i>        | Reads a target attribute property from the preparation register.                      |
| <i>"BestTargetAttrLineProg" on page 129</i>       | Programs the target attribute memory's preparation register to a memory line.         |
| <i>"BestTargetAttrLineRead" on page 130</i>       | Reads a target attribute memory line to the preparation register.                     |
| <i>"BestTargetAttrGroupLineProg" on page 127</i>  | Programs a group of target attributes from the preparation register to a memory line. |
| <i>"BestTargetAttrGroupLineRead" on page 128</i>  | Reads a group of target attributes from a memory line to the preparation register.    |
| <i>"BestTargetAttrPageSelect" on page 132</i>     | Selects a target attribute memory page for transfer.                                  |

How to use the functions is described in *"Programming the Exerciser as a Target Device"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestTargetAttrGroupLineProg

**Call** `b_errtype BestTargetAttrGroupLineProg(  
     b_handletype       handle,  
     b_tattrgrouptype   group,  
     b_int32            page_num,  
     b_int32            offset );`

**Description** Writes the attributes of a group from the preparation register to a line in the attribute memory.

After you have programmed all memory lines within a group, set the individual loop bit B\_T\_DOLOOP with “*BestTargetAttrPropSet*” on page 134 to indicate the last line of the current group.

**NOTE** Within a page you can use either group or non-group memory programming functions, but not both together.

**CLI Equivalent** `BestTargetAttrGroupLineProg group=<group> page_num=<page_num>  
 offset=<offset>`

**CLI Abbreviation** `tagrplprog grp=<group> page=<page_num> offs=<offset>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**group** Identifier of the group; see “*b\_tattrgrouptype*” on page 287.

**page\_num** Number of the page on which the line is to be programmed.

Valid page numbers are:

- 1 ... 63 if page size is 4
- 1 ... 7 if page size is 32

The page size is set by the generic exerciser property B\_EGEN\_ATTRPAGESIZE with “*BestExerciserGenPropSet*” on page 100.

**offset** Number of the line to be programmed. Line counting starts from 0.

**NOTE** You can program more lines than are available on a page. This results in concatenated pages.

**See also** “*BestTargetAttrPageInit*” on page 131  
 “*BestTargetAttrGroupLineRead*” on page 128  
 “*BestTargetAttrLineProg*” on page 129

## BestTargetAttrGroupLineRead

**Call** `b_errtype BestTargetAttrGroupLineRead(  
           b_handletype        handle,  
           b_tattrgrouptype   group,  
           b_int32            page_num,  
           b_int32            offset );`

**Description** Reads the attributes of a group from a line in the attribute memory to the preparation register.

**CLI Equivalent** `BestTargetAttrGroupLineRead group=<group> page_num=<page_num>  
 offset=<offset>`

**CLI Abbreviation** `tagrplread grp=<group> page=<page_num> offs=<offset>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**group** Identifier of the group; see “*b\_tattrgrouptype*” on page 287.

**page\_num** Number of the page on which the line is to be read.

Valid page numbers are:

- 1 ... 63 if page size is 4
- 1 ... 7 if page size is 32

The page size is set by the generic exerciser property  
 B\_EGEN\_ATTRPAGESIZE with “*BestExerciserGenPropSet*” on  
 page 100.

**offset** Number of the line to be read. Line counting starts from 0.

**See also** “*BestTargetAttrPageInit*” on page 131  
 “*BestTargetAttrGroupLineProg*” on page 127  
 “*BestTargetAttrLineRead*” on page 130



## BestTargetAttrLineProg

**Call** `b_errtype BestTargetAttrLineProg(  
           b_handletype handle,  
           b_int32 page_num,  
           b_int32 offset );`

**Description** Writes the contents of the preparation register to a line in the attribute memory.

After you have programmed all memory lines, set the general loop bit B\_T\_DOLOOP with “*BestTargetAttrPropSet*” on page 134 to indicate the last line in the attribute page.

**NOTE** Within a page you can use either group or non-group memory programming functions, but not both together.

**CLI Equivalent** `BestTargetAttrLineProg page_num=<page_num> offset=<offset>`

**CLI Abbreviation** `talprog page=<page_num> offs=<offset>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**page\_num** Number of the page on which the line is to be programmed.

Valid page numbers are:

- 1 ... 63 if page size is 4
- 1 ... 7 if page size is 32

The page size is set by the generic exerciser property B\_EGEN\_ATTRPAGESIZE with “*BestTargetAttrPropSet*” on page 134.

**offset** Number of the line to be programmed. Line counting starts from 0.

**NOTE** You can program more lines than are available on a page. This results in concatenated pages.

**See also** “*BestTargetAttrPageInit*” on page 131  
 “*BestTargetAttrGroupLineProg*” on page 127  
 “*BestTargetAttrLineRead*” on page 130

## BestTargetAttrLineRead

**Call**     `b_errtype BestTargetAttrLineRead(  
                   b_handletype    handle,  
                   b_int32        page_num,  
                   b_int32        offset );`

**Description**   Reads the contents of an attribute memory's line to the preparation register.

**CLI Equivalent**   `BestTargetAttrLineRead page_num=<page_num> offset=<offset>`

**CLI Abbreviation**   `talread page=<page_num> offs=<offset>`

**Return Value**     Error code; see “*b\_errtype*” on page 250.

**Input Parameters**   **handle**     Handle to identify the session.

**page\_num**     Number of the page from which the line is to be read.

**offset**     Number of the line to be read.

**See also**     “*BestTargetAttrPageInit*” on page 131  
                   “*BestTargetAttrGroupLineRead*” on page 128  
                   “*BestTargetAttrLineProg*” on page 129

## BestTargetAttrPageInit

**Call** `b_errtype BestTargetAttrPageInit (`  
           `b_handletype  handle,`  
           `b_int32      page_num );`

**Description** Initializes a target attribute memory page (and all pages concatenated to this page).

This function must be called once before an attribute page can be programmed or read.

**NOTE** BestTargetAttrPageInit fails if you call this function while a transaction is running.

**CLI Equivalent** `BestTargetAttrPageInit page_num=<page_num>`

**CLI Abbreviation** `tapginit page=<page_num>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**page\_num** Number of the memory page to be initialized.

Valid page numbers are:

- 1 ... 63 if page size is 4
- 1 ... 7 if page size is 32

The page size is set by the generic exerciser property

B\_EGEN\_ATTRPAGESIZE with “*BestTargetAttrPropSet*” on page 134.

**See also** “*BestTargetAttrGroupLineProg*” on page 127

“*BestTargetAttrGroupLineRead*” on page 128

“*BestTargetAttrLineProg*” on page 129

“*BestTargetAttrLineRead*” on page 130

## BestTargetAttrPageSelect

**Call** `b_errtype BestTargetAttrPageSelect (`  
     `b_handletype handle,`  
     `b_int32 page_num );`

**Description** Selects a target attribute page.

**CLI Equivalent** `BestTargetAttrPageSelect page_num=<page_num>`

**CLI Abbreviation** `tapgsel page=<page_num>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**page\_num** Number of the memory page to be selected.

Valid page numbers are:

- 1 ... 63 if page size is 4
- 1 ... 7 if page size is 32

The page size is set by the generic exerciser property

`B_EGEN_ATTRPAGESIZE` with “*BestTargetAttrPropSet*” on page 134.

**See also** –

## BestTargetAttrPropDefaultSet

**Call** `b_errtype BestTargetAttrPropDefaultSet( b_handletype handle );`

**Description** Sets the preparation register of the target attribute memory to default values. For a description of attributes and default values, see “*b\_tattrproptype*” on page 288.

**CLI Equivalent** `BestTargetAttrPropDefaultSet`

**CLI Abbreviation** `taprpdefset`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestMasterAttrPropSet*” on page 117  
 “*BestMasterAttrPropGet*” on page 116

## BestTargetAttrPropGet

**Call**    `b_errtype BestTargetAttrPropGet (`  
                  `b_handletype        handle,`  
                  `b_tattrproptype    tattrprop,`  
                  `b_int32                *value );`

**Description**    Reads a target attribute from the attribute preparation register.

**CLI Equivalent**    `BestTargetAttrPropGet tattrprop=<tattrprop>`

**CLI Abbreviation**    `taprpget prop=<tattrprop>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**tattrprop**    Target attribute to be read; see “*b\_tattrproptype*” on page 288.

**Output Parameters**    **value**    Value of the attribute; see “*b\_tattrproptype*” on page 288.

**See also**    “*BestMasterAttrPropDefaultSet*” on page 116  
                  “*BestMasterAttrPropSet*” on page 117

## BestTargetAttrPropSet

**Call** `b_errtype BestTargetAttrPropSet (`  
     `b_handletype       handle,`  
     `b_tattrproptype   tattrprop,`  
     `b_int32            value );`

**Description** Sets a target attribute in the preparation register of the target attribute memory.

After you have set all attributes to the required values in the register, you can program the complete register to a memory line with “*BestTargetAttrLineProg*” on page 129.

**CLI Equivalent** `BestTargetAttrPropSet tattrprop=<tattrprop> value=<value>`

**CLI Abbreviation** `taprpset prop=<tattrprop> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**tattrprop** Target attribute to be set; see “*b\_tattrproptype*” on page 288.

**value** Value to which the attribute is set; see “*b\_tattrproptype*” on page 288.

**See also** “*BestMasterAttrPropDefaultSet*” on page 116  
 “*BestMasterAttrPropGet*” on page 116

## BestTargetDecoderPowerUpProg

**Call** `b_errtype BestTargetDecoderPowerUpProg(  
           b_handletype handle,  
           b_decodertype decoder );`

**Description** Programs power up properties of a decoder with the properties set in the preparation register by means of “*BestTargetDecoderPropSet*” on page 140. (Use *BestTargetDecoderPropSet* to change decoder base address register settings or decoder size when the card is operating.)

The function also checks whether the property values in the target decoder preparation register are consistent with the specified decoder, except when the decoder is set to “custom” behavior.

**NOTE** *BestTargetDecoderPowerUpProg* fails if you call this function while a transaction is running.

**CLI Equivalent** `BestTargetDecoderPowerUpProg decoder_num=<decoder>`

**CLI Abbreviation** `tdpuprog dec=<decoder>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**decoder** Decoder to be programmed; see “*b\_decodertype*” on page 243.

**See also** “*BestTargetDecoderPowerUpRead*” on page 136  
 “*BestTargetDecoderProg*” on page 137

## BestTargetDecoderPowerUpRead

**Call**     `b_errtype BestTargetDecoderPowerUpRead(  
                  b_handletype    handle,  
                  b_decodertype   decoder );`

**Description**   Reads back a set of decoder power up properties to the decoder property preparation register.

**CLI Equivalent**   `BestTargetDecoderPowerUpRead decoder_num=<decoder>`

**CLI Abbreviation**   `tdpuread dec=<decoder>`

**Return Value**     Error code; see “*b\_errtype*” on page 250.

**Input Parameters**   **handle**     Handle to identify the session.

**decoder**     Decoder to be read; see “*b\_decodertype*” on page 243.

**See also**     “*BestTargetDecoderPowerUpProg*” on page 135  
                  “*BestTargetDecoderRead*” on page 141



## BestTargetDecoderProg

**Call** `b_errtype BestTargetDecoderProg(  
           b_handletype    handle,  
           b_decodertype   decoder );`

**Description** Programs a decoder using the properties set in the preparation register.

This function also checks whether the property values in the target decoder preparation register are consistent with the specified decoder, except when the decoder is set to “custom” behavior.

**NOTE** BestTargetDecoderProg fails if you call this function while a transaction is running.

**CLI Equivalent** `BestTargetDecoderProg decoder_num=<decoder>`

**CLI Abbreviation** `tdprog dec=<decoder>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**decoder** Decoder to be programmed; see “*b\_decodertype*” on page 243.

**See also** “*BestTargetDecoderPowerUpProg*” on page 135  
 “*BestTargetDecoderRead*” on page 141

## BestTargetDecoderPropDefaultSet

**Call** `b_errtype BestTargetDecoderPropDefaultSet (`  
     `b_handletype     handle,`  
     `b_decodertype    decoder_num );`

**Description** Sets the preparation register to the default values of a decoder. Each decoder provides a specific parameter set.

**CLI Equivalent** `BestTargetDecoderPropDefaultSet decoder_num=<decoder_num>`

**CLI Abbreviation** `tdprpdefset dec=<decoder_num>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**decoder\_num** Number of the decoder of which the default values are to be set in the preparation register. See “*b\_decodertype*” on page 243.

**See also** “*BestTargetDecoderPropSet*” on page 140  
           “*BestTargetDecoderPropGet*” on page 139

## BestTargetDecoderPropGet

**Call** `b_errtype BestTargetDecoderPropGet (`  
           `b_handletype     handle,`  
           `b_decproptype   decoder_prop,`  
           `b_int32           *value );`

**Description** Reads a decoder property from the decoder preparation register.

**NOTE** BestTargetDecoderPropGet fails if you call this function while a transaction is running.

**CLI Equivalent** `BestTargetDecoderPropGet decoder_prop=<decoder_prop>`

**CLI Abbreviation** `tdprpget prop=<decoder_prop>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**decoder\_prop** Decoder property to be read; see “*b\_decproptype*” on page 243.

**Output Parameters** **value** Property value; see “*b\_decproptype*” on page 243.

**See also** “*BestTargetDecoderPropDefaultSet*” on page 138  
 “*BestTargetDecoderPropSet*” on page 140

## BestTargetDecoderPropSet

**Call**    `b_errtype BestTargetDecoderPropSet (`  
                  `b_handletype     handle,`  
                  `b_decproptype   decoder_prop,`  
                  `b_int32            value );`

**Description**    Sets a property of a decoder in the preparation register.

**NOTE**    This function overrides configuration space settings made with “*BestConfRegSet*” on page 144.

**CLI Equivalent**    `BestTargetDecoderPropSet decoder_prop=<decoder_prop> value=<value>`

**CLI Abbreviation**    `tdprpset prop=<decoder_prop> val=<value>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**decoder\_prop**    Defines the property to be set; see “*b\_decproptype*” on page 243.

**value**    Value to which the specified property is set; see “*b\_decproptype*” on page 243.

**See also**    “*BestTargetDecoderPropDefaultSet*” on page 138  
               “*BestTargetDecoderPropGet*” on page 139

## BestTargetDecoderRead

**Call**     `b_errtype BestTargetDecoderRead(  
                  b_handletype    handle,  
                  b_decodertype   decoder );`

**Description**   Reads back a set of decoder properties to the decoder property preparation register.

**CLI Equivalent**   `BestTargetDecoderRead decoder_num=<decoder>`

**CLI Abbreviation**   `tdread   dec=<decoder>`

**Return Value**     Error code; see “*b\_errtype*” on page 250.

**Input Parameters**   **handle**     Handle to identify the session.

**decoder**     Decoder to be read; see “*b\_decodertype*” on page 243.

**See also**     “*BestTargetDecoderProg*” on page 137  
                  “*BestTargetDecoderPowerUpRead*” on page 136

# Configuration Space Programming Functions

The following functions program the configuration space:

| Function                                 | Result   |
|--|--|
| <i>"BestConfRegSet" on page 144</i>      | Sets a register in the configuration space.                            |
| <i>"BestConfRegGet" on page 143</i>      | Reads a register in the configuration space.                           |
| <i>"BestConfigScan" on page 147</i>      | Scans configuration space header registers.                            |
| <i>"BestConfigScanPrint" on page 147</i> | Returns configuration space header registers.                          |
| <i>"BestConfRegMaskSet" on page 146</i>  | Sets a bit mask for read-only bits in a configuration space register.  |
| <i>"BestConfRegMaskGet" on page 145</i>  | Reads a bit mask for read-only bits in a configuration space register. |
| <i>"BestPCIConfigCheck" on page 148</i>  | Checks the configuration space header settings for PCI compliance.     |

How to use the functions is described in *"Configuration Space Header Programming"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestConfRegGet

**Call**    `b_errtype BestConfRegGet (  
                  b_handletype    handle,  
                  b_int32        offset,  
                  b_int32        *value );`

**Description**    Reads a value of a configuration space register.

**CLI Equivalent**    `BestConfRegGet offset=<offset>`

**CLI Abbreviation**    `conrget offs=<offset>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**offset**    Address offset in the configuration space (00 ... 3C\h, dword aligned).

**Output Parameters**    **value**    Value of the register (32-bit value).

**See also**    “*BestConfRegSet*” on page 144

## BestConfRegSet

**Call**     `b_errtype BestConfRegSet (  
                 b_handletype    handle,  
                 b_int32          offset,  
                 b_int32          value );`

**Description**     Sets a register of the configuration space to the specified value.

**NOTE**     BestConfRegSet fails if you call this function while a transaction is running or if the settings conflict with those of a decoder property or decoder behavior. Refer to “*b\_decproptype*” on page 243.

**CLI Equivalent**     `BestConfRegSet offset=<offset> value=<value>`

**CLI Abbreviation**     `conrset offs=<offset> val=<value>`

**Return Value**     Error code; see “*b\_errtype*” on page 250.

**Input Parameters**     **handle**     Handle to identify the session.

**offset**     Address offset in the configuration space (00 ... 3C\h, dword aligned).

**value**     Value to which the register is set (32-bit value).

**See also**     “*BestConfRegGet*” on page 143



## BestConfRegMaskGet

**Call** `b_errtype BestConfRegMaskGet (`  
           `b_handletype  handle,`  
           `b_int32      offset,`  
           `b_int32      *value );`

**Description** Reads the mask that defines which bits in a configuration space register are set to read-only and which to read/write for configuration accesses.

**CLI Equivalent** `BestConfRegMaskGet offset=<offset>`

**CLI Abbreviation** `conrmaskget offs=<offset>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**offset** Address offset in the configuration space (00 ... 3C\h, dword aligned).

**Output Parameters** **value** 32-bit mask:

- 0 – read-only
- 1 – read/write

**See also** “*BestConfRegMaskSet*” on page 146

## BestConfRegMaskSet

**Call**    `b_errtype BestConfRegMaskSet (`  
              `b_handletype    handle,`  
              `b_int32            offset,`  
              `b_int32            value );`

**Description**    Sets the mask that defines which bits in a configuration space register are set to read-only and which to read/write for configuration accesses.

**NOTE**            This function fails if the settings conflict with those of a decoder property or decoder behavior. Refer to “*b\_decproptype*” on page 243.

**CLI Equivalent**    `BestConfRegMaskSet offset=<offset> value=<value>`

**CLI Abbreviation**    `conrmaskset offs=<offset> val=<value>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**offset**    Address offset in the configuration space (00 ... 3C\h, dword aligned).

**value**    32-bit mask:

- 0 – read-only
- 1 – read/write

**See also**    “*BestConfRegMaskGet*” on page 145

## BestConfigScan

**Call** `b_errtype BestConfigScan( b_handletype handle );`

**Description** Scans the information stored in the registers of the configuration space header of each PCI device connected to the PCI bus of the system under test.

After this function call you can use “*BestConfigScanPrint*” on page 147 to print the scanned information.

**CLI Equivalent** `BestConfigScan`

**CLI Abbreviation** `cscan`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** –

## BestConfigScanPrint

**Call** `b_errtype BestConfigScanPrint(  
          b_handletype handle,  
          b_charptrtype *res_string );`

**Description** Returns the information stored in the registers of the configuration space header of each PCI device connected to the PCI bus of the system under test.

Before using this function, call “*BestConfigScan*” on page 147 to get the current information from the configuration space headers.

**CLI Equivalent** `BestConfigScanPrint`

**CLI Abbreviation** `cscanprnt`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**Output Parameters** **res\_string** Text string.

**See also** –

## BestPCIConfigCheck

**Call** `b_errtype BestPCIConfigCheck( b_handletype handle );`

**Description** Checks the current settings in the configuration space header of the testcard for PCI compliance.

**CLI Equivalent** `BestPCIConfigCheck`

**CLI Abbreviation** `cfcgcheck`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** –

## Expansion ROM Programming Functions

The following functions are used to program the expansion ROM memory space of the testcard:

| Function                                 | Result                               |
|--|--------------------------------------|
| <i>“BestExpRomByteWrite” on page 150</i> | Writes a byte to the expansion ROM.  |
| <i>“BestExpRomByteRead” on page 149</i>  | Reads a byte from the expansion ROM. |

How to use the functions is described in “*Expansion ROM Programming*” in the *Agilent E2927A Opt. 320 C-API/PPR Programmer’s Guide*.

## BestExpRomByteRead

**Call**    `b_errtype BestExpRomByteRead(  
                  b_handletype  handle,  
                  b_int32      offset,  
                  b_int32      *value );`

**Description**    Reads a byte from the specified offset in the expansion ROM.

**CLI Equivalent**    `BestExpRomByteRead offset=<offset>`

**CLI Abbreviation**    `erbyteread offs=<offset>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**offset**    Address offset (in byte) in the expansion ROM  
                                  (0000h ... FFFFh).

**Output Parameters**    **value**    Read data byte (bits 0 ... 7).

**NOTE**    When a byte is written into the expansion rom and read immediatly after that can return incorrect data. The written data is correct, you cannot directly read from the expansion rom after you have written a byte.

**See also**    “*BestExpRomByteWrite*” on page 150

## BestExpRomByteWrite

**Call** `b_errtype BestExpRomByteWrite(  
     b_handletype handle,  
     b_int32 offset,  
     b_int32 value );`

**Description** Writes one byte to the specified offset in the expansion ROM.

**CLI Equivalent** `BestExpRomByteWrite offset=<offset> value=<value>`

**CLI Abbreviation** `erbytewrite offs=<offset> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**offset** Address offset (in byte) in the expansion ROM (0000h ... FFFFh).

**value** Data byte to be written (bits 0 ... 7).

**NOTE** When a byte is written into the expansion ROM and immediately read, it is possible that incorrect data is returned. Bytes cannot be read immediately after data has been written to them.

**See also** “*BestExpRomByteRead*” on page 149

## Data Memory Functions

The following functions access the data memory for master and target transactions:

| Function                              | Result  |
|---------------------------------------|---|
| <i>“BestDataMemInit” on page 151</i>  | Initializes the data memory of the testcard by filling it with zeros. |
| <i>“BestDataMemWrite” on page 152</i> | Writes to the data memory of the testcard.                            |
| <i>“BestDataMemRead” on page 151</i>  | Reads from the data memory of the testcard.                           |

How to use the functions is described in “*Data Memory and Compare Unit Programming*” in the *Agilent E2927A Opt. 320 C-API/PPR Programmer’s Guide*.

## BestDataMemInit

**Call** `b_errtype BestDataMemInit( b_handletype handle );`

**Description** Initializes the internal data memory of the testcard and fills it completely with zeros.

**CLI Equivalent** `BestDataMemInit`

**CLI Abbreviation** `dataminit`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** –

## BestDataMemRead

**Call** `b_errtype BestDataMemRead(  
           b_handletype handle,  
           b_int32 int_addr,  
           b_int32 num_of_bytes,  
           b_int8 *data_ptr );`

**Description** Reads a data block from the data memory of the testcard to the control PC memory via the control interface.

**NOTE** Using the CLI restricts the number of bytes to the internal buffer size for the CLI output (8000h). Within C programs the number of bytes is only restricted by the size of the testcard’s data memory.

**CLI Equivalent** `BestDataMemRead int_addr=<int_addr> num_of_bytes=<num_of_bytes>  
 [data_ptr>"file path"]`

**CLI Abbreviation** `datamrd iad=<int_addr> nob=<num_of_bytes> [data>"file path"]`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**int\_addr** Start address of source data within data memory of the testcard. The range depends on the available memory size.

**num\_of\_bytes** Number of bytes to be read (maximum of 8000\h when using the CLI).

**data\_ptr** **CLI only.** This parameter allows export of the data to a file. Use a redirection operator, for example: data\_ptr>"file path".

**Output Parameters** **data\_ptr** Destination buffer in the memory of the control PC.

**See also** *"BestDataMemInit" on page 151*  
*"BestDataMemWrite" on page 152*

## BestDataMemWrite

**Call**

```
b_errtype BestDataMemWrite(
    b_handletype  handle,
    b_int32       int_addr,
    b_int32       num_of_bytes,
    b_int8        *data_ptr );
```

**Description** Writes a data block from the memory of the control PC to the data memory of the testcard. The data memory is shared by master and target. Both can use the data for further testing.

**CLI Equivalent** BestDataMemWrite int\_addr=<int\_addr> num\_of\_bytes=<num\_of\_bytes>  
data\_ptr=<{data\_list}>

**CLI Abbreviation** datamwr iad=<int\_addr> nob=<num\_of\_bytes> data=<{data\_list}>

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**int\_addr** Destination start address for the data within the data memory of the testcard. The range depends on the available memory size.

**num\_of\_bytes** Number of bytes to be written.

**data\_ptr** Source data in the system memory of the control PC.

**data\_ptr** **CLI only.** List of data to be transferred when using the CLI (for example, data={1\h, 2\h, 3\h, 4\h, 5\h, 6\h, 7\h, 8\h}).



Data may also be imported from a file using a redirection operator (for example, `data_ptr<"file path"`). The input file must contain a sequence of byte values in hexadecimal format, as shown in the following example:

```
01 23 45 67 89 AB CD EF 01 23 45 67 89 AB CD EF
EF CD AB 89 67 45 23 01 EF CD AB 89 67 45 23 01
```

**See also** *“BestDataMemInit” on page 151*  
*“BestDataMemRead” on page 151*

## Host Access Functions

The following functions are used for data transfer to and from the host (control PC):

| Function   | Result   |
|--|--|
| <i>“BestHostSysMemAccessPrepare” on page 156</i> | Prepares a transfer between the testcard’s data memory and the host. |
| <i>“BestHostSysMemDump64” on page 157</i>        | Transfers data from a PCI device to the host system memory.          |
| <i>“BestHostSysMemFill64” on page 159</i>        | Transfers data from the host system memory to a PCI device.          |
| <i>“BestHostPCIRegSet” on page 155</i>           | Sets register of a PCI device.                                       |
| <i>“BestHostPCIRegGet” on page 154</i>           | Reads register of a PCI device.                                      |

How to use the functions is described in *“Host Access Programming”* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer’s Guide*.

## BestHostPCIRegGet

**Call**

```
b_errtype BestHostPCIRegGet (
    b_handletype    handle,
    b_addrspacetype addrspace,
    b_int32         bus_addr,
    b_int32         *reg_value,
    b_sizetype      size );
```

**Description** Reads the value from a specific PCI device register in a 32-bit address space—the type of address space determines a Configuration, Memory or I/O Read.

The bus address is a byte address. The function performs single cycle transactions, automatically setting the correct byte enables corresponding to the word size and bus address.

This function only applies to a 32 bit address space.

**NOTE** “*BestConfRegSet*” on page 144 and “*BestConfRegGet*” on page 143 access the configuration space registers of the testcard.

**CLI Equivalent** HostPCIRegGet addrspace=<addrspace> bus\_addr=<bus\_addr>  
wordsize=<size>

**CLI Abbreviation** hprgget space=<addrspace> bad=<bus\_addr> size=<size>

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**addrspace** Type of address space for the register access; see “*b\_addrspacetype*” on page 237.

**bus\_addr** PCI bus address.

**size** Defines the width of the value to be read from the register; see “*b\_sizetype*” on page 284.

**Output Parameters** **reg\_value** Value read from the register.

**See also** “*BestHostPCIRegSet*” on page 155

## BestHostPCIRegSet

**Call** `b_errtype BestHostPCIRegSet (`  
     `b_handletype      handle,`  
     `b_addrspacetype    addrspace,`  
     `b_int32            bus_addr,`  
     `b_int32            reg_value,`  
     `b_sizetype        size );`

**Description** Writes a value to a specific PCI device register—the type of address space determines a Configuration Write, Memory Write or I/O Write.

The bus address is a byte address. The function performs single cycle transactions, automatically setting the correct byte enables corresponding to word size and bus address.

This function only applies to a 32 bit address space.

**NOTE** “*BestConfRegSet*” on page 144 and “*BestConfRegGet*” on page 143 access the configuration space registers of the testcard.

**CLI Equivalent** `BestHostPCIRegSet addrspace=<addrspace> bus_addr=<bus_addr>`  
`reg_value=<reg_value> wordsize=<size>`

**CLI Abbreviation** `hprgset space=<addr_space> bad=<bus_addr> val=<reg_value>`  
`size=<size>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**addrspace** Type of address space for the register access; see “*b\_addrspacetype*” on page 237.

**bus\_addr** PCI bus address.

**reg\_value** Value to be written to the register.

**size** Defines the width of the values to be set in the register. See “*b\_sizetype*” on page 284.

**See also** “*BestHostPCIRegGet*” on page 154

## BestHostSysMemAccessPrepare

**Call** `b_errtype BestHostSysMemAccessPrepare(  
     b_handletype handle,  
     b_int32 buscmd,  
     b_int32 bufsize );`

**Description** Prepares the internal address, the command in the master block properties, and a memory buffer for a transfer through the data memory of the testcard. A data verification can be activated.

Call this function before calling “*BestHostSysMemFill64*” on page 159 or “*BestHostSysMemDump64*” on page 157.

**CLI Equivalent** `BestHostSysMemAccessPrepare buscmd=<buscmd> bufsize=<bufsize>`

**CLI Abbreviation** `hsmarep cmd=<buscmd> buf=<bufsize>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**buscmd** Specifies the bus command. See “*b\_blkproptype*” on page 238. The command to be specified depends on the access function:

- If using “*BestHostSysMemFill64*” on page 159, the command must be set to B\_CMD\_MEM\_WRITE (Memory Write).
- If using “*BestHostSysMemDump64*” on page 157, the command must be set to B\_CMD\_MEM\_READ (Memory Read).

To enable data verification, OR-combine the bus command with B\_HOST\_VERIFY. If the verification fails, the access function returns a data compare error or a master abort.

**bufsize** Specifies the internal memory buffer size in dwords (minimum: 2 dwords).

**NOTE** The bufsize must be equal to or larger than the block size specified in “*BestHostSysMemFill64*” on page 159 or “*BestHostSysMemDump64*” on page 157.

**See also** –

## BestHostSysMemDump64

**Call** `b_errtype BestHostSysMemDump64 (`  
     `b_handletyp    handle,`  
     `b_int32        bus_addr_low,`  
     `b_int32        bus_addr_high,`  
     `b_int32        num_of_bytes,`  
     `b_int32        blocksize,`  
     `b_int8         *data_ptr );`

**Description** Moves data blocks from a device or from the memory in the system under test through the internal data memory of the testcard to the control PC (host). The device is specified by its PCI bus address.

During the master block transfer, the default master attributes are used (master attribute page 0). See “*b\_mattrproptype*” on page 260.

**NOTE** This function changes the settings of generic master properties. See “*b\_mastergenproptype*” on page 257.

Each master block transfer can consist of one or more bursts depending on the overall bus traffic and latency timer setting. If a data transfer results in a master abort, the function will *not* return an error, but the master abort is indicated by bit 7 of the status register (block aborted flag). You can read the status register with “*BestStatusRegGet*” on page 39.

Before you call this function for the first time, set the command to **Memory Read** and the size of the buffer in the testcard’s internal data memory with “*BestHostSysMemAccessPrepare*” on page 156 .

**CLI Equivalent** `BestHostSysMemDump64 bus_addr_low=<bus_addr_low>`  
     `bus_addr_high=<bus_addr_high> num_of_bytes=<num_of_bytes>`  
     `blocksize=<blocksize> [data>"file path"]`

**CLI Abbreviation** `hsmdump64 badl=<bus_addr_low> badh=<bus_addr_high>`  
     `nob=<num_of_bytes> blk=<blocksize> [data>"file path"]`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**bus\_addr\_low / bus\_addr\_high** PCI bus address from where the data is read. Addresses need not be aligned.

If the high bus address parameter is not equal to zero, the transfers are performed using dual address cycles.

**num\_of\_bytes** Number of bytes to be transferred:

- maximum of 128 Kbytes when you use the CLI.
- maximum of 4 Gbytes when you use the C-API.

**blocksize** Size of the master block transfers in bytes. This blocksize must be smaller or equal to the buffer size specified by *“BestHostSysMemAccessPrepare” on page 156*.

**data\_ptr** Destination in the memory of the control PC.

**data** **CLI only.** This parameter allows export of the data to a file. Use a redirection operator, for example: `data>"file path"`.

**See also** *“BestHostSysMemFill64” on page 159*

## BestHostSysMemFill64

**Call**

```
b_errtype BestHostSysMemFill64(
    b_handletype    handle,
    b_int32         bus_addr_low,
    b_int32         bus_addr_high,
    b_int32         num_of_bytes,
    b_int32         blocksize,
    b_int8          *data_ptr );
```

**Description** Moves data blocks from the control PC (host) through the internal data memory of the testcard to a device or to the memory in the system under test. The device is specified by its PCI bus address.

During the master block transfer, the default master attributes are used (master attribute page 0). See “*b\_mattrproptype*” on page 260.

**NOTE** This function changes the settings of generic master properties. See “*b\_mastergenproptype*” on page 257.

Each master block transfer can consist of one or more bursts depending on the overall bus traffic and latency timer setting. If a data transfer results in a master abort, the function will *not* return an error, but the master abort is indicated by bit 7 of the status register (block aborted flag). You can read the status register with “*BestStatusRegGet*” on page 39.

Before you call this function for the first time, set the command to **Memory Write** and the size of the buffer in the testcard’s internal data memory with “*BestHostSysMemAccessPrepare*” on page 156.

**CLI Equivalent**

```
BestHostSysMemFill64 bus_addr_low=<bus_addr_low>
bus_addr_high=<bus_addr_high> num_of_bytes=<num_of_bytes>
blocksize=<blocksize> ( data=<{data_list}> ) | ( data<"file path"> )
```

**CLI Abbreviation**

```
hsmfill64 badl=<bus_addr_low> badh=<bus_addr_high>
nob=<num_of_bytes>
blk=<blocksize> ( data=<{data_list}> ) | ( data<"file path"> )
```

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**bus\_addr\_low / bus\_addr\_high** PCI bus address in PCI memory space to where the data is written. Addresses need not be aligned.

If the high bus address parameter is not equal to zero, the transfers are performed using dual address cycles.

**num\_of\_bytes** Number of bytes to be transferred:

- maximum of 128 Kbytes when you use the CLI.
- maximum of 4 Gbytes when you use the C-API.

**blocksize** Size of the master block transfers in bytes. This blocksize must be smaller or equal to the buffer size specified with *“BestHostSysMemAccessPrepare”* on page 156.

**data\_ptr** Source data in the memory of the control PC.

**data** **CLI only.** List of data to be transferred when using the CLI (for example, data={1\h, 2\h, 3\h, 4\h, 5\h, 6\h, 7\h, 8\h}).

Data may also be imported from a file using a redirection operator (for example, data<"file path"). The input file must contain a sequence of byte values in hexadecimal format, as shown in the following example:

```
01 23 45 67 89 AB CD EF 01 23 45 67 89 AB CD EF
EF CD AB 89 67 45 23 01 EF CD AB 89 67 45 23 01
```

**See also** *“BestHostSysMemDump64”* on page 157



# Interrupt Generation Function

The following function is used to generate a PCI interrupt:

| Function                                   | Result                     |
|--|----------------------------|
| <i>"BestInterruptGenerate" on page 161</i> | Generates a PCI interrupt. |

How to use the functions is described in *"Interrupt Programming"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestInterruptGenerate

**Call** `b_errtype BestInterruptGenerate(  
          b_handletype handle,  
          b_int32 pci_int );`

**Description** Sets the specified PCI interrupt request pin INTA# ... INTD#.

To reset the interrupt, clear the corresponding status bit in the status register with *"BestStatusRegClear" on page 38*.

**CLI Equivalent** `BestInterruptGenerate pci_int=<pci_int>`

**CLI Abbreviation** `intgen int=<pci_int>`

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**pci\_int** Interrupts to be generated. See table below.

Multiple interrupts can be set by OR-combining the interrupt values, for example, `pci_int=B_INTA | B_INTC`.

| Value (CLI Abbreviation) | Description  |
|--------------------------|--|
| B_INTA (inta)            | PCI Interrupt Request A ... D<br>(INTA# ... INTD#) |
| ...                      |  |
| B_INTD (intd)            |  |

**See also** –

# Built-In Test Functions

The following functions program built-in tests:

| Function                                    | Result  |
|---|---|
| <i>"BestTestPropDefaultSet" on page 162</i> | Sets the built-in test property to defaults.      |
| <i>"BestTestPropSet" on page 163</i>        | Sets a built-in test property.                    |
| <i>"BestTestRun" on page 166</i>            | Starts a built-in test.                           |
| <i>"BestTestResultDump" on page 165</i>     | Dumps report and waveform file to the control PC. |
| <i>"BestTestProtErrDetect" on page 164</i>  | Detects protocol errors.                          |

How to use the functions is described in *"Built-In Test Programming"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestTestPropDefaultSet

**Call** `b_errtype BestTestPropDefaultSet( b_handletype handle );`

**Description** Sets all the test properties to default values. For a description of properties and default values, see *"b\_testproptype" on page 293*.

**CLI Equivalent** `BestTestPropDefaultSet`

**CLI Abbreviation** `testprpdefset`

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**See also** *"BestTestPropSet" on page 163*

## BestTestPropSet

**Call**    `b_errtype BestTestPropSet(  
                  b_handletype    handle,  
                  b_testproptype testprop,  
                  b_int32        value );`

**Description**    Sets a test property. The tests use different properties. Refer to “*Test Commands*” on page 167.

**CLI Equivalent**    `BestTestPropSet testprop=<testprop> value=<value>`

**CLI Abbreviation**    `testprpset prop=<testprop> val=<value>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**testprop**    Property to be set; see “*b\_testproptype*” on page 293.

**value**    Value to which the property is set, see “*b\_testproptype*” on page 293.

**See also**    “*BestTestPropDefaultSet*” on page 162

## BestTestProtErrDetect

**Call** `b_errtype BestTestProtErrDetect( b_handletype handle );`

**Description** Sets up the analyzer as a PCI protocol error checker. This performs the following:

- Clears the observer status register.
- Clears the “Trace Run Bit” in the Card Status Register.
- Sets the analyzer trigger pattern to trigger on protocol errors.
- Starts the protocol observer and the analyzer.

For further information see:

- “Protocol Observer Functions” on page 42
- “Card Status Functions” on page 38
- “Pattern Term Function” on page 62

**CLI Equivalent** `BestTestProtErrDetect`

**CLI Abbreviation** `testpedet`

**Return Value** Error code; see “b\_errtype” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** –

## BestTestResultDump

**Call** `b_errtype BestTestResultDump(  
           b_handletype    handle,  
           b_charptrtype   *filename );`

**Description** Saves the test results in a textual report file (<filename>.rpt) and in a waveform file (<filename>.wfm):

- The *report file* contains the analyzer status, the observer status, and compare errors.
- The *waveform file* contains the trace memory content and can be viewed with the listers of the graphical user interface.

**CLI Equivalent** `BestTestResultDump file=<filename>`

**CLI Abbreviation** `testrdump file=<filename>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle**    Handle to identify the session.

**filename**    String containing path and filename (no suffix), for example, “c:\temp\xx”.

**See also**    –

## BestTestRun

**Call**    `b_errtype BestTestRun(  
                  b_handletype handle,  
                  b_int32 testcmd );`

**Description**    Starts the specified test.

Before calling this command, you must set the properties for the specified type of test. The “*Test Commands*” on page 167 show which properties the tests use. You can set the properties with “*BestTestPropSet*” on page 163.

Depending on the generic master property B\_MGEN\_REPEATMODE, the test runs either once or loops infinitely. Refer to “*BestMasterGenPropSet*” on page 105.

You can stop the test with “*BestMasterStop*” on page 106. The test stops automatically after a data compare error (if data comparison is enabled).

**CLI Equivalent**    `BestTestRun testcmd=<testcmd>`

**CLI Abbreviation**    `testrun cmd=<testcmd>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**testcmd**    Test command, specifying the type of the test to be run. See “*Test Commands*” on page 167.

**See also**    –

## Test Commands

| Test Commands<br>(CLI Abbreviation)   | Description   | Test properties used by the<br>command  |
|---------------------------------------|---|---|
| B_TSTCMD_TRAFFICMAKE<br>(trafficmake) | Enables the card's master to write data to the card's target over the full PCI bus bandwidth. | B_TST_BANDWIDTH<br>B_TST_BLKLENGTH<br>B_TST_DATAPATTERN<br>B_TST_PROTOCOL   |
| B_TSTCMD_WRITEREAD<br>(writeread)     | Performs writes and reads to a PCI device or to system memory.                                | B_TST_BANDWIDTH<br>B_TST_BLKLENGTH<br>B_TST_DATAPATTERN<br>B_TST_PROTOCOL<br>B_TST_COMPARE<br>B_TST_SOURCEADDR<br>B_TST_NOBYTES                     |
| B_TSTCMD_BLOCKMOVE<br>(blockmove)     | Moves a data block from one system memory address to another.                                 | B_TST_BANDWIDTH<br>B_TST_BLKLENGTH<br>B_TST_DATAPATTERN<br>B_TST_PROTOCOL<br>B_TST_COMPARE<br>B_TST_SOURCEADDR<br>B_TST_DESTINADDR<br>B_TST_NOBYTES |
| B_TSTCMD_READ<br>(read)               | Performs read transactions from a PCI device or from system memory.                           | B_TST_BANDWIDTH<br>B_TST_BLKLENGTH<br>B_TST_PROTOCOL<br>B_TST_SOURCEADDR<br>B_TST_NOBYTES   |

“” in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*





# Interface Control Functions

The PCI Interface Control functions are divided into the following sections:

- *“CPU Port Programming Functions” on page 169*
- *“Static I/O Port Programming Functions” on page 177*
- *“Trigger I/O Sequencer Programming Functions” on page 181*
- *“Display Functions” on page 188*
- *“Mailbox Functions” on page 189*
- *“Power Management Event Functions” on page 194*

## CPU Port Programming Functions

The following functions are used to program the CPU port:

| Function                                       | Result                                     |
|--|--|
| <i>“BestCPUportPropSet” on page 173</i>        | Sets a property of the CPU port.           |
| <i>“BestCPUportWordBlockWrite” on page 171</i> | Writes a data block to the CPU port.       |
| <i>“BestCPUportWordBlockRead” on page 170</i>  | Reads a data block from the CPU port.      |
| <i>“BestCPUportWrite” on page 176</i>          | Writes data to the CPU port.               |
| <i>“BestCPUportRead” on page 174</i>           | Reads data from the CPU port.              |
| <i>“BestCPUportIntrClear” on page 172</i>      | Clears a CPU port interrupt.               |
| <i>“BestCPUportIntrStatusGet” on page 172</i>  | Gets the interrupt status of the CPU port. |
| <i>“BestCPUportRST” on page 175</i>            | Sets the RST# signal on the CPU port.      |

How to use the functions is described in *“CPU Port Programming”* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer’s Guide*.

## BestCPUportWordBlockRead

**Call**    `b_errtype BestCPUportWordBlockRead(  
           b_handletype    handle,  
           b_int32        device_num,  
           b_int32        address,  
           b_int32        *data_ptr,  
           b_int32        num_of_bytes );`

**Description**    Reads a data block of words from a device connected to the CPU port. Reading starts at the CPU port address given by the address parameter.

You should not use this function if a memory-mapped resource is connected to the CPU port.

**CLI Equivalent**    `BestCPUportWordBlockRead device_num=<device_num> address=<address>  
                          num_of_bytes=<num_of_bytes>`

**CLI Abbreviation**    `cpuwbread dev=<device_num> ad=<address> nob=<num_of_bytes>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**device\_num**    Selects one of two possible devices for the access (0 or 1). The device select lines will be set during the read.

**address**    Start address for the read (0 ... 64 k).

**num\_of\_bytes**    Number of bytes to be read.

**Output Parameters**    **data\_ptr**    Data. Range is 0 ... 64 k.

**See also**    “*BestCPUportWordBlockWrite*” on page 171  
               “*BestCPUportRead*” on page 174

## BestCPUportWordBlockWrite

**Call** `b_errtype BestCPUportWordBlockWrite(  
     b_handletype handle,  
     b_int32 device_num,  
     b_int32 address,  
     b_int16 *data_ptr,  
     b_int32 num_of_bytes );`

**Description** Writes a data block of words to a device connected to the CPU port. Writing starts at the CPU port address given by the address parameter. You should not use this function if a memory-mapped resource is internally connected to the CPU port.

**CLI Equivalent** `BestCPUportWordBlockWrite device_num=<device_num> address=<address>  
 data_ptr=<data_ptr> num_of_bytes=<num_of_bytes>`

**CLI Abbreviation** `cpuwbwrite dev=<device_num> ad=<address> data=<data_ptr>  
 nob=<num_of_bytes>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**device\_num** Selects one of two possible devices for the access (0 ... 1). The device select lines will be set during the write.

**address** Start address used on the CPU port for the write (0 ... 64 k).

**data\_ptr** Data. Range is 0 ... 64 k.

**num\_of\_bytes** Number of bytes to be written.

**See also** “*BestCPUportWordBlockRead*” on page 170  
 “*BestCPUportWrite*” on page 176

## BestCPUportIntrClear

**Call** `b_errtype BestCPUportIntrClear( b_handletype handle );`

**Description** Clears the CPU port interrupt.

**CLI Equivalent** `BestCPUportIntrClear`

**CLI Abbreviation** `cpuintclear`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestCPUportIntrStatusGet*” on page 172

## BestCPUportIntrStatusGet

**Call** `b_errtype BestCPUportIntrStatusGet (
 b_handletype handle,
 b_int32 *intvalue_ptr );`

**Description** Reads the interrupt state of the CPU port.

**CLI Equivalent** `BestCPUportIntrStatusGet`

**CLI Abbreviation** `cpuistatget`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**Output Parameters** **intvalue\_ptr** Status of the CPU port’s interrupt line:

- 0 – no interrupt
- 1 – interrupt is pending

**See also** “*BestCPUportIntrClear*” on page 172

## BestCPUportPropSet

**Call**     `b_errtype BestCPUportPropSet (`  
             `b_handletype     handle,`  
             `b_cpuproptype     cpuprop,`  
             `b_int32             value );`

**Description**     Sets the value of a CPU port property.

**CLI Equivalent**     `BestCPUportPropSet cpuprop=<cpuprop> value=<value>`

**CLI Abbreviation**     `cpuprpset prop=<cpuprop> val=<value>`

**Return Value**     Error code; see “*b\_errtype*” on page 250

**Input Parameters**     **hantrdle**     Handle to identify the session.

**cpuprop**     Property to be set; see “*b\_cpuproptype*” on page 242.

**value**     Value of the property to be set; see “*b\_cpuproptype*” on page 242.

**See also**     –

## BestCPUportRead

**Call** `b_errtype BestCPUportRead(  
     b_handletype handle,  
     b_int32 device_num,  
     b_int32 address,  
     b_sizetype size,  
     b_int32 *data_ptr );`

**Description** Reads data from a device connected to the CPU port using the specified address on the CPU port's address lines.

**CLI Equivalent** `BestCPUportRead device_num=<device_num> address=<address>  
 size=<size>`

**CLI Abbreviation** `cpuread dev=<device_num> ad=<address> size=<size>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**device\_num** Selects one of two possible devices for the access (0 ... 1). The device select lines will be set during the read.

**address** Address used for the read (0 ... 64 k).

**size** Size of the data to be read (data width). 8 or 16 bits. Must be aligned. Valid values are:

| size (CLI Abbreviations) |
|--------------------------|
| B_SIZE_BYTE (1)          |
| B_SIZE_WORD (2)          |

**Output Parameters** **data\_ptr** Data value to be read.

Data ranges are:

- 0 ... 64 k if size is 16 bits
- 0 ... 255 if size is 8 bits

**See also** “*BestCPUportWordBlockRead*” on page 170  
 “*BestCPUportWrite*” on page 176

## BestCPUportRST

**Call**     `b_errtype BestCPUportRST(  
                  b_handletype    handle,  
                  b_int32         value );`

**Description**    Controls the RST# line of the CPU port.

For example, to pulse RST#, call this function successively with the first call setting the value to 0, the second to 1 and so on. To achieve a defined pulse duration, use a timed loop between the calls.

**CLI Equivalent**    `BestCPUportRST    value=<value>`

**CLI Abbreviation**   `cpurst val=<value>`

**Return Value**     Error code; see “*b\_errtype*” on page 250.

**Input Parameters**   **handle**     Handle to identify the session.  
  
                  **value**     Value of the RST# signal (0 or 1).

**See also**     –

## BestCPUportWrite

**Call** `b_errtype BestCPUportWrite(  
     b_handletype handle,  
     b_int32 device_num,  
     b_int32 address,  
     b_int32 data,  
     b_sizetype size );`

**Description** Writes data to a device connected to the CPU port using the specified address on the CPU port's address lines.

**CLI Equivalent** `BestCPUportWrite device_num=<device_num> address=<address>  
 data=<data> size=<size>`

**CLI Abbreviation** `cpuwrite dev=<device_num> ad=<address> val=<data> size=<size>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**device\_num** Selects one of two possible devices for the access (0 ... 1). The device select lines will be set during the write.

**address** Address used for the write (0 ... 64 k).

**data** Data value to be written.

Data ranges are:

- 0 ... 64 k if size is 16 bits
- 0 ... 255 if size is 8 bits

**size** Size of the data to be written (data width). 8 or 16 bits. Must be aligned. Valid values are:

| size (CLI Abbreviations) |
|--------------------------|
| B_SIZE_BYTE (1)          |
| B_SIZE_WORD (2)          |

**See also** “*BestCPUportRead*” on page 174  
 “*BestCPUportWordBlockWrite*” on page 171



# Static I/O Port Programming Functions

The following functions are used to program the static I/O port:

| Function                                | Result   |
|---|--|
| <i>"BestStaticPropSet" on page 179</i>  | Sets a property value of the static I/O port.  |
| <i>"BestStaticPropGet" on page 178</i>  | Reads a property value of the static I/O port. |
| <i>"BestStaticWrite" on page 180</i>    | Writes a byte to the static I/O port.          |
| <i>"BestStaticPinWrite" on page 177</i> | Writes a bit to the static I/O port.           |
| <i>"BestStaticRead" on page 180</i>     | Reads a byte from the static I/O port.         |

How to use the functions is described in *"Static I/O Port Programming"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestStaticPinWrite

**Call** `b_errtype BestStaticPinWrite(  
           b_handletype  handle,  
           b_int32      pin_num,  
           b_int32      value );`

**Description** Writes a bit to a pin of the static I/O port, or inverts it for a period of time. The other pins remain unchanged.

**CLI Equivalent** `BestStaticPinWrite pin_num=<pin_num> value=<value>`

**CLI Abbreviation** `spwrite pin=<pin_num> val=<value>`

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**pin\_num** Identifier for the pin (0 ... 7, no default).

**value** Bit value for the pin (0, 1 or 2). 2 inverts the pin for approximately 130 ms and then restores its previous value.

**See also** *"BestStaticWrite" on page 180*

## BestStaticPropGet

**Call** `b_errtype BestStaticPropGet (`  
     `b_handletype        handle,`  
     `b_int32            pin_num,`  
     `b_staticproptype   staticprop,`  
     `b_int32            *value );`

**Description** Reads each pin of the 8-bit static I/O port as either:

- input only
- open-drain output
- totem-pole output

**CLI Equivalent** `BestStaticPropGet pin_num=<pin_num> staticprop=<staticprop>`

**CLI Abbreviation** `sprpget pin=<pin_num> prop=<staticprop>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**pin\_num** Pin for which the property is to be read (0 ... 7).

**staticprop** Property to be read; see “*b\_staticproptype*” on page 285.

**Output Parameters** **value** Setting for the pin; see “*b\_staticproptype*” on page 285.

**See also** “*BestStaticPropSet*” on page 179

## BestStaticPropSet

**Call** `b_errtype BestStaticPropSet (`  
     `b_handletype           handle,`  
     `b_int32                pin_num,`  
     `b_staticproptype     staticprop,`  
     `b_int32                value );`

**Description** Sets each pin of the 8-bit static I/O port as either:

- input only
- open-drain output
- totem-pole output

**CLI Equivalent** `BestStaticPropSet pin_num=<pin_num> staticprop=<staticprop>`  
`value=<value>`

**CLI Abbreviation** `sprpset pin=<pin_num> prop=<staticprop> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**pin\_num** Pin for which the property is to be set (0 ... 7).

**staticprop** Property to be set; see “*b\_staticproptype*” on page 285.

**value** Setting for the pin; see “*b\_staticproptype*” on page 285.

**See also** “*BestStaticPropGet*” on page 178

## BestStaticRead

**Call**     `b_errtype BestStaticRead(  
                  b_handletype    handle,  
                  b_int32        *value );`

**Description**   Reads a byte from the static I/O port.

**CLI Equivalent**   `BestStaticRead`

**CLI Abbreviation**   `sread`

**Return Value**   Error code; see “*b\_errtype*” on page 250.

**Input Parameters**   **handle**     Handle to identify the session.

**Output Parameters**   **value**     Value read from the static I/O port.

**See also**     “*BestStaticWrite*” on page 180

## BestStaticWrite

**Call**     `b_errtype BestStaticWrite(  
                  b_handletype    handle,  
                  b_int32        value );`

**Description**   Writes a byte to the static I/O port.

**CLI Equivalent**   `BestStaticWrite value=<value>`

**CLI Abbreviation**   `swrite val=<value>`

**Return Value**   Error code; see “*b\_errtype*” on page 250.

**Input Parameters**   **handle**     Handle to identify the session.

**value**     Value to be written (0 ... 255, no default).

**See also**     “*BestStaticRead*” on page 180

# Trigger I/O Sequencer Programming Functions

The following functions are used to program the external trigger I/O lines:

| Function   | Result   |
|--|--|
| <i>"BestTrigIOGenPropDefaultSet" on page 182</i>     | Sets all generic trigger I/O sequencer properties to default values.                                 |
| <i>"BestTrigIOGenPropGet" on page 182</i>            | Reads the value of a generic trigger I/O sequencer property.   |
| <i>"BestTrigIOGenPropSet" on page 183</i>            | Sets the value of a generic trigger I/O sequencer property.  |
| <i>"BestTrigIOSeqPropDefaultSet" on page 184</i>     | Sets all properties in the trigger I/O sequencer description table to default values.                |
| <i>"BestTrigIOSeqTranPropDefaultSet" on page 186</i> | Sets all properties of a transient in the trigger I/O sequencer description table to default values. |
| <i>"BestTrigIOSeqTranPropSet" on page 187</i>        | Sets numeric transition properties ("state" or "next state").  |
| <i>"BestTrigIOSeqTranCondPropSet" on page 185</i>    | Sets conditions in the trigger I/O sequencer description table.                                      |
| <i>"BestTrigIOSeqProg" on page 183</i>               | Writes the sequencer description table to sequencer memory.  |
| <i>"BestTrigIORun" on page 184</i>                   | Starts the trigger I/O sequencer.  |
| <i>"BestTrigIOStop" on page 187</i>                  | Stops the trigger I/O sequencer.   |

How to use the functions is described in *"Trigger I/O Sequencer Programming"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestTrigIOGenPropDefaultSet

**Call** `b_errtype BestTrigIOGenPropDefaultSet( b_handletype handle );`

**Description** Sets all generic properties of the trigger I/O sequencer to default values.

**CLI Equivalent** `BestTrigIOGenPropDefaultSet`

**CLI Abbreviation** `tiogenprpdefset`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*b\_trigioseqgenproptype*” on page 297  
 “*BestTrigIOGenPropGet*” on page 182  
 “*BestTrigIOGenPropSet*” on page 183

## BestTrigIOGenPropGet

**Call** `b_errtype BestTrigIOGenPropGet (
 b_handletype handle,
 b_trigioengenproptype trigenprop,
 b_int32 *value );`

**Description** Queries generic properties of the trigger I/O sequencer. Generic properties determine the preload value of the feedback counter and the trigger I/O output settings.

**CLI Equivalent** `BestTrigIOGenPropGet trigenprop=<trigenprop>`

**CLI Abbreviation** `tiogenprpget prop=<trigenprop>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**trigenprop** Property to be queried; see “*b\_trigioseqgenproptype*” on page 297.

**Output Parameters** **value** Value of the queried property; see “*b\_trigioseqgenproptype*” on page 297.

**See also** “*BestTrigIOGenPropDefaultSet*” on page 182  
 “*BestTrigIOGenPropSet*” on page 183

## BestTrigIOGenPropSet

**Call** `b_errtype BestTrigIOGenPropSet (`  
     `b_handletype           handle,`  
     `b_trigiogenproptype   trigiogenprop,`  
     `b_int32                value );`

**Description** Sets generic properties of the trigger I/O sequencer. Generic properties determine the preload value of the feedback counter and the trigger I/O output settings.

**CLI Equivalent** `BestTrigIOGenPropSet trigiogenprop=<trigiogenprop> value=<value>`

**CLI Abbreviation** `tioenprpset prop=<trigiogenprop> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**trigiogenprop** Property to be set; see “*b\_trigioeqgenproptype*” on page 297.

**value** Value to which the property is set; see “*b\_trigioeqgenproptype*” on page 297.

**See also** “*BestTrigIOGenPropDefaultSet*” on page 182  
 “*BestTrigIOGenPropGet*” on page 182

## BestTrigIOSeqProg

**Call** `b_errtype BestTrigIOSeqProg( b_handletype handle );`

**Description** Writes the information stored in the trigger I/O sequencer description table to the sequencer memory.

This function also checks whether transition conditions of one state are consistent. If they are not, the function returns an error.

To complete sequencer programming, the pattern terms must be programmed with “*BestPattSet*” on page 62.

**CLI Equivalent** `BestTrigIOSeqProg`

**CLI Abbreviation** `tiosprog`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** –

## BestTrigIORun

**Call** `b_errtype BestTrigIORun( b_handletype handle );`

**Description** Starts the trigger I/O sequencer (and the performance counters).

**CLI Equivalent** `BestTrigIORun`

**CLI Abbreviation** `tiorun`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestPerfRun*” on page 89  
“*BestTrigIOStop*” on page 187

## BestTrigIOSeqPropDefaultSet

**Call** `b_errtype BestTrigIOSeqPropDefaultSet( b_handletype handle );`

**Description** Initializes the trigger I/O sequencer description table and sets all properties to default values.

For a description of properties and default values, refer to  
“*b\_trigioseqtranproptype*” on page 299 and  
“*b\_trigseqtrancondproptype*” on page 300.

**CLI Equivalent** `BestTrigIOSeqPropDefaultSet`

**CLI Abbreviation** `tiosprpdefset`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** –



## BestTrigIOSeqTranCondPropSet

**Call** `b_errtype BestTrigIOSeqTranCondPropSet (`  
     `b_handletype                           handle,`  
     `b_int32                                 transient,`  
     `b_trigioseqtrancondproptype   trigioseqtrancondprop,`  
     `b_charptrtype                    condition );`

**Description** Sets condition properties in the trigger I/O sequencer description table.

Condition properties can be the transition condition, trigger condition, storage qualifier condition, or conditions to decrement or preload the feedback counter.

**CLI Equivalent** `BestTrigIOSeqTranCondPropSet transient=<transient>`  
`trigioseqtrancondprop=<trigioseqtrancondprop> condition=<condition>`

**CLI Abbreviation** `tiostrancprpset tran=<transient> prop=<trigioseqtrancondprop>`  
`con=<condition>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**transient** Number of the transient (0 ... 251).

**trigioseqtrancondprop** Property to be set; see “*b\_trigioseqtrancondproptype*” on page 298.

**condition** Condition string to which the property is set. The string must be written in quotation marks. See “*Conditions Reference*” on page 72.

**See also** “*BestTrigIOSeqTranPropDefaultSet*” on page 186  
 “*BestTrigIOSeqTranPropSet*” on page 187

## BestTrigIOSeqTranPropDefaultSet

**Call** `b_errtype BestTrigIOSeqTranPropDefaultSet (`  
           `b_handletype  handle,`  
           `b_int32       transient );`

**Description** Sets all properties of a transient in the trigger I/O sequencer description table to default values.

For a description of properties and default values, refer to  
*“b\_trigioseqtrancondproptype” on page 298* and  
*“b\_trigioseqtranproptype” on page 299.*

**CLI Equivalent** `BestTrigIOSeqTranPropDefaultSet transient=<transient>`

**CLI Abbreviation** `tiostranprpdefset tran=<transient>`

**Return Value** Error code; see *“b\_errtype” on page 250.*

**Input Parameters** **handle** Handle to identify the session.

**transient** Transient number (0 ... 251).

**See also** *“BestTrigIOSeqTranCondPropSet” on page 185*  
*“BestTrigIOSeqTranPropSet” on page 187*

## BestTrigIOSeqTranPropSet

**Call** `b_errtype BestTrigIOSeqTranPropSet (`  
     `b_handletype                   handle,`  
     `b_int32                         transient,`  
     `b_trigioseqtranproptype    trigioseqtranprop,`  
     `b_int32                         value );`

**Description** Sets a numeric transition property (“state” or “next state”) in the trigger I/O sequencer description table.

**CLI Equivalent** `BestTrigIOSeqTranPropSet transient=<transient>`  
`trigioseqtranprop=<trigioseqtranprop> value=<value>`

**CLI Abbreviation** `tiostranprpset tran=<transient> prop=<trigioseqtranprop>`  
`val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**transient** Number of the transient (0 ... 251).

**trigioseqtranprop** Property to be set; see “*b\_trigioseqtranproptype*” on page 299.

**value** Value to which the property is set; see “*b\_trigioseqtranproptype*” on page 299.

**See also** “*BestTrigIOSeqTranPropDefaultSet*” on page 186  
“*BestTrigIOSeqTranCondPropSet*” on page 185

## BestTrigIOStop

**Call** `b_errtype BestTrigIOStop( b_handletype handle );`

**Description** Stops the trigger I/O sequencer (and the performance counters).

**CLI Equivalent** `BestTrigIOStop`

**CLI Abbreviation** `tiostop`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestTrigIORun*” on page 184  
“*BestPerfStop*” on page 95

# Display Functions

The following functions are used to control the LED display:

| Function                                | Result                         |
|---|--------------------------------|
| <i>"BestDisplayPropSet" on page 188</i> | Sets the display mode.         |
| <i>"BestDisplayWrite" on page 189</i>   | Writes a value to the display. |

How to use the functions is described in *"LED Controlling and Display Functions Overview"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestDisplayPropSet

**Call** `b_errtype BestDisplayPropSet (`  
           **b\_hndletype**    **handle**,  
           **b\_int32**       **value** );

**Description** Sets the mode of the LED display. Use this function to select "user mode" before writing values to the display.

**CLI Equivalent** `BestDisplayPropSet value=<value>`

**CLI Abbreviation** `dprpset val=<value>`

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**value** The display provides the following modes:

| Display Mode (CLI abbreviation)       | Description   |
|---------------------------------------|---|
| B_DISP_USER (user)                    | User mode.<br>Select this mode to write to the display. |
| <b>default:</b><br>B_DISP_CARD (card) | The display shows internal system states.               |

**See also** *"BestDisplayWrite" on page 189*

## BestDisplayWrite

**Call** `b_errtype BestDisplayWrite(  
          b_handletype handle,  
          b_int32 value );`

**Description** Writes a value to the LED display.

Before using this function, ensure that “user mode” is selected for the display. You can query the selected mode with “*BestDisplayPropSet*” on page 188.

**CLI Equivalent** `BestDisplayWrite value=<value>`

**CLI Abbreviation** `dwrite val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**value** 5-bit value to be displayed.

**See also** –

## Mailbox Functions

The following functions are used for communication between testcard and test environment via the PCI interface:

| Function   | Result                                 |
|--|--|
| <i>“BestMailboxReceiveRegRead” on page 190</i>       | Reads data from mailbox to control PC. |
| <i>“BestMailboxSendRegWrite” on page 191</i>         | Sends data from control PC to mailbox. |
| <i>“BestPCICfgMailboxReceiveRegRead” on page 192</i> | Reads data from mailbox via PCI.       |
| <i>“BestPCICfgMailboxSendRegWrite” on page 193</i>   | Sends data to mailbox via PCI.         |

How to use the functions is described in “*Mailbox Programming*” in the *Agilent E2927A Opt. 320 C-API/PPR Programmer’s Guide*.

## BestMailboxReceiveRegRead

**Call** `b_errtype BestMailboxReceiveRegRead(  
           b_handletype  handle,  
           b_int32      *value,  
           b_int32      *status );`

**Description** This function is used on the control PC to receive a value from the mailbox via the control interface. The function returns the mailbox value and a status flag.

**CLI Equivalent** BestMailboxReceiveRegRead

**CLI Abbreviation** mrregread

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**Output Parameters** **value** Value read from the mailbox.

**status** Status flag:

- 0 – The mailbox was empty. The transfer failed.
- 1 – The transfer was successful.

**See also** “*BestMailboxSendRegWrite*” on page 191  
 “*BestPCICfgMailboxReceiveRegRead*” on page 192

## BestMailboxSendRegWrite

**Call** `b_errtype BestMailboxSendRegWrite(  
           b_handletype  handle,  
           b_int32      value,  
           b_int32      *status );`

**Description** This function is used on the control PC to send a value to the mailbox via the control interface.

If the mailbox is occupied (by unread data, see status flag below), the function will not write the value to the mailbox.

**CLI Equivalent** `BestMailboxSendRegWrite value=<value>`

**CLI Abbreviation** `msregwrite val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**value** Value to send to the mailbox.

**Output Parameters** **status** Status flag:

- 0 – Unread data was found in the mailbox. The transfer failed.
- 1 – The transfer was successful.

**See also** “*BestMailboxReceiveRegRead*” on page 190  
 “*BestPCICfgMailboxSendRegWrite*” on page 193

## BestPCICfgMailboxReceiveRegRead

**Call** `b_errtype BestPCICfgMailboxReceiveRegRead (`  
     `b_int32 devid,`  
     `b_int32 *value,`  
     `b_int32 *status );`

**Description** This function is used on the system under test to read a value from the mailbox via the PCI bus. The function returns the mailbox value and the status flag.

To identify the testcard within the PCI system, first call “*BestDevIdentifierGet*” on page 19. The device identifier returned by this function can be used directly.

**NOTE** When checking for errors, use *non-handle-based error checking*. See “*Error Handling*” on page 233.

**CLI Equivalent** No CLI equivalent.

**CLI Abbreviation** No CLI abbreviation.

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **devid** Device identifier of the testcard as returned by “*BestDevIdentifierGet*” on page 19.

**Output Parameters** **value** Value read from the mailbox.

**status** Status flag:

- 0 – The mailbox was empty. The transfer failed.
- 1 – The transfer was successful.

**See also** “*BestMailboxReceiveRegRead*” on page 190  
 “*BestPCICfgMailboxSendRegWrite*” on page 193



## BestPCICfgMailboxSendRegWrite

**Call** `b_errtype BestPCICfgMailboxSendRegWrite(  
     b_int32 devid,  
     b_int32 value,  
     b_int32 *status );`

**Description** This function is used on the system under test to write a value to the mailbox via the PCI bus.

To identify the testcard within the PCI system, first call “*BestDevIdentifierGet*” on page 19. The device identifier returned by this function can be used directly.

If the mailbox is occupied (by unread data), the function will not write the value to the mailbox.

**NOTE** When checking for errors, use *non-handle-based error checking*. See “*Error Handling*” on page 233.

**CLI Equivalent** No CLI equivalent.

**CLI Abbreviation** No CLI abbreviation.

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **devid** Device identifier of the testcard as returned by “*BestDevIdentifierGet*” on page 19.

**value** Value to send to the mailbox.

**Output Parameters** **status** Status flag:

- 0 – Unread data was found in the mailbox. The transfer failed.
- 1 – The transfer was successful.

**See also** “*BestPCICfgMailboxReceiveRegRead*” on page 192  
 “*BestMailboxSendRegWrite*” on page 191

# Power Management Event Functions

The following functions are used to control power management events:

| Function                          | Result   |
|-----------------------------------|--|
| <i>"BestPMEWrite" on page 195</i> | Issues a power management event.                       |
| <i>"BestPMERead" on page 194</i>  | Determines the occurrence of a power management event. |

How to use the functions is described in *"Power Management Event Programming"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestPMERead

**Call** `b_errtype BestPMERead(  
          b_handletype  handle,  
          b_int32      *value );`

**Description** Reads the status of the power management event lines.

**CLI Equivalent** BestPMERead

**CLI Abbreviation** pmeread

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**Output Parameters** **value** Status of the power management lines:

- 0 – no event
- 1 – event

**See also** *"BestPMEWrite" on page 195*

## BestPMEWrite

**Call** `b_errtype BestPMEWrite(  
          b_handletype handle,  
          b_int32 value);`

**Description** Issues a power management event.

**CLI Equivalent** `BestPMEWrite value=<value>`

**CLI Abbreviation** `pmewrite val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**value** Value to be written:

- 0 – resets the event
- 1 – sets the event

**See also** “*BestPMERead*” on page 194



# Protocol Permutator and Randomizer Functions

The PCI Protocol Permutator and Randomizer (PPR) functions can be divided into several categories:

- “*PPR Administrating Functions*” on page 197
- “*Master Block Permutation Functions*” on page 202
- “*Master Attribute Permutation Functions*” on page 210
- “*PPR Report Functions*” on page 218
- “*Target Attribute Permutation Functions*” on page 224

## PPR Administrating Functions

The following functions are used to initialize and deinitialize the PCI Protocol Permutator and Randomizer software:

| Function  | Result   |
|---|--|
| “ <i>BestPprInit</i> ” on page 201              | Initializes the PPR software.                      |
| “ <i>BestPprDelete</i> ” on page 198            | Frees all memory allocated by the software.        |
| “ <i>BestPprGenPropDefaultSet</i> ” on page 198 | Sets all PPR generic properties to default values. |
| “ <i>BestPprGenPropSet</i> ” on page 200        | Sets values of PPR generic properties.             |
| “ <i>BestPprGenPropGet</i> ” on page 199        | Reads values of PPR generic properties.            |

How to use the functions is described in “*PPR Administration*” in the *Agilent E2927A Opt. 320 C-API/PPR Programmer’s Guide*.

## BestPprDelete

**Call** `b_errtype BestPprDelete( b_handletype handle );`

**Description** Frees all the memory space allocated by the PCI PPR software. You must use this function when you finish working with the software.

**CLI Equivalent** `BestPprDelete`

**CLI Abbreviation** `pprd`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestPprInit*” on page 201

## BestPprGenPropDefaultSet

**Call** `b_errtype BestPprGenPropDefaultSet( b_handletype handle );`

**Description** Sets all generic properties of the PCI PPR software to default values. For a description of properties and default values, see “*bppr\_genproptype*” on page 310.

**CLI Equivalent** `BestPprGenPropDefaultSet`

**CLI Abbreviation** `pprgpds`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestPprGenPropSet*” on page 200  
“*BestPprGenPropGet*” on page 199

## BestPprGenPropGet

**Call** `b_errtype BestPprGenPropGet (`  
           `b_handletype      handle,`  
           `bppr_genproptype  prop,`  
           `b_int32              *value );`

**Description** Reads generic property values of the PCI PPR software.

The generic properties are PCI bus speed and width, expected number of clocks per data transfer and random seed.

**CLI Equivalent** `BestPprGenPropGet prop=<prop>`

**CLI Abbreviation** `pprgpg p=<prop>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**prop** Property to be read; see “*bppr\_genproptype*” on page 310.

**Output Parameters** **value** Value of the read property; see “*bppr\_genproptype*” on page 310.

**See also** “*BestPprGenPropSet*” on page 200  
 “*BestPprGenPropDefaultSet*” on page 198

## BestPprGenPropSet

**Call** `b_errtype BestPprGenPropSet (`  
     `b_handletype        handle,`  
     `bppr_genproptype   prop,`  
     `b_int32             value );`

**Description** Sets generic property values of the PCI PPR software.

The generic properties are PCI bus speed and width, expected number of clocks per data transfer and random seed.

**CLI Equivalent** `BestPprGenPropSet prop=<prop> value=<value>`

**CLI Abbreviation** `pprgps p=<prop> val=<value>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**prop** Property to be set; see “*bppr\_genproptype*” on page 310.

**value** Value to which the property is set; see “*bppr\_genproptype*” on page 310.

**See also** “*BestPprGenPropGet*” on page 199  
 “*BestPprGenPropDefaultSet*” on page 198



## BestPprInit

**Call** `b_errtype BestPprInit( b_handletype handle );`

**Description** Initializes the PCI PPR software and sets all properties of this software to default values. This function must be called before any other PCI PPR function.

At the end of the test you must call “*BestPprDelete*” on page 198 to free the memory.

If you just want to set the generic properties to default values, see “*BestPprGenPropDefaultSet*” on page 198.

If you just want to set specific parts of the program to default values, see “*BestPprBlockInit*” on page 204, “*BestPprMAttrInit*” on page 212, “*BestPprTAttrInit*” on page 226, or “*BestPprReportPropDefaultSet*” on page 219.

**CLI Equivalent** `BestPprInit`

**CLI Abbreviation** `ppri`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**NOTE** If you call `BestPprInit` twice with the same session handle, the memory allocated by the first call is freed automatically.

**See also** “*bppr\_genproptype*” on page 310

# Master Block Permutation Functions

The following functions are used to prepare and to perform a master block permutation:

| Function   | Result   |
|--|--|
| <i>"BestPprBlockInit" on page 204</i>                | Initializes the block permutator.  |
| <i>"BestPprBlockPermPropDefaultSet" on page 205</i>  | Sets all block permutation properties to default values.                             |
| <i>"BestPprBlockPermPropSet" on page 206</i>         | Sets values of block permutation properties.   |
| <i>"BestPprBlockPermPropGet" on page 205</i>         | Reads values of block permutation properties.  |
| <i>"BestPprBlockVariationDefaultSet" on page 207</i> | Sets all block variation parameters to default values.                               |
| <i>"BestPprBlockVariationSet" on page 209</i>        | Sets the value list and the corresponding algorithm of a block variation parameter.  |
| <i>"BestPprBlockVariationGet" on page 208</i>        | Reads the value list and the corresponding algorithm of a block variation parameter. |
| <i>"BestPprBlockGenerate" on page 204</i>            | Generates a compound block.  |
| <i>"BestPprBlockResultGet" on page 207</i>           | Reads a result parameter.  |
| <i>"BestPprBlockCoverageGet" on page 203</i>         | Returns coverage and repetition length.  |

How to use the functions is described in *"Programming Master Block Permutations"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestPprBlockCoverageGet

**Call** `b_errtype BestPprBlockCoverageGet (`  
     `b_handletype handle,`  
     `b_int32 nmb_param,`  
     `b_int32 *param_list,`  
     `b_int32 *rep_len,`  
     `b_int32 *covered );`

**Description** Internally performs the permutation algorithm with the parameters currently set. “Internally” means nothing is downloaded to the testcard. This function determines whether all permutations of a given tuple of block variation parameters are covered, and returns the repetition length of the tuple.

**CLI Equivalent** No CLI equivalent. To get coverage information with the CLI, use “*BestPprReportWrite*” on page 222.

**CLI Abbreviation** No CLI abbreviation.

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**nmb\_param** Number of block variation parameters in the tuple.

**param\_list** Array that defines the tuple of which the coverage is to be tested. The tuple holds nmb\_param entries. The entries must be valid values of “*bppr\_blkvarparamtype*” on page 307.

**Output Parameters** **rep\_len** Returns the repetition length of the tuple (that is the number of permutations of the tuple).

**covered** Returned values:

- 1 – if the tuple is covered after complete execution of the master block page created by the permutation algorithm.
- 0 – if this is not the case.

**See also** –

## BestPprBlockGenerate

**Call** `b_errtype BestPprBlockGenerate( b_hndletype handle );`

**Description** Generates a compound block using the properties shown in the table below and downloads this compound block to the testcard.

A compound block consists of blocks with differing block variation parameters. The number of these blocks depends on the master block page size (MBPS).

| Properties of Master Block Page Generation | Adjusted by Function                          |
|--|---|
| Generic setup properties                   | <i>"BestPprGenPropSet" on page 200</i>        |
| Master block properties                    | <i>"BestMasterBlockPropSet" on page 123</i>   |
| Block permutation properties               | <i>"BestPprBlockPermPropSet" on page 206</i>  |
| Block variation parameters                 | <i>"BestPprBlockVariationSet" on page 209</i> |

**CLI Equivalent** `BestPprBlockGenerate`

**CLI Abbreviation** `pprbg`

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**See also** *"bppr\_blkvarparamtype" on page 307*

## BestPprBlockInit

**Call** `b_errtype BestPprBlockInit( b_hndletype handle );`

**Description** Initializes the block permutator and sets all block permutation properties to default values.

To set individual properties to default values, use either the function *"BestPprBlockPermPropDefaultSet" on page 205* or *"BestPprBlockVariationDefaultSet" on page 207*.

**CLI Equivalent** `BestPprBlockInit`

**CLI Abbreviation** `pprbi`

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**See also** –

## BestPprBlockPermPropDefaultSet

**Call** `b_errtype BestPprBlockPermPropDefaultSet( b_handletype handle );`

**Description** Sets the block permutation properties to default values. For a description of properties and default values, see *“bppr\_blkpermproptype” on page 304.*

**CLI Equivalent** `BestPprBlockPermPropDefaultSet`

**CLI Abbreviation** `pprbppds`

**Return Value** Error code; see *“b\_errtype” on page 250.*

**Input Parameters** **handle** Handle to identify the session.

**See also** *“BestPprBlockPermPropSet” on page 206*  
*“BestPprBlockPermPropGet” on page 205*

## BestPprBlockPermPropGet

**Call** `b_errtype BestPprBlockPermPropGet (
 b_handletype handle,
 bppr_blkpermproptype prop,
 b_int32 *value );`

**Description** Reads the value of a block permutation property.

**CLI Equivalent** `BestPprBlockPermPropGet prop=<prop>`

**CLI Abbreviation** `pprbppg p=<prop>`

**Return Value** Error code; see *“b\_errtype” on page 250.*

**Input Parameters** **handle** Handle to identify the session.

**prop** Property to be read; see *“bppr\_blkpermproptype” on page 304.*

**Output Parameters** **value** Value of the read property; see *“bppr\_blkpermproptype” on page 304.*

**See also** *“BestPprBlockPermPropDefaultSet” on page 205*  
*“BestPprBlockPermPropSet” on page 206*

## BestPprBlockPermPropSet

**Call**    `b_errtype BestPprBlockPermPropSet (`  
                  `b_handletype          handle,`  
                  `bppr_blkpermproptype  prop,`  
                  `b_int32                  value );`

**Description**    Sets a block permutation property.

**CLI Equivalent**    `BestPprBlockPermPropSet prop=<prop> value=<value>`

**CLI Abbreviation**    `pprbpps p=<prop> val=<value>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**prop**    Property to be set; see “*bppr\_blkpermproptype*” on page 304.

**value**    Value to which the property is set; see “*bppr\_blkpermproptype*” on page 304.

**See also**    “*BestPprBlockPermPropDefaultSet*” on page 205  
                  “*BestPprBlockPermPropGet*” on page 205

## BestPprBlockResultGet

**Call** `b_errtype BestPprBlockResultGet (`  
     `b_handletype                    handle,`  
     `bppr_blkresultparamtype    param,`  
     `b_int32                        *value );`

**Description** Internally performs the permutation algorithm with the parameters currently set. “Internally” means nothing is downloaded to the testcard. This function reads the value of a selected master block result parameter.

**CLI Equivalent** `BestPprBlockResultGet param=<param>`

**CLI Abbreviation** `pprbrg p=<param>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**param** Result parameter to be read; see “*bppr\_blkresultparamtype*” on page 306.

**Output Parameters** **value** Value of the read result parameter.

**See also** –

## BestPprBlockVariationDefaultSet

**Call** `b_errtype BestPprBlockVariationDefaultSet ( b_handletype handle );`

**Description** Sets the block variation parameters to default values. For a description of properties and default values, see “*bppr\_blkvarparamtype*” on page 307.

**CLI Equivalent** `BestPprBlockVariationDefaultSet`

**CLI Abbreviation** `pprbvds`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestPprBlockVariationGet*” on page 208  
 “*BestPprBlockVariationSet*” on page 209

## BestPprBlockVariationGet

**Call** `b_errtype BestPprBlockVariationGet (`  
     `b_handletype                   handle,`  
     `bppr_blkvarparamtype       param,`  
     `b_charptrtype                *value_list,`  
     `bppr_algorithmtype         *algorithm );`

**Description** Reads the value list and the corresponding algorithm of a block variation parameter.

The block variation parameters are alignments, block sizes, byte enables or PCI bus commands.

**CLI Equivalent** `BestPprBlockVariationGet param=<param>`

**CLI Abbreviation** `pprbvg p=<param>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**param** Parameter to be read; see “*bppr\_blkvarparamtype*” on page 307.

**Output Parameters** **value\_list** List of permuted values. For the syntax of the list, see “*Value Lists*” on page 308.

If the PCI PPR software has not yet been initialized with “*BestPprInit*” on page 201, the list will be empty and an error will be returned.

**algorithm** Returns the algorithm used to pick values from the value list. See “*bppr\_algorithmtype*” on page 302.

**See also** “*BestPprBlockVariationDefaultSet*” on page 207  
 “*BestPprBlockVariationSet*” on page 209



## BestPprBlockVariationSet

**Call** `b_errtype BestPprBlockVariationSet (`  
     `b_handletype                    handle,`  
     `bppr_blkvarparamtype        param,`  
     `b_charptrtype                value_list,`  
     `bppr_algorithmtype        algorithm );`

**Description** Defines the value list for a block variation parameter and selects the algorithm used to pick values from this list.

Block variation parameters are alignments, block sizes, byte enables or PCI bus commands. To select PCI bus commands, specific algorithms are available. See “*bppr\_algorithmtype: B\_BLK\_CMDs Details*” on page 302.

**CLI Equivalent** `BestPprBlockVariationSet param=<param> value_list=<value_list>`  
`algorithm=<algorithm>`

**CLI Abbreviation** `pprbvs p=<param> list=<value_list> alg=<algorithm>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**param** Block variation parameter; see “*bppr\_blkvarparamtype*” on page 307.

**value\_list** List of values to be permuted according to the selected algorithm. For the syntax of the list, see “*Value Lists*” on page 308.

**algorithm** Algorithm used to pick values from the value list; see “*bppr\_algorithmtype*” on page 302.

**See also** “*BestPprBlockVariationDefaultSet*” on page 207  
 “*BestPprBlockVariationGet*” on page 208

# Master Attribute Permutation Functions

The following functions are used to prepare and to perform the permutation of the master attributes:

| Function   | Result  |
|--|---|
| <i>"BestPprMAttrInit" on page 212</i>                | Initializes the master attribute permutator.                                |
| <i>"BestPprMAttrPermPropDefaultSet" on page 213</i>  | Sets all master attribute properties to default values.                     |
| <i>"BestPprMAttrPermPropSet" on page 214</i>         | Sets values of master attribute permutation properties.                     |
| <i>"BestPprMAttrPermPropGet" on page 213</i>         | Reads values of master attribute permutation properties.                    |
| <i>"BestPprMAttrVariationDefaultSet" on page 215</i> | Sets all master attribute variation parameters to default values.           |
| <i>"BestPprMAttrVariationSet" on page 217</i>        | Sets the value list and the corresponding algorithm of a master attribute.  |
| <i>"BestPprMAttrVariationGet" on page 216</i>        | Reads the value list and the corresponding algorithm of a master attribute. |
| <i>"BestPprMAttrGenerate" on page 212</i>            | Generates a master attribute page.  |
| <i>"BestPprMAttrResultGet" on page 215</i>           | Reads a result parameter.   |
| <i>"BestPprMAttrCoverageGet" on page 211</i>         | Returns coverage and repetition length.                                     |

How to use the functions is described in *"Programming Master Attribute Permutations"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestPprMAttrCoverageGet

**Call** `b_errtype BestPprMAttrCoverageGet (`  
     `b_handletype        handle,`  
     `b_int32            nmb_attr,`  
     `b_mattrproptype   *attr_list,`  
     `b_int32            *rep_len,`  
     `b_int32            *covered );`

**Description** Internally performs the permutation algorithm with the parameters currently set. “Internally” means that nothing is downloaded to the testcard. This function determines whether all permutations of a given tuple of master attributes are covered, and returns the repetition length of the tuple.

**CLI Equivalent** No CLI equivalent. To get coverage information with the CLI, use “*BestPprReportWrite*” on page 222.

**CLI Abbreviation** No CLI abbreviation.

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**nmb\_attr** Number of master attributes in the tuple.

**attr\_list** This array defines the tuple with the coverage to be tested. The tuple holds nmb\_attr entries. The entries must be valid values of “*b\_mattrproptype*” on page 260.

**Output Parameters** **rep\_len** Returns the repetition length of the tuple (that is the number of permutations of the tuple).

**covered** Returned values:

- 1 – if the tuple is covered after complete execution of the master attribute page created by the permutation algorithm.
- 0 – if this is not the case.

**See also** –

## BestPprMAttrGenerate

**Call** `b_errtype BestPprMAttrGenerate( b_handletype handle );`

**Description** Generates a master attribute page using the properties shown in the table below and downloads this master attribute page to the testcard.

| Properties of Master Attribute Page Generation | Adjusted by Function                          |
|--|---|
| Generic setup properties                       | <i>"BestPprGenPropSet" on page 200</i>        |
| Master attribute permutation properties        | <i>"BestPprMAttrPermPropSet" on page 214</i>  |
| Master attribute variation parameters          | <i>"BestPprMAttrVariationSet" on page 217</i> |

**CLI Equivalent** `BestPprMAttrGenerate`

**CLI Abbreviation** `pprmag`

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**See also** *"b\_mattrproptype" on page 260*

## BestPprMAttrInit

**Call** `b_errtype BestPprMAttrInit( b_handletype handle );`

**Description** Initializes the master attribute permutator and sets all of its properties to default values.

To set individual properties to default values, use either the function *"BestPprMAttrPermPropDefaultSet" on page 213* or *"BestPprMAttrVariationDefaultSet" on page 215*.

**CLI Equivalent** `BestPprMAttrInit`

**CLI Abbreviation** `pprmai`

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**See also** —

## BestPprMAttrPermPropDefaultSet

**Call** `b_errtype BestPprMAttrPermPropDefaultSet( b_handletype handle );`

**Description** Sets the master attribute permutation properties to default values. For a description of properties and default values, see *“bppr\_mattrpermproptype” on page 311*.

**CLI Equivalent** `BestPprMAttrPermPropDefaultSet`

**CLI Abbreviation** `pprmappds`

**Return Value** Error code; see *“b\_errtype” on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**See also** *“BestPprMAttrPermPropGet” on page 213*  
*“BestPprMAttrPermPropSet” on page 214*

## BestPprMAttrPermPropGet

**Call** `b_errtype BestPprMAttrPermPropGet (
 b_handletype handle,
 bppr_mattrpermproptype prop,
 b_int32 *value );`

**Description** Reads a master attribute permutation property.

**CLI Equivalent** `BestPprMAttrPermPropGet prop=<prop>`

**CLI Abbreviation** `pprmappg p=<prop>`

**Return Value** Error code; see *“b\_errtype” on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**prop** Property to be read; see *“bppr\_mattrpermproptype” on page 311*.

**Output Parameters** **value** Value of the read property; see *“bppr\_mattrpermproptype” on page 311*.

**See also** *“BestPprMAttrPermPropDefaultSet” on page 213*  
*“BestPprMAttrPermPropSet” on page 214*

## BestPprMAttrPermPropSet

**Call**    `b_errtype BestPprMAttrPermPropSet (`  
          `b_handletype            handle,`  
          `bppr_mattrpermproptype prop,`  
          `b_int32                 value );`

**Description**    Sets a master attribute permutation property.

**CLI Equivalent**    `BestPprMAttrPermPropSet prop=<prop> value=<value>`

**CLI Abbreviation**    `pprmapps p=<prop> val=<value>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**prop**    Property to be set; see “*bppr\_mattrpermproptype*” on page 311.

**value**    Value to which the property is set; see  
                          “*bppr\_mattrpermproptype*” on page 311.

**See also**    “*BestPprMAttrPermPropDefaultSet*” on page 213  
              “*BestPprMAttrPermPropGet*” on page 213

## BestPprMAttrResultGet

**Call** `b_errtype BestPprMAttrResultGet (`  
           `b_handletype                    handle,`  
           `bppr_mattrresultparamtype param,`  
           `b_int32                          *value );`

**Description** Internally performs the permutation algorithm with the parameters currently set. “Internally” means nothing is downloaded to the testcard. This function reads the value of a selected master attribute result parameter.

**CLI Equivalent** `BestPprMAttrResultGet param=<param>`

**CLI Abbreviation** `pprmarg p=<param>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**param** Result parameter to be read; see “*bppr\_mattrresultparamtype*” on page 312.

**Output Parameters** **value** Value of the read result parameter; see “*bppr\_mattrresultparamtype*” on page 312.

**See also** –

## BestPprMAttrVariationDefaultSet

**Call** `b_errtype BestPprMAttrVariationDefaultSet ( b_handletype handle );`

**Description** Sets the master attribute variation parameters to default values. For a description of parameters and default values, see “*b\_mattrproptype*” on page 260.

**CLI Equivalent** `BestPprMAttrVariationDefaultSet`

**CLI Abbreviation** `pprmavds`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestPprMAttrVariationGet*” on page 216  
 “*BestPprMAttrVariationSet*” on page 217

## BestPprMAttrVariationGet

**Call**    `b_errtype BestPprMAttrVariationGet (`  
                  `b_handletype            handle,`  
                  `bppr_mattrproptype    attribute,`  
                  `b_charptrtype            *value_list,`  
                  `bppr_algorithmtype    *algorithm );`

**Description**    Reads the variation properties of a master attribute.

**CLI Equivalent**    `BestPprMAttrVariationGet attribute=<attribute>`

**CLI Abbreviation**    `pprmvg attr=<attribute>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**attribute**    Master attribute to be read; see “*b\_mattrproptype*” on page 260.

For more information on master attributes, see *Agilent E2927A Exerciser User's Guide*.

**Output Parameters**    **value\_list**    Returns a pointer to a string containing a comma-separated list of values. The values in this list are permuted according to the selected algorithm. For a description of the string's contents, refer to “*Numeric values*” on page 309.

If the PCI PPR software has not yet been initialized with “*BestPprBlockInit*” on page 204 or “*BestPprInit*” on page 201, the list will be empty and an error will be returned.

**algorithm**    Returns the algorithm used to pick values from the value list; see “*bppr\_algorithmtype*” on page 302.

**See also**    “*BestPprMAttrVariationDefaultSet*” on page 215  
               “*BestPprMAttrVariationSet*” on page 217



## BestPprMAttrVariationSet

**Call** `b_errtype BestPprMAttrVariationSet (`  
     `b_handletype           handle,`  
     `b_mattrproptype       attribute,`  
     `b_charptrtype        value_list,`  
     `bppr_algorithmtype   algorithm );`

**Description** Defines the value list for a master attribute and selects the algorithm used to pick values from this list.

**CLI Equivalent** `BestPprMAttrVariationSet attribute=<attribute>`  
     `value_list=<value_list> algorithm=<algorithm>`

**CLI Abbreviation** `pprmavs attr=<attribute> list=<value_list> alg=<algorithm>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**attribute** Master attribute; see “*b\_mattrproptype*” on page 260.

**value\_list** The values in this list are permuted according to the selected algorithm. For the syntax of the list, see “*Numeric values*” on page 309.

**algorithm** Determines the algorithm used to pick values from the value list. See “*bppr\_algorithmtype*” on page 302.

**See also** “*BestPprMAttrVariationDefaultSet*” on page 215  
     “*BestPprMAttrVariationGet*” on page 216

# PPR Report Functions

The following functions are used to set the report properties and to generate reports:

| Function   | Result  |
|--|---|
| <i>"BestPprReportPropDefaultSet" on page 219</i> | Sets all report properties to default values. |
| <i>"BestPprReportPropSet" on page 221</i>        | Sets a report property.                       |
| <i>"BestPprReportPropGet" on page 220</i>        | Reads the value of a report property.         |
| <i>"BestPprReportWrite" on page 222</i>          | Generates a report.                           |
| <i>"BestPprReportFile" on page 219</i>           | Writes a report to a file.                    |
| <i>"BestPprReportDelete" on page 218</i>         | Frees memory used for a generated report.     |

How to use the functions is described in *"Generating PPR Reports"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.

## BestPprReportDelete

**Call** `b_errtype BestPprReportDelete( b_handletype handle );`

**Description** Frees the memory space allocated by *"BestPprReportWrite" on page 222*.

**CLI Equivalent** `BestPprReportDelete`

**CLI Abbreviation** `pprrd`

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**See also** —

## BestPprReportFile

**Call** `b_errtype BestPprReportFile(  
    b_handletype handle,  
    b_charptrtype filename );`

**Description** Generates a report and writes the content to the specified file. If the file already exists, it will be overwritten. For more information on reports, see “*BestPprReportWrite*” on page 222.

**CLI Equivalent** `BestPprReportFile filename=<filename>`

**CLI Abbreviation** `prrf file=<filename>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**filename** String holding a valid filename for the report file.

**See also** –

## BestPprReportPropDefaultSet

**Call** `b_errtype BestPprReportPropDefaultSet( b_handletype handle );`

**Description** Sets the report properties to default values. For a description of properties and default values, see “*bppr\_reportproptype*” on page 313.

**CLI Equivalent** `BestPprReportPropDefaultSet`

**CLI Abbreviation** `prrpds`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestPprReportPropGet*” on page 220  
“*BestPprReportPropSet*” on page 221

## BestPprReportPropGet

**Call** `b_errtype BestPprReportPropGet (`  
     `b_handletype           handle,`  
     `bppr_reportproptype   prop,`  
     `b_int32                *value );`

**Description** Reads the value of a report property. The report properties determine which information is printed to the report.

**CLI Equivalent** `BestPprReportPropGet prop=<prop>`

**CLI Abbreviation** `pprrpg p=<prop>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**prop** Property to be read; see “*bppr\_reportproptype*” on page 313.

**Output Parameters** **value** Value of the read report property; see “*bppr\_reportproptype*” on page 313.

**See also** “*BestPprReportPropDefaultSet*” on page 219  
 “*BestPprReportPropSet*” on page 221

## BestPprReportPropSet

**Call**    `b_errtype BestPprReportPropSet (`  
                  `b_handletype          handle,`  
                  `bppr_reportproptype  prop,`  
                  `b_int32              value );`

**Description**    Sets a report property. The report properties determine which information is printed to the report.

**CLI Equivalent**    `BestPprReportPropSet prop=<prop> value=<value>`

**CLI Abbreviation**    `pprrps p=<prop> val=<value>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**prop**    Property to be set; see “*bppr\_reportproptype*” on page 313.

**value**    Value to which the property is set; see “*bppr\_reportproptype*” on page 313.

**See also**    “*BestPprReportPropDefaultSet*” on page 219  
                  “*BestPprReportPropGet*” on page 220

## BestPprReportWrite

**Call** `b_errtype BestPprReportWrite(  
 b_handletype handle,  
 b_charptrtype *report_ptr );`

**Description** Generates a report. This function performs the permutations using the properties of master block variations, master attributes and target attributes, and reports the results.

Block or attribute pages do not need to be generated and downloaded to the testcard before calling this function. The permutations are performed internally, nothing is downloaded to or uploaded from the testcard.

The contents of the report can be determined with “*BestPprReportPropSet*” on page 221. The report may also contain hints and warnings, which are described in the table below.

**NOTE** The function returns a pointer to the generated report string. The required memory space is allocated automatically. When the report is no longer needed, the memory space must be freed with “*BestPprReportDelete*” on page 218.

**CLI Equivalent** `BestPprReportWrite.`  
The report string will be displayed in the CLI window.

**CLI Abbreviation** `pprrw`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**Output Parameters** **report\_ptr** Pointer to the report string.

**NOTE** If this function is used twice with the same handle, the memory of a previously generated report is automatically deleted.

**See also** –

## Hints and Warnings in the Report String

| Text in the Report String  | Description  |
|--|--|
| <b>HINT:</b> Coverage assumes that all permutation numbers from 1 to first permutation = <N> have been executed before.  | Issued if the first permutation number is not equal to 1.  |
| <b>HINT:</b> This master attribute page should be called with a block size = <N> dwords to avoid shortened bursts.   | Always issued. Informs you about the minimum block size to be used if the master attribute page is downloaded and called.  |
| <b>WARNING:</b> Burstlengths cannot be guaranteed, because the largest block size (<N> dwords) is smaller than the sum of all burstlengths (<M>).  | Issued if the parameter BPPR_BLK_SIZE is smaller than the sum of all burstlengths N and therefore the performance of all burstlengths cannot be guaranteed.  |
| <b>WARNING:</b> There is no permutation of block parameters, which allows the usage of the MWI command according to PCI Specification.   | The variation properties for block variation parameters are defined in a way that no MWI command can be generated by the PCI PPR software. See <i>"BestPprBlockVariationSet" on page 209</i> .         |
| <b>WARNING:</b> MWI bursts must have sizes of multiple cachelines. Therefore make sure you use infinite burstlength when generating attribute pages by PPR, or set up your own attribute page so that a LAST bit will not interrupt transfer within a cacheline! | Burstlengths are set to values shorter than cacheline size, and simultaneously MWI is contained in the value list of commands. This will result in MWI transfers being interrupted within a cacheline. |

# Target Attribute Permutation Functions

The following functions are used to prepare and perform the permutation of the target attributes:

| Function   | Result  |
|--|---|
| <i>"BestPprTAttrInit" on page 226</i>                | Initializes the target attribute permutator.                                |
| <i>"BestPprTAttrPermPropDefaultSet" on page 227</i>  | Sets all target attribute properties to default values.                     |
| <i>"BestPprTAttrPermPropSet" on page 228</i>         | Sets values of target attribute permutation properties.                     |
| <i>"BestPprTAttrPermPropGet" on page 227</i>         | Reads values of target attribute permutation properties.                    |
| <i>"BestPprTAttrVariationDefaultSet" on page 229</i> | Sets all target attribute variation parameters to default values.           |
| <i>"BestPprTAttrVariationSet" on page 231</i>        | Sets the value list and the corresponding algorithm of a target attribute.  |
| <i>"BestPprTAttrVariationGet" on page 230</i>        | Reads the value list and the corresponding algorithm of a target attribute. |
| <i>"BestPprTAttrGenerate" on page 226</i>            | Generates a target attribute page.  |
| <i>"BestPprTAttrResultGet" on page 229</i>           | Reads result parameters.  |
| <i>"BestPprTAttrCoverageGet" on page 225</i>         | Returns coverage and repetition length.                                     |

How to use the functions is described in *"Programming Target Attribute Permutations"* in the *Agilent E2927A Opt. 320 C-API/PPR Programmer's Guide*.



## BestPprTAttrCoverageGet

**Call** `b_errtype BestPprTAttrCoverageGet (`  
     `b_handletype       handle,`  
     `b_int32            nmb_attr,`  
     `b_tattrproptype   *attr_list,`  
     `b_int32            *rep_len,`  
     `b_int32            *covered );`

**Description** Internally performs the permutation algorithm with the parameters currently set. “Internally” means that nothing is downloaded to the testcard. This function determines whether all permutations of a given tuple of target attributes are covered, and returns the repetition length of the tuple.

**CLI Equivalent** No CLI equivalent. To get coverage information with the CLI, use “*BestPprReportWrite*” on page 222.

**CLI Abbreviation** No CLI abbreviation.

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**nmb\_attr** Number of target attributes in the tuple.

**attr\_list** This array defines the tuple with the coverage to be tested. The tuple holds nmb\_attr entries. The entries must be valid values of “*b\_tattrproptype*” on page 288.

**Output Parameters** **rep\_len** Returns the repetition length of the tuple (that is the number of permutations of the tuple).

**covered** Returned values:

- 1 – if the tuple is covered after complete execution of the target attribute page created by the permutation algorithm.
- 0 – if this is not the case.

**See also** –

## BestPprTAttrGenerate

**Call** `b_errtype BestPprTAttrGenerate( b_handletype handle );`

**Description** Generates a target attribute page using the properties shown in the table below. The target attribute page is downloaded to the testcard.

| Properties of Target Attribute Page Generation | Adjusted by Function                          |
|--|---|
| Generic setup properties                       | <i>"BestPprGenPropSet" on page 200</i>        |
| Target attribute permutation properties        | <i>"BestPprTAttrPermPropSet" on page 228</i>  |
| Target attribute variation parameters          | <i>"BestPprTAttrVariationSet" on page 231</i> |

**CLI Equivalent** `BestPprTAttrGenerate`

**CLI Abbreviation** `pprtag`

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**See also** *"b\_tattrproptype" on page 288*

## BestPprTAttrInit

**Call** `b_errtype BestPprTAttrInit( b_handletype handle );`

**Description** Initializes the target attribute permutator and sets all of its properties to default values.

To set individual properties to default values, use either the function *"BestPprTAttrPermPropDefaultSet" on page 227* or *"BestPprTAttrVariationDefaultSet" on page 229*.

**CLI Equivalent** `BestPprTAttrInit`

**CLI Abbreviation** `pprtai`

**Return Value** Error code; see *"b\_errtype" on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**See also** –

## BestPprTAttrPermPropDefaultSet

**Call** `b_errtype BestPprTAttrPermPropDefaultSet( b_handletype handle );`

**Description** Sets the target attribute permutation properties to default values. For a description of properties and default values, see *“bppr\_tattrpermproptype” on page 314*.

**CLI Equivalent** `BestPprTAttrPermPropDefaultSet`

**CLI Abbreviation** `pprbvds`

**Return Value** Error code; see *“b\_errtype” on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**See also** *“BestPprTAttrPermPropGet” on page 227*  
*“BestPprTAttrPermPropSet” on page 228*

## BestPprTAttrPermPropGet

**Call** `b_errtype BestPprTAttrPermPropGet (
 b_handletype handle,
 bppr_tattrpermproptype prop,
 b_int32 *value );`

**Description** Reads the value of a target attribute permutation property.

**CLI Equivalent** `BestPprTAttrPermPropGet prop=<prop>`

**CLI Abbreviation** `pprtappg p=<prop>`

**Return Value** Error code; see *“b\_errtype” on page 250*.

**Input Parameters** **handle** Handle to identify the session.

**prop** Property to be read; see *“bppr\_tattrpermproptype” on page 314*.

**Output Parameters** **value** Value of the read property; see *“bppr\_tattrpermproptype” on page 314*.

**See also** *“BestPprTAttrPermPropDefaultSet” on page 227*  
*“BestPprTAttrPermPropSet” on page 228*

## BestPprTAttrPermPropSet

**Call**    `b_errtype BestPprTAttrPermPropSet (`  
                  `b_handletype                  handle,`  
                  `bppr_tattrpermproptype  prop,`  
                  `b_int32                      value );`

**Description**    Sets a target attribute permutation property.

**CLI Equivalent**    `BestPprTAttrPermPropSet prop=<prop> value=<value>`

**CLI Abbreviation**    `pprtapps p=<prop> val=<value>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**prop**    Property to be set; see “*bppr\_tattrpermproptype*” on page 314.

**value**    Value to which the property is set; “*bppr\_tattrpermproptype*” on page 314.

**See also**    “*BestPprTAttrPermPropDefaultSet*” on page 227  
                  “*BestPprTAttrPermPropGet*” on page 227

## BestPprTAttrResultGet

**Call** `b_errtype BestPprTAttrResultGet (`  
           `b_handletype                    handle,`  
           `bppr_tattrresultparamtype param,`  
           `b_int32                         *value );`

**Description** Internally performs the permutation algorithm with the parameters currently set. “Internally” means nothing is downloaded to the testcard. This function reads the value of a selected result parameter.

**CLI Equivalent** `BestPprTAttrResultGet param=<param>`

**CLI Abbreviation** `pprtarg p=<param>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**param** Result parameter to be read; see “*bppr\_tattrresultparamtype*” on page 315.

**Output Parameters** **value** Value of the read result parameter; see “*bppr\_tattrresultparamtype*” on page 315.

**See also** –

## BestPprTAttrVariationDefaultSet

**Call** `b_errtype BestPprTAttrVariationDefaultSet ( b_handletype handle );`

**Description** Sets the target attribute variation parameters to default values. For a description of parameters and default values, see “*b\_tattrproptype*” on page 288.

**CLI Equivalent** `BestPprTAttrVariationDefaultSet`

**CLI Abbreviation** `pprtavds`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestPprTAttrVariationGet*” on page 230  
 “*BestPprTAttrVariationSet*” on page 231

## BestPprTAttrVariationGet

**Call**    `b_errtype BestPprTAttrVariationGet (`  
                  `b_handletype          handle,`  
                  `b_tattrproptype      attribute,`  
                  `b_charptrtype          *value_list,`  
                  `bppr_algorithmtype      *algorithm );`

**Description**    Reads variation properties of a target attribute.

**CLI Equivalent**    `BestPprTAttrVariationGet attribute=<attribute>`

**CLI Abbreviation**    `pprtvg attr=<attribute>`

**Return Value**    Error code; see “*b\_errtype*” on page 250.

**Input Parameters**    **handle**    Handle to identify the session.

**attribute**    Target attribute to be read; see “*b\_tattrproptype*” on page 288.

**Output Parameters**    **value\_list**    Returns a pointer to a string containing a comma-separated list of values. The values in this list are permuted according to the selected algorithm. For a description of the string’s contents, refer to “*Numeric values*” on page 309.

If the PCI PPR software has not yet been initialized with “*BestPprBlockInit*” on page 204 or “*BestPprInit*” on page 201, the list will be empty and an error will be returned.

**algorithm**    Returns the algorithm used for this attribute. See “*bppr\_algorithmtype*” on page 302.

**See also**    “*BestPprTAttrVariationDefaultSet*” on page 229  
               “*BestPprTAttrVariationSet*” on page 231

## BestPprTAttrVariationSet

**Call** `b_errtype BestPprTAttrVariationSet (`  
     `b_handletype           handle,`  
     `b_tattrproptype       attribute,`  
     `b_charptrtype        value_list,`  
     `bppr_algorithmtype   algorithm );`

**Description** Defines the value list for a target attribute and selects the algorithm used to pick values from this list.

**CLI Equivalent** `BestPprTAttrVariationSet attribute=<attribute>`  
`value_list=<value_list> algorithm=<algorithm>`

**CLI Abbreviation** `pprtavs attr=<attribute> list=<value_list> alg=<algorithm>`

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**attribute** Target attribute; see “*b\_tattrproptype*” on page 288.

**value\_list** List of values to be permuted according to the selected algorithm. For the syntax of the list, see “*Numeric values*” on page 309.

**algorithm** Algorithm used to pick values from the value list; see “*bppr\_algorithmtype*” on page 302.

**See also** “*BestPprTAttrVariationDefaultSet*” on page 229  
 “*BestPprTAttrVariationGet*” on page 230





# Error Handling

The application programming interface of the testcard provides several functions to handle occurred errors.

Error functions can be used to query the error code and the error string of the last error that occurred in the specified session. But it is also possible to query the meaning of an error code if no handle is available.

## Error Functions

The following functions are used for error handling:

| Function                                    | Result                                      |
|---|---|
| <i>"BestLastErrorGet" on page 234</i>       | Returns the last error.                     |
| <i>"BestLastErrorStringGet" on page 234</i> | Returns the error string of the last error. |
| <i>"BestErrorStringGet" on page 235</i>     | Returns the error string of an error.       |

## BestLastErrorGet

**Call** `b_errtype BestLastErrorGet( b_handletype handle );`

**Description** Returns the code of the last error that occurred with the handle.

**CLI equivalent** No CLI equivalent.

**CLI abbreviation** No CLI abbreviation.

**Return Value** Error code; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestLastErrorStringGet*” on page 234  
“*BestErrorStringGet*” on page 235

## BestLastErrorStringGet

**Call** `b_charptrtype BestLastErrorStringGet( b_handletype handle );`

**Description** Returns the error string of the last error that occurred with the handle.

**CLI equivalent** No CLI equivalent.

**CLI abbreviation** No CLI abbreviation.

**Return Value** Error string; see “*b\_errtype*” on page 250.

**Input Parameters** **handle** Handle to identify the session.

**See also** “*BestLastErrorGet*” on page 234  
“*BestErrorStringGet*” on page 235

## BestErrorStringGet

**Call** `b_charptrtype BestErrorStringGet( b_errtype err );`

**Description** Returns the error string of an error. Use this function if no handle is available (for example, if “*BestOpen*” on page 21 fails).

**CLI equivalent** No CLI equivalent.

**CLI abbreviation** No CLI abbreviation.

**Return Value** Error string; see “*b\_errtype*” on page 250.

**Input Parameters** **err** Error define.

**See also** “*BestLastErrorGet*” on page 234  
“*BestLastErrorStringGet*” on page 234



# Type Definitions

All type definitions are listed in alphabetical order.

Several definitions differ in their naming conventions. So in the first part the type names begin with “b\_”, in the second part the names begin with “bppr\_”. The suffix “ppr” identifies type definitions which are only used in Protocol Permutator and Randomizer functions.

For further information on the naming conventions, see “*Conventions*” on page 14.

## b\_addrspacetype

| addrspace (CLI Abbreviation)     | Description                    |
|----------------------------------|--------------------------------|
| B_ASP_CONFIG (config)            | Type 0 access to config space. |
| B_ASP_CONFIG_TYPE1 (configtype1) | Type 1 access to config space. |
| B_ASP_IO (io)                    | Access to I/O space.           |
| B_ASP_MEM (mem)                  | Access to memory space.        |

# b\_blkproptype

| blk_prop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation)  | Description  |
|--------------------------------|--|--|
| B_BLK_ATTRPAGE<br>(apage)      | 0 ... 63; <b>default:</b> 0 – if page size is 4 ,<br>or<br>0 ... 7; <b>default:</b> 0 – if page size is 32<br>(for compatibility reasons). | Pointer to a master attribute page to define PCI protocol<br>behavior for the block transfer.  |
| B_BLK_BUSADDR<br>(bad)         | 32 bits.   | PCI bus address, used as the starting address for the block<br>transfer (64-bit address width only if master block property<br>B_BLK_BUSDAC is enabled). |
| B_BLK_BUSADDR_HI<br>(badhi)    | Upper 32 bits of a 64-bit address.   |  |
| B_BLK_BUSCMD<br>(cmd)          | PCI bus command for the block transfer (C/BE#[3:0] during address phases).   |  |
|                                | B_CMD_INT_ACK<br>(int_ack)   | Interrupt Acknowledge  |
|                                | B_CMD_SPECIAL<br>(special)   | Special Cycle  |
|                                | B_CMD_IO_READ<br>(io_read)   | I/O Read   |
|                                | B_CMD_IO_WRITE<br>(io_write)   | I/O Write  |
|                                | B_CMD_RESERVED_4<br>(reserved_4)   | Reserved   |
|                                | B_CMD_RESERVED_5<br>(reserved_5)   | Reserved   |
|                                | <b>default:</b><br>B_CMD_MEM_READ (mem_read)   | Memory Read  |
|                                | B_CMD_MEM_WRITE<br>(mem_write)   | Memory Write   |
|                                | B_CMD_RESERVED_8<br>(reserved_8)   | Reserved   |
|                                | B_CMD_RESERVED_9<br>(reserved_9)   | Reserved   |
|                                | B_CMD_CONFIG_READ<br>(config_read)   | Configuration Read   |

| blk_prop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation)   | Description   |
|--------------------------------|---|---|
| B_BLK_BUSCMD<br>(cmd)          | B_CMD_CONFIG_WRITE<br>(config_write)  | Configuration Write   |
|                                | B_CMD_MEM_READMULTIPLE<br>(mem_readmultipl )  | Memory Read Multiple (MRM)  |
|                                | B_CMD_MEM_READLINE<br>(mem_readline)  | Memory Read Line (MRL)  |
|                                | B_CMD_MEM_WRITEINVALIDATE,<br>(writeinvalidate)   | Memory Write & Invalidate (MWI)<br><br>MWI must be enabled in the configuration space and must start at a cacheline boundary. If not, it is replaced by Memory Write.<br><br><b>Note:</b> When used in combination with B_M_LAST, illegal burstlengths could be generated.  |
| B_BLK_BUSDAC<br>(dac)          | Determines whether 64-bit or 32-bit addresses are transferred during an address phase.  |   |
|                                | <b>default:</b> 0   | Single address cycle with 32-bit address.   |
|                                | 1   | Dual address cycle with 64-bit address. Uses B_BLK_BUSADDR_HI.  |
| B_BLK_BYTEN<br>(ben)           | 0x00 ... 0xFF<br><b>default:</b> 00   | Pointer to the byte enable memory.<br><br>00 – all byte enables are active during all data phases of the block transfer.<br><br>1 ... 15 – values for the lower byte enables (C/BE[3:0]). Valid for all data phases of the block transfer. (The upper byte enables are set to the same value.)<br><br>16 ... 255 – pointer to the byte enable memory storing a series of byte enables (8 bit) that can be worked through during the block transfer. |
| B_BLK_BYTEN_VAR<br>(benv)      | Determines whether the byte enables are fixed or variable during the block transfer.  |   |
|                                | <b>default:</b> 0   | Byte enables are fixed.<br>The pointer to the byte enable memory is not changed.  |
|                                | 1   | Byte enables are variable.<br>The next byte enables are taken from byte enable memory after each successful data transfer.  |
| B_BLK_COMPFLAG<br>(cflag)      | The compare flag determines whether read data are compared with data stored in the testcard's internal data memory. The comparison takes place while the data is read. The read data is not stored in the memory. |   |
|                                | <b>default:</b> 0   | No comparison. Data is stored. Start address of internal memory is given by B_BLK_INTADDR.  |
|                                | 1   | Data is compared with data stored in internal memory, beginning with the start address given by B_BLK_COMPOFFS.   |
| B_BLK_COMPOFFS<br>(coffs)      | 0 ... 0x1FFFC<br><b>default:</b> 0  | Dword-aligned compare offset, see B_BLK_COMPFLAG.   |

| blk_prop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation)   | Description  |
|--------------------------------|---|--|
| B_BLK_CONDSTART<br>(cond)      | The conditional start flag determines whether the master run of a block is performed immediately or whether a start condition must first be fulfilled.  |  |
|                                | <b>default:</b> 0   | Start immediately.   |
|                                | 1   | Wait until start condition is fulfilled. The start condition must be set using <i>"BestMasterCondStartPattSet"</i> on page 106.<br><br>The generic master property B_MGEN_RUNMODE must be set at 1. If not, conditional start is generally disabled. |
| B_BLK_CONTATTR<br>(contattr)   | The continue attributes flag determines whether an attribute page is restarted after a block transfer has been completed or whether transfer is continued from the current pointer position.<br><br>After startup or reset, the master attribute page is always started at the beginning.<br><br><b>Note:</b> If this flag is 0, it can still be overridden by the setting of the generic master property B_MGEN_ATTRMODE. See <i>"BestMasterGenPropSet"</i> on page 105. |  |
|                                | <b>default:</b> 0   | The master attribute page as given by B_BLK_ATTRPAGE is restarted.   |
|                                | 1   | Transfer is continued from the current pointer position.   |
| B_BLK_INTADDR<br>(iad)         | 00000\h ... FFFFF\h<br><b>default:</b> 0<br><br>Must be dword-aligned and have the same alignment as B_BLK_BUSADDR.   | Internal Address of the testcard's data memory. Used to access data to be written or to store data to be read. (NOT used for data compare!)  |



# b\_boardproptype

| boardprop (CLI Abbreviation) | value (CLI Abbreviation)  | Description  |
|------------------------------|---|--|
| B_BOARD_PERREN<br>(perren)   | Enables/disables parity error (PERR#) reporting by setting the configuration space command register bit 6.        |  |
|                              | <b>default:</b> 0   | Disables parity error reporting.   |
|                              | 1   | Enables parity error reporting.  |
| B_BOARD_RSTMODE<br>(rstmode) | B_RSTMODE_RESETALL<br>(resetall, 0)   | After a PCI reset, a testcard reset is performed.  |
|                              | B_RSTMODE_RESETSM<br>(resetasm, 1)  | After a PCI reset, the master, target, and analyzer state machines are reset, without affecting configuration space, decoders or other onboard properties.                       |
|                              | <b>default:</b> B_RSTMODE_RESETCONFIG<br>(resetconfig, 2)   | After a PCI reset, the configuration space is reset.<br><br>The read/write bits are restored according to the setting of B_PU_CONFRESTORE (see “BestPowerUpPropSet” on page 37). |
| B_BOARD_SERREN<br>(serren)   | Enables/disables parity system error (SERR#) reporting by setting the configuration space command register bit 8. |  |
|                              | <b>default:</b> 0   | Disables system error reporting.   |
|                              | 1   | Enables system error reporting.  |

## b\_cpuproptype

| cpuprop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation)                    | Description   |
|-------------------------------|--|---|
| B_CPU_MODE<br>(mode)          | <b>default:</b><br>B_CM_DISABLED<br>(disabled) | Disables the CPU port and sets the outputs to high impedance.                                       |
|                               | B_CM_MASTER<br>(master)                        | Enables the CPU port and sets it to master.   |
| B_CPU_PROTOCOL<br>(proto)     | <b>default:</b><br>B_CP_INTEL<br>(intel)       | Sets the protocol type to be Intel-compatible.  |
| B_CPU_RDYTYPE<br>(rdy)        | <b>default:</b><br>B_CR_AUTO<br>(auto)         | Sets the RDY# signal generation to internal. The target device is assumed to be ready after 300 ns. |
|                               | B_CR_EXTERNAL<br>(external)                    | Sets the RDY# signal generation to external.  |

## b\_decodertype

| decoder<br>(CLI Abbreviation)     | Description   |
|-----------------------------------|---|
| B_DEC_CONFIG<br>(config)          | Configuration decoder   |
| B_DEC_EXPROM<br>(exprom)          | Expansion ROM decoder   |
| B_DEC_FULL_CONFIG<br>(fullconfig) | Full configuration decoder (configuration cycle type 1 decoder) |
| B_DEC_STANDARD_1<br>(std1)        | Standard decoder 1  |
| B_DEC_STANDARD_2<br>(std2)        | Standard decoder 2  |
| B_DEC_STANDARD_3<br>(std3)        | Standard decoder 3  |
| B_DEC_STANDARD_4<br>(std4)        | Standard decoder 4  |
| B_DEC_STANDARD_5<br>(std5)        | Standard decoder 5  |
| B_DEC_STANDARD_6<br>(std6)        | Standard decoder 6  |
| B_DEC_SUBTRACTIV<br>(subtract)    | Subtractive decoder   |

## b\_decproptype

The target decoder properties of type b\_decproptype can be grouped according to their functions:

- “Decoding Properties” on page 244
- “Info Properties” on page 248
- “Resource Properties” on page 249

## Decoding Properties

| decoder_prop<br>(CLI Abbreviation) | Affects<br>Decoder  | value<br>(CLI Abbreviation)  | Description  |
|------------------------------------|---|------------------------------|--|
| B_DEC_BASEADDR<br>(base)           | All except<br>Subtractive   | Bits[31:4] of a 32-bit value | Base address of the decoder and also the entry of the Base Address Register of the decoder in the configuration space.<br><br>Specific bits have specific meanings during configuration cycles. If you change their values while the testcard is being used as a target, the behavior of the device/system under test can change.<br><br><b>Warning:</b> Setting multiple devices with overlapping address ranges within a system may result in hardware damage. |
| B_DEC_BASEADDR_HI<br>(basehi)      | All except<br>Subtractive   | 32 Bit                       | Upper 32-bit of a 64-bit base address.<br>See B_DEC_BASEADDR.<br><br>B_DEC_DAC must be set to B_DAC_YES.   |
| B_DEC_BASEDEC<br>(basedec)         | Standard  | 1 ... 6                      | Base decoder identifier. To be used with overlay behavior.<br><br><b>Note:</b> The decoders 1 ... 6 provide increasing priority.<br>Therefore, the decoder with overlay behavior must have a higher number than the base decoder. Otherwise the base decoder cannot decode.  |
| B_DEC_BEHAVIOR<br>(behavior)       | Determines the behavior of Standard Decoders 1 ... 6.<br>This property influences the availability and ranges of other target decoder properties. |                              |  |
|                                    | Standard  | B_BEH_NORMAL<br>(normal)     | Decoder behaves PC compliant, according to the settings in the configuration space.  |
|                                    |   | B_BEH_OVERLAY<br>(overlay)   | Decoder is an overlay decoder. All bits of its base address register return 0 on a read.<br><br>B_DEC_BASEDEC must contain the base decoder identifier.  |
|                                    |   | B_BEH_CONFIG<br>(config)     | Decoder behaves like a configuration decoder. All bits of its base address register return 0 on a read.  |
|                                    |   | B_BEH_CUSTOM<br>(custom)     | Decoder can be set up non-PCI compliant. Property values are not checked (see <i>"BestTargetDecoderProg"</i> on page 137).   |

| decoder_prop<br>(CLI Abbreviation) | Affects<br>Decoder  | value<br>(CLI Abbreviation)              | Description  |
|------------------------------------|---|--|--|
| B_DEC_BUSCMD<br>(cmd)              | Commands to be decoded. Use a logical OR-connection to decode multiple commands . |  |  |
|                                    | All except<br>Expan. ROM  | B_CMDBIT_INT_ACK<br>(0x0001)             | Interrupt Acknowledge  |
|                                    |   | B_CMDBIT_SPECIAL<br>(0x0002)             | Special Cycle  |
|                                    |   | B_CMDBIT_IO_READ<br>(0x0004)             | I/O Read   |
|                                    |   | B_CMDBIT_IO_WRITE<br>(0x0008)            | I/O Write  |
|                                    |   | B_CMDBIT_RESERVED_4<br>(0x0010)          | Reserved   |
|                                    |   | B_CMDBIT_RESERVED_5<br>(0x0020)          | Reserved   |
|                                    |   | B_CMDBIT_MEM_READ<br>(0x0040)            | Memory Read  |
|                                    |   | B_CMDBIT_MEM_WRITE<br>(0x0080)           | Memory Write   |
|                                    |   | B_CMDBIT_RESERVED_8<br>(0x0100)          | Reserved   |
|                                    |   | B_CMDBIT_RESERVED_9<br>(0x0200)          | Reserved   |
|                                    |   | B_CMDBIT_CONFIG_READ<br>(0x0400)         | Configuration Read   |
|                                    |   | B_CMDBIT_CONFIG_WRITE<br>(0x0800)        | Configuration Write  |
|                                    |   | B_CMDBIT_MEM_READMULTIPLE<br>(0x1000)    | Memory Read Multiple (MRM)   |
|                                    |   | B_CMDBIT_MEM_READLINE<br>(0x4000)        | Memory Read Line (MRL)   |
|                                    |   | B_CMDBIT_MEM_WRITEINVALIDATE<br>(0x8000) | Memory Write & Invalidate (MWI)<br>MWI must be enabled in the configuration space and must start at a cacheline boundary. Otherwise "Memory Write" replaces MWI. |

| decoder_prop<br>(CLI Abbreviation) | Affects<br>Decoder  | value<br>(CLI Abbreviation)    | Description  |
|------------------------------------|---|--------------------------------|--|
| B_DEC_DAC<br>(dac)                 | Specifies whether the decoder is 64-bit capable. For 64-bit address decoding, two standard decoders are used as a pair:<br><br>Standard decoder 1 decodes the lower 32 bits, standard decoder 2 the upper 32.<br><br>Standard decoder 3 decodes the lower 32 bits, standard decoder 4 the upper 32.<br><br>Standard decoder 5 decodes the lower 32 bits, standard decoder 6 the upper 32. |                                |  |
|                                    | Standard<br>Subtractive   | B_DAC_NO<br>(no, 0)            | Decoder is not 64-bit capable.   |
|                                    |   | B_DAC_YES<br>(yes, 1)          | Decoder is 64-bit capable.   |
|                                    | Subtractive   | B_DAC_BOTH<br>(both, 2)        | Decodes both 32 and 64-bit addresses.  |
| B_DEC_IDSEL<br>(idsel)             | Specifies whether IDSEL must be asserted to recognize a configuration command.  |                                |  |
|                                    | Standard<br>Configuration<br>Subtractive  | B_IDSEL_ASSERT<br>(assert)     | IDSEL must be asserted.  |
|                                    |   | B_IDSEL_DEASSERT<br>(deassert) | IDSEL must be de-asserted.   |
|                                    |   | B_IDSEL_DONTCARE<br>(dontcare) | IDSEL is ignored.  |
| B_DEC_MASK<br>(mask)               | Standard  | 32 Bit                         | Specifies the bits in the base address register relevant for decoding:<br><br>1 – read/write (relevant)<br><br>0 – read-only (not relevant)<br><br><b>Warning:</b> Setting multiple devices with overlapping address ranges within a system may result in hardware damage. |
| B_DEC_MASK_HI<br>(maskhi)          | Standard  | 32 Bit                         | Upper 32-bit of a 64-bit address. See B_DEC_MASK.<br><br>B_DEC_DAC must be set to B_DAC_YES or B_DAC_BOTH.   |
| B_DEC_PRIMARY_BUS<br>(primary)     | Config. Type 1  | 0 ... 255                      | Specifies the lowest bus number for which type 1 configuration cycles are decoded.   |
| B_DEC_SECONDARY_BUS<br>(secondary) | Config. Type 1  | 0 ... 255                      | Specifies the highest bus number for which type 1 configuration cycles are decoded.  |

| decoder_prop<br>(CLI Abbreviation) | Affects<br>Decoder  | value<br>(CLI Abbreviation)  | Description   |
|------------------------------------|---|------------------------------|---|
| B_DEC_SIZE<br>(size)               | Standard  | 0, 5 ... 64                  | <p>Determines the size of the decoded address range (actual size = <math>2^{B\_DEC\_SIZE}</math>).</p> <p>A size of 0 switches off the decoder by setting all bits of its base address register to zero, unless behavior is set to "custom". In this case, you have to set them to zero with B_DEC_BASEADDR.</p> <p><b>Warning:</b> Setting multiple devices with overlapping address ranges within a system may result in hardware damage.</p> |
| B_DEC_SPEED<br>(speed)             | <p>Determines the decode speed (that is the delay until DEVSEL# is asserted after request).</p> <p>Fast decode has the following restrictions:</p> <p><b>Resource</b> must be the data memory or the compare unit.</p> <p>For <b>memory decoders</b>, the size value must be set between 20 and 43 (or mask must be set accordingly). "Memory Read Multiple" is not allowed.</p> <p>For <b>I/O decoders</b>, the size value must be between 2 and 16 (or mask must be set accordingly).</p> <p><b>Configuration decoders</b> require IDSEL to be set. Only the bits carrying the "function" information are decoded.</p> <p><b>Expansion ROM decoders and Full configuration decoders</b> cannot fast decode.</p> <p><b>Bursts</b> may run out of range.</p> <p>At one time, only <b>one decoder of each type</b> (standard, config, subtractive) can do fast decoding.</p> |                              |   |
|                                    | All   | B_DSP_MEDIUM<br>(medium)     | Sets double speed to medium.  |
|                                    |   | B_DSP_SLOW<br>(slow)         | Sets decode speed to slow.  |
|                                    |   | B_DSP_NODEVSEL<br>(nodevsel) | <p>The decoder never asserts DEVSEL#.</p> <p><b>Note:</b> The transaction can still be claimed by a subtractive decoder.</p>  |

## Info Properties

| decoder_prop<br>(CLI Abbreviation) | Affects<br>Decoder   | value<br>(CLI Abbreviation)    | Description                                |
|------------------------------------|--|--------------------------------|--|
| B_DEC_LOCATION<br>(loc)            | Sets the bits in the configuration space that specify the location where the BIOS has to place the memory range. |                                |  |
|                                    | Standard   | B_LOC_SPACE32<br>(space32)     | In 32-bit memory address range.            |
|                                    |  | B_LOC_BELOW1MEG<br>(below1meg) | Below 1 MB memory address range.           |
|                                    |  | B_LOC_SPACE64<br>(space64)     | Location is a 64-bit memory address range. |
|                                    |  | B_LOC_IO<br>(io)               | Location is an I/O address range.          |
| B_DEC_PREFETCH<br>(prefetch)       | Sets the “prefetch” bit in the configuration space. This specifies that a device provides “prefetchable” memory. |                                |  |
|                                    | Standard   | <b>default:</b> 0              | “Prefetch” bit is not set.                 |
|                                    |  | 1                              | “Prefetch” bit is set.                     |



## Resource Properties

| decoder_prop<br>(CLI Abbreviation) | Affects<br>Decoder   | value<br>(CLI Abbreviation)            | Description  |
|------------------------------------|--|--|--|
| B_DEC_RESBASE<br>(resbase)         | All  | 32 Bit                                 | Internal Base Address of the resource.<br>The value must be a multiple of the resource size ( $2^{B\_DEC\_RESSIZE}$ ).   |
| B_DEC_RESOURCE<br>(res)            | Specifies the internal resource to which the decoder is connected. |  |  |
|                                    | All  | B_RES_DATA<br>(data)                   | Internal data memory. Protocol attributes are used as defined in the target attribute page.  |
|                                    |  | B_RES_DATA_<br>DEFATTR<br>(datadef)    | Internal data memory. Default protocol attributes are used.  |
|                                    |  | B_RES_COMPARE<br>(comp)                | Compare unit. For write transactions only. Incoming data is not stored. Protocol attributes are used as defined in the target attribute page.  |
|                                    |  | B_RES_COMPARE_<br>DEFATTR<br>(compdef) | Compare unit. For write transactions only. Incoming data is not stored. Default protocol attributes are used.  |
|                                    |  | B_RES_REGFILE<br>(regfile)             | Configuration Space, public and private section (this is also referred to as "internal register file").<br><b>Note:</b> The private section is reserved. Overwriting can result in the loss of the connection to the testcard. |
| B_DEC_RESSIZE<br>(ressize)         | All  | 0 ... 64                               | Determines the size of the resource (actual size of the address range = $2^{B\_DEC\_RESSIZE}$ ).   |

# b\_errtype

There are two types of error codes:

- **Software Errors** (codes beginning with “B\_E\_”).

The software errors are issued by the testcard software running on the control PC.

- **Firmware Errors** (codes beginning with “B\_EFW\_”).

Firmware errors are issued by the firmware running directly on the card.

This information may help you to localize some errors.

## Software Errors

| err/Error Code               | Error String   | Notes/Recommendations/Help  |
|------------------------------|--|---|
| B_E_ALIGN                    | Parameter <i>&lt;parameter&gt;</i> must be aligned to <i>&lt;align-<br/>mt&gt;</i> . Its value is <i>&lt;value&gt;</i> .   |   |
| B_E_BAD_DECODER_NUMBER       | No valid decoder number.   |   |
| B_E_BAD_FILE_FORMAT          | Bad format for file <i>&lt;filename&gt;</i> .  |   |
| B_E_BAD_HANDLE               | Bad handle. You may only use handles obtained from “BestOpen()”.   |   |
| B_E_BAUDRATE                 | Could not set new baud rate.   |   |
| B_E_BOARD_RESET              | Could not re-connect after board reset.  |   |
| B_E_CANNOT_CONNECT           | The specified port <i>&lt;port&gt;</i> is unable to connect. Please check cable and connectors and try again. If your card is E2925A it may be connected exclusively.  | Another reason may be that the card is busy, for example, with transferring data via the fast host interface. |
| B_E_CANNOT_CONNECT_CORE      | <i>&lt;port&gt;</i> port is unable to connect because the card is operating in core mode. Please reset the card. If the problem persists, run a hardware update.   |   |
| B_E_CANNOT_CONNECT_EXCLUSIVE | The specified port <i>&lt;port&gt;</i> cannot get an exclusive connection to the card. Check that no other port is connected exclusively and try again. If the problem persists, try a conventional (non-exclusive) connection. This message applies to the E2925A only. | Another reason may be that the card is busy, for example, with transferring data via the fast host interface. |
| B_E_CONF_REG                 | Could not write to config space OR the value to be written is invalid for that register.   |   |
| B_E_CONFIG_MASK_INVALID      | Mask <i>&lt;mask&gt;</i> for config space register <i>&lt;cfreg&gt;</i> is invalid or not PCI compliant.   |   |

| err/Error Code               | Error String   | Notes/Recommendations/Help  |
|------------------------------|--|---|
| B_E_CONFIG_VALUE_INVALID     | Value <i>&lt;value&gt;</i> for config space register <i>&lt;cfreg&gt;</i> is invalid or not PCI compliant.   |   |
| B_E_CONNECTION_LOST          | <i>&lt;port&gt;</i> port lost its connection. Please check cable connection, and try to close and re-open the connection.  |   |
| B_E_CONNECTION_LOST_CMD      | <i>&lt;port&gt;</i> port lost its connection. Please check cable connection and try to close and re-open the connection.<br><i>&lt;first cmdbyte&gt;&lt;second cmdbyte&gt;</i> |   |
| B_E_CORE_VERSION_MISMATCH    | Minimal core version for firmware <i>&lt;version1&gt;</i> is <i>&lt;version2&gt;</i> . Current core version is <i>&lt;version3&gt;</i> .                                       | Try hardware update from graphical user interface.                                  |
| B_E_CPU_MISALIGNED           | CPU Port address misaligned.   |   |
| B_E_DEC_BASE_NOT_0           | Base address must be 0 when size is 0.   |   |
| B_E_DEC_BASE_WRITE           | Error while writing base address into hardware.  | Base address is not PCI compliant.  |
| B_E_DEC_CHECK                | Unknown decoder error. (Perhaps there is no PCI clock or hardware is busy.)  |   |
| B_E_DEC_CONFIG_ACCESS        | Error while accessing hardware (config space busy, or no PCI clock).   |   |
| B_E_DEC_INVALID_MODE         | Invalid mode specified for this decoder.   |   |
| B_E_DEC_INVALID_SIZE         | Invalid size specified for this decoder.   |   |
| B_E_DEC_ROM_SIZE_0_ENABLED   | Inconsistency: Generic ROM enable active, but size is 0.   |   |
| B_E_DEC_SIZE_BASE_MISMATCH   | There is a decoder size and base address mismatch.   |   |
| B_E_DRIVER_VERSION_DIFF      | The PCI driver version is incompatible with this C-API.  | Re-install the software.  |
| B_E_DYNAMIC_CAPABILITY       | Error while checking dynamic capabilities.   |   |
| B_E_ERROR                    | Error during command transfer.   |   |
| B_E_FCT_PARAM                | Function <i>&lt;function&gt;</i> rejected parameter <i>&lt;param&gt;</i> : <i>&lt;value&gt;</i> .  |   |
| B_E_FILE_OPEN                | Could not open file <i>&lt;filename&gt;</i> .  |   |
| B_E_FUNC                     | Functional onboard error.  | Runtime error on the hardware. Try again, if the problem persists, reboot the card. |
| B_E_HIF_OPEN                 | Could not open BEST Fast Host Interface driver.  |   |
| B_E_HOST_MEM_FULL            | Not enough memory.   |   |
| B_E_HW_BUSY                  | Board hardware is busy, or no PCI clock.   |   |
| B_E_INVALID_OBS_RULE         | Observer rule unknown or not implemented.  |   |
| B_E_MASK_REG                 | Could not write config mask OR the mask value is invalid for that register.  |   |
| B_E_MASTER_ABORT             | Master abort occurred, no target response.   |   |
| B_E_NO_BEST_PCI_DEVICE_FOUND | Specified device not found on PCI Bus.   |   |

| err/Error Code       | Error String   | Notes/Recommendations/Help |
|----------------------|--|----------------------------|
| B_E_NO_HANDLE_LEFT   | Cannot open another port because no handle is left. Close some ports and try again.  |                            |
| B_E_NO_PCI_CLOCK     | No or slow PCI clock.  |                            |
| B_E_NOT_COMPACT      | This command is not supported by the E2940A.   |                            |
| B_E_NOT_CONNECTED    | The specified port <port> is not connected. Please execute the BestConnect() command. This error message applies to the E2925A only.                           |                            |
| B_E_NOT_E2925A       | This command is not supported by the E2925A.   |                            |
| B_E_OK               | No error.  |                            |
| B_E_ONLY_COMPACT     | This command is only supported by the E2940A.  |                            |
| B_E_ONLY_E2925A      | This command is only supported by the E2925A.  |                            |
| B_E_ONLY_E2925A_DEEP | This command is only supported by the E2925A deep trace.   |                            |
| B_E_OVERFLOW         | The value returned is larger than 0xFFFFFFFF. Actual value was <value>.  |                            |
| B_E_PARALLEL_OPEN    | Could not open BEST EPP port driver.   |                            |
| B_E_PARAM            | An internal parameter was out of range, unable to continue. Please check parameters of this call.  |                            |
| B_E_PARAM_NOT_EXIST  | In group <group> parameter <parameter> does not exist. Either the parameter or group index is out of range or the capability for the parameter is not enabled. |                            |
| B_E_PCI_NT_ONLY      | Win32 PCI access is only supported under Windows NT.   |                            |
| B_E_PCI_OPEN         | Could not open BEST PCI driver.  |                            |
| B_E_PROG_DEC_ENABLE  | BEST IO programming decoder not enabled, or could not read from config space.  |                            |
| B_E_RANGE            | Parameter <parameter> has to be in a range from <min> through <max>. Its value is <value>.   |                            |
| B_E_RS232_OPEN       | Could not open RS232 port.   |                            |
| B_E_SELFTEST_FAILED  | Self test failed. Please contact support.  |                            |
| B_E_SYNTAX           | Syntax error.  |                            |
| B_E_TEST_NO_DECODER  | First decoder not enabled.   |                            |
| B_E_UNDERFLOW        | The value returned is less than 0. Actual value was <value>.   |                            |
| B_E_UNKNOWN_ERR      | Unknown error code <errorcode>.  |                            |
| B_E_UNKNOWN_HARDWARE | The connected hardware on the <port> Port could not be identified.   |                            |

| err/Error Code       | Error String   | Notes/Recommendations/Help                         |
|----------------------|--|--|
| B_E_VALUE            | Parameter <i>&lt;parameter&gt;</i> cannot take value <i>&lt;value&gt;</i> in this context.                                     |  |
| B_E_VERSION_MISMATCH | Expected version <i>&lt;version1&gt;</i> for <i>&lt;programmable device&gt;</i> . Current version is <i>&lt;version2&gt;</i> . | Try hardware update from graphical user interface. |
| B_E_WRONG_BUSADDR    | Invalid PCI bus address specified in that address space—see PCI Spec.  |  |
| B_E_WRONG_PARAMETER  | At least one parameter value is out of range.  |  |
| B_E_WRONG_PORT       | This action is only applicable to the <i>&lt;port1&gt;</i> port. You are connected to the <i>&lt;port2&gt;</i> port.           |  |

## Firmware Errors

| Error Code                   | Error string  | Notes/<br>Recommendations/Help                         |
|------------------------------|---|--|
| B_EFW_ALIGN                  | Parameter <i>&lt;parameter&gt;</i> must be aligned to <i>&lt;alignmt&gt;</i> . Its value is <i>&lt;value&gt;</i> .                          |  |
| B_EFW_ATTR_PROGMODE_MIXED    | <i>&lt;memory type&gt;</i> <i>&lt;page&gt;</i> has to be programmed with <i>&lt;progmode&gt;</i> .  |  |
| B_EFW_BASEDEC_NOT_NORMAL     | Decoder <i>&lt;decoder1&gt;</i> cannot be a base decoder for decoder <i>&lt;decoder2&gt;</i> because it is not switched to normal behavior. | Set <i>&lt;decoder1&gt;</i> to normal behavior.        |
| B_EFW_BLOCKPAGE_CONCATENATED | Block page <i>&lt;blockpage&gt;</i> is concatenated.  |  |
| B_EFW_CARD_MEM_FULL          | Not enough onboard memory.  |  |
| B_EFW_CMDBUSY                | Cannot execute command <i>&lt;command&gt;</i> . Related resource is busy from port <i>&lt;port&gt;</i> .                                    |  |
| B_EFW_CMDLOCKED              | Cannot execute command <i>&lt;command&gt;</i> . Related resource is locked by port <i>&lt;port&gt;</i> .                                    |  |
| B_EFW_CONFIG_CMD_SUBSET      | Decoder <i>&lt;decoder&gt;</i> accepts only config commands in the command parameter. The current command mask is <i>&lt;mask&gt;</i> .     | Select configuration commands for this decoder.        |
| B_EFW_DATA_CHECKSUM          | Data transmission error detected by checksum algorithm.   |  |
| B_EFW_DECODER_GRABBED        | Decoder <i>&lt;decoder&gt;</i> is not accessible because its base address register is used as the upper half of a 64-bit decoder.           |  |
| B_EFW_DECODER_NO_DAC         | Decoder <i>&lt;decoder&gt;</i> cannot decode DACs. Only odd numbered decoders have this capability.   |  |
| B_EFW_DEEPTTRACE_FUNC        | Deep Trace board may be defect!   |  |
| B_EFW_ERROR                  | Error during command transfer.  |  |
| B_EFW_EX_INIMODESET_FAILED   | Programming failed, because <i>&lt;reason&gt;</i> .   | The testcard was unable to switch to programming mode. |

| Error Code                      | Error string   | Notes/<br>Recommendations/Help  |
|---------------------------------|--|---|
| B_EFW_FASTDECODER_COMMAND_ERROR | Decoder <i>&lt;decoder&gt;</i> cannot be set to fast decoding speed because the command pattern <i>&lt;cmdpatt&gt;</i> is not supported at this speed. Please refer to the documentation for supported command sets.           | Only the following decoders are capable of fast decoding:<br>– Memory decoders,<br>– I/O decoders,<br>– Configuration (type 0) decoders.<br>Use appropriate commands. |
| B_EFW_FASTDECODER_GRABBED       | Decoder <i>&lt;decoder1&gt;</i> cannot be set to fast decoding speed because decoder <i>&lt;decoder2&gt;</i> is still set to fast speed.   |   |
| B_EFW_FUNC                      | Functional onboard error.  | Runtime error on the hardware. Try again. If the problem persists, reboot the card.   |
| B_EFW_ILLEGAL_PAGE_OFFSET       | <i>&lt;memory type&gt;</i> <i>&lt;page&gt;</i> has only <i>&lt;lines&gt;</i> lines to address. Requested offset is <i>&lt;offset&gt;</i> .   |   |
| B_EFW_IMAGE_TOO_LARGE           | The image is too large to be programmed into the ROM. Either the selected sector is too small or the RAM is too small to shadow the image.   | Error while performing firmware update.   |
| B_EFW_INVALID_ATTRPOINTER       | Block <i>&lt;block&gt;</i> of page <i>&lt;blockpage&gt;</i> contains a pointer to attribute page <i>&lt;attrpage&gt;</i> , which is either empty, non-existent or concatenated.  |   |
| B_EFW_INVALID_PREP_ATTRPOINTER  | Preparation register contains a pointer to attribute page <i>&lt;attrpage&gt;</i> , which is either empty, non-existent or concatenated.   |   |
| B_EFW_LIFELOCK_PREVENTION       | The last command failed to prevent the system from a lifelock situation. One possible reason is a data memory up/download when the card is connected by PCI at the same time. Please refer to the manual for more information. | Use data memory accesses with less than 120 bytes.  |
| B_EFW_MBLOCKCMD_DAC             | Master cannot issue the DAC command directly. Use B_BLK_BUSDAC property to enable Dual Address Cycles.   |   |
| B_EFW_MEN_NOT_SET               | Master cannot run, because the master enable bit is not set in config space.   |   |
| B_EFW_NO_CAPABILITY             | Your hardware is not capable of <i>&lt;capability&gt;</i> action.  |   |
| B_EFW_NO_EOP_FOUND              | Could not find an EndOfPage (EOP) block in page <i>&lt;blockpage&gt;</i> .   |   |
| B_EFW_OK                        | No error.  |   |
| B_EFW_OVERLAY_CMD_SUBSET        | The base decoder's <i>&lt;decoder1&gt;</i> decode commands have to be a superset of the overlaid decoder's <i>&lt;decoder2&gt;</i> range.  |   |
| B_EFW_OVERLAY_RANGE_SUBSET      | The base decoder's <i>&lt;decoder1&gt;</i> decode range has to be a superset of the overlaid decoder's <i>&lt;decoder2&gt;</i> range.  |   |
| B_EFW_PAGE_CONCATENATED         | <i>&lt;memory type&gt;</i> <i>&lt;page&gt;</i> is concatenated to its predecessor. Choose another page or initialize it first.   |   |

| Error Code                | Error string  | Notes/<br>Recommendations/Help   |
|---------------------------|---|--|
| B_EFW_PAGE_EMPTY          | <memory type> <page> is empty (initialized but not programmed yet).   |  |
| B_EFW_PARAM               | An internal parameter was out of range, unable to continue. Please check parameters of this call.   |  |
| B_EFW_PARAM_NOT_EXIST     | In group <group> parameter <parameter> does not exist. Either the parameter index is out of range or the capability for the parameter is not enabled. |  |
| B_EFW_PATT_MTO_LISTSIGNAL | OR-operation on different list types in pattern.  | Set the pattern string in such a way that signals of only one list type are OR-combined. |
| B_EFW_PATT_SYNTAX         | Syntax error in <pattern>.  | Syntax error in pattern string.<br>The error can be found left of the marker (^).        |
| B_EFW_PATT_UNDEF_TOKEN    | Undefined token found in <pattern>.   | The error can be found left of the marker (^).   |
| B_EFW_PATT_UNKNOWN_ERROR  | Error parsing <pattern>.  | The error can be found left of the marker (^).   |
| B_EFW_PCI_CLK_TOO_SLOW    | PCI clock too slow or not running.  |  |
| B_EFW_PCI_NO_POWER        | PCI bus is not powered.   |  |
| B_EFW_RANGE               | Parameter <parameter> has to be in a range from <min> through <max>. Its value is <value>.  |  |
| B_EFW_SEQ_BUNDLE_SIGNAL   | <sequencer>:<br>Cannot output this value (too big) in Line: <line>.   |  |
| B_EFW_SEQ_TRAN_EXCLUSIVE  | <sequencer>:<br>"Next State" Transitions conditions of state <line> not exclusive.  |  |
| B_EFW_SEQ_SYNTAX_ERR      | <sequencer>: Syntax Error in Line: <line>.  |  |
| B_EFW_SEQ_TOO_MANY_INPUTS | <sequencer>: Line <line>:<br>Too many Inputs (max is <maxinput> with <maxstates> states).   |  |
| B_EFW_SEQ_TOO_MANY_STATES | <sequencer>: Line <line>:<br>Too many states (max is <maxstates>).  |  |
| B_EFW_SEQ_UNKNOWN_OUTPUT  | <sequencer>:<br>Unknown output in Line: <line>  |  |

| Error Code              | Error string   | Notes/<br>Recommendations/Help   |
|-------------------------|--|--|
| B_EFW_SEQ_UNKNOWN_TOKEN | <sequencer>:<br>Unknown token in line: <line>  |  |
| B_EFW_SYNTAX            | Syntax error.  |  |
| B_EFW_TARGET_BUSY       | Programming failed, because the target is still active. Maybe the PCI clock is too slow or the target hangs. | The testcard was unable to switch to programming mode because the target is busy or the PCI clock is too slow. The PCI bus may be hanging. Resetting the statema-chines may solve the problem. |
| B_EFW_VALUE             | Parameter <parameter> cannot take value <value> in this context.   |  |
| B_EFW_WRONGPORT         | Command <command> has to be issued from port <port>.   |  |

## b\_exercisergenproptype

### CAUTION

Changing the page size property of the attribute memory deletes all attribute pages!

| exeprop<br>(CLI Abbreviation)         | value       | Description   |
|---------------------------------------|-------------|---|
| B_EGEN_ATTRPAGESIZE<br>(attrpagesize) | default: 32 | The attribute memory consists of 8 pages of <b>32 lines</b> each.                                     |
|                                       | 4           | The attribute memory consists of 64 pages of <b>4 lines</b> each.<br>This provides more entry points. |



# b\_mastergenproptype

| mastergenprop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation)  | Description   |
|-------------------------------------|--|---|
| B_MGEN_ATTRMODE<br>(attrmode)       | The attribute mode property defines when the master attribute line pointer should be reset to the start of a master attribute page.<br><br><b>Note:</b> The block attribute B_BLK_CONTATTR must be set accordingly for this property to have an effect. <i>"BestMasterBlockPropSet" on page 123</i> sets this attribute. |   |
|                                     | <b>default:</b><br>B_ATTRMODE_BLOCK<br>(block, 0)  | Pointer is reset to start after each block.   |
|                                     | B_ATTRMODE_SEQUENTIAL<br>(sequential, 1)   | Pointer is not reset. The exerciser continues with the attributes of the next attribute memory line.  |
|                                     | B_ATTRMODE_PAGE<br>(page, 2)   | Pointer is reset to start after each block page.  |
| B_MGEN_CACHELINESIZE<br>(clinesize) | 0 ... 0xFF<br><b>default:</b> 0xFF   | Sets the cacheline size in the register of the configuration space.   |
| B_MGEN_DELAYCTR<br>(delayctr)       | 32-bit value<br><b>default:</b> 0  | Number of PCI clocks that are added to the inevitable delay between the occurrence of the conditional start pattern and the start of the master.<br><br>The inevitable delay is caused by the testcard always inserting 4 clock cycles, and further delays if the bus is not granted to the master. |
| B_MGEN_INVDAT_ENABLE<br>(invdataen) | 0, 1<br><b>default:</b> 1  | When this bit is set, outgoing data bytes are inverted if they are masked by byte enables.  |
| B_MGEN_LATCTR<br>(latctr)           | 0 ... 255 clocks<br><b>default:</b> 0  | Sets the latency timer value of the testcard.   |
| B_MGEN_LATMODE<br>(latmode)         | The latency mode property determines whether the master ignores or adheres to its latency timer. Ignoring the latency timer allows you to test beyond the PCI specification.   |   |
|                                     | <b>default:</b><br>B_LATMODE_OFF (off)   | Ignores the latency timer.  |
|                                     | B_LATMODE_ON (on)  | Adheres to the latency timer.   |
| B_MGEN_MASTERENABLE<br>(men)        | 0, 1<br><b>default:</b> 0  | Sets the master enable bit in the command register of the testcard's configuration space.   |
| B_MGEN_MWIENABLE<br>(mwien)         | 0, 1<br><b>default:</b> 0  | Sets memory write and invalidate enable bit in the command register of the testcard's configuration space.  |
| B_MGEN_REPEATMODE<br>(repmode)      | <b>default:</b><br>B_REPEATMODE_SINGLE<br>(single, 1)  | Sets the master to perform a master run only once.  |
|                                     | B_REPEATMODE_INFINITE<br>(infinite, 0)   | Sets the master to repeat a master run until <i>"BestMasterStop" on page 106</i> is called.   |

| mastergenprop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation)  | Description   |
|-------------------------------------|--|---|
| B_MGEN_RUNMODE<br>(runmode)         | The run mode property determines whether the master should start immediately, after a delay, or after a trigger event has occurred.<br><br><b>Note:</b> This property has priority over the master block property for conditional start (see “ <i>b_blkproptype</i> ” on page 238, B_BLK_CONDSTART). |   |
|                                     | <b>default:</b><br>B_RUNMODE_IMMEDIATE,<br>(immediate, 0)  | Sets the master to start without waiting for the master start condition.                        |
|                                     | B_RUNMODE_WONDELAY,<br>(wondelay, 1)   | Sets the master to wait until the master trigger pattern occurs, and the delay counter expires. |
| B_MGEN_TRYBACK_ENABLE<br>(backen)   | 0, 1<br><b>default:</b> 0  | Sets the Fast Back-to-Back bit in the command register of the configuration space.              |
| B_MGEN_WARN64BIT<br>(warn64)        | If a 64-bit transfer moves an <i>odd</i> number of dwords to a memory block starting with an <i>even</i> address—and in the reverse case—a warning is issued because this can cause compare errors.<br><br>If the warning disturbs your test, use this property to suppress it.                      |   |
|                                     | <b>default:</b> 1  |   |
|                                     | 0  |   |

# b\_mattrgroupype

**NOTE** The master attributes marked with an asterisk must **not** be used with the PCI Permutator and Randomizer software.

| Master Attribute Group   | group<br>(CLI Abbreviation)  | Master Attributes in the Group<br>(CLI Abbreviation)  |
|--------------------------|------------------------------|---|
| Address Phase Attributes | B_MATTR_GRP_MA0<br>(magrpa0) | B_M_DELAY (delay)<br>B_M_DOLLOOP* (loop)  |
|                          | B_MATTR_GRP_MA1<br>(magrpa1) | B_M_DOLLOOP* (loop)<br>B_M_STEPS (steps)<br>B_M_TRYBACK (back)  |
|                          | B_MATTR_GRP_MA2<br>(magrpa2) | B_M_LOCK* (lock)<br>B_M_REQ64 (req64)<br>B_M_RELREQ (rreq)<br>B_M_DOLLOOP* (loop)   |
|                          | B_MATTR_GRP_MA3<br>(magrpa3) | B_M_DOLLOOP* (loop)<br>B_M_RESUMEDELAY (resdel)   |
|                          | B_MATTR_GRP_MA4<br>(magrpa4) | B_M_APERR (aperr)<br>B_M_AWRPAR (awp)<br>B_M_AWRPAR64 (awp64)<br>B_M_DACPERR (dacperr)<br>B_M_DACWRPAR (dacwp)<br>B_M_DACWRPAR64 (dacwp64)<br>B_M_DOLLOOP* (loop) |
| Data Phase Attributes    | B_MATTR_GRP_MD0<br>(magrpd0) | B_M_DOLLOOP* (loop)<br>B_M_WAITS (w)  |
|                          | B_MATTR_GRP_MD1<br>(magrpd1) | B_M_DOLLOOP* (loop)<br>B_M_DPERR (dperr)<br>B_M_DSERR (dserr)<br>B_M_DRELREQ* (drreq)   |
|                          | B_MATTR_GRP_MD2<br>(magrpd2) | B_M_DOLLOOP* (loop)<br>B_M_DWRPAR (dwp)<br>B_M_DWRPAR64 (dwp64)<br>B_M_MARKER* (mark)   |
| Control Attributes       | B_MATTR_GRP_ML<br>(magrpl)   | B_M_DOLLOOP* (loop)<br>B_M_LAST (last)<br>B_M_REPEAT* (repeat)  |

# b\_mattrproptype

The master attribute properties of type b\_mattrproptype are divided into the following sections:

- “Address Phase Attributes (Master)” on page 260,
- “Data Phase Attributes (Master)” on page 263,
- “Control Attributes (Master)” on page 265.

## Address Phase Attributes (Master)

**NOTE** The master attribute marked with an asterisk must **not** be used with the PCI Permutator and Randomizer software.

| mattrprop<br>(CLI Abbreviation) | value   | Description              |
|---------------------------------|---|--------------------------|
| B_M_APERR<br>(aperr)            | Asserts a system error (SERR#) in the address phase (SERR# is used to signal address parity errors). This property also sets the SERR flag in the configuration space and in the status register of the testcard.<br><br>System errors must be enabled in the configuration space for this property to have an effect.                  |                          |
|                                 | default: 0  | No error is signaled.    |
|                                 | 1   | Error is signaled.       |
| B_M_AWRPAR<br>(awp)             | A wrong parity is set one clock after the address phase.  |                          |
|                                 | default: 0  | Parity remains as it is. |
|                                 | 1   | Parity is inverted.      |
| B_M_AWRPAR64<br>(awp64)         | A wrong parity (PAR64) is set one clock after the address phase or in the first half of a dual address cycle.   |                          |
|                                 | default: 0  | Parity remains as it is. |
|                                 | 1   | Parity is inverted.      |
| B_M_DACPERR<br>(dACPerr)        | Asserts a system error (SERR#) in the second cycle of a dual address cycle (SERR# is used for address parity errors). This property also sets the SERR flag in the configuration space and in the status register of the testcard.<br><br>System errors must be enabled in the configuration space for this property to have an effect. |                          |
|                                 | default: 0  | No error is signaled.    |
|                                 | 1   | Error is signaled.       |
| B_M_DACWRPAR<br>(dacwp)         | A wrong parity signaled in the second cycle of a dual address cycle.  |                          |
|                                 | default: 0  | Parity remains as it is. |
|                                 | 1   | Parity is inverted.      |

| mattrprop<br>(CLI Abbreviation) | value  | Description  |
|---------------------------------|--|--|
| B_M_DACWRPAR64<br>(dacwp64)     | A wrong parity (PAR64) signaled in the second cycle of a dual address cycle.   |  |
|                                 | <b>default:</b> 0  | Parity remains as it is.   |
|                                 | 1  | Parity is inverted.  |
| B_M_DELAY<br>(delay)            | 0 ... 2 <sup>21</sup><br><b>default:</b> 0   | <p>Number of clocks a master transaction is delayed before its start. The transactions must reside in the same block, otherwise an additional gap of 15 clocks is inserted.</p> <p>The master stays idle only if B_M_TRYBACK (Fast Back-to-Back) is not activated.</p> <p><b>Note:</b> The master must also stay idle until the arbiter grants the bus access to the master. This can result in additional delay clocks.</p> |
| B_M_LOCK*<br>(lock)             | To achieve exclusive ownership of a target during an access, the master can lock the target with the LOCK# signal. This property controls whether or not an access is exclusive and should remain unchanged during the entire transaction, unless "hide_lock" is used. Exclusive access can only be established with a "read" command.   |  |
|                                 | B_LOCK_LOCK<br>(lock, 1)   | The master tries to achieve exclusive access to the target.  |
|                                 | B_LOCK_HIDELOCK<br>(hidelock, 3)   | <p>The exerciser will not release LOCK# after the address phase. This is useful to test the lock protocol of the target with one master only. The exerciser can pretend to be another master trying to access the locked target.</p> <p><b>Note:</b> This value is used only once. After a retry, the next attribute line is used.</p>   |
|                                 | B_LOCK_UNCHANGE<br>(unchange, 0)   | LOCK# state remains unchanged.   |
|                                 | <b>default:</b><br>B_LOCK_UNLOCK (unlock, 2)   | Releases LOCK# at the end of this transaction. The master will not try to achieve exclusive access to the target.  |
| B_M_RELREQ<br>(rreq)            | Allows you to deassert REQ# earlier than usual (usual means: one clock after the bus enters the idle state after a transfer).  |  |
|                                 | B_RELREQ_ON (rreqon, 0)  | REQ# is released directly after FRAME# is asserted.  |
|                                 | 1 ... 14   | Number of cycles after which REQ# is released after assertion of FRAME#.   |
|                                 | <b>default:</b><br>B_RELREQ_OFF (rreqoff, 15)  | The master releases REQ# as usual.   |
| B_M_REQ64<br>(req64)            | <p>The exerciser tries a 64-bit transfer and asserts REQ64# together with FRAME#. The address must be aligned to a qword boundary, otherwise REQ64# is not asserted.</p> <p>If an intended 64-bit data transfer has been denied once within a block, the exerciser assumes that this address range is not capable of handling 64-bit data accesses and will not try to transfer 64-bit data in this block again.</p> |  |
|                                 | <b>default:</b> 0  | No 64-bit access is tried.   |
|                                 | 1  | 64-bit access is tried.  |

| mattrprop<br>(CLI Abbreviation) | value  | Description  |
|---------------------------------|--|--|
| B_M_RESUMEDELAY<br>(resume)     | 2 ... 127<br><b>default:</b> 10  | <p>Number of clocks after which the master resumes after a target termination.</p> <p>The PCI Specification requires the master to deassert REQ# after a target termination. This parameter specifies the number of clock cycles after which the master reasserts REQ#. The purpose of this parameter is to give other masters a programmable chance to obtain the bus.</p> <p><b>Note:</b> If the exerciser has already been granted access to the bus (parking master), the master resumes after 2 clocks.</p> |
| B_M_STEPS<br>(steps)            | 0 ... 15<br><b>default:</b> 0  | Number of additional clocks during an address phase. They are added between assertion of GNT# and assertion of FRAME#.   |
| B_M_TRYBACK<br>(back)           | The exerciser can try a Fast Back-to-Back cycle if this is allowed by PCI specification in the current situation, and if this is enabled in the configuration space. |  |
|                                 | <b>Notes:</b> Only transactions within one block transfer can be executed with Fast Back-to-Back. The number of additional address steps (B_M_STEPS) must be 0.      |  |
|                                 | <b>default:</b> 0  | Does not try Fast Back-to-Back cycle.  |
|                                 | 1  | Tries Fast Back-to-Back cycle.   |

## Data Phase Attributes (Master)

**NOTE** The master attributes marked with an asterisk must **not** be used with the PCI Permutator and Randomizer software.

| mattrprop<br>(CLI Abbreviation) | value   | Description   |
|---------------------------------|---|---|
| B_M_DPERR<br>(dperr)            | Asserts a parity error (PERR#) two clocks after the read data transfer and sets the PERR flag in the configuration space and in the status register of the testcard. Parity errors must be enabled in the configuration space for this property to have an effect.<br><br><b>Note:</b> This attribute is ignored in write transfers. It is also ignored in read transfers when data comparison is enabled (B_BLK_COMPFLAG is 1 and B_BLK_BUSCMD is a “read” command; see “ <i>b_mattrproptype</i> ” on page 260). |   |
|                                 | <b>default:</b> 0   | No error is signaled.   |
|                                 | 1   | Error is signaled.  |
| B_M_DRELREQ*<br>(drreq)         | Allows you to deassert REQ# earlier than usual (usual means: one clock after the bus enters the idle state after the transfer).<br><br><b>Note:</b> This entry is ignored if the master has already been instructed to release REQ# (with B_M_RELREQ in the address phase, or with B_M_DRELREQ in a previous data phase within the same burst).   |   |
|                                 | B_DRELREQ_ON<br>(drreqon, 0)  | REQ# is released immediately after the start of the data phase. |
|                                 | 1 ... 30  | Number of cycles after the start of the data phase.             |
|                                 | <b>default:</b><br>B_DRELREQ_OFF (drreqoff, 31)   | The master releases REQ# as usual.                              |
| B_M_DSERR<br>(dserr)            | Asserts a system error (SERR#) in the data phase along with IRDY# and sets the SERR flag in the configuration space and in the status register of the testcard.<br><br>System errors must be enabled in the configuration space for this property to have an effect.  |   |
|                                 | <b>default:</b> 0   | No error is signaled.   |
|                                 | 1   | Error is signaled.  |
| B_M_DWRPAR<br>(dwp)             | A wrong parity (PAR) is set one clock after a write data transfer. This attribute is ignored in read transfers.   |   |
|                                 | <b>default:</b> 0   | Parity remains as it is.  |
|                                 | 1   | Parity is inverted.   |

| mattrprop<br>(CLI Abbreviation) | value   | Description  |
|---------------------------------|---|--|
| B_M_DWRPAR64<br>(dwp64)         | A wrong parity (PAR64) is set one clock after a write data transfer. This attribute is ignored in read transfers.   |  |
|                                 | <b>default:</b> 0   | Parity remains as it is.   |
|                                 | 1   | Parity is inverted.  |
| B_M_MARKER*<br>(marker)         | 0 ... 15<br><b>default:</b> 0   | Number issued during the address and data phase. It can be observed by pattern terms, for example, for synchronization.<br><br>By convention, the value "0" is used to indicate "nothing special". |
|                                 |   |  |
| B_M_WAITS<br>(w)                | Number of waits. This is controlled by the IRDY# behavior per data phase.<br><br><b>Note for 64-bit data transfers:</b> If the target does not accept an intended 64-bit data transfer, then the master asserts IRDY# at the earliest two clocks after the target has asserted DEVSEL#. In practice, this means no performance degradation, because single 64-bit data accesses are not recommended by the PCI Specification, because the master has to insert initial waits until the target accepts 64-bit data accesses. |  |
|                                 | B_M_WAITS_HANG<br>(whang, -1)   | Simulates a hanging master (IRDY# is not asserted).  |
|                                 | 0 ... 30<br><b>default:</b> 0   | Number of waits inserted.  |



## Control Attributes (Master)

**NOTE** The master attributes marked with an asterisk must **not** be used with the PCI Permutator and Randomizer software.

| mattrprop<br>(CLI Abbreviation) | value  | Description  |
|---------------------------------|--|--|
| B_M_DOLOOP*<br>(loop)           | Loop bit.<br><br>This bit indicates the last line in the attribute page. The next line executed after this one is again the first line of the same page.<br><br>Used with “ <i>BestMasterAttrGroupLineProg</i> ” on page 111, the loop bit is set for the corresponding master attribute <i>group</i> .<br><br>Used with “ <i>BestMasterAttrLineProg</i> ” on page 113, the <i>general loop bit</i> for the complete page is set (previously set group loop bits for this page are overwritten). |  |
|                                 | default: 0   | Continues with the next attribute line.  |
|                                 | 1  | Continues with the first line of the same page.  |
|                                 |  |  |
| B_M_LAST<br>(last)              | 1 ... $2^{32}$<br>default: 0   | Indicates the last phase of a burst.<br><br><b>Note:</b> When used in combination with the block command B_CMD_MEM_WRITEINVALIDATE, illegal burstlengths can be generated. |
|                                 |  |  |
| B_M_REPEAT*<br>(repeat)         | One attribute line can be performed repeatedly.  |  |
|                                 | 0  | Line used for all transfers.   |
|                                 | 1 ... $2^{32}$<br>default: 1   | Number of repetitions.   |

# b\_obsruletype

| obsrule<br>(CLI Abbreviation) | obsrule<br>(CLI Abbreviation) | obsrule<br>(CLI Abbreviation) |
|-------------------------------|-------------------------------|-------------------------------|
| B_R_FRAME_0 (frame_0)         | B_R_LOCK_1 (lock_1)           | B_R_SEM_0 (sem_0)             |
| B_R_FRAME_1 (frame_1)         | B_R_LOCK_2 (lock_2)           | B_R_SEM_1 (sem_1)             |
| B_R_IRDY_0 (irdy_0)           | B_R_CACHE_0 (cache_0)         | B_R_SEM_2 (sem_2)             |
| B_R_IRDY_1 (irdy_1)           | B_R_CACHE_1 (cache_1)         | B_R_SEM_3 (sem_3)             |
| B_R_IRDY_2 (irdy_2)           | B_R_PARITY_0 (par_0)          | B_R_SEM_4 (sem_4)             |
| B_R_IRDY_3 (irdy_3)           | B_R_PARITY_1 (par_1)          | B_R_SEM_5 (sem_5)             |
| B_R_IRDY_4 (irdy_4)           | B_R_PARITY_2 (par_2)          | B_R_SEM_6 (sem_6)             |
| B_R_DEVSEL_0 (devsel_0)       | B_R_W64_0 (w64_0)             | B_R_SEM_7 (sem_7)             |
| B_R_DEVSEL_1 (devsel_1)       | B_R_W64_1 (w64_1)             | B_R_SEM_8 (sem_8)             |
| B_R_DEVSEL_2 (devsel_2)       | B_R_W64_2 (w64_2)             | B_R_SEM_9 (sem_9)             |
| B_R_DEVSEL_3 (devsel_3)       | B_R_W64_3 (w64_3)             | B_R_SEM_10 (sem_10)           |
| B_R_TRDY_0 (trdy_0)           | B_R_ARB_0 (arb_0)             | B_R_SEM_11 (sem_11)           |
| B_R_TRDY_1 (trdy_1)           | B_R_ARB_1 (arb_1)             | B_R_SEM_12 (sem_12)           |
| B_R_TRDY_2 (trdy_2)           | B_R_ARB_2 (arb_2)             | B_R_SEM_13 (sem_13)           |
| B_R_STOP_0 (stop_0)           | B_R_PARITY_3 (par_3)          | B_R_LAT_0 (lat_0)             |
| B_R_STOP_1 (stop_1)           | B_R_PARITY_4 (par_4)          | B_R_LAT_1 (lat_1)             |
| B_R_STOP_2 (stop_2)           | B_R_PARITY_5 (par_5)          | B_R_IRDY_5 (irdy_5)           |
| B_R_LOCK_0 (lock_0)           | B_R_PARITY_6 (par_6)          |                               |

For detailed rule descriptions, see *Agilent E2927A PCI Analyzer User's Guide*.

# b\_obsstatustype

The following identifiers define the observer register to be read with “BestObsStatusGet” on page 49.

| obsstatus (CLI Abbreviation) | Description  |
|------------------------------|--|
| B_OBS_FIRSTERR (firsterr)    | First Error Register (bit 0 ... 31).<br>The returned values indicate the first error that has occurred after observer startup.                   |
| B_OBS_FIRSTERR2 (firsterr2)  | First Error Register (bit 32 ... 63).  |
| B_OBS_ACCUERR (accuerr)      | Accumulated Error Register (bit 0 ... 31).<br>The returned value indicates all protocol errors that have occurred since startup of the observer. |
| B_OBS_ACCUERR2 (accuerr2)    | Accumulated Error Register (bit 32 ... 63).  |
| B_OBS_OBSSTAT (obsstat)      | Observer Status Register.<br>The returned value indicates the observer status, see table below.  |

**Error Register** Each bit in the error registers represents one rule. If the rule has been masked, the bit value is undefined. For a list of rule identifiers refer to “b\_obsruletype” on page 266.

**Observer Status Register** The observer status register indicates the status of the observer.

| Bits    | Meaning                              |
|---------|--------------------------------------|
| 0       | 1= observer is currently in run mode |
| 1       | 1= observer out of sync              |
| 2       | 1= protocol error detected           |
| [31::3] | not used, returns 0                  |

## b\_perfgenproptype

| perfgenprop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation)                  | Description  |
|-----------------------------------|--|--|
| B_PERFGEN_CAMODE<br>(mode)        | <b>default:</b><br>B_CAMODE_INCR1 (incr1, 0) | Counter A is incremented by 1 after each count enable.   |
|                                   | B_CAMODE_INCRBYTEN<br>(incrbyten, 1)         | If the count enable is active, counter A is incremented by the number of byte enables set (C/BE3[3:0]) in order to count data. |
| B_PERFGEN_CTRC_PREL<br>(cprel)    | 0 ... $2^{32} - 1$<br><b>default:</b> 0      | Preload value for feedback counter C.  |

## b\_perfseqtrancondproptype

| perfseqtrancondprop<br>(CLI Abbreviation) | Default values of condition<br>(CLI Abbreviation) | Description  |
|---|---|--|
| B_PERFSEQ_CA_EN<br>(caen)                 | "0" (false)                                       | Condition to increment counter A (nominator counter).                                |
| B_PERFSEQ_CB_EN<br>(cben)                 | "0" (false)                                       | Condition to increment counter B (denominator counter).                              |
| B_PERFSEQ_CE<br>(ce)                      | "0" (false)                                       | Condition to decrement feedback counter C.   |
| B_PERFSEQ_CLOAD<br>(cload)                | "0" (false)                                       | Condition to preload feedback counter C.   |
| B_PERFSEQ_XCOND<br>(x)                    | "1" (true)  | Transition condition for a performance sequencer to move from one state to the next. |

## b\_perfseqtranproptype

| perfseqtranprop<br>(CLI Abbreviation) | value                         | Description   |
|---------------------------------------|-------------------------------|---|
| B_PERFSEQ_STATE<br>(state)            | 0 ... 63<br><b>default:</b> 0 | State identifier. The sequencer starts at state 0.                    |
| B_PERFSEQ_NEXTSTATE<br>(nextstate)    | 0 ... 63<br><b>default:</b> 0 | Identifier for the state after the transition condition has been met. |

# b\_porttype

| port            | portnum   | Description   |
|-----------------|---|---|
| B_PORT_FASTHIF  | 0 ... n   | <p>Specifies a connection via fast host interface card.</p> <p>The first fast host interface card to be found is assigned to "0", the second to "1", and so forth. The order in which the cards are found is not predictable. After the BestOpen call, use <i>"BestPing"</i> on page 22 and watch the LEDs to see which card is connected to the session.</p>   |
| B_PORT_OFFLINE  | <p>Result of:<br/><i>assumed capabilities</i><br/>OR<br/><i>assumed hardware</i></p> <p>Values for <i>assumed hardware</i> are:</p> <p>B_HW_E2925B,<br/>B_HW_E2925B_DEEP,<br/>B_HW_E2926A,<br/>B_HW_E2926A_DEEP,<br/>B_HW_E2926B,<br/>B_HW_E2926B_DEEP,<br/>B_HW_E2927A,<br/>B_HW_E2927A_DEEP,<br/>B_HW_E2928A,<br/>B_HW_E2928A_DEEP,<br/>B_HW_E2940A,<br/>B_HW_E2940A_DEEP</p> | <p>Specifies the Offline/Demo Mode.</p> <p>The Offline/Demo Mode allows you to call and try out functions during test development without hardware.</p> <p>In Offline/Demo Mode, the calls still perform most of the runtime range checking. The port number must be the result of the logical OR-combination of the assumed hardware and the assumed activated capability codes.</p> <p>The values for activated capabilities are returned by the BestCapabilityCheck call, see <i>"Capability Code Values"</i> on page 25.</p> <p><b>Example:</b></p> <p>(B_HW_E2926A_DEEP   B_CAPABILITY_64_BIT)</p> |
| B_PORT_PCI_CONF | 32 Bit  | <p>Device number of the testcard, as used by PCI BIOS and host bridge.</p> <p>This number may be requested using the function <i>"BestDevIdentifierGet"</i> on page 19 in a system with PCI BIOS. If the system does not have a PCI BIOS, then you must enter a specific system identifier to identify the testcard (see <i>"Device Identifier Format"</i> on page 20).</p>   |
| B_PORT_RS232    | B_PORT_COM1,<br>B_PORT_COM2,<br>B_PORT_COM3,<br>B_PORT_COM4   | Specifies a serial port.  |
| B_PORT_CURRENT  |   | Specifies the current session.  |

# b\_puproptype

| pu_prop (CLI Abbreviation)        | value (CLI Abbreviation)  | Description   |
|-----------------------------------|---|---|
| B_PU_MASTERRRUNMODE<br>(mrunmode) | 0 ... 15  | Provides the number of the block page the master executes after power up. |
|                                   | <b>default:</b> M_RUN_OFF   | The master does not start automatically after power up.                   |
| B_PU_TRCRUNMODE<br>(trcrunmode)   | <b>default:</b> 0   | The trace memory is started not automatically after power up.             |
|                                   | 1   | The trace memory runs after power up.                                     |
| B_PU_CONFRESTORE<br>(confrestore) | Behavior of the testcard's configuration space after power up or execution of " <i>BestAllPropStore</i> " on page 34 (this affects primarily the base address registers).                   |   |
|                                   | <b>default:</b> 0   | Read/write bits of the configuration space is set to 0.                   |
|                                   | 1   | Read/write bits of the configuration space are left as they are.          |
| B_PU_SSTRUNMODE<br>(sstrunmode)   | Used by the Agilent E2974A system stress test software. Do not directly set it to 1, but it may be set to 0 to disable a test which has already been set up using the stress test software. |   |
|                                   | <b>default:</b> 0   | No subsystem test is run at power up.                                     |
|                                   | 1   | An Agilent E2974A test is run at power up.                                |

## b\_resourcetype

| resource (CLI Abbreviation)    | Description          |
|--------------------------------|----------------------|
| B_RESLOCK_CPUPORT (cpuport)    | CPU port             |
| B_RESLOCK_EXERCISER (exe)      | Exerciser            |
| B_RESLOCK_MAILBOX (mailbox)    | Mailbox registers    |
| B_RESLOCK_OBSERVER (obs)       | Observer             |
| B_RESLOCK_PATT_TERM0 (pt0)     | Pattern terms        |
| ...                            |                      |
| B_RESLOCK_PATT_TERM23 (pt23)   |                      |
| B_RESLOCK_PERFORMANCE (perfor) | Performance measures |
| B_RESLOCK_STATIC_IO (staticio) | Static I/O           |
| B_RESLOCK_TRACEMEM (trc)       | Trace memory         |



## b\_signaltype (for Timing Check)

The following table shows the signals checked by the PCI Analyzer's timing checker to detect timing violations. The remaining signals are not checked due to different timing behavior.

| signal                                   | Timing Checker Syntax        | Description  |
|--|------------------------------|--|
| B_SIG_ACK64                              | ACK64                        | Target acknowledgment of a 64-bit transfer.                    |
| B_SIG_AD32                               | AD32                         | Lower 32 address and data bus lines.                           |
| B_SIG_AD64                               | AD64                         | Upper 32 address and data bus lines.                           |
| B_SIG_CBE3_0                             | CBE3_0                       | Lower 4 PCI command and byte enables sideband signals.         |
| B_SIG_CBE7_4                             | CBE7_4                       | Upper 4 PCI command and byte enables sideband signals.         |
| B_SIG_DEVSEL                             | DEVSEL                       | Device Select  |
| B_SIG_FRAME                              | FRAME                        | Cycle Frame.<br>Indicates start and duration of a transaction. |
| B_SIG_IDSEL                              | IDSEL                        | Own Initialization Device Select                               |
| B_SIG_IRDY                               | IRDY                         | Initiator (= Master) Ready                                     |
| B_SIG_LOCK                               | LOCK                         | Lock   |
| B_SIG_PAR                                | PAR                          | Parity Bit   |
| B_SIG_PAR64                              | PAR64                        | Parity Bit for AD[63::32]                                      |
| B_SIG_PERR                               | PERR                         | Parity Error   |
| B_SIG_REQ64                              | REQ64                        | Master request for a 64-bit transfer.                          |
| B_SIG_SBO                                | SBO                          | Snoop Back Off   |
| B_SIG_SDONE                              | SDONE                        | Snoop Done   |
| B_SIG_SERR                               | SERR                         | System Error   |
| B_SIG_STOP                               | STOP                         | Stop   |
| B_SIG_TRDY                               | TRDY                         | Target Ready   |
| B_SIG_trigger0<br>...<br>B_SIG_trigger11 | trigger0<br>...<br>trigger11 | External trigger lines of the testcard.                        |

## b\_signaltype (List of Signals)

The topics included in this section list bus signals, bus states, etc. that are stored in the trace memory and/or can be used in pattern terms. Any restrictions in the use of signals are stated explicitly.

The signals are grouped as follows:

- *“Bus Signals” on page 276*  
These signals are directly taken from the PCI bus lines.
- *“Bus States” on page 277*  
Bus states are internally generated by the testcard to provide easy trigger setup for the multiplexed PCI bus.
- *“Static I/O Signals” on page 280*  
The trigger and static I/O lines on the testcard.
- *“Transaction Attributes” on page 280*  
These signals can be used for triggering on the occurrence of certain transfer attributes, for example, commands
- *“Markers” on page 282*  
Markers are issued by the master or target on the testcard.
- *“Exerciser Signals” on page 283*  
Exerciser signals are derived from the states of master and target.
- *“Internal Counters” on page 283*  
Values of internal counters (stored in trace memory).
- *“Checks” on page 283*  
These signals can be used for triggering on the basis of results of data compare and detected timing or protocol errors.
- *“Gap Information” on page 284* (if gap mode is set to “performance”)  
Gap information stored in trace memory instead of samples due to selective recording by the storage qualification.

## Signal Types

The following table shows the types of signals, their capabilities and where they can be applied. The “type” column of the tables in the following topics shows the type of each signal.

| Criteria Type | Capability   | Application  |
|---------------|--|--|
| 10X           | For the individual bits, you can determine whether they are 1, or 0, or “don’t care” (“don’t care” = X).   | Single-bit signals, for example, FRAME# or IRDY#.  |
| 10X vector    | For multiple bit signals, you can determine for each bit, whether it is 1, or 0, or “don’t care” (“don’t care” = X).   | Bitwise signals, for example, byte enables.  |
| 10 vector     | For multiple bit signals, you can determine for each bit, whether it is 1, or 0.   | Bitwise signals, for example, some bus addresses.  |
| list          | For multiple bit signals that are encoded in multiple bit values, you can determine these values. The single bits have no meaning.   | For example, commands.<br><br>The elements of each list are numbered, for example, command “I/O Read” equals 2. Only the numeric values may be used. |
| range         | For multiple bit signals of this type, you can determine whether their value is within a certain range.<br><br>The following constraints can be set: <, >, <>, >=, <=, ==, don’t care. | Buses that represent a value, for example, address or counters.  |

## Bus Signals

These signals are directly taken from the PCI bus lines.

| signal                                   | Pattern and Trace Memory Syntax | Type       | Description   |
|--|---------------------------------|------------|---|
| B_SIG_ACK64                              | ACK64                           | 10X        | 64-Bit Acknowledge  |
| B_SIG_AD32                               | AD32                            | 10X vector | Lower 32 address and data bus lines.                        |
| B_SIG_AD64                               | AD64                            | 10X vector | Upper 32 address and data bus lines.                        |
| –  | addr_phase                      | 10X        | Indicates an address phase.                                 |
| B_SIG_CBE3_0                             | CBE3_0                          | 10X vector | Lower 4 PCI command and byte enables sideband signals.      |
| B_SIG_CBE7_4                             | CBE7_4                          | 10X vector | Upper 4 PCI command and byte enables sideband signals.      |
| B_SIG_DEVSEL                             | DEVSEL                          | 10X        | Device Select   |
| B_SIG_FRAME                              | FRAME                           | 10X        | Cycle Frame.<br>Indicates whether a transaction is running. |
| B_SIG_GNT                                | GNT                             | 10X        | Own Grant   |
| B_SIG_IDSEL                              | IDSEL                           | 10X        | Own Initialization Device Select                            |
| B_SIG_INTA<br>...<br>B_SIG_INTD          | INTA<br>...<br>INTD             | 10X        | Interrupt A ... D   |
| B_SIG_IRDY                               | IRDY                            | 10X        | Initiator (= Master) Ready                                  |
| B_SIG_LOCK                               | LOCK                            | 10X        | Lock  |
| B_SIG_PAR                                | PAR                             | 10X        | Parity Bit  |
| B_SIG_PAR64                              | PAR64                           | 10X        | Parity Bit for upper 32 address and data bus lines.         |
| B_SIG_PERR                               | PERR                            | 10X        | Parity Error  |
| B_SIG_REQ                                | REQ                             | 10X        | Own Bus Request   |
| B_SIG_REQ64                              | REQ64                           | 10X        | 64-bit Request  |
| B_SIG_RESET                              | RESET                           | 10X        | Reset   |
| B_SIG_SBO                                | SBO                             | 10X        | Snoop Back Off  |
| B_SIG_SDONE                              | SDONE                           | 10X        | Snoop Done  |
| B_SIG_SERR                               | SERR                            | 10X        | System Error  |
| B_SIG_STOP                               | STOP                            | 10X        | Stop  |
| B_SIG_TRDY                               | TRDY                            | 10X        | Target Ready  |
| B_SIG_trigger0<br>...<br>B_SIG_trigger11 | trigger0<br>...<br>trigger11    | 10X        | External trigger lines of the testcard.                     |

## Bus States

Bus states are detected and made available by the analyzer when observing protocol rules.

| signal        | Pattern and Trace Memory Syntax | Type | Meaning and Values  |
|---------------|---------------------------------|------|---|
| B_SIG_b_state | b_state                         | list | The current state of the bus as detected by the observer.<br><b>Note:</b> To identify the state use the <b>numeric values</b> .   |
|               |                                 |      | unsync = 0<br>After reset, this state is active until an idle state (FRAME# & IRDY#) is detected.<br>After a protocol violation, this state is used if no other state can be assumed.   |
|               |                                 |      | idle = 1<br>The bus is idle (FRAME# & IRDY#).   |
|               |                                 |      | dac1 = 2<br>The first half of a dual address cycle.   |
|               |                                 |      | addr = 3<br>Single address phase.   |
|               |                                 |      | dac2 = 4<br>The second half of a dual address cycle.  |
|               |                                 |      | decoding = 5<br>The address phase has been completed, however, a target has not yet claimed the access.   |
|               |                                 |      | wait = 6<br>A target has accepted the transfer (asserted DEVSEL#). The target, the master, or both are inserting wait cycles.<br>If IRDY# is deasserted, the waits are inserted by the master, if TRDY# is deasserted, they are inserted by the target. |
|               |                                 |      | transfer = 7<br>Currently a data transfer is taking place (both IRDY# and TRDY# are asserted).  |

| signal            | Pattern and Trace Memory Syntax | Type | Meaning and Values  |
|-------------------|---------------------------------|------|---|
| –                 | decode                          | list | Decode speed of the current transaction.<br><br>Once a decode speed has been determined, the information stays valid during the entire transaction. Master aborts are detected by the termination state.  |
|                   |                                 |      | dont_know = 0<br><br>A transaction has not been started or a started transaction has not yet been claimed by a target. This state is left when b_state is “wait” or “transfer”.   |
|                   |                                 |      | fast = 1<br><br>The current transaction has been decoded with fast decode speed.  |
|                   |                                 |      | medium = 2<br><br>The current transaction has been decoded with medium decode speed.  |
|                   |                                 |      | slow = 3<br><br>The current transaction has been decoded with slow decode speed.  |
|                   |                                 |      | subtractive = 4<br><br>The current transaction has been decoded with subtractive decode speed or one clock slower than “slow”.  |
|                   |                                 |      | too_slow = 5<br><br>The current transaction has been decoded slower than a subtractive decoder decodes.   |
| B_SIG_burst_order | burst_order                     | list | Indicates at which position of a burst a data phase was started by the master, independent of whether it is actually terminated earlier.<br><br>This information is only valid in the last clock of a data phase and can be used to count types of data phases with respect to its position within the burst.<br><br><b>Note:</b> A target termination with no data transfer and target limited waits is counted as one data phase. |
|                   |                                 |      | invalid = 0<br><br>This is not the last clock of a data phase.  |
|                   |                                 |      | single = 1<br><br>This data phase belongs to a burst of length one.   |
|                   |                                 |      | first = 2<br><br>This is the first data phase in a burst of a minimum of two data phases.   |
|                   |                                 |      | middle = 3<br><br>This is a data phase between first and last in a burst of minimum three data phases.  |
|                   |                                 |      | last = 4<br><br>This is the last data phase in a burst of a minimum of two data phases.   |

| signal     | Pattern and Trace Memory Syntax | Type | Meaning and Values  |
|------------|---------------------------------|------|---|
| B_SIG_term | term                            | list | Termination of a data phase.<br><br>It can be used to count the occurrence of certain terminations. A target termination with no data transfer and target limited waits is counted as one data phase.<br><br>This information is only valid in the last data phase of a transaction. Therefore, the information is valid only if the signal burst is valid simultaneously (any burst value except "invalid"). |
|            |                                 |      | no_term = 0      This data phase is not the end of a transaction.   |
|            |                                 |      | m_abort = 1      The transaction is terminated by the master with the master abort protocol.  |
|            |                                 |      | m_complete = 2      The master has concluded its intended transaction.<br><br>This may also be because its latency timer has expired and it has to conclude the transaction.  |
|            |                                 |      | t_disconnectA = 3      This transaction is terminated by the target. Data has been transferred successfully.  |
|            |                                 |      | t_disconnectB = 4      This transaction is terminated by the target. Data has been transferred successfully.  |
|            |                                 |      | t_disconnect1/t_retry1 = 5      This transaction is terminated by the target without any data transfers.<br><br>When burst is in the "single" or "first" state, this is a "retry", otherwise a "disconnect" (formerly "disconnect-C").  |
|            |                                 |      | t_disconnect2 / t_retry2 = 6      This transaction is terminated by the target without any data transfers.<br><br>When "burst" is in the "single" or "first" state, this is a "retry", otherwise a "disconnect" (formerly "disconnect-C").  |
|            |                                 |      | t_abort = 7      This transaction is terminated by the target using a target abort protocol.  |
| –          | berr                            | 10X  | Protocol violation detected.  |

## Static I/O Signals

The Static I/O signals can only be used in standard pattern terms.

| Signal                   | Type | Description |
|--------------------------|------|-------------|
| static_0 ...<br>static_7 | 10X  | Static I/O  |

## Transaction Attributes

Protocol attributes accompanying the transaction.

| signal         | Pattern and Trace<br>Memory Syntax | Type | Meaning and Values  |
|----------------|------------------------------------|------|---|
| –              | xact_fb2b                          | 10X  | Fast Back-to-Back.<br><br>This signal indicates whether the current transaction has started back-to-back to the previous one. Otherwise, the current transaction has started after at least one idle state.<br><br>The signal is valid between the address phase and the end of the transaction (while FRAME# or IRDY# is low). |
| B_SIG_xact_dac | xact_dac                           | 10X  | Dual Address Cycle.<br><br>This signal indicates that a current transaction addresses to a 64-bit address space. Otherwise, it addresses a 32-bit space.<br><br>The signal is valid between the address phase and the end of the transaction (while FRAME# or IRDY# is low).  |



| signal         | Pattern and Trace Memory Syntax | Type | Meaning and Values   |
|----------------|---------------------------------|------|--|
| B_SIG_xact_cmd | xact_cmd                        | list | Command used for the current transaction.  |
|                |                                 |      | This information is valid between the address phase and the end of the transaction (while FRAME# or IRDY# is low).       |
|                |                                 |      | In a dual address cycle it shows "DAC" in the first address cycle, and the bus command used in the second address cycle. |
|                |                                 |      | 0  |
|                |                                 |      | Interrupt Acknowledge  |
|                |                                 |      | 1  |
|                |                                 |      | Special Cycle  |
|                |                                 |      | 2  |
|                |                                 |      | I/O Read   |
|                |                                 |      | 3  |
|                |                                 |      | I/O Write  |
|                |                                 |      | 4  |
|                |                                 |      | Reserved   |
|                |                                 |      | 5  |
|                |                                 |      | Reserved   |
|                |                                 |      | 6  |
|                |                                 |      | Memory Read  |
|                |                                 |      | 7  |
|                |                                 |      | Memory Write   |
|                |                                 |      | 8  |
|                |                                 |      | Reserved   |
|                |                                 |      | 9  |
|                |                                 |      | Reserved   |
|                |                                 |      | A  |
|                |                                 |      | Configuration Read   |
|                |                                 |      | B  |
|                |                                 |      | Configuration Write  |
|                |                                 |      | C  |
|                |                                 |      | Memory Read Multiple   |
|                |                                 |      | D  |
|                |                                 |      | Dual Address Cycle   |
|                |                                 |      | E  |
|                |                                 |      | Memory Read Line   |
|                |                                 |      | F  |
|                |                                 |      | Memory Write & Invalidate  |

| signal                             | Pattern and Trace Memory Syntax  | Type | Meaning and Values   |                 |                  |                 |                       |                                    |                    |
|------------------------------------|----------------------------------|------|--|-----------------|------------------|-----------------|-----------------------|------------------------------------|--------------------|
| B_SIG_xact_burst                   | xact_burst                       | list | <p>Indicates the dword ordering of a burst to cacheable memory space for the current transaction. For details, refer to PCI Specification, “Addressing”.</p> <p>During accesses to memory space, this signal equals AD[1::0] in the address phase, otherwise it is set to “linear_incr”.</p> <p>This information is valid between the address phase and the end of the transaction (while FRAME# or IRDY# low).</p> <table><tr><td>linear_incr = 0</td><td>linear increment</td></tr><tr><td>wrap_around = 2</td><td>cacheline wrap around</td></tr><tr><td>reserved_01 = 1<br/>reserved_11 = 3</td><td>reserved encodings</td></tr></table> | linear_incr = 0 | linear increment | wrap_around = 2 | cacheline wrap around | reserved_01 = 1<br>reserved_11 = 3 | reserved encodings |
| linear_incr = 0                    | linear increment                 |      |  |                 |                  |                 |                       |                                    |                    |
| wrap_around = 2                    | cacheline wrap around            |      |  |                 |                  |                 |                       |                                    |                    |
| reserved_01 = 1<br>reserved_11 = 3 | reserved encodings               |      |  |                 |                  |                 |                       |                                    |                    |
| B_SIG_xact_req64                   | xact_req64                       | 10X  | <p>This signal indicates whether the master requested a 64-bit data transfer in the address phase.</p> <p>This information is valid between the address phase and the end of the transaction (while FRAME# or IRDY# low).</p>  |                 |                  |                 |                       |                                    |                    |
| –                                  | xact_lock                        | 10X  | <p>This signal indicates whether the current transaction is an exclusive access.</p> <p>This information is valid between the address phase and the end of the transaction (while FRAME# or IRDY# low).</p>  |                 |                  |                 |                       |                                    |                    |
| –                                  | xact_trig0<br>...<br>xact_trig11 | 10X  | <p>Shows the states of trigger 0 to 11 one clock before the most recent address phase.</p> <p>If the trigger inputs are connected to the GNT# signals of other devices, this signal can be used to identify the master of the current transaction.</p> <p>This information is valid during the whole transaction (while FRAME# or IRDY# are low).</p>  |                 |                  |                 |                       |                                    |                    |

## Markers

The markers can be used in standard pattern terms only.

| signal | Pattern Syntax | Type | Description   |
|--------|----------------|------|---|
| –      | t_marker[3::0] | 10X  | <p>Marker generated by the testcard's Exerciser target.</p> <p>This signal can be used to relate trigger outputs to target transactions. Aligned to data.</p> |
| –      | m_marker[3::0] | 10X  | <p>Marker generated by the testcard's Exerciser master.</p> <p>This signal can be used to relate trigger outputs to master transactions. Aligned to data.</p> |

## Exerciser Signals

Exerciser signals are derived from status of master and target.

| signal       | Pattern and Trace Memory Syntax | Type | Meaning   |
|--------------|---------------------------------|------|---|
| B_SIG_m_act  | m_act                           | 10X  | Master of the testcard is active.   |
| B_SIG_m_lock | m_lock                          | 10X  | Master of the testcard has established a lock.  |
| B_SIG_t_act  | t_act                           | 10X  | Target of the testcard is active.   |
| –            | master_done                     | 10X  | The intended master block page has been completed.<br><br>This signal can be used to inform the sequencer about completion.<br><br>A pattern output using this signal will be true 8 clocks after the master has completed the last data phase. |
| B_SIG_t_lock | t_lock                          | 10X  | Target of the testcard is locked.   |

## Internal Counters

Values of internal counters.

| Counter          | Pattern and Trace Memory Syntax | Type   | Meaning   |
|------------------|---------------------------------|--------|---|
| B_SIG_block_xfer | block_xfer                      | number | 8 bits. Identifies the currently executed block number.   |
| B_SIG_attr_ctr   | attr_ctr                        | number | 8 bits. Attribute counter. This is the <i>master</i> attribute counter whenever the master is involved in the stored transaction. Otherwise, it is the <i>target</i> attribute counter. |

## Checks

Results of data compare and detected protocol errors.

| signal          | Pattern and Trace Memory Syntax | Type | Meaning  |
|-----------------|---------------------------------|------|--|
| B_SIG_prot_rule | prot_rule                       | 10X  | Protocol violation has occurred.<br><br>The signal is aligned to bus signals.                |
| B_SIG_chck_data | chck_data                       | 10X  | Data discrepancy (miscompare) has occurred.<br><br>The signal is aligned to a data transfer. |
| –               | timing_err                      | 10X  | Timing error has occurred.   |

## Gap Information

Gap information stored instead of samples due to storage qualification.

| signal                               | Pattern and Trace Memory Syntax | Type      | Meaning   |
|--------------------------------------|---------------------------------|-----------|---|
| B_SIG_gap                            | gap                             | 10X       | <p>Gap flag, indicating that a gap precedes this sample due to storage qualification.</p> <p>The information stored in this memory line depends on the gap mode setting.</p> <p>If gap mode is set to "performance", information is as described in the <b>rows below in this table</b> (gap clocks, current address).</p> <p>If gap mode is set to "E2925A compatibility", information is as described in the <b>tables above</b> (signals, states, etc.).</p> <p>The gap mode is set by the "performance analyzer mode" property.</p> |
| B_SIG_gap_clocks                     | gap_clocks                      | number    | Number of clocks during which storage was suppressed.   |
| B_SIG_crr_addrlo<br>B_SIG_crr_addrhi | crr_addrlo<br>crr_addrhi        | 10 vector | <p>Current address (high and low 32 bits, the upper bits are available only with a 64-bit testcard) , that is the address of the data phase that follows the gap.</p> <p>This allows address information even if data phases only are sampled (address phases are suppressed by storage qualification).</p>   |

## b\_sizetype

| size (CLI Abbreviation) |
|-------------------------|
| B_SIZE_BYTE (1)         |
| B_SIZE_WORD (2)         |
| B_SIZE_DWORD (4)        |

## b\_staticproptype

| staticprop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation)                     | Description        |
|----------------------------------|---|--------------------|
| B_STAT_PINMODE<br>(pinmode)      | <b>default:</b><br>B_PMD_INPONLY<br>(inonly, 0) | Pin is input only. |
|                                  | B_PMD_TOTEMPOLE<br>(totempole, 1)               | Pin is totem-pole. |
|                                  | B_PMD_OPENDRAIN<br>(opendrain, 2)               | Pin is open-drain. |

## b\_systeminfotype

| infoprop (CLI Abbreviation) | value        | Description                 |
|-----------------------------|--------------|-----------------------------|
| B_SINFO_BUSWIDTH<br>(width) | 32, 64       | PCI bus width: 32 or 64 bit |
| B_SINFO_BUSSPEED<br>(speed) | 33000, 66000 | PCI bus speed in Hz         |

# b\_targetgenproptype

| targetgenprop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation)   | Description   |
|-------------------------------------|---|---|
| B_TGEN_BACKCAPABLE<br>(backcapable) | Sets the Fast Back-to-Back capability bit in the command register of the configuration space (bit 9).   |   |
|                                     | <b>default:</b> 0   | Disables Fast Back-to-Back.   |
|                                     | 1   | Enables Fast Back-to-Back.  |
| B_TGEN_IOSPACE<br>(iospace)         | Sets the I/O decoder enable bit in the command register of the configuration space (bit 0).<br>Normally the bit is set during the system configuration routine to enable the decoder after system initialization.                 |   |
|                                     | <b>default:</b> 0   | Disables I/O decoders.  |
|                                     | 1   | Enables I/O decoders.   |
| B_TGEN_MEMSPACE<br>(memspace)       | Sets the memory decoder enable bit in the command register of the configuration space (bit 1).<br>Normally the bit is set during the system configuration routine to enable the decoder after system initialization.              |   |
|                                     | <b>default:</b> 0   | Disables memory decoders.   |
|                                     | 1   | Enables memory decoders.  |
| B_TGEN_ROMENABLE<br>(romenable)     | Sets the Expansion ROM decoder enable bit in the expansion ROM register of the configuration space (bit 0).<br>Normally the bit is set during the system configuration routine to enable the decoder after system initialization. |   |
|                                     | <b>default:</b> 0   | Disables Expansion ROM.   |
|                                     | 1   | Enables Expansion ROM decoder.  |
| B_TGEN_RUNMODE<br>(runmode)         | B_RUNMODE_ADDRRESTART<br>(addrrestart, 0)   | Sets the target to restart from the beginning of the attribute page with every address phase.   |
|                                     | <b>default:</b><br>B_RUNMODE_SEQUENTIAL<br>(sequential, 1)  | Sets the target to loop the attribute page only at the end of the page, thus providing a kind of random attribute set with respect to the address phases. |

# b\_tattrgroupstype

**NOTE** The target attributes marked with an asterisk must **not** be used with the PCI Permutator and Randomizer software.

| Target Attribute Group   | group (CLI Abbreviation)  | Target Attributes in the Group (CLI Abbreviation)   |
|--------------------------|---------------------------|---|
| Address Phase Attributes | B_TATTR_GRP_TA0 (tagrpa0) | B_T_ACK64 (ack64)<br>B_T_APERR (aperr)<br>B_T_DACPERR (dacperr)<br>B_T_DOLOOP* (loop)                                     |
| Data Phase Attributes    | B_TATTR_GRP_TD0 (tagrpd0) | B_T_WAITS (w)<br>B_T_DOLOOP* (loop)   |
|                          | B_TATTR_GRP_TD1 (tagrpd1) | B_T_TERM (term)<br>B_T_DPERR (dperr)<br>B_T_DSERR (dserr)<br>B_T_WRPARR (wp)<br>B_T_WRPARR64 (wp64)<br>B_T_DOLOOP* (loop) |
|                          | B_TATTR_GRP_TD2 (tagrpd2) | B_T_MARKER* (mark)<br>B_T_DOLOOP* (loop)  |
| Control Attributes       | B_TATTR_GRP_TC (tagrpc)   | B_T_REPEAT* (repeat)<br>B_T_DOLOOP* (loop)  |

# b\_tattrproptype

The target attribute properties of type b\_tattrproptype are divided into the following sections:

- “Address Phase Attributes (Target)” on page 288
- “Data Phase Attributes (Target)” on page 289
- “Control Attributes (Target)” on page 291

## Address Phase Attributes (Target)

| tattrprop<br>(CLI Abbreviation) | value   | Description                           |
|---------------------------------|---|---------------------------------------|
| B_T_ACK64<br>(ack64)            | default: 0  | 64-bit requests are not acknowledged. |
|                                 | 1   | 64-bit requests are acknowledged.     |
| B_T_APERR<br>(aperr)            | Asserts a system error (SERR#) two clocks after the address phase (SERR# is used to signal address parity errors).                                  |                                       |
|                                 | This property also sets the SERR flag in the configuration space and in the status register of the testcard.  |                                       |
|                                 | System errors must be enabled in the configuration space for this property to have an effect.   |                                       |
|                                 | <b>Note:</b> If the subtractive decoder is enabled, SERR# is asserted, although the transaction can later be claimed by another device.             |                                       |
| B_T_DACPERR<br>(daccperr)       | default: 0  | No error is signaled.                 |
|                                 | 1   | Error is signaled.                    |
|                                 | Asserts a system error (SERR#) two clocks after the second cycle of a dual address cycle (SERR# is used to signal address parity errors).           |                                       |
|                                 | This property also sets the SERR flag in the configuration space and in the status register of the testcard.  |                                       |
|                                 | System errors must be enabled in the configuration space for this property to have an effect.   |                                       |
|                                 | <b>Note:</b> Be cautious if the subtractive decoder is enabled. SERR# is asserted, although the transaction can later be claimed by another device. |                                       |
|                                 | default: 0  | No error is signaled.                 |
|                                 | 1   | Error is signaled.                    |



## Data Phase Attributes (Target)

**NOTE** The target attributes marked with an asterisk must **not** be used with the PCI Permutator and Randomizer software.

| tattprop<br>(CLI Abbreviation) | value   | Description   |
|--------------------------------|---|---|
| B_T_DPERR<br>(dperr)           | Asserts a parity error (PERR#) two clocks after the write data transfer, along with TRDY# or STOP#.   |   |
|                                | This property also sets the PERR flag in the configuration space and in the status register of the testcard.  |   |
|                                | Parity errors must be enabled in the configuration space for this property to have an effect.   |   |
|                                | <b>Note:</b> This attribute is ignored both in read transfers and in write transfers when data comparison is enabled.   |   |
| B_T_DSERR<br>(dserr)           | default: 0  | No error is signaled.   |
|                                | 1   | Error is signaled.  |
|                                | Asserts a system error (SERR#) two clocks after the data phase and sets the SERR flag in the configuration space and in the status register of the testcard.                            |   |
| B_T_MARKER*<br>(marker)        | System errors must be enabled in the configuration space for this property to have an effect.   |   |
|                                | <b>Note:</b> If the subtractive decoder is enabled, SERR# is asserted, although the transaction can later be claimed by another device. That device asserts DEVSEL# and TRDY# or STOP#. |   |
|                                | 0 ... 15<br>default: 0  | Number issued during address and data phase. It can be observed by pattern terms, for example, for synchronization. |
|                                |   | By convention, the value "0" is used to indicate "nothing special".   |

| tattprop<br>(CLI Abbreviation) | value  | Description   |
|--------------------------------|--|---|
| B_T_TERM<br>(term)             | Specifies a termination applied after the number of clocks as specified by B_T_WAITS.                        |   |
|                                | B_T_NOTERM<br>(noterm)   | Data is accepted. No termination.   |
|                                | B_T_TERM_RETRY<br>(retry)  | Data is not accepted.<br>A disconnect-C or retry—depending on the time within the burst—is signaled.  |
|                                | B_T_DISCONNECT<br>(discon)   | Data is accepted.<br>A disconnect-A or disconnect-B—depending on the time within the burst—is signaled.   |
|                                | B_T_ABORT<br>(abort)   | A target abort is signaled.   |
| B_T_WAITS                      | Number of waits. This is controlled by the TRDY# behavior per data phase.                                    |   |
|                                | B_T_WAITS_HANG<br>(whang), -1  | Simulates a hanging target (TRDY# is not asserted).   |
|                                | 0 ... 30<br><b>default: 0</b>  | Number of waits inserted.<br><br>In the first data phase, this value refers to the number of clocks between the end of the address phase and the assertion of TRDY#. If prefetching was successful, 0 waits can be achieved, otherwise at least 7 waits are inserted. |
|                                | 31   | For compatibility reasons only. 30 waits are generated.   |
| B_T_WRPAR<br>(wp)              | A wrong parity is set one clock after a read transfer. This attribute is ignored in write transfers.         |   |
|                                | <b>default: 0</b>  | Parity remains as it is.  |
|                                | 1  | Parity is inverted.   |
| B_T_WRPAR64<br>(wp64)          | A wrong parity (PAR64) is set one clock after a read transfer. This attribute is ignored in write transfers. |   |
|                                | <b>default: 0</b>  | Parity remains as it is.  |
|                                | 1  | Parity is inverted.   |

## Control Attributes (Target)

**NOTE** The target attributes marked with an asterisk must **not** be used with the PCI Permutator and Randomizer software.

| tattrprop<br>(CLI Abbreviation) | value   | Description                                    |
|---------------------------------|---|--|
| B_T_DOLOOP*<br>(loop)           | Loop bit.<br><br>This bit indicates the last line in the attribute page. The next line executed after this one is again the first line of the same page.<br><br>Used with “ <i>BestTargetAttrGroupLineProg</i> ” on page 127, the loop bit is set for the referring target attribute group.<br><br>Used with “ <i>BestTargetAttrLineProg</i> ” on page 129, the <i>general loop bit</i> for the complete page is set (previously set group loop bits of this page are overwritten). |  |
|                                 | default: 0  | Continue with the next attribute line.         |
|                                 | 1   | Continue with the first line of the same page. |
| B_T_REPEAT*<br>(repeat)         | One attribute line can be performed repeatedly.   |  |
|                                 | 0   | This line is used for all transfers.           |
|                                 | 1 ... 2 <sup>32</sup><br>default: 1   | Number of repetitions.                         |

## b\_tcgenproptype

| tcgenprop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation) | Description   |
|---------------------------------|-----------------------------|---|
| B_TCGEN_SPEC<br>(spec)          | default: 1                  | PCI Specification determines the permissible values of setup and hold time (preparation register values are ignored). |
|                                 | 0                           | Preparation register values can be written to the testcard.   |

## b\_tcproptype

| tcprop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation)  | Description   |
|------------------------------|--|---|
| B_TC_SETUP_TIME<br>(setup)   | With a 29 – 35 MHz PCI bus:<br>5000 ... 9000<br><b>default:</b> 7000<br><br>With a 35 – 67 MHz PCI bus:<br>1000 ... 5000<br><b>default:</b> 3000 | Setup time $t_{su}$ in ps (picoseconds).<br><br>The value must be divisible by 250. |
| B_TC_HOLD_TIME<br>(hold)     | 0 ... 2000<br><b>default:</b> 0  | Absolute value of the hold time $t_h$ .<br><br>The value must be divisible by 250.  |
| B_TC_HSIGN<br>(hsign)        | 0, 1<br><b>default:</b> 0  | Sign of the hold time $t_h$ .<br>1 = negative hold time.                            |

## b\_tcstatustype

| status<br>(CLI Abbreviation) | value                   | Description  |
|------------------------------|-------------------------|--|
| B_TC_TCSTAT<br>(tcstat)      | B_TC_VIOLATION<br>(0x2) | Bit 1 indicates whether a timing violation has occurred.   |
|                              | B_TC_ERROR<br>(0x1)     | Bit 0 indicates whether the timing check works properly.<br><br><b>Note:</b> The current timing check results are invalid if this error flag is set. |

# b\_testproptype

| testprop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation)   | Description   |
|--------------------------------|---|---|
| B_TST_BANDWIDTH<br>(bw)        | 0 ... 100<br><b>default:</b> 50   | Requested bus bandwidth in percent. The test tries to occupy the bus with this bandwidth.<br><br><b>Note:</b> Bandwidth values of more than 95 % may slow down the device under test so much that it appears to be hanging. |
| B_TST_BLKLENGTH<br>(blklen)    | 1 ... 64k<br>(dword aligned)<br><b>default:</b> 8192  | Number of bytes for a data block. The tests transfer data blockwise.  |
| B_TST_COMPARE<br>(comp)        | 0, 1<br><b>default:</b> 0   | Enable/disable data compare between written and read data.  |
| B_TST_DATAPATTERN<br>(dpat)    | Sets a data pattern for the tests.<br>The patterns are generated during the test run.<br>The testcard's internal data memory is not used. |   |
|                                | <b>default:</b><br>B_DATAPATTERN_RANDOM<br>(dprando )   | Random data pattern.  |
|                                | B_DATAPATTERN_FIX<br>(dpfix)  | Data is always 00000000\h.  |
|                                | B_DATAPATTERN_TOGGLE<br>(dptoggle)  | Data toggles between 00000000\h and FFFFFFFF\h.   |
| B_TST_DESTINADDR<br>(dest)     | 32 Bit, dword aligned<br><b>default:</b> 00000000\h   | PCI address of source data. Used when moving blocks within PCI system.  |
| B_TST_NOBYTES<br>(nob)         | 32 Bit<br><b>default:</b> 4   | Number of transferred bytes.  |

| testprop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation)                         | Desfription   |
|--------------------------------|---|---|
| B_TST_PROTOCOL<br>(prot)       | Sets the protocol stress level.                     |   |
|                                | <b>default:</b><br>B_PROTOCOL_LITE<br>(lite)        | Stresses the system as little as possible.                                    |
|                                | B_PROTOCOL_MEDIUM<br>(medium)                       | Generates medium protocol stress.   |
|                                | B_PROTOCOL_HARD<br>(hard)                           | Generates maximum protocol stress.  |
| B_TST_STARTADDR<br>(start)     | 32 Bit, dword aligned<br><b>default:</b> 00000000\h | PCI address of the memory range to be accessed by a test (except block move). |
| B_TST_SOURCEADDR<br>(source)   | 32 Bit, dword aligned<br><b>default:</b> 00000000\h | PCI address of source data. Used when moving blocks within PCI system.        |

## b\_tracepattproptype

| tracepattprop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation) | Description        |
|-------------------------------------|-----------------------------|--------------------|
| B_PT_TRIGGER<br>(trig)              | "<pattern_string>"          | Trigger pattern.   |
| B_PT_SQ<br>(sq)                     | "<pattern_string>"          | Storage qualifier. |

**NOTE** After initialization, the properties may hold any value because the properties are not preset to default values. This can affect your measurement. Therefore, set both properties with *"BestTracePattPropSet"* on page 80.

To build pattern terms, refer to *"Pattern Term Identifiers"* on page 63.

# b\_traceproptype

| traceprop<br>(CLI Abbreviation)    | value<br>(CLI abbreviation)   | Description  |
|------------------------------------|---|--|
| B_TRC_HEARTBEATMODE<br>(hbmode)    | B_HBMODE_ON<br>(on, 1)  | Heartbeat trigger mode.  |
|                                    | <b>default:</b><br>B_HBMODE_OFF<br>(off, 0)   | Normal trigger mode.   |
| B_TRC_HEARTBEATVALUE<br>(hbvalue)  | 32 bits   | Heartbeat trigger value in PCI clocks.   |
| B_TRC_PERFANALYZER_MODE<br>(amode) | <b>default:</b><br>B_E2925_COMPATIBLE<br>(e2925, 0)   | Only the gap flag indicates a gap. Programs written for the trace memory of the Agilent E2925A run if this is activated.   |
|                                    | B_PERFORMANCE<br>(perf, 1)  | Exhaustive gap information is stored (normal gap mode).  |
| B_TRC_PATT0_MODE<br>(pt0mode)      | <b>default:</b><br>B_PATT0MODE_STANDARD<br>(standard, 1)  | Pattern term pt0 is used as a standard pattern term.   |
|                                    | B_PATT0MODE_DIFFERENTIAL<br>(differential, 1)   | Pattern term pt0 is used as a transitional (same as: differential) pattern term.   |
| B_TRC_TRIG_HISTORY<br>(trighist)   | 0 ... ffff0\h in steps of 4 ;<br><br>The default depends on the available trace memory space: it is set so that the trigger can be found in the center of the trace memory. | Trigger counter preload value defining how many lines are captured after a trigger event. Only multiples of 4 are allowed. |

## b\_tracestatustype

| tracestatus<br>(CLI Abbreviation) | Description   |
|-----------------------------------|---|
| B_TRC_STAT<br>(stat)              | Contents of the trace status register; see table below. |
| B_TRC_TRIGPOINT<br>(trig)         | Line number corresponding to the trigger event.         |
| B_TRC_LINESCAPT<br>(lines)        | Number of lines captured.                               |

**Trace Status Register** The following table shows the meanings of the single bits of the trace status register:

| Bit      | Meaning                                     | Description  |
|----------|---|--|
| 0        | 1 = Trace memory stopped.                   |  |
| 1        | 1 = Trigger occurred.                       |  |
| 2        | 0 = B_E2925_COMPATIBLE<br>1 = B_PERFORMANCE | Current mode of the performance analyzer.<br>See also<br>B_TRC_PERFANALYZER_MODE of<br><i>"b_traceproptype"</i> on page 295. |
| 3 ... 31 | Not used.                                   |  |



# b\_trigioseqgenproptype

| trigigenprop<br>(CLI Abbreviation)  | value<br>(CLI Abbreviation)  | Description                           |
|-------------------------------------|--|---------------------------------------|
| B_TRIGIOSEQGEN_OUT_0<br>(io0)       | Sets the output mode of trigger line 0 ... 11. (Input is always possible.) |                                       |
| ...                                 | B_TRIGIO_INONLY<br>(inonly)  | Disables the output.                  |
| B_TRIGIOSEQGEN_OUT_11<br>(io11)     | B_TRIGIO_TOTEMPOLE<br>(totempole)  | Sets the output to totem-pole.        |
|                                     | B_TRIGIO_OPENDRAIN<br>(opendrain)  | Sets the output to open-drain.        |
| B_TRIGIOSEQGEN_CTRC_PREL<br>(cprel) | 0 ... $2^{32} - 1$<br><b>default: 0</b>                                    | Preload value for feedback counter C. |

## b\_trigioseqtrancondproptype

| trigioseqgenprop<br>(CLI Abbreviation)                              | default value<br>(CLI Abbreviation) | Description  |
|---|-------------------------------------|--|
| B_TRIGIOSEQ_XCOND<br>(x)  | "1" (true)                          | Transition condition for the trigger I/O sequencer to move from one state to the next. |
| B_TRIGIOSEQ_OUT_0<br>(out0)<br>...<br>B_TRIGIOSEQ_OUT_11<br>(out11) | "0" (false)                         | Output condition on trigger I/O pins 0 ... 11.   |
| B_TRIGIOSEQ_CDEC (cdec)   | "0" (false)                         | Condition to decrement feedback counter C.   |
| B_TRIGIOSEQ_DDEC (ddec)   | "0" (false)                         | Condition to decrement feedback counter D  |
| B_TRIGIOSEQ_CINC (cinc)   | "0" (false)                         | Condition to increment feedback counter C  |
| B_TRIGIOSEQ_DINC (dinc)   | "0" (false)                         | Condition to increment feedback counter D  |
| B_TRIGIOSEQ_CLOAD<br>(cload)  | "0" (false)                         | Condition to preload feedback counter C.   |
| B_TRIGIOSEQ_DLOAD<br>(dload)  | "0" (false)                         | Condition to preload feedback counter D  |

## b\_trigioseqtranproptype

| trigioseqtranprop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation) | Description   |
|---|-----------------------------|---|
| B_TRIGIOSEQ_STATE<br>(state)            | 0 ... 63<br>default: 0      | State identifier. The sequencer starts at state 0.                    |
| B_TRIGIOSEQ_NEXTSTATE<br>(nextstate)    | 0 ... 63<br>default: 0      | Identifier for the state after the transition condition has been met. |

## b\_trigseqgenproptype

| trigseqgenprop<br>(CLI Abbreviation) | value<br>(CLI Abbreviation)      | Description                           |
|--------------------------------------|----------------------------------|---------------------------------------|
| B_TRIGSEQGEN_CTRL_CPREL<br>(cprel)   | 0 ... $2^{32} - 1$<br>default: 0 | Preload value for feedback counter C. |

## b\_trigseqtrancondproptype

| trigseqtrancondprop<br>(CLI Abbreviation) | Default values<br>(CLI Abbreviation) | Description   |
|---|--------------------------------------|---|
| B_TRIGSEQ_XCOND<br>(x)                    | "1" (true)                           | Transition condition for the trigger sequencer to move from one state to the next.                |
| B_TRIGSEQ_TRIGCOND<br>(trig)              | "0" (false)                          | Trigger condition.  |
| B_TRIGSEQ_SQCOND<br>(sq)                  | "0" (false)                          | Condition to store the current trace data line in the trace memory (Storage Qualifier Condition). |
| B_TRIGSEQ_CDEC (cdec)                     | "0" (false)                          | Condition to decrement feedback counter C.  |
| B_TRIGSEQ_DDEC (ddec)                     | "0" (false)                          | Condition to decrement feedback counter D   |
| B_TRIGSEQ_CINC (cinc)                     | "0" (false)                          | Condition to increment feedback counter C   |
| B_TRIGSEQ_DINC (dinc)                     | "0" (false)                          | Condition to increment feedback counter D   |
| B_TRIGSEQ_CLOAD<br>(cload)                | "0" (false)                          | Condition to preload feedback counter C.  |
| B_TRIGSEQ_DLOAD<br>(dload)                | "0" (false)                          | Condition to preload feedback counter D   |

## b\_trigseqtranproptype

| trigseqtranprop<br>(CLI Abbreviation) | value                  | Description   |
|---------------------------------------|------------------------|---|
| B_TRIGSEQ_STATE<br>(state)            | 0 ... 63<br>default: 0 | State identifier. The sequencer starts at state 0.                    |
| B_TRIGSEQ_NEXTSTATE<br>(nextstate)    | 0 ... 63<br>default: 0 | Identifier for the state after the transition condition has been met. |

## b\_versionproptype

| versionprop (CLI Abbreviation)        | Description   |
|---------------------------------------|---|
| B_VER_BOARD<br>(board)                | Returns the testcard ID (for example, B3835-0069).                |
| B_VER_CAPI<br>(capi)                  | Returns the C-API version.  |
| B_VER_CORE<br>(core)                  | Returns the date code of the onboard core BIOS.                   |
| B_VER_FIRMWARE<br>(firmware)          | Returns the version code of the onboard firmware.                 |
| B_VER_FIRMWARE_DATE<br>(firmwaredate) | Returns the version date of the onboard firmware.                 |
| B_VER_PRODUCT<br>(product)            | Returns the testcard product number (for example, E2927A).        |
| B_VER_SERIAL<br>(serial)              | Returns the serial number of the testcard hardware.               |
| B_VER_TEAM<br>(team)                  | Returns the members of the developer team.                        |
| B_VER_XILDATE<br>(xildate)            | Returns the date code of the Xilinx FPGA chain architecture file. |

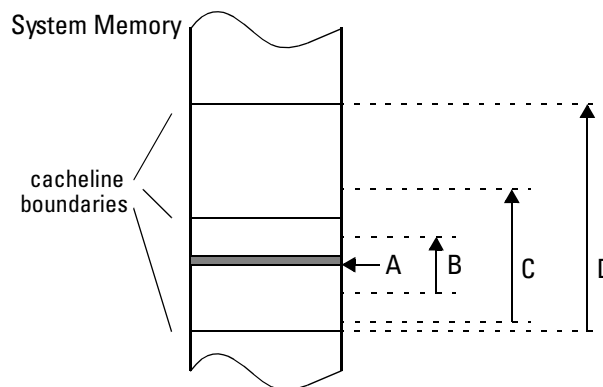
# bppr\_algorithmtype

| Values<br>(CLI Abbreviation)               | Description  |
|--|--|
| <b>default:</b><br>BPPR_ALG_PERM<br>(perm) | All values are picked one after the other as listed in the value list of the referring function.   |
| BPPR_ALG_RAND<br>(rand)                    | Values are picked at random from the value list. Using this algorithm cannot guarantee that <b>all</b> values are picked.  |
| BPPR_ALG_RECOMM<br>(recomm)                | Only for the block variation parameter B_BLK_CMDS.<br><br>The PCI bus commands from the value list will be picked one after the other, but the algorithm honors PCI Specification recommendations (see description below). |
| BPPR_ALG_BEST<br>(best)                    | Only for the block variation parameter B_BLK_CMDS.<br><br>Only best suitable PCI bus commands from the value list will be chosen for each permutation (see description below).   |

## bppr\_algorithmtype: B\_BLK\_CMDS Details

For the block variation parameters additional algorithms are available. Here the selection of parameters can be constrained to recommended or best suitable PCI bus commands.

When using PCI bus commands to perform memory write or read transfers, the following relationships between block start and end address and cacheline boundaries can occur and influence the commands usage:



| Scenario | Description   |
|----------|---|
| A        | Spans one dword only.   |
| B        | Spans multiple dwords, but range does not cross any cacheline boundary.   |
| C        | Spans multiple dwords, and crosses at least one cacheline boundary.       |
| D        | Spans multiple dwords. Start and end address are on a cacheline boundary. |

**Possible Commands** The following table shows, which commands the PCI PPR software can choose, depending on the specified algorithm and a specific scenario.

| Direction | Scenario     | Algorithm    |              |      |
|-----------|--------------|--------------|--------------|------|
|           |              | PERM, RAND   | RECOMM       | BEST |
| READ      | A            | MR, MRL, MRM | MR           | MR   |
|           | B            | MR, MRL, MRM | MR, MRL      | MRL  |
|           | C, D         | MR, MRL, MRM | MR, MRL, MRM | MRM  |
| WRITE     | all except D | MW           | MW           | MW   |
|           | D            | MW, MWI      | MW, MWI      | MWI  |

In this table the following abbreviations are used for the PCI memory commands—numbers in brackets are their decimal equivalents:

- MR – Read (6)
- MRL – Read Line (14)
- MRM – Read Multiple Lines (12)
- MW – Write (7)
- MWI – Write & Invalidate (15)

# bppr\_blkpermproptype

The following table lists the available block permutation properties and the appropriate value ranges. Values in these ranges can be used in the corresponding value lists.

| prop<br>(CLI Abbreviation)            | value/ranges   | Description  |
|---------------------------------------|--|--|
| BPPR_BLK_DIR<br>(dir)                 | B_DIR_WRITE  | Transfer from the testcard's internal to system memory.  |
|                                       | <b>default:</b> B_DIR_READ   | Transfer from system to the testcard's internal memory.  |
| BPPR_BLK_BUSADDR<br>(busaddr)         | 32-bit integer<br><b>default:</b> 0  | Start address on the bus for the compound block transfer.  |
| BPPR_BLK_BUSADDR_HI<br>(busaddrhi)    | 32-bit integer<br><b>default:</b> 0  | Upper 32 bits of a 64-bit bus address (requires BPPR_BLK_BUSDAC to be enabled).  |
| BPPR_BLK_BUSDAC<br>(busdac)           | Determines whether 32-bit or 64-bit addresses are transferred during an address phase. |  |
|                                       | <b>default:</b> 0  | Single address cycle (SAC) with 32-bit address.  |
|                                       | 1  | Dual address cycle (DAC) with 64-bit address. Uses BPPR_BLK_BUSADDR_HI.  |
| BPPR_BLK_INTADDR<br>(intaddr)         | 0 ... 0xFFFFC<br><b>default:</b> 0   | Start address in the testcard's internal data memory for the compound block transfer. Refer to " <i>b_blkproptype</i> " on page 238.   |
| BPPR_BLK_NOFDWORDS<br>(nod)           | 31-bit value<br><b>default:</b> 1  | Compound block size.<br>Specifies the total number of dwords to be transferred.  |
| BPPR_BLK_ATTRPAGE<br>(apage)          | 0 ... 255<br><b>default:</b> 1   | Specifies the master attribute page used for the compound block transfer (refer to " <i>b_blkproptype</i> " on page 238).<br><br>With each data phase in the compound block the lines of this attribute page are sequentially executed. To determine when the line pointer should be reset to the start of the attribute page, the property B_MGEN_ATTRMODE of " <i>b_mastergenproptype</i> " on page 257 can be used. |
| BPPR_BLK_COMPFLAG<br>(compflag)       | 0, 1<br><b>default:</b> 0  | Refer to " <i>b_blkproptype</i> " on page 238.   |
| BPPR_BLK_COMPOFFS<br>(compoffs)       | 0 ... 0xFFFFC<br><b>default:</b> 0   | Refer to " <i>b_blkproptype</i> " on page 238.   |
| BPPR_BLK_PAGENUM<br>(pagenum)         | 0 ... 15<br><b>default:</b> 0  | Block page number.   |
| BPPR_BLK_PAGESIZEMAX<br>(pagesizemax) | 1 ... 256<br><b>default:</b> 1   | Maximum page size, measured in blocks.   |



| prop<br>(CLI Abbreviation)        | value/ranges  | Description   |
|-----------------------------------|---|---|
| BPPR_BLK_FIRSTPERM<br>(firstperm) | 1 ... 10 <sup>9</sup><br><b>default:</b> 1  | Master block first permutation number.<br><br>Starts the permutation algorithm at the specific value. If not all desired permutations have been exercised previously, you can continue the permutation by setting this number to where the last permutation ended.<br>The number of the last permutation can be queried with BPPR_BLK_LASTPERM; see <i>"bppr_blkresultparamtype"</i> on page 306. |
| BPPR_BLK_CACHELINE<br>(cacheline) | <div>Informs the block permutator about the system's cacheline size, which is required to know which PCI bus commands are legal.</div> <div><b>default:</b><br/>BPPR_BLK_CACHELINE_NO</div>   | The system's cacheline size is either unknown or no cache exists.   |
|                                   | A power of 2 between 2 and 2048.  | Cacheline size, measured in dwords.   |
|                                   |   |   |
| BPPR_BLK_FILLGAPS<br>(fillgaps)   | Determines whether or not remaining gaps are filled after fitting block permutations into the compound block size.<br><br>Filling the gaps guarantees that the whole block is transferred. On the other hand, filling gaps may also require that address alignments or byte enable values are used that are (perhaps intentionally) not specified in <i>"BestPprBlockVariationSet"</i> on page 209. |   |
|                                   | <b>default:</b> 1   | All bytes are transferred. Unlisted variation values may be used.   |
|                                   | 0   | No unlisted variation values are used. Not all bytes may be transferred.  |

## bppr\_blkresultparamtype

| param<br>(CLI Abbreviation)           | Description  |
|---------------------------------------|--|
| BPPR_BLK_PAGESIZEACT<br>(pagesizeact) | Number of blocks that have actually been used to implement the compound block (that is the master block page size).  |
| BPPR_BLK_LASTPERM<br>(lastperm)       | Number of the last permutation of block variation parameters that did fit into the compound block. This number can be used to determine which permutations will be covered after complete execution of the compound block.   |
| BPPR_BLK_TIME<br>(time)               | Estimated testing time in $\mu$ s required to execute the compound block once. This does not include the initial programming time. To determine the testing time, the generic properties BPPR_GEN_BUSSPEED and BPPR_GEN_XFERCLKS of <i>"bppr_genproptype"</i> on page 310 are evaluated. |

# bppr\_blkvarparamtype

The following table lists the available block variation parameters and the corresponding value ranges. Values within these ranges can be used in the corresponding value lists. A syntax description for the entries of a value list can be found in “*Value Lists*” on page 308.

| param<br>(CLI Abbreviation) | value ranges for<br>entries in value_lists  | Description   |
|-----------------------------|---|---|
| BPPR_BLK_ALIGN<br>(align)   | <b>Granularity:</b><br>Power of 2 between cacheline size and 8192.<br><br><b>Offset:</b><br>Multiple of 4 between 0 and 8188.<br><br><b>default:</b> (%32=0)  | Restricts the start of a block to selected offsets using a given granularity. For example, “(%16=4)” means 4 bytes after a 16 byte boundary (address modulo 16 must be 4).<br><br>The generation algorithm will perform a check against the specified cacheline size.   |
| BPPR_BLK_BYTEN<br>(byten)   | 0 ... 15<br><b>default:</b> 0   | Numeric values for the C/BE lines in the address phase.<br><br>Be aware that gaps may occur if BPPR_BLK_FILLGAPS of “ <i>bppr_blkpermproptype</i> ” on page 304 is set to “0”.  |
| BPPR_BLK_CMDS<br>(cmds)     | int_ack<br>special<br>io_read<br>io_write<br>reserved_4<br>reserved_5<br><b>default:</b> mem_read<br><b>default:</b> mem_write<br>reserved_8<br>reserved_9<br>config_read<br>config_write<br>mem_readmultiple<br>mem_dac<br>mem_readline<br>mem_writeinvalidate | List of PCI bus commands to be used for variations. Only suitable values according to specified transfer direction (BPPR_BLK_DIR) will be used. Therefore, the default value also depends on “BPPR_BLK_DIR”. Refer to “ <i>bppr_blkpermproptype</i> ” on page 304.<br><br>Make sure, that you use extended read commands in prefetchable area only.<br><br>For more information on the usage of the PCI bus commands, refer to “ <i>bppr_algorithmtype: B_BLK_CMDS Details</i> ” on page 302. |
| BPPR_BLK_SIZE<br>(size)     | Multiple of 4 between 4 and 128k.<br><b>default:</b> 4  | Numeric value for a block size, measured in bytes.  |

## Value Lists

**Content** The parameter `value_list` holds the values to be used for permutation or randomization. They must always be suited to the corresponding variation parameter.

Values may be held in the list repeatedly:

- to determine how many times this value will appear in the generated tuples.

For example, if a value is held twice in the list, it will appear twice as much as values that appear once in the list.

- to generate an individual sequence of tuples, in which this value should appear repeatedly.

For this purpose the algorithm `BPPR_ALG_PERM` is used. See *"bppr\_algorithmtype"* on page 302.

**Syntax** `value_list` is a quoted string with a comma-separated list of values (non case-sensitive). The format of a valid value entry depends on the variation parameter.

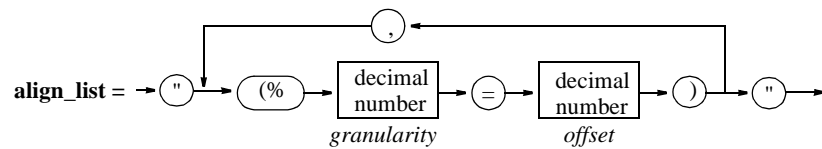
**Example:**

- `"io_read, mem_read, mem_readmultiple, mem_readline"`

There are two different kinds of values. Alignment values and numeric values.

## Alignment values

Alignment values must be given in brackets. They start with a percent sign followed by the granularity, an equality sign, and the offset. This syntax is shown in the following diagram:



**Example:**

- `"(%4=0)"`

Means a granularity of 4 and an offset of 0 bytes.

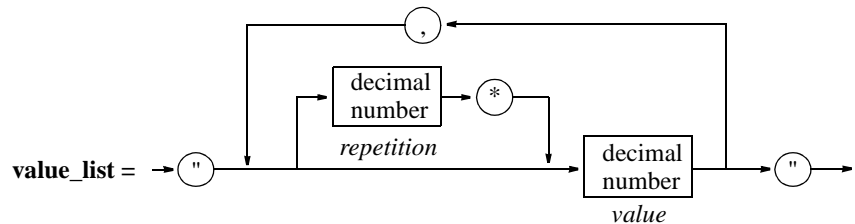
**NOTE** For more information on granularity, offset and value, see *"bppr\_blkvarparamtype"* on page 307.

## Numeric values

Numeric values can be commands, byte enables and block sizes. In this case the value list is a quoted string consisting of a comma-separated list of **symbols** or **numbers**, for example, "B\_RELREQ\_OFF, 2, 4, 6, 8".

Use given symbols instead of numbers whenever possible. If using numbers, they must be specified in decimal number format.

The syntax for value lists is shown in the following diagram—a repetition factor can be given to repeat a value in the list:



### Example:

- "99\*0,1"

Generates a list of 100 entries with 99 zeros and one 1. In one percent of all cases the attribute value is set to 1.

**NOTE** *Repetition* values increase the actual length of the list. This length must be less than 100 entries.

# bppr\_genproptype

| prop<br>(CLI Abbreviation)      | value                               | Description  |
|---------------------------------|-------------------------------------|--|
| BPPR_GEN_BUSSPEED<br>(busspeed) | 1 ... 33<br><b>default:</b> 33      | PCI bus speed in MHz. Used to calculate times from clock cycles.   |
| BPPR_GEN_BUSWIDTH<br>(buswidth) | 32, 64<br><b>default:</b> 32        | PCI bus width of the used system in bits.  |
| BPPR_GEN_SEED<br>(seed)         | 32-bit integer<br><b>default:</b> 0 | Start value for the generation of pseudo random number sequences used by the permutations algorithm. Use this to make "random" numbers reproducible. |
| BPPR_GEN_XFERCLKS<br>(xferclks) | 1 ... 1000<br><b>default:</b> 10    | Expected number of clocks per data transfer. Used to estimate testing times.   |

# bppr\_mattrpermproptype

| prop<br>(CLI Abbreviation)           | value                                      | Description  |
|--------------------------------------|--|--|
| BPPR_MA_FIRSTPERM<br>(firstperm)     | 1 ... 10 <sup>9</sup><br><b>default:</b> 1 | Master attribute first permutation number. Starts the permutation algorithm at the specific point. If not all desired permutations have been exercised previously, you can continue the permutation by setting this number to where the last permutation ended.<br><br>The last permutation can be queried from property BPPR_MA_LASTPERM. Refer to <i>"bppr_mattrresultparamtype" on page 312</i> . |
| BPPR_MA_PAGENUM<br>(pagenum)         | 1 ... 255<br><b>default:</b> 1             | Master attribute page number to be used for the permutation of attributes.   |
| BPPR_MA_PAGESIZEMAX<br>(pagesizemax) | 0 ... 255<br><b>default:</b> 1             | Maximum master attribute page size, measured in attribute lines. All desired master attribute permutations must fit into this page.  |
| BPPR_MA_TUPLES<br>(tuples)           | 0 ... 9<br><b>default:</b> 3               | Maximum number of groups that are permuted against each other for calculation of coverage.   |

# bppr\_mattrresultparamtype

| param<br>(CLI Abbreviation)          | value          | Description   |
|--------------------------------------|----------------|---|
| BPPR_MA_DATA<br>(data)               | 0 ... $2^{32}$ | Maximum amount of data in bytes that must be transferred to achieve complete coverage of master attribute variations. Depends on the buswidth. See BPPR_GEN_BUSWIDTH of type <i>"bppr_genproptype"</i> on page 310.   |
| BPPR_MA_LASTPERM<br>(lastperm)       | 0 ... $2^{32}$ | Number of the last permutation of master attributes in the allocated attribute page. This number can be used to determine which permutations will be covered after complete execution of the attribute page.  |
| BPPR_MA_PAGESIZEACT<br>(pagesizeact) | 0 ... 255      | Actually used attribute page size. This size is always below the value specified by BPPR_MA_PAGESIZEMAX.  |
| BPPR_MA_TIME<br>(time)               | 0 ... $2^{32}$ | Estimated testing time in $\mu$ s required to go through the complete attribute page. This does not include the initial programming time. To determine the testing time, the general properties BPPR_GEN_BUSSPEED and BPPR_GEN_XFERCLKS of <i>"bppr_genproptype"</i> on page 310 are evaluated. |
| BPPR_MA_TUPLES_DATA<br>(tuplesdata)  | 0 ... $2^{32}$ | Maximum amount of data in bytes that must be transferred to achieve complete coverage of all required group tuples. See BPPR_MA_TUPLES of type <i>"bppr_mattrpermproptype"</i> on page 311.   |
| BPPR_MA_TUPLES_TIME<br>(tuplestime)  | 0 ... $2^{32}$ | Estimated time in $\mu$ s required to transfer enough data to achieve complete coverage of all required group tuples. See BPPR_MA_TUPLES of type <i>"bppr_mattrpermproptype"</i> on page 311.   |
| BPPR_MA_RUNS<br>(runs)               | 0 ... $2^{32}$ | Number of block page runs needed for complete coverage if block variations are set to be permuted.  |
| BPPR_MA_TUPLES_RUNS<br>(tuplesruns)  | 0 ... $2^{32}$ | Number of block page runs needed for group tuple coverage if block variations are set to be permuted.   |



# bppr\_reportproptype

| prop<br>(CLI Abbreviation)           | value                                       | Description   |
|--------------------------------------|---|---|
| BPPR_REP_BLOCK<br>(block)            | 0, 1<br><b>default:</b> 1                   | Report of block permutation.  |
| BPPR_REP_MA<br>(ma)                  | 0, 1<br><b>default:</b> 1                   | Report of master attribute permutation.   |
| BPPR_REP_REPORT<br>(report)          | 0, 1<br><b>default:</b> 1                   | Report of report properties.  |
| BPPR_REP_TA<br>(ta)                  | 0, 1<br><b>default:</b> 1                   | Report of target attribute permutation.   |
| BPPR_REP_ORDER_TUPLE<br>(order)      | 1 ... 9<br><b>default:</b> 3                | Specifies how many elements of a tuple are used for coverage report (maximum tuple order). "1" means that the coverage for individual attributes is reported, "2" means that the coverage for pairs of attributes is reported, and so on. Valid for reporting of block, master and target permutations. |
| BPPR_REP_CAPI<br>(capi)              | 0, 1<br><b>default:</b> 1                   | Includes C-API abbreviations into the report.   |
| BPPR_REP_BLOCKCONTENT<br>(blockcont) | 0 ... 256<br><b>default:</b> 30             | Specifies the maximum number of lines of generated permutations to be reported.   |
| BPPR_REP_MACONTENT<br>(macont)       | 0 ... 2 <sup>32</sup><br><b>default:</b> 30 | Specifies the maximum number of master attribute page lines to be reported, starting with first page line.  |
| BPPR_REP_TACONTENT<br>(tacont)       | 0 ... 2 <sup>32</sup><br><b>default:</b> 30 | Specifies the maximum number of target attribute page lines to be reported, starting with the first page line.  |

# bppr\_tattrpermproptype

| prop<br>(CLI Abbreviation)           | value                                      | Description   |
|--------------------------------------|--|---|
| BPPR_TA_FIRSTPERM<br>(firstperm)     | 1 ... 10 <sup>9</sup><br><b>default:</b> 1 | Target attribute first permutation. Starts the permutation algorithm at the specific point. If not all desired permutations have been exercised previously, you can continue the permutation by setting this number to where the last permutation ended.<br><br>The last iteration can be queried from property BPPR_TA_LASTPERM. Refer to <i>"bppr_tattrresultparamtype" on page 315</i> . |
| BPPR_TA_PAGENUM<br>(pagenum)         | 1 ... 255<br><b>default:</b> 1             | Target attribute page number to be used for permutation of attributes.  |
| BPPR_TA_PAGESIZEMAX<br>(pagesizemax) | 0 ... 255<br><b>default:</b> 1             | Maximum target attribute page size, measured in attribute lines. All desired target attribute permutations must fit into this page.   |
| BPPR_TA_TUPLES<br>(tuples)           | 0 ... 5<br><b>default:</b> 3               | Maximum number of groups that are permuted against each other for calculation of coverage.  |

# bppr\_tattrresultparamtype

| param<br>(CLI Abbreviation)          | value          | Description   |
|--------------------------------------|----------------|---|
| BPPR_TA_DATA<br>(data)               | 0 ... $2^{32}$ | Maximum amount of data in bytes that must be transferred to achieve complete coverage of target attribute variations. Depends on the buswidth. See BPPR_GEN_BUSWIDTH of type <i>"bppr_genproptype" on page 310</i> .  |
| BPPR_TA_LASTPERM<br>(lastperm)       | 0 ... $2^{32}$ | Number of the last permutation of target attributes in the allocated attribute page. This number can be used to determine which permutations will be covered after complete execution of the attribute page.  |
| BPPR_TA_PAGESIZEACT<br>(pagesizeact) | 0 ... 255      | Actually used attribute page size. This size is always below the value specified by BPPR_TA_PAGESIZEMAX.  |
| BPPR_TA_TIME<br>(time)               | 0 ... $2^{32}$ | Estimated testing time in $\mu$ s required to go through the complete attribute page. This does not include the initial programming time. To determine the testing time, the general properties BPPR_GEN_BUSSPEED and BPPR_GEN_XFERCLKS of <i>"bppr_genproptype" on page 310</i> are evaluated. |
| BPPR_TA_TUPLES_DATA<br>(tuplesdata)  | 0 ... $2^{32}$ | Maximum amount of data in bytes that must be transferred to achieve complete coverage of all required group tuples. See BPPR_TA_TUPLES of type <i>"bppr_tattrpermproptype" on page 314</i> .  |
| BPPR_TA_TUPLES_TIME<br>(tuplestime)  | 0 ... $2^{32}$ | Estimated testing time in $\mu$ s required to transfer enough data to achieve complete coverage of all required group tuples. See BPPR_TA_TUPLES of type <i>"bppr_tattrpermproptype" on page 314</i> .  |



# Index

## A

Address Phase Attributes (Master) 260  
 Address Phase Attributes (Target) 288  
 alignment 307, 308  
 application  
   programming interface (C-API) 13

## B

bandwidth 293  
 base  
   address 244  
 baudrate 22  
 behavior  
   decoder property 244  
 block  
   move 167  
 board  
   properties 241  
 built-in test functions 162

## C

cacheline boundaries 302  
 cacheline size 257  
 capability 24  
 C-API 13  
 captured lines (in trace memory) 296  
 card  
   status register 38  
 compatibility  
   mode 295, 296  
 condition  
   for preloading the feedback counters 71  
   reference 72  
 configuration  
   space  
     programming Functions 142  
 connection 21  
 connection functions 17  
 Control Attributes (Master) 265  
 Control Attributes (Target) 291  
 counter 85  
   byte enable 268  
   identifier 86  
 CPU port  
   programming functions 169  
   properties 242

## D

data  
   memory  
     functions 150  
   pattern 293  
 Data Phase Attributes (Master) 263  
 Data Phase Attributes (Target) 289  
 decoding  
   speed 247  
 defaults 201  
 delay 257  
 denominator counter 86  
 device ID 19  
 display  
   functions 188  
 download 226

## E

Error Code 250

## F

fast  
   host interface 270  
 feedback counter 86  
 Firmware Errors 253  
 function reference 197  
 functions  
   BestAllPropDefaultLoad() 33  
   BestAllPropLoad() 34  
   BestAllPropStore() 34  
   BestAllResourceUnlock() 23  
   BestAnalyzerRun() 75  
   BestAnalyzerStop() 76  
   BestBoardPropGet() 35  
   BestBoardPropSet() 35  
   BestCapabilityCheck() 21  
   BestCapabilityRead() 26  
   BestClose() 22  
   BestConfigScan() 147  
   BestConfigScanPrint() 147  
   BestConfRegGet() 143  
   BestConfRegMaskGet() 145  
   BestConfRegMaskSet() 146  
   BestConfRegSet() 144  
   BestCPUportIntrClear() 172  
   BestCPUportIntrStatusGet() 172  
   BestCPUportPropSet() 173  
   BestCPUportRead() 174  
   BestCPUportRST() 175  
   BestCPUportWordBlockRead() 170  
   BestCPUportWordBlockWrite() 171  
   BestCPUportWrite() 176

BestDataMemInit() 151  
 BestDataMemRead() 151  
 BestDataMemWrite() 152  
 BestDevIdentifierGet() 18  
 BestDisplayPropSet() 188  
 BestDisplayWrite() 189  
 BestErrStringGet() 235  
 BestExerciserGenPropGet() 99  
 BestExerciserGenPropSet() 100  
 BestExpRomByteRead() 149  
 BestExpRomByteWrite() 150  
 BestHostMemDump64() 157  
 BestHostMemFill64() 159  
 BestHostPCIRegGet() 154  
 BestHostPCIRegSet 155  
 BestHostSysMemAccessPrepare() 156  
 BestInterruptGenerate() 161  
 BestLastErrGet() 234  
 BestLastErrStringGet() 234  
 BestMailboxReceiveRegRead() 190  
 BestMailboxSendRegWrite() 191  
 BestMasterAttrGroupLineProg() 111  
 BestMasterAttrGroupLineRead() 112  
 BestMasterAttrLineProg() 113  
 BestMasterAttrLineRead() 114  
 BestMasterAttrPageInit() 115  
 BestMasterAttrPropDefaultSet() 116  
 BestMasterAttrPropGet() 116  
 BestMasterAttrPropSet() 117  
 BestMasterBlockEndLineProg() 118  
 BestMasterBlockLineProg() 119  
 BestMasterBlockLineRead() 120  
 BestMasterBlockPageInit() 121  
 BestMasterBlockPageRun() 122  
 BestMasterBlockPropDefaultSet() 122  
 BestMasterBlockPropGet() 122  
 BestMasterBlockPropSet() 123  
 BestMasterBlockRun() 102  
 BestMasterByteEnableProg() 124  
 BestMasterByteEnableRead() 124  
 BestMasterCondStartPattSet() 106  
 BestMasterGenPropDefaultSet() 125  
 BestMasterGenPropGet() 104  
 BestMasterGenPropSet() 105  
 BestMasterStop() 106  
 BestObsBitPositionFind() 43  
 BestObsErrStringGet() 45  
 BestObsMaskGet() 46  
 BestObsMaskSet() 47  
 BestObsPropDefaultSet() 48  
 BestObsRuleErrTypeGet() 48  
 BestObsStatusClear() 49  
 BestObsStatusGet() 49  
 BestOpen() 21  
 BestPattSet() 62  
 BestPCICfgMailboxReceiveRegRead() 192

BestPCICfgMailboxSendRegWrite() 193  
 BestPCIconfigCheck() 148  
 BestPerfCtrRead() 85  
 BestPerfGenPropDefaultSet() 86  
 BestPerfGenPropGet() 87  
 BestPerfGenPropSet() 88  
 BestPerfRun() 89  
 BestPerfSeqProg() 89  
 BestPerfSeqPropDefaultSet() 90  
 BestPerfSeqTranCondPropSet() 91  
 BestPerfSeqTranPropDefaultSet() 92  
 BestPerfSeqTranPropSet() 93  
 BestPerfStatusGet() 94  
 BestPerfStop() 95, 187  
 BestPerfUpdate() 95  
 BestPing() 22  
 BestPMERead() 194  
 BestPMEWrite() 195  
 BestPowerUpPropGet() 36  
 BestPowerUpPropSet() 37  
 BestPprBlockCoverageGet 203  
 BestPprBlockGenerate 204  
 BestPprBlockInit 204  
 BestPprBlockPermPropDefaultSet 205  
 BestPprBlockPermPropGet 205  
 BestPprBlockPermPropSet 206  
 BestPprBlockResultGet 207  
 BestPprBlockVariationDefaultSet 207  
 BestPprBlockVariationGet 208  
 BestPprBlockVariationSet 209  
 BestPprDelete 198  
 BestPprGenPropDefaultSet 198  
 BestPprGenPropGet 199  
 BestPprGenPropSet 200  
 BestPprInit 201  
 BestPprMAttrCoverageGet 211  
 BestPprMAttrGenerate 212  
 BestPprMAttrInit 212  
 BestPprMAttrPermPropDefaultSet 213  
 BestPprMAttrPermPropGet 213  
 BestPprMAttrPermPropSet 214  
 BestPprMAttrResultGet 215  
 BestPprMAttrVariationDefaultSet 215  
 BestPprMAttrVariationGet 216  
 BestPprMAttrVariationSet 217  
 BestPprReportDelete 218  
 BestPprReportFile 219  
 BestPprReportPropDefaultSet 219  
 BestPprReportPropGet 220  
 BestPprReportPropSet 221  
 BestPprReportWrite 222  
 BestPprTAttrCoverageGet 225  
 BestPprTAttrGenerate 226  
 BestPprTAttrInit 226  
 BestPprTAttrPermPropDefaultSet 227  
 BestPprTAttrPermPropGet 227  
 BestPprTAttrPermPropSet 228  
 BestPprTAttrResultGet 229  
 BestPprTAttrVariationDefaultSet 229  
 BestPprTAttrVariationGet 230  
 BestPprTAttrVariationSet 231  
 BestResourceIsLocked() 23

BestResourceLock() 29  
 BestResourceUnlock() 29  
 BestRS232BaudRateSet() 22  
 BestSMReset() 37  
 BestStaticPinWrite() 177  
 BestStaticPropSet() 179  
 BestStaticRead() 180  
 BestStaticWrite() 180  
 BestStatusRegClear() 38  
 BestStatusRegGet() 39  
 BestSystemInfoGet() 31  
 BestTargetAttrGroupLineRead() 128  
 BestTargetAttributeGroupLineProg() 127  
 BestTargetAttributeLineProg() 129  
 BestTargetAttributeLineRead() 130  
 BestTargetAttrPageInit() 131  
 BestTargetAttrPageSelect() 132  
 BestTargetAttrPropDefaultSet() 132  
 BestTargetAttrPropGet() 133  
 BestTargetAttrPropSet() 134  
 BestTargetDecoderPowerUpProg() 135  
 BestTargetDecoderPowerUpRead() 136  
 BestTargetDecoderProg() 137  
 BestTargetDecoderPropGet() 139  
 BestTargetDecoderPropSet() 140  
 BestTargetDecoderRead() 141  
 BestTargetGenPropDefaultSet() 142  
 BestTargetGenPropGet() 107  
 BestTargetGenPropSet() 108  
 BestTestPropDefaultSet() 162  
 BestTestPropSet() 163  
 BestTestProtErrDetect() 164  
 BestTestResultDump() 165  
 BestTestRun() 166  
 BestTimCheckDefaultSet() 51  
 BestTimCheckGenPropGet() 52  
 BestTimCheckGenPropSet() 53  
 BestTimCheckMaskGet() 54  
 BestTimCheckMaskSet() 55  
 BestTimCheckProg() 56  
 BestTimCheckPropGet() 57  
 BestTimCheckPropSet() 58  
 BestTimCheckRead() 59  
 BestTimCheckResultGet() 59  
 BestTimCheckStatusClear() 60  
 BestTimCheckStatusGet() 61  
 BestTraceBitPosGet() 77  
 BestTraceBytePerLineGet() 78  
 BestTraceDataGet() 79  
 BestTracePattPropSet() 80  
 BestTracePropSet() 81  
 BestTraceRun() 82  
 BestTraceStatusGet() 82  
 BestTraceStop() 83  
 BestTrigIOGenPropDefaultSet() 182  
 BestTrigIOGenPropGet() 182  
 BestTrigIOGenPropSet() 183  
 BestTrigIORun() 184  
 BestTrigIOSeqProg() 183  
 BestTrigIOSeqPropDefaultSet() 184  
 BestTrigIOSeqTranCondPropSet() 185

BestTrigIOSeqTranPropDefaultSet() 186  
 BestTrigIOSeqTranPropSet() 187  
 BestTrigSeqGenPropDefaultSet() 68  
 BestTrigSeqGenPropGet() 68  
 BestTrigSeqGenPropSet() 69  
 BestTrigSeqProg() 69  
 BestTrigSeqPropDefaultSet 70  
 BestTrigSeqTranCondPropSet() 71  
 BestTrigSeqTranPropDefaultSet() 73  
 BestTrigSeqTranPropSet() 74  
 BestVersionGet() 31

## G

generating a report 222  
 generic  
   exerciser properties 98  
   performance properties 88  
 granularity 308

## H

heartbeat trigger 295  
 hints in the report 223  
 host access functions 153

## I

increment (de-) nominator/feedback counter 268  
 initialization 17  
 interrupt  
   generation 161

## L

lines captured (in trace memory) 296  
 location 248  
 logical operators  
   standard pattern terms 64  
   transitional pattern term 66

## M

mailbox  
   functions 189  
 Markers  
   Pattern Terms 282  
 mask 47, 246

master  
   attribute  
     programming functions 113  
     properties 260  
 block  
   page  
     run 101  
   properties 238  
   run 124  
 conditional start 106  
 enable bit 257  
 groups 111  
 programming functions 102

## N

next state 269, 299, 300  
 nominator counter 86

## O

observer  
   programming 42  
   rule type 266  
   status 267  
 offset 308

## P

pattern term  
   function 62  
 performance  
   counter 85  
   generic properties 88  
   measure  
     functions 84  
 port 21  
 power management event  
   functions 194  
 power-up  
   properties 271  
 prefetch 248  
 preload  
   performance counter 86  
   trigger counter 295  
   trigger I/O counter 298, 300  
 programming  
   transitional pattern term 295  
 properties/values  
   B\_ASP\_CONFIG 237  
   B\_ASP\_CONFIG\_TYPE1 237  
   B\_ASP\_IO 237  
   B\_ASP\_MEM 237  
   B\_BLK\_ATTRPAGE 238  
   B\_BLK\_BUSADDR 238  
   B\_BLK\_BUSADDR\_HI 238  
   B\_BLK\_BUSCMD 238, 239  
   B\_BLK\_BUSDAC 239  
   B\_BLK\_BYTEN 239  
   B\_BLK\_BYTEN\_VAR 239  
   B\_BLK\_CMDS 302

B\_BLK\_COMPFLAG 239  
 B\_BLK\_COMPOFFS 239  
 B\_BLK\_CONDSTART 240  
 B\_BLK\_CONTATTR 240  
 B\_BLK\_INTADDR 240  
 B\_BOARD\_PERREN 241  
 B\_BOARD\_RSTMODE 241  
 B\_BOARD\_SERREN 241  
 B\_CAMODE\_INCR1 268  
 B\_CAMODE\_INCRBYTEN 268  
 B\_CAPABILITY\_64\_BIT 25  
 B\_CAPABILITY\_66\_MHZ\_AN 25  
 B\_CAPABILITY\_66\_MHZ\_EX 25  
 B\_CAPABILITY\_ALL 25  
 B\_CAPABILITY\_ANALYZER 25  
 B\_CAPABILITY\_EXERCISER 25  
 B\_CAPABILITY\_HOSTINT 25  
 B\_CAPABILITY\_NO 25  
 B\_CAPABILITY\_PERFSEQ 26  
 B\_CAPABILITY\_TRACEDEPTH 25  
 B\_CM\_DISABLED 242  
 B\_CM\_MASTER 242  
 B\_CMDBIT\_INT\_ACK 245  
 B\_CMDBIT\_IO\_READ 245  
 B\_CMDBIT\_IO\_WRITE 245  
 B\_CMDBIT\_MEM 245  
 B\_CMDBIT\_MEM\_READ 245  
 B\_CMDBIT\_SPECIAL 245  
 B\_CP\_INTEL 242  
 B\_CPU\_MODE 242  
 B\_CPU\_PROTOCOL 242  
 B\_CPU\_RDYTYPE 242  
 B\_CR\_AUTO 242  
 B\_CR\_EXTERNAL 242  
 B\_DAC\_BOTH 246  
 B\_DAC\_NO 246  
 B\_DAC\_YES 246  
 B\_DATAPATTERN\_FIX 293  
 B\_DATAPATTERN\_RANDOM 293  
 B\_DATAPATTERN\_TOGGLE 293  
 B\_DEC\_BASEADDR 244  
 B\_DEC\_BASEADDR\_HI 244  
 B\_DEC\_BASEDEC 244  
 B\_DEC\_BEHAVIOR 244  
 B\_DEC\_BUSCMD 245  
 B\_DEC\_CONFIG 243  
 B\_DEC\_DAC 246  
 B\_DEC\_EXPROM 243  
 B\_DEC\_FULL\_CONFIG 243  
 B\_DEC\_IDSEL 246  
 B\_DEC\_LOCATION 248  
 B\_DEC\_MASK 246  
 B\_DEC\_MASK\_HI 246  
 B\_DEC\_PREFETCH 248  
 B\_DEC\_PRIMARY\_BUS 246  
 B\_DEC\_RESBASE 249  
 B\_DEC\_RESOURCE 249  
 B\_DEC\_RESSIZE 249  
 B\_DEC\_SECONDARY\_BUS 246  
 B\_DEC\_SIZE 247  
 B\_DEC\_SPEED 247  
 B\_DEC\_STANDARD\_1 243

B\_DEC\_STANDARD\_2 243  
 B\_DEC\_STANDARD\_3 243  
 B\_DEC\_STANDARD\_4 243  
 B\_DEC\_STANDARD\_5 243  
 B\_DEC\_STANDARD\_6 243  
 B\_DEC\_SUBTRACTIV 243  
 B\_DIR\_READ 304  
 B\_DIR\_WRITE 304  
 B\_E2925\_COMPATIBLE 295, 296  
 B\_EGEN\_ATTRPAGESIZE 256  
 B\_HBMODE\_OFF 295  
 B\_HBMODE\_ON 295  
 B\_IDSEL\_ASSERT 246  
 B\_IDSEL\_DEASSERT 246  
 B\_IDSEL\_DONTCARE 246  
 B\_INTD 161  
 B\_LOC\_BELOWIMEG 248  
 B\_LOC\_SPACE32 248  
 B\_LOC\_SPACE64 248  
 B\_M\_APERR 260  
 B\_M\_AWRPAR 260  
 B\_M\_AWRPAR64 260  
 B\_M\_DACPERR 260  
 B\_M\_DACWRPAR 260  
 B\_M\_DACWRPAR64 261  
 B\_M\_DELAY 261  
 B\_M\_DOLOOP 265  
 B\_M\_DPERR 263  
 B\_M\_DRELREQ 263  
 B\_M\_DSERR 263  
 B\_M\_DWRPAR 263  
 B\_M\_DWRPAR64 264  
 B\_M\_LAST 265  
 B\_M\_LOCK 261  
 B\_M\_MARKER 264  
 B\_M\_RELREQ 261  
 B\_M\_REPEAT 265  
 B\_M\_REQ64 261  
 B\_M\_RESUMEDELAY 262  
 B\_M\_STEPS 262  
 B\_M\_TRYBACK 262  
 B\_M\_WAITS 264  
 B\_MGEN\_ATTRMODE 257  
 B\_MGEN\_CACHELINESIZE 257  
 B\_MGEN\_DELAYCTR 257  
 B\_MGEN\_INVDAT\_ENABLE 257  
 B\_MGEN\_LATCTR 257  
 B\_MGEN\_LATMODE 257  
 B\_MGEN\_MASTERENABLE 257  
 B\_MGEN\_MWIENABLE 257  
 B\_MGEN\_REPEATMODE 257  
 B\_MGEN\_RUNMODE 258  
 B\_MGEN\_TRYBACK\_ENABLE 258  
 B\_OBS\_ACCUERR 267  
 B\_OBS\_FIRSTERR 267  
 B\_OBS\_OBSSTAT 267  
 B\_PATT\_TERM 63  
 B\_PATTOMODE\_DIFFERENTIAL 295  
 B\_PATTOMODE\_STANDARD 295  
 B\_PERFCTR\_A 86  
 B\_PERFCTR\_B 86  
 B\_PERFCTR\_C 86

- B\_PERFGEN\_CAMODE 268
- B\_PERFGEN\_CTRC\_PREL 268
- B\_PERFMEAS 85
- B\_PERFORMANCE 295, 296
- B\_PERFSEQ\_NEXTSTATE 269, 300
- B\_PERFSEQ\_CA\_EN 268
- B\_PERFSEQ\_CB\_EN 268
- B\_PERFSEQ\_CE 268
- B\_PERFSEQ\_CLOAD 268
- B\_PERFSEQ\_STATE 269, 300
- B\_PERFSEQ\_XCOND 268
- B\_PMD\_INPONLY 285
- B\_PMD\_OPENDRAIN 285
- B\_PMD\_TOTEMPOLE 285
- B\_PORT\_CURRENT 270
- B\_PORT\_FASTHIF 270
- B\_PORT\_OFFLINE 270
- B\_PORT\_PCI\_CONF 270
- B\_PORT\_RS232 270
- B\_PROTOCOL\_HARD 294
- B\_PROTOCOL\_LITE 294
- B\_PROTOCOL\_MEDIUM 294
- B\_PT\_SQ 294
- B\_PT\_TRIGGER 294
- B\_PU\_CONFRESTORE 271
- B\_PU\_MASTERRRRUNMODE 271
- B\_PU\_SSTRUNMODE 271
- B\_PU\_TRCRUNMODE 271
- B\_R\_ARB 266
- B\_R\_CACHE 266
- B\_R\_DEVSEL 266
- B\_R\_FRAME 266
- B\_R\_IRDY 266
- B\_R\_LAT 266
- B\_R\_LOCK 266
- B\_R\_PARITY 266
- B\_R\_SEM 266
- B\_R\_STOP 266
- B\_R\_TRDY 266
- B\_R\_W64 266
- B\_REFCTR 86
- B\_RES\_COMPARE 249
- B\_RES\_COMPARE\_DEFATTR 249
- B\_RES\_DATA 249
- B\_RES\_DATA\_DEFATTR 249
- B\_RES\_REGFILE 249
- B\_RES\_STATICIO 249
- B\_RESLOCK\_CUPORT 272
- B\_RESLOCK\_EXERCISER 272
- B\_RESLOCK\_MAILBOX 272
- B\_RESLOCK\_OBSERVER 272
- B\_RESLOCK\_PATT\_TERM 272
- B\_RESLOCK\_PERFORMANCE 272
- B\_RESLOCK\_STATIC\_IO 272
- B\_RESLOCK\_TRACEMEM 272
- B\_RUNMODE\_ADDRRESTART 286
- B\_RUNMODE\_SEQUENTIAL 286
- B\_SINFO\_BUSSPEED 285
- B\_SINFO\_BUSWIDTH 285
- B\_SIZE\_BYTE 174, 176, 284
- B\_SIZE\_DWORD 284
- B\_SIZE\_WORD 174, 176, 284
- B\_STAT\_PINMODE 285
- B\_T\_ACK64 288
- B\_T\_APERR 288
- B\_T\_DACPERR 288
- B\_T\_DOLOOP 291
- B\_T\_DPERR 289
- B\_T\_DSERR 289
- B\_T\_MARKER 289
- B\_T\_REPEAT 291
- B\_T\_TERM 290
- B\_T\_WAITS 290
- B\_T\_WRPAR 290
- B\_T\_WRPAR64 290
- B\_TATTR\_GRP\_TAO 287
- B\_TATTR\_GRP\_TC 287
- B\_TATTR\_GRP\_TD 287
- B\_TC\_ERROR 292
- B\_TC\_HOLD\_TIME 292
- B\_TC\_HSIGN 292
- B\_TC\_SETUP\_TIME 292
- B\_TC\_TCSTAT 292
- B\_TC\_VIOLATION 292
- B\_TCGEN\_SPEC 291
- B\_TGEN\_BACKCAPABLE 286
- B\_TGEN\_IOSPACE 286
- B\_TGEN\_ROMENABLE 286
- B\_TGEN\_RUNMODE 286
- B\_TRC\_HEARTBEATMODE 295
- B\_TRC\_HEARTBEATVALUE 295
- B\_TRC\_PERFANALYZER\_MODE 295
- B\_TRC\_LINESCAPT 296
- B\_TRC\_PATT0\_MODE 295
- B\_TRC\_PERFANALYZER\_MODE 296
- B\_TRC\_STAT 296
- B\_TRC\_TRIG\_HISTORY 295
- B\_TRC\_TRIGPOINT 296
- B\_TRGIOSEQ\_CDEC 298, 300
- B\_TRGIOSEQ\_CINC 298, 300
- B\_TRGIOSEQ\_DDEC 298, 300
- B\_TRGIOSEQ\_DINC 298, 300
- B\_TRIGIO\_INPOLY 297
- B\_TRIGIO\_OPENDRAIN 297
- B\_TRIGIO\_TOTEMPOLE 297
- B\_TRIGIOSEQ\_OUT\_0...11 298
- B\_TRIGIOSEQ\_CLOAD 298, 300
- B\_TRIGIOSEQ\_DLOAD 298, 300
- B\_TRIGIOSEQ\_NEXTSTATE 299
- B\_TRIGIOSEQ\_STATE 299
- B\_TRIGIOSEQ\_XCOND 298
- B\_TRIGIOSEQGEN\_CTRC\_PREL 297
- B\_TRIGIOSEQGEN\_OUT 297
- B\_TRIGSEQ\_TRIGCOND 300
- B\_TRIGSEQ\_SQCOND 300
- B\_TRIGSEQ\_XCOND 300
- B\_TRIGSEQGEN\_CTRC\_PREL 299
- B\_TST\_BANDWIDTH 293
- B\_TST\_BLKLENGTH 293
- B\_TST\_COMPARE 293
- B\_TST\_DATAPATTERN 293
- B\_TST\_DESTINADDR 293
- B\_TST\_NOBYTES 293
- B\_TST\_PROTOCOL 294
- B\_TST\_SOURCEADDR 294
- B\_TST\_STARTADDR 294
- B\_TSTCMD\_BLOCKMOVE 167
- B\_TSTCMD\_READ 167
- B\_TSTCMD\_TRAFFICMAKE 167
- B\_TSTCMD\_WRITEREAD 167
- B\_VER\_BOARD 301
- B\_VER\_CAPI 301
- B\_VER\_CORE 301
- B\_VER\_FIRMWARE 301
- B\_VER\_FIRMWARE\_DATE 301
- B\_VER\_PRODUCT 301
- B\_VER\_SERIAL 301
- B\_VER\_TEAM 301
- B\_VER\_XILDATE 301
- BPPR\_ALG\_BEST 302
- BPPR\_ALG\_PERM 302
- BPPR\_ALG\_RAND 302
- BPPR\_ALG\_RECOMM 302
- BPPR\_BLK\_ALIGN 307
- BPPR\_BLK\_ATTRPAGE 304
- BPPR\_BLK\_BUSADDR 304
- BPPR\_BLK\_BUSADDR\_HI 304
- BPPR\_BLK\_BYTEN 307
- BPPR\_BLK\_CACHELINE 305
- BPPR\_BLK\_CACHELINE\_NO 305
- BPPR\_BLK\_CMDS 307
- BPPR\_BLK\_COMPFLAG 304
- BPPR\_BLK\_COMPOFFS 304
- BPPR\_BLK\_DIR 304, 307
- BPPR\_BLK\_FILLGAPS 305, 307
- BPPR\_BLK\_FIRSTPERM 305
- BPPR\_BLK\_INTADDR 304
- BPPR\_BLK\_LASTPERM 306
- BPPR\_BLK\_NOFDWORDS 304
- BPPR\_BLK\_PAGENUM 304
- BPPR\_BLK\_PAGESIZEACT 306
- BPPR\_BLK\_PAGESIZEMAX 304
- BPPR\_BLK\_SIZE 307
- BPPR\_BLK\_TIME 306
- BPPR\_GEN\_BUSSPEED 306, 310, 312, 315
- BPPR\_GEN\_BUSWIDTH 310
- BPPR\_GEN\_SEED 310
- BPPR\_GEN\_XFERCLKS 306, 310, 312, 315
- BPPR\_MA\_DATA 312
- BPPR\_MA\_FIRSTPERM 311
- BPPR\_MA\_LASTPERM 312
- BPPR\_MA\_PAGENUM 311
- BPPR\_MA\_PAGESIZEACT 312
- BPPR\_MA\_PAGESIZEMAX 311, 312
- BPPR\_MA\_RUNS 312
- BPPR\_MA\_TIME 312
- BPPR\_MA\_TUPLES 311
- BPPR\_MA\_TUPLES\_DATA 312
- BPPR\_MA\_TUPLES\_TIME 312
- BPPR\_REP\_BLOCK 313
- BPPR\_REP\_BLOCKCONTENT 313
- BPPR\_REP\_CAPI 313
- BPPR\_REP\_MA 313
- BPPR\_REP\_MACONTENT 313



- BPPR\_REP\_ORDER\_TUPLE 313
- BPPR\_REP\_REPORT 313
- BPPR\_REP\_TA 313
- BPPR\_REP\_TACONTENT 313
- BPPR\_TA\_DATA 315
- BPPR\_TA\_FIRSTPERM 314
- BPPR\_TA\_LASTPERM 315
- BPPR\_TA\_PAGENUM 314
- BPPR\_TA\_PAGESIZEACT 315
- BPPR\_TA\_PAGESIZEMAX 314, 315
- BPPR\_TA\_TIME 315
- BPPR\_TA\_TUPLES 314
- BPPR\_TA\_TUPLES\_DATA 315
- BPPR\_TA\_TUPLES\_TIME 315
- protocol
  - observer
    - functions 42
  - stress 294
- R**

---

- reference counter 86
- report
  - generating 222
  - hints and warnings 223
- resource
  - properties 272
- S**

---

- serial
  - interface 270
- Signal Types 275
- Signals
  - Trigger Signals 280
- Software Errors 250
- speed 247
- standard
  - pattern term operators 64
- state 269, 299, 300
- static I/O port
  - programming functions 177
  - properties 285
- status register (card) 38
- steps 262
- storage qualifier
  - condition 71
- stress 294
- syntax of value lists 308
- system
  - checking (info) 30
- T**

---

- target
  - attribute
    - groups 127
    - page 131
    - properties 288
  - decoder
    - power-up behavior 135
    - properties 243
    - programming functions 125
- test
  - built-in 162
  - properties 162
- trace
  - memory
    - functions 75
    - properties 295
    - trigger sequencer
      - functions 67
- traffic make 167
- transition condition 71, 268, 298
- transitional pattern term 66, 295
- trigger
  - condition 71
  - counter preload 295
  - mode (heartbeat/normal) 295
  - occurred (bit of the status register) 296
  - point (line number in trace memory) 296
- trigger I/O
  - programming functions 181
- Trigger Signals 280
- Type Definitions 237
- types
  - b\_addrspacetype 237
  - b\_blkproptype 238
  - b\_boardproptype 241
  - b\_cpuproptype 242
  - b\_decodertype 243
  - b\_decproptype 243
  - b\_errtype 250
  - b\_exercisergenproptype 101, 256
  - b\_masterergenproptype 257
  - b\_mattrgrouptype 259
  - b\_mattrproptype 260
  - b\_obsruletype 266
  - b\_obsstatustype 267
  - b\_perfgenproptype 268
  - b\_perfseqtrancondproptype 268
  - b\_perfseqtranproptype 269
  - b\_porttype 22, 270
  - b\_puproptype 271
  - b\_resourcetype 272
  - b\_signaltype (for Timing Check) 273
  - b\_signaltype (for Trace Memory) 274
  - b\_sizetype 284
  - b\_staticproptype 285
  - b\_systeminfotype 285
  - b\_targetgenproptype 286
  - b\_tattrgrouptype 287
  - b\_tattrproptype 288
  - b\_tcgenproptype 291
  - b\_tcproptype 59
  - b\_tctesttype 292
  - b\_testproptype 162
  - b\_traceattproptype 81
  - b\_traceproptype 295
  - b\_tracestatustype 296
  - b\_trigioseqgenproptype 297
  - b\_trigioseqtrancondproptype 298
  - b\_trigioseqtranproptype 299
  - b\_trigseqgenproptype 299
  - b\_trigseqtranproptype 300
  - b\_versionproptype 301
  - bppr\_algorithmtype 302
  - bppr\_blkpermproptype 304
  - bppr\_blkresultparamtype 306
  - bppr\_blkvarparamtype 307
  - bppr\_genproptype 310
  - bppr\_mattrpermproptype 215
  - bppr\_mattrresultparamtype 312
  - bppr\_reportproptype 313
  - bppr\_tattrpermproptype 314
  - bppr\_tattrresultparamtype 315
- V**

---

- value lists (syntax) 308
- version
  - properties 301
- W**

---

- warnings in the report 223

