

Agilent E2929/E2930 Opt. 300 PCI-X Exerciser

User's Guide



Agilent Technologies

Important Notice

All information in this document is valid for both Agilent E2929 and Agilent E2930 testcards unless otherwise noted.

© Agilent Technologies, Inc. 2003

Revision

June 2003

Printed in Germany

Agilent Technologies
Herrenberger Straße 130
D-71034 Böblingen
Germany

Authors: t3 medien GmbH

Warranty

The material contained in this document is provided "as is," and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Agilent Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

Safety Notices

CAUTION

A CAUTION notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a CAUTION notice until the indicated conditions are fully understood and met.

WARNING/DANGER

A WARNING notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met.

Trademarks

Windows NT ® and MS Windows ® are U.S. registered trademarks of Microsoft Corporation.

Contents

Documentation Overview	7
PCI-X Exerciser Overview	9
Gaining an Active Part in Optimization and Validation	10
PCI-X Exerciser Architecture	12
PCI-X Exerciser Configurations	15
Setting Up a PCI-X Exerciser Test	17
Typical PCI-X Transactions	17
Running A Sample PCI-X Exerciser Session	19
Example Scenarios	20
Preparing for the Exerciser Guided Tour	21
Guided Tour: Generating Requester-Initiator Block Transfer	23
Loading the Setup Files	23
Setting up a Requester-Initiator Transaction	24
Running the Test	25
Viewing the Results	26
Guided Tour: Specifying Requester-Initiator Behavior	29
Loading the Setup Files	30
Setting up the Requester-Initiator Behaviors	30
Running the Test	31
Viewing the Results	32
Guided Tour: Specifying Completer-Target Behavior	34
Loading the Setup Files	35
Setting up the Target Decoders	36
Specifying the Completer-Target Protocol Behavior	37
Viewing the Results	38

Guided Tour: Defining the Transfers that will be given a Split Response	40
Starting the Command Line Interface	40
Entering the Required CLI Commands	41
Guided Tour: Specifying Completer-Initiator Behavior	43
Loading the Setup Files	44
Setting up the Completer-Initiator Behaviors	44
Running the Test	44
Viewing the Results	45
Guided Tour: Specifying Requester-Target Behavior	46
Loading the Setup Files	47
Setting up the Requester-Target Decoder	48
Specifying the Requester-Target Protocol Behavior	49
Viewing the Results	50
The PCI-X Exerciser as a Requester-Initiator Device	53
Programming Requester-Initiator Transactions	54
Requester-Initiator Transactions Overview	55
Available Transaction Properties	59
How to Specify Requester-Initiator Transactions	62
Programming the Requester-Initiator Behaviors	63
Available Requester-Initiator Behaviors	64
Latencies between Requester-Initiator Transactions	65
Starting the Requester-Initiator	66
Preparing Test Execution	66
Running the Requester-Initiator	68
Programming Requester-Initiator Execution Order of Blocks	70
Programming Requester-Initiator Reaction to Terminations	71
Programming Injection of Errors, Interrupts and Triggers	71
Stopping the Requester-Initiator	73

The PCI-X Exerciser as a Completer-Target Device	75
Configuration Space and Completer-Target Decoders	76
Configuration Space Header	77
Target Decoder Properties	79
Data Resources	83
Target Decoder Setup	84
Programming the Completer-Target Decoders	85
Modifying the Configuration Space Header	87
Overwriting BIOS Settings	90
Standard and Power-Up Databases	91
Programming Completer-Target Behaviors	93
Available Completer-Target Behaviors	94
Initial Latencies	95
Defining a Split Response Condition	96
The PCI-X Exerciser as Completer-Initiator Device	97
Programming Completer-Initiator Behaviors	98
Available Completer-Initiator Behaviors	99
Defining Split Completion Messages	101
Generating Split Completion Transactions	102
The PCI-X Exerciser as a Requester-Target Device	103
Requester-Target Decoder Setup	104
Programming Requester-Target Behaviors	105
Available Requester-Target Behaviors	106
Using Data Resources	109
Data Memory	109
Data Memory Organization	110
How to Use the Data Memory Editor	112
Data Generator	114
Performing Unidirectional Data Path Verification	115
How to Use the Data Generator	116
Real-Time Data Compare	118
Programmable Data Path Configurations	119

Generating Interrupts	121
Interrupt Capabilities of the Testcard	122
How to Assert and Deassert Interrupts	123
Using the Command Line Interface	125
How to Start the CLI	126
Basic CLI Command Syntax	127
Using CLI Scripts	128

Documentation Overview

This section shows you the different types of documents offered by Agilent Technologies and gives you an overview of which documents are available when you work with the Agilent E2929/E2930 PCI-X Exerciser and Analyzer.

All information is valid for both Agilent E2929 and Agilent E2930 testcards unless otherwise noted. The following documents are available:

Getting Started Guide

- **Getting Started Guide**

Introduces standard analysis features and provides an example of how to set up the protocol observer.

Provides an overview of how the Agilent E2930 testcard supports the PCI-X 2.0 features.

Gives detailed information about hardware and interfaces.

User's Guides

- **Agilent E2929/E2930 Opt. 300 PCI-X Exerciser User's Guide**

Provides information on programming the testcard as an initiator and/or target device. It shows you how to actively stimulate the PCI-X bus.

This guide shows how to:

- Initiate data transfers on the PCI-X bus (act as requester-initiator).
- Act as completer-target.
- Handle split-completion transactions (act as completer-initiator).
- Handle open requests (act as requester-target).

- **Agilent E2929/E2930 PCI-X Analyzer User's Guide**

Provides information on how to examine the behavior and performance of a PCI-X device on the bus and shows how to perform functional tests such as data compares.

- **E2929 Opt. 200 PCI-X Performance Optimizer User's Guide**

Provides all features that are needed to evaluate and optimize any device under test in terms of the performance (post-processed performance analysis and optimization). Option 200 is available for Agilent E2929 testcards only.

- **Agilent E2920 PCI-X Series Opt. 320 C-API/PPR Programmer's Guide**

Provides information on how to set up test programs using the C functions described in the corresponding C-API/PPR Reference.

GUI and C-API/PPR References

- **Agilent E2929/E2930 Windows and Dialog Boxes Reference**

Provides reference information on all windows and dialog boxes of the Agilent E2920 graphical user interface (GUI).

- **Agilent E2929/E2930 Opt. 320 C-API/PPR Reference**

Describes all C functions, types and definitions of the application programming interface of the Agilent E2929/E2930 PCI-X testcard.

This reference also provides the commands and abbreviations that are used in the command line interface (CLI) of the graphical user interface.

- **Agilent E2922/E2923 Opt. 320 C-API/PPR Reference**

Describes all C functions, types and definitions of the application programming interface of the Agilent E2922/E2923 PCI-X testcard.

This reference also provides the commands and abbreviations that are used in the command line interface (CLI) of the graphical user interface.

PCI-X Exerciser Overview

Agilent Technologies' E2920 Verification Tools, PCI-X Series is your “window into the system” during product development, giving you access to almost all of the system components located on the PCI-X bus, as well as devices and adapters on secondary buses or within the system.

The PCI-X Exerciser (option 300) allows you to overcome the passive role of monitoring the PCI-X bus. With the PCI-X Exerciser, the testcard can be programmed to behave as a requester-initiator, a completer-target, a completer-initiator and/or requester-target device to generate any kind of traffic on the bus.

“Gaining an Active Part in Optimization and Validation” on page 10 gives an overview of how you can use the Exerciser during the various phases of the design cycle.

“PCI-X Exerciser Architecture” on page 12 introduces the structure of the PCI-X Exerciser and shows the interaction of single components.

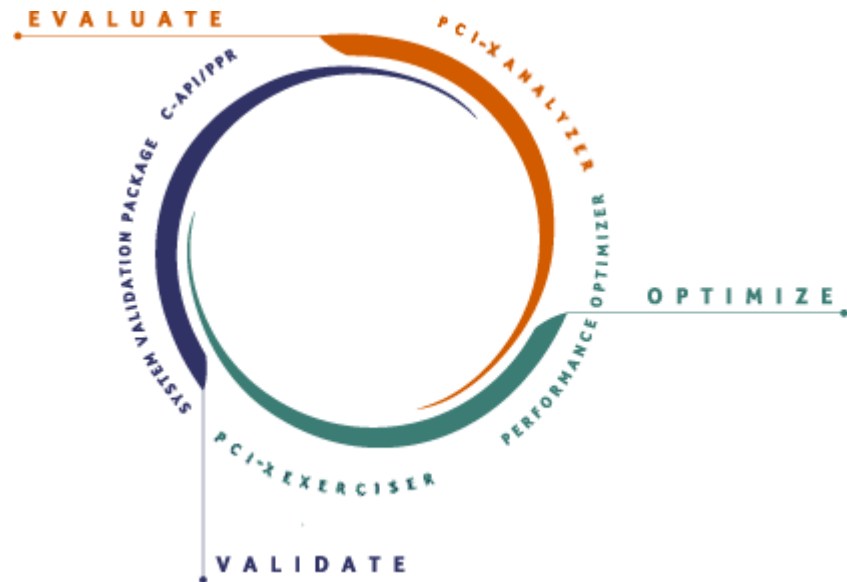
“PCI-X Exerciser Configurations” on page 15 shows examples of possible configurations for PCI-X Exerciser tests.

“Setting Up a PCI-X Exerciser Test” on page 17 outlines the major setup steps that are required.

In most tests, the Exerciser will act as a initiator device and/or a target device on the PCI-X bus, initiating or serving transactions respectively. *“Typical PCI-X Transactions” on page 17* introduces in this basic term.

Gaining an Active Part in Optimization and Validation

The Agilent Technologies PCI-X Exerciser's focus in the design cycle clearly lies on those phases, when you are optimizing and validating your PCI-X design for reliability.



Optimization Phase The Exerciser is designed to enable you to optimize your PCI-X design for reliability.

The Exerciser plays an active role in the analysis of complex PCI-X scenarios by letting you set up worst-case test patterns quickly. It provides the full flexibility to set up all the test scenarios you need, with your own choice of protocol and traffic variations, wait states, retries and disconnects, latencies, and burst sizes. You can also control the use of memory as well as of read and write commands, which is especially important when testing the correct functioning of bridge devices.

This not only saves you the tedious chore of having to test your device with various other PCI-X components one after the other, but also allows you to test in a repeatable way, which means that you can reproduce any errors for deeper investigation.

You can also run functional tests, directing the Exerciser to generate and transmit large blocks of data in specified time intervals, thus testing how much PCI-X traffic your device can handle.

Validation Phase Validating your PCI-X device means ensuring its long-term reliability. This means ensuring that it remains stable under any application conditions, with any combination of plug-ins and any kind of traffic on the bus.

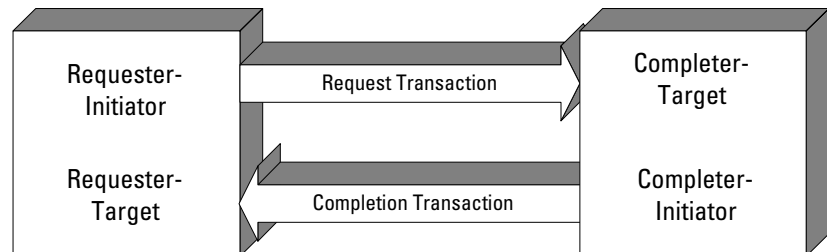
Using the PCI-X Exerciser as an initiator or a target device, you can test your design repeatedly and consistently from the earliest opportunity, from the earliest stages during bring-up and integration, before a complete system is available..

In combination with the C-API/PPR options, the Exerciser offers fully controllable system-test, with wide coverage, reproducibility, and root-cause-analysis capabilities that will reveal system-critical problems faster than any hot mock-up testing could.

PCI-X Exerciser Architecture

The Agilent E2929/E2930 testcard can act as an initiator or a target device on the PCI-X bus to generate any kind of PCI-X transfer.

Because PCI-X allows you to perform split transactions, initiator and target have been subdivided. This is shown in the following figure:



The **requester-initiator (RI)** is the initiator of a transaction. It initiates all transactions except split completion transactions.

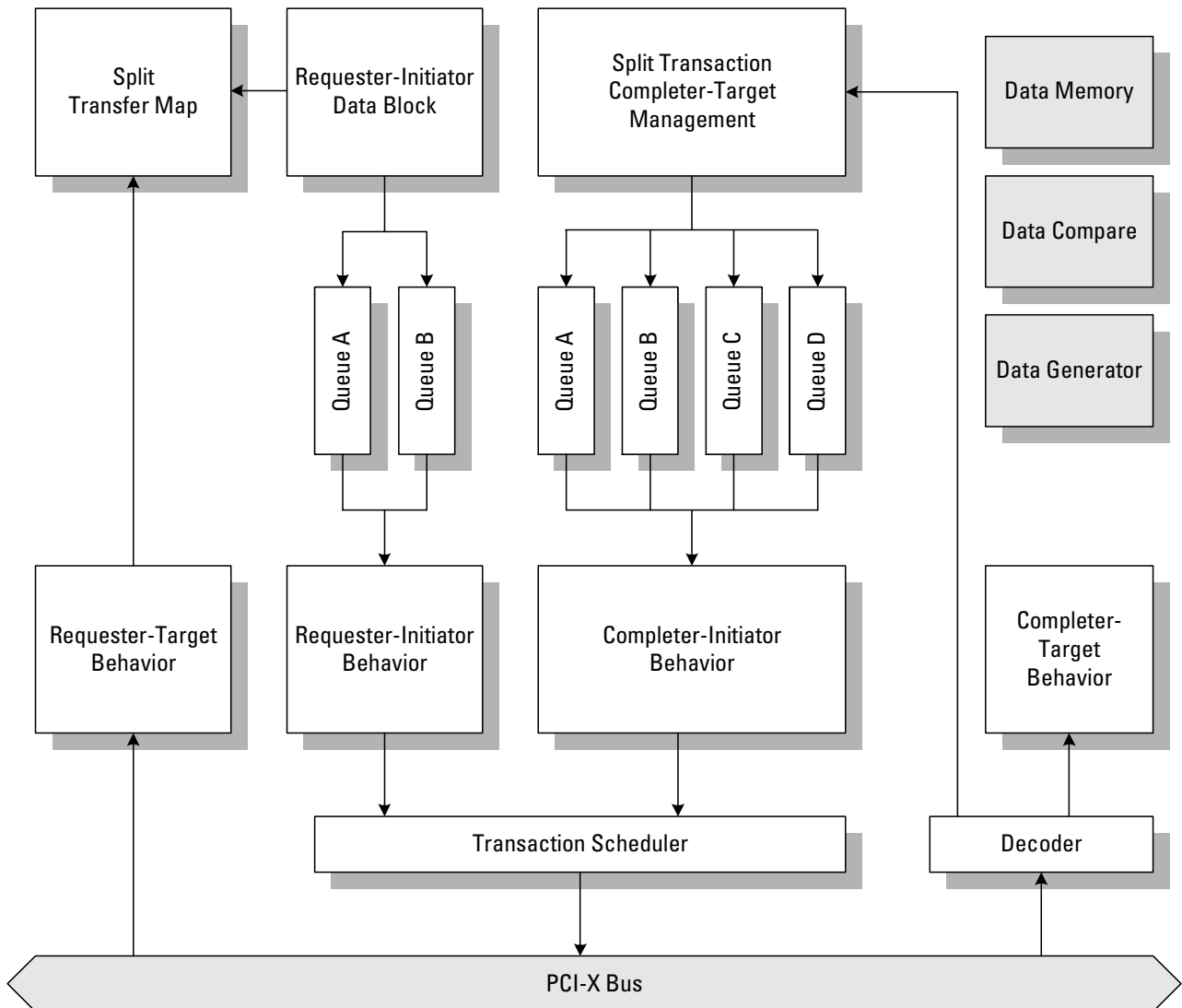
The **completer-initiator (CI)** initiates split completion transactions only.

The **completer-target (CT)** is the target for a transaction. It decodes all transactions except split completion transactions.

The **requester-target (RT)** decodes split transactions only.

PCI-X Exerciser Architecture Overview

The following figure shows the architecture of the PCI-X Exerciser and the connections between single components:



The exerciser is based on the following concepts:

1. Requester-initiator data blocks are defined.

The blocks describe **what** data is transferred over the PCI-X bus.

2. Requester-initiator behaviors are defined.

The behaviors describe **how** data transferred over the PCI-X bus is executed.

For the requester-initiator, up to 256 blocks of data transfers can be set up. In addition, **requester-initiator behaviors** are set up to specify how the requester-initiator intends to transfer the data blocks over the PCI-X bus.

If any target replies to a transfer and requests a split transaction, the data block attributes are moved internally to the split transaction map for further use. The transaction map can manage up to 32 split transactions. When completing split transactions, the **requester-target behaviors** are used to control the transfer.

The **completer-target behaviors** define how the target of the Agilent E2929/E2930 testcard acts. The completer-target can manage up to four split transaction queues.

To control the initiation of a split transaction completion, the **completer-initiator behaviors** are used.

The **transaction scheduler** decides whether the completer-initiator or the requester-initiator transaction is performed.

All data is supplied by the onboard **data memory** or from the onboard **real-time generator**.

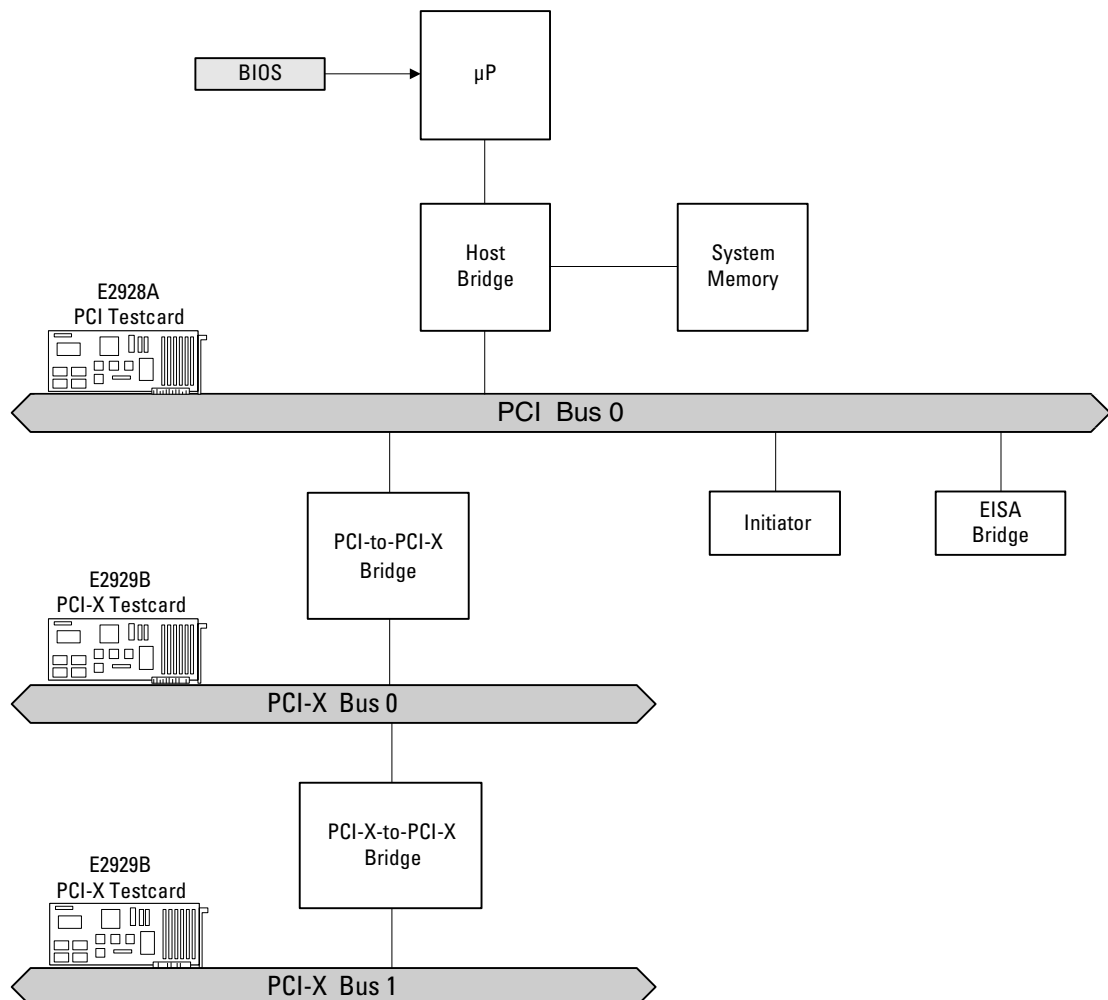
The **real-time data compare** unit is used to compare data that is written to the memory against the actual memory content.

Data compare can also be performed on the data generator.

PCI-X Exerciser Configurations

The configuration possibilities provided by the testcard and the GUI are basically the same as described for the testcard in the *Agilent E2929/E2930 PCI-X Getting Started Guide*.

When using the Exerciser, your test requirements determine where to insert a testcard into the system under test. The following figure gives an overview of a typical computer system, where a single Agilent E2928A PCI testcard and two Agilent E2929B PCI-X testcards have been inserted.



You can use the Exerciser's initiator and target features to test each individual device—or the system as a whole:

- Testing devices

The testcard's requester-initiator can be used to initiate transactions to a target device under test. The testcard's target can be used to react to transactions initiated by the requester-initiator device under test. Initiator devices can also be bridges, for instance host bridges or PCI-X-to-PCI-X bridges.

The testcard can also be used for functional tests. For example, when testing communication devices (LAN cards, ATM cards), the testcard can be set up to generate and receive data blocks at specified data rates, thus allowing to test device and system behavior under full load.

- Testing a PCI-X-to-PCI-X bridge

To test the interfaces of a PCI-X-to-PCI-X bridge, testcards can communicate with other devices (or testcards) over a bridge or on the same bus. Thus, you can test whether the bridge forwards transactions to the correct bus.

During start-up, you can test whether configuration cycles are transferred correctly.

- Testing the whole system

When testing a system as a whole, the Exerciser can be used to:

- Set up any PCI-X traffic scenario quickly and in a repeatable way.
- Simulate PCI-X devices that are not yet available.
- Generate interrupts to test the system's interrupt processing capabilities.

Agilent E2930 Testcard for PCI-X 2.0

The Agilent E2930 testcard fully supports PCI-X mode 1 and mode 2, allowing to test PCI-X 2.0 systems and devices. It allows to determine the states of the PCIXCAP and M66EN pins through SW-controlled relays (there is no PCIXCAP jumper as on E2929 testcards). Because these pins signal a card's capabilities to the booting system, the testcard is able to emulate other types of PCI and PCI-X cards.

Setting Up a PCI-X Exerciser Test

Setting up a PCI-X Exerciser test includes the steps already described for the PCI-X standard analysis test (see *Agilent E2929/E2930 Getting Started Guide*).

Briefly, the following steps are necessary:

1. Inserting the testcard.
2. Connecting to the testcard.
3. Additionally, you have to set up the Exerciser as required for your test. This is described in detail in:
 - “*The PCI-X Exerciser as a Requester-Initiator Device*” on page 53
 - “*The PCI-X Exerciser as a Completer-Target Device*” on page 75
 - “*The PCI-X Exerciser as Completer-Initiator Device*” on page 97
 - “*The PCI-X Exerciser as a Requester-Target Device*” on page 103
 - “*Using Data Resources*” on page 109

Typical PCI-X Transactions

On every PCI-X system the different devices communicate via the PCI-X bus. This communication is controlled by the bus arbiter that determines which device may actively use the bus at any given time. This mechanism is used to avoid data collision and possible hardware damage. Basically, there are two different types of devices that operate on the bus: **initiators** (actors) and **targets** (reactors).

The Agilent E2929/E2930 PCI-X Exerciser testcard can simulate a requester-initiator, a completer-target, a completer-initiator or a requester-target device or all together at the same time. This enables the testcard to emulate and/or test any device in your system under test.

The transfer of data between an initiator and a target device is performed in one or more data transactions. These transactions are initiated and controlled by the initiator, while the target reacts depending on the type of transaction.

Transactions without Split Completion

A successful transaction without split completion transaction includes the following steps:

1. The requester-initiator requests bus access from the bus arbiter.
2. The arbiter grants the requester-initiator to use the bus for a transaction.
3. The requester-initiator starts the transaction by sending the address of the completer-target, the bus command and the transaction attributes on the bus.
4. The completer-target that owns the corresponding address range responds by asserting the `DEVSEL#` signal.
5. The requester-initiator initiates one or more data phases on the bus. If the command was a write command, the requester-initiator also sends the data. Otherwise, the data is sent by the completer-target.
6. On reaching the last data phase of the transaction, the requester-initiator deasserts the `FRAME#` signal to indicate that it is ready to complete the transaction.

Transactions with Split Completion

A successful transaction with a split completion transaction includes the following steps:

1. The requester-initiator initiates the transaction by requesting access to the bus, asserting `FRAME#`, and driving the command and address on the bus.
2. The completer-target terminates the transaction with a split response.
3. The data block properties are moved internally to a split transaction map, which is able to manage up to 32 open split transactions.
4. The completer-initiator initiates the split transaction completion, where Completer-Initiator behaviors are used.
5. The requester-target behaviors are used to terminate the split transactions.

Running A Sample PCI-X Exerciser Session

The following application examples explain how the testcard can be used in various tests. After introducing the major scenarios for the PCI-X Exerciser and showing how to prepare for the sample sessions, you will be guided through the following examples:

- *“Guided Tour: Generating Requester-Initiator Block Transfer” on page 23*
- *“Guided Tour: Specifying Requester-Initiator Behavior” on page 29*
- *“Guided Tour: Specifying Completer-Target Behavior” on page 34*
- *“Guided Tour: Defining the Transfers that will be given a Split Response” on page 40*
- *“Guided Tour: Specifying Completer-Initiator Behavior” on page 43*
- *“Guided Tour: Specifying Requester-Target Behavior” on page 46*

Example Scenarios

The Agilent E2929/E2930 PCI-X Exerciser and Analyzer testcard can be used for basically any test that might be required when developing devices, chipsets, or drivers for the PCI-X environment. Here you can find five examples that give a guided introduction to some of the main features of the testcard and its Graphical User Interface (GUI).

Testing a Completer-Target Device

Let us say that you are integrating a PCI-X chip into an adapter or a system, or that your validation team (or even your customer) reports that your PCI-X chip or system is not working properly under certain circumstances. In both cases you will very likely need to generate or reproduce a given PCI-X scenario to find out how your chip behaves in a real environment. You may need to investigate for instance your chip's reaction to certain PCI-X commands or initiator protocol variations such as disconnect or sequence length, or to error conditions such as wrong parity.

In a case like this you can set up the Agilent PCI-X testcard as a requester-initiator to access your chip or system under test with certain commands or behaviors (stimuli from the testcard). This is explained in *“Guided Tour: Generating Requester-Initiator Block Transfer” on page 23*.

Testing a Requester-Initiator Device

If you are developing, debugging, validating, or characterizing a PCI-X bus initiator device, you need a programmable target that can react deterministically in the required fashion. The Agilent PCI-X Exerciser provides this functionality. It has a 1-MB data memory that can be accessed with either memory or I/O transactions. The target protocol behavior is fully programmable for every transaction. This includes the number of wait states, the termination type and so forth.

An example, in which the Exerciser is set up as a completer-target, can be found in *“Guided Tour: Specifying Completer-Target Behavior” on page 34*.

Making C Function Calls Directly

Besides the Graphical User Interface, the Agilent E2929/E2930 testcard also features a C Application Programming Interface (option 320), providing full access to all functions of the testcard.

If you do not have the API installed or do not want to write complete C programs, you can use the Command Line Interface (CLI) instead, which is part of the GUI. With the CLI you can run scripts written in a simple script language.

An example of how this is done is presented in “*Guided Tour: Defining the Transfers that will be given a Split Response*” on page 40.

Testing Split Transaction Completion

If a target replies to a transfer and requests a split transaction, the data block attributes are moved internally to the split transaction map for further use. The transaction map can manage up to 32 split transactions.

The initiation of a split-transaction completion is controlled by the completer-initiator device. For setting up the testcard as a completer-initiator device, completer-initiator behaviors must be specified. This is explained in “*Guided Tour: Specifying Completer-Initiator Behavior*” on page 43.

When completing split transactions, the requester-target behaviors are used to control the transfer. Setting up the testcard as requester-target device is explained in “*Guided Tour: Specifying Requester-Target Behavior*” on page 46.

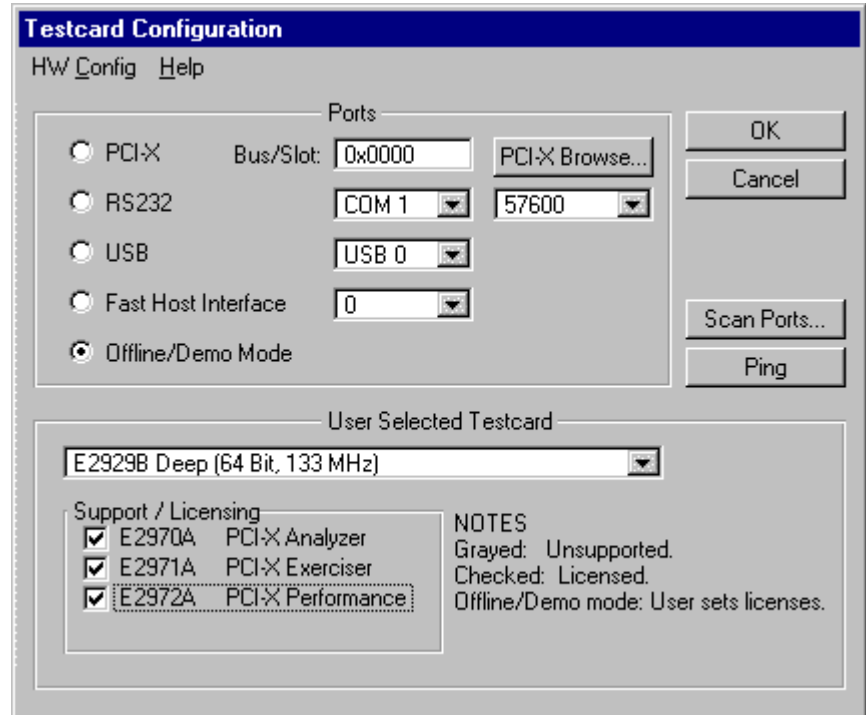
Preparing for the Exerciser Guided Tour

The example described in the guided tour is designed to be performed in Offline/Demo Mode—without hardware.

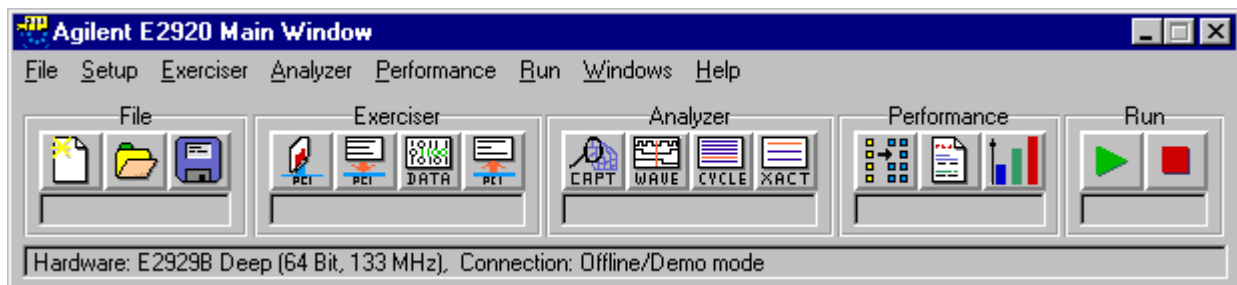
All the setup files (*.bst) and logic analyzer trace files (*.wfm) that are mentioned in the following text can be found under
<your_installation_directory>\samples\demo. If you did not change the default setting during installation, <your_installation_directory> will be
c:\Program Files\Agilent\E2920 PCI-X Series.

To prepare for the guided tour:

- 1 Launch the Agilent E2920 software.
- 2 From the *Setup* menu, choose *Testcard Configuration*.
- 3 In the Testcard Configuration window, select *Offline/Demo Mode*.



- 4 Now choose *E2929B Deep (64 bit, 133 MHz)* from the *User Selected Testcard* list, and select all licenses in the *Support/Licensing* group. Your display should look like the window shown above.
- 5 Click *OK* and the main window looks like this.



You are now ready to start the guided tours.

Guided Tour: Generating Requester-Initiator Block Transfer

This example shows how to set up the testcard as a requester-initiator device that initiates data transfers on the PCI-X bus. More specifically, the following transfers are generated:

- A read of 10 bytes from I/O space at address **0x10000d**.
- A memory read of 273 bytes from memory at address **0x10003000**.
- A memory write of 18 bytes generated by the data generator. The lower part of the write address is **0x2000f0f**, the upper part is **0x30000000**.
- A memory write of 130 bytes and 10 byte enables to the memory at address **0x1000310d**.

Loading the Setup Files

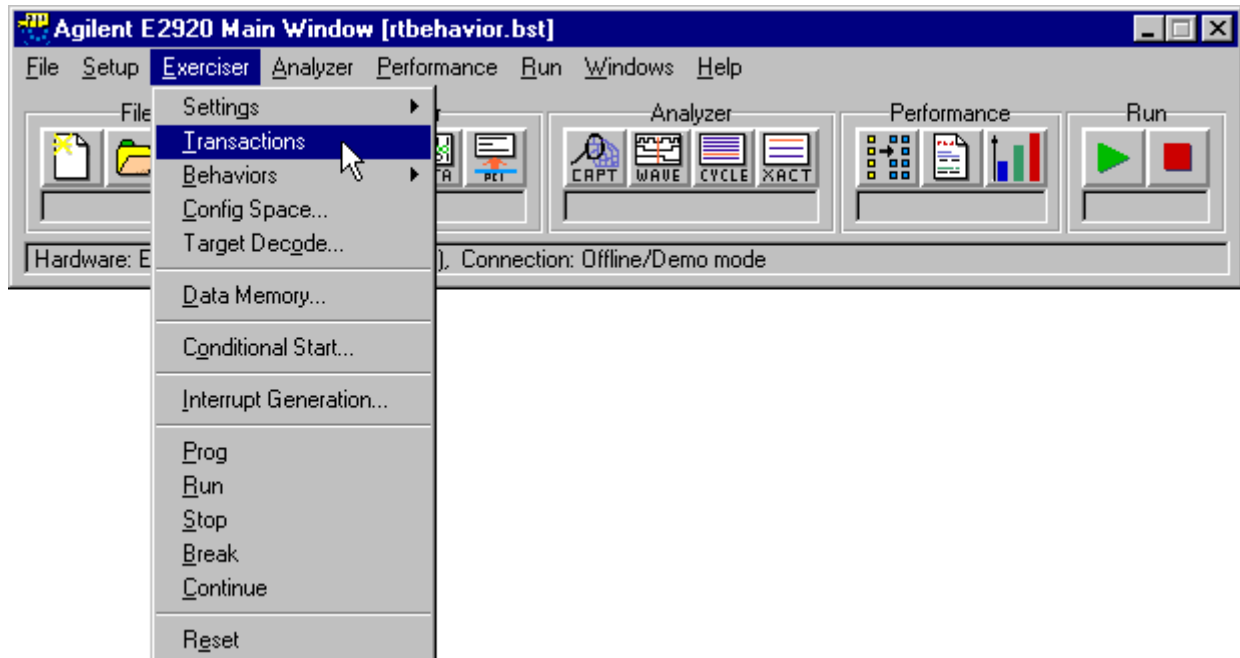
If you were connected to the testcard, the results of the test could be viewed on the screen. However, as you are working in offline mode, you need to start by loading the required files:

- 1 Load the setup file for this example (`block_transfer.bst`) by selecting *Load* from the *File* menu in the main window.
- 2 Load the PCI-X signal waveform file for this example (`block_transfer.wfm`) by selecting *Load from file* from the *File* menu in the Waveform Viewer window.

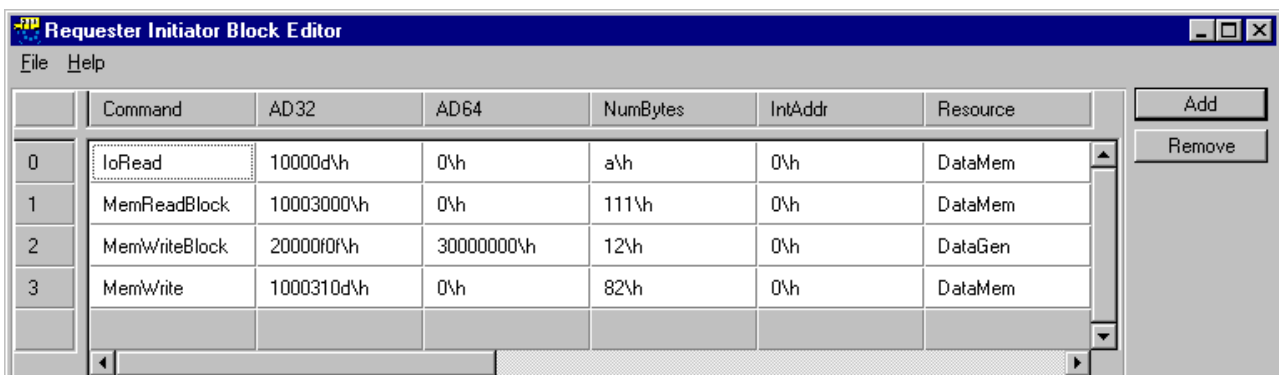
Setting up a Requester-Initiator Transaction

Once you have loaded the setup file and the waveform file, continue by setting up the requester-initiator block for the test.

- ◆ Open the Requester-Initiator Block Editor dialog box by clicking the Transactions button in the main window or by selecting the *Transactions* item in the *Exerciser* menu.



In the Requester-Initiator Block Editor you see that the block transfers for this test are already displayed. They are stored in the setup file (block_transfer.bst) together with all other settings.




For reordering of blocks the testcard provides two queues into which each block can be transferred. You can transfer a block to queue A, to queue B or automatically to any free queue.

Requester-initiator behaviors define which block from which queue is executed. This is shown in “*Guided Tour: Specifying Requester-Initiator Behavior*” on page 29.

In this example, block 0 is transferred to queue A. The other blocks are transferred automatically to any free queue.


Running the Test

If you were connected to a testcard, you would start the test:

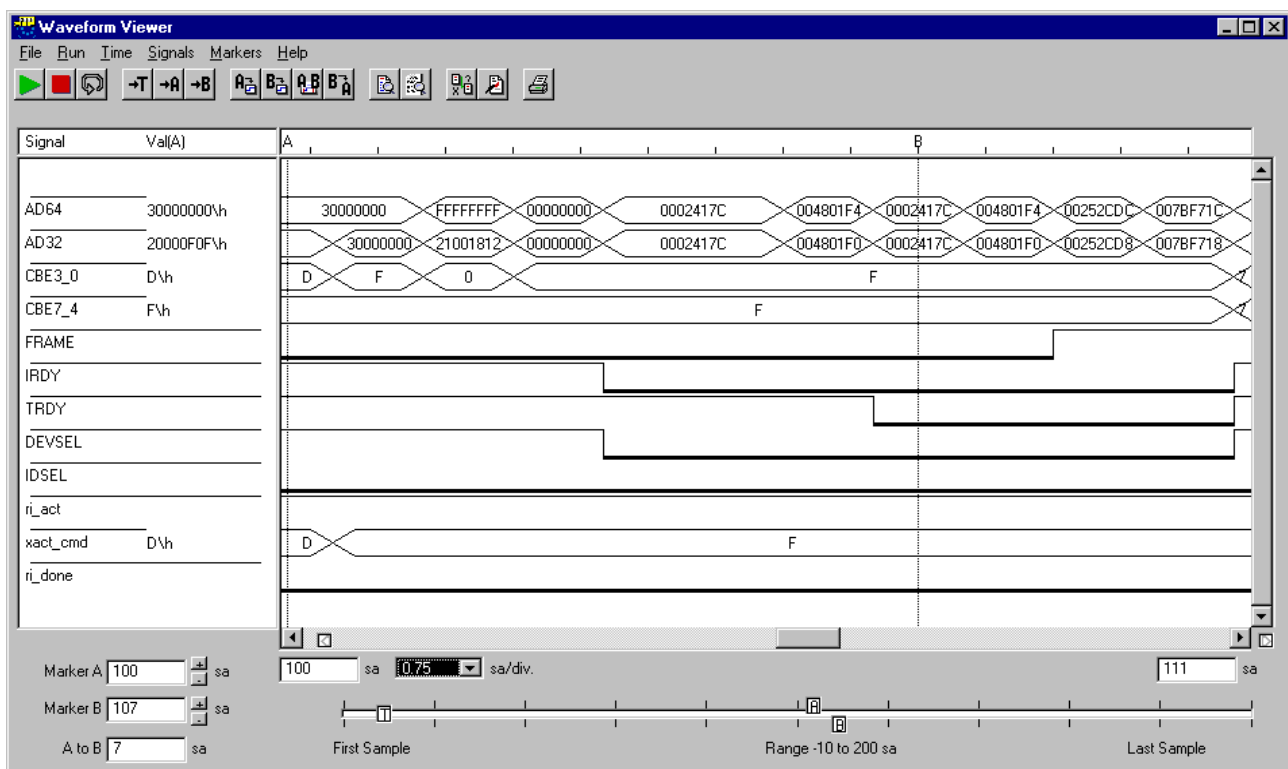
- by clicking the Run button  in the main window, or
- by choosing *Run* from the *Exerciser* menu.

Viewing the Results

Use the Analyzer's Waveform Viewer to inspect the test results. The waveforms were loaded from the waveform file `block_transfer.wfm`.

- 1 Click the Waveform Viewer button  in the main window (or use the *Waveform Lister* item in the Analyzer menu) to open the waveform viewer.

In the following window, you see a section of the waveform diagram in which the third transaction (memory write block to data generator) was executed.



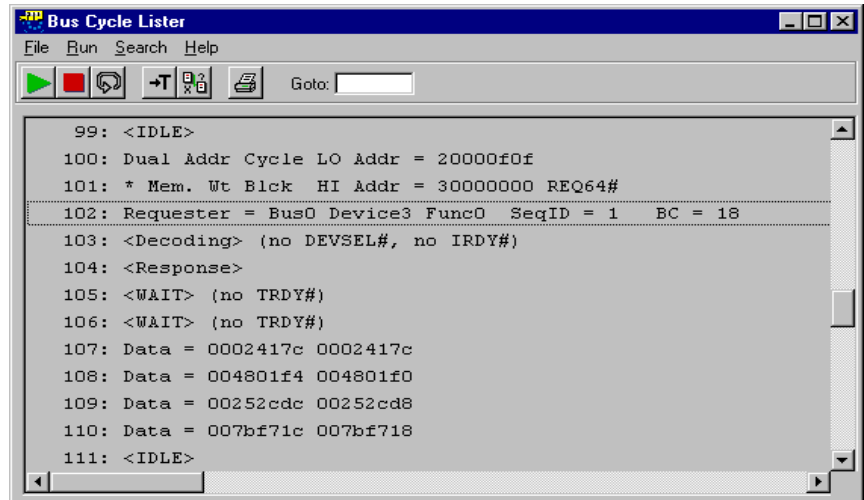
At marker A the requester-initiator drives the dual address cycle command (coding 0xd) with the lower and upper address onto the bus. In the next cycle, it drives the memory write block command (coding 0xf) onto the bus.

It is also easy to identify that during this transaction:


- The requester-initiator is active (ri_act is high).
- The AD32 signal first holds the target address (address phase) and then the transferred data in the four data phases (shown at marker B).

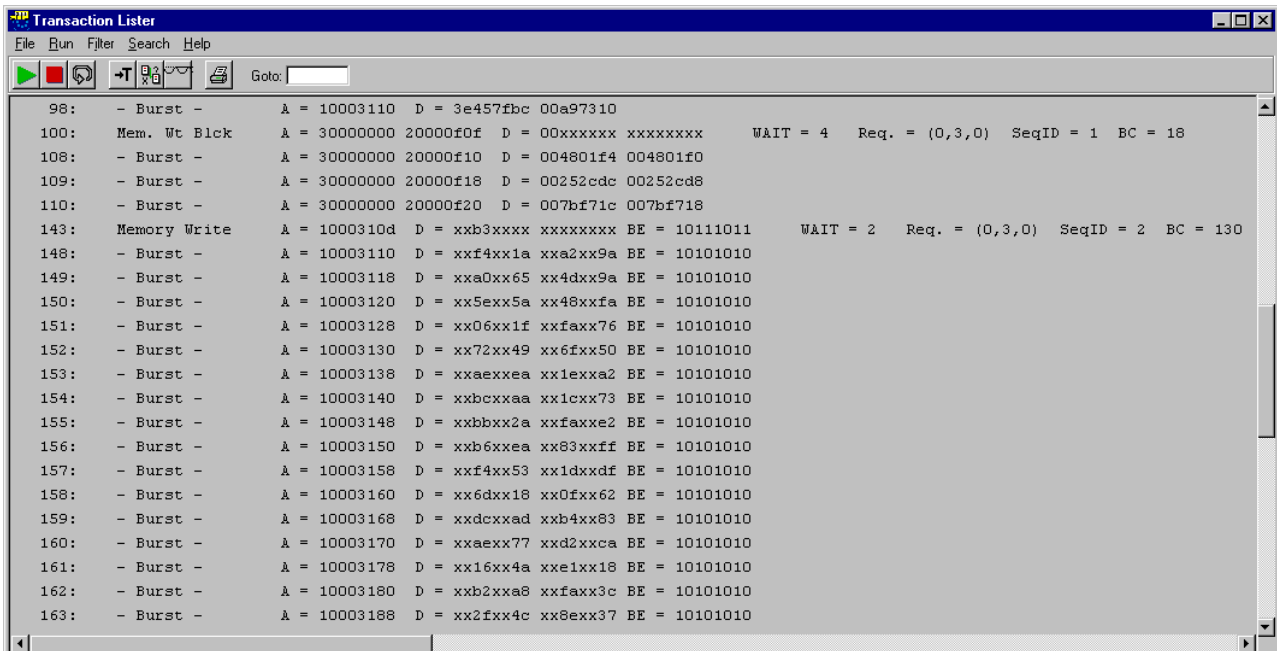
To see the corresponding section in the Bus Cycle Lister window:

- Click the Bus Cycle Lister button  in the main window (or use the *Bus Cycle Lister* item in the *Analyzer* menu) to open the bus cycle lister.



To see the corresponding section in the Transaction Lister window:

- Click the Transaction Lister button  in the main window (or use the *Transaction Lister* item in the *Analyzer* menu) to open the transaction lister.



The following table gives an overview of all of the traffic generated by the exerciser:

Lines	Transfer	Block Line
0	I/O Read bytes 0 ... 2	0
17	I/O Read bytes 3 ... 6	0
34	I/O Read bytes 7 ... 9	0
51 ... 98	Memory Read Block 273 bytes	1
100 ... 110	Memory Write Block 18 bytes	2
143 ... 163	Memory Write 130 bytes	3

In the transaction lister above, the last two transactions are displayed:

- Memory write block
- Memory write

Additionally, you can assign behaviors to each transaction block, which defines further protocol behaviors for the requester-initiator. These behaviors can be found in the Requester-Initiator Behavior Editor dialog box and are introduced in the following section.

Guided Tour: Specifying Requester-Initiator Behavior

Whereas requester-initiator blocks describe **what** data is transferred over the PCI-X bus, requester-initiator behaviors describe **how** data is transferred over the PCI-X bus. In particular, behaviors control the partitioning of blocks into sequences and the reordering of blocks.

This guided tour gives an example of setting up the requester-initiator behaviors for the blocks specified in “*Guided Tour: Generating Requester-Initiator Block Transfer*” on page 23.

The properties of four behaviors are programmed as follows:

- Behavior 0
 - The block is selected from queue A.
 - The sequence length is 64 bytes.
 - The sequence is broken into transactions at every second allowable disconnect boundary (ADB).
 - The sequence is assigned tag 17.
 - The number of steps is two.
- Behaviors 1 to 3
 - The block is selected on a toggle basis from the queue that follows the last used queue.
 - The sequence length is 64 bytes.
 - For behavior 2, the sequence is broken into transactions at each ADB.
 - The sequences for behaviors 1, 2 and 3 are assigned with an automatically selected free tag, tag 8 and tag 30 respectively.

The following table shows the summary of the specified behaviors:

Behavior	Queue	Bytes	Disconnect	Tag	Steps
0	A	64	at every second ADB	17	2
1	Next	64	-	Auto	-
2	Next	64	at each ADB	8	-
3	Next	64	-	30	-

For the remaining properties, the default values are used.

Loading the Setup Files

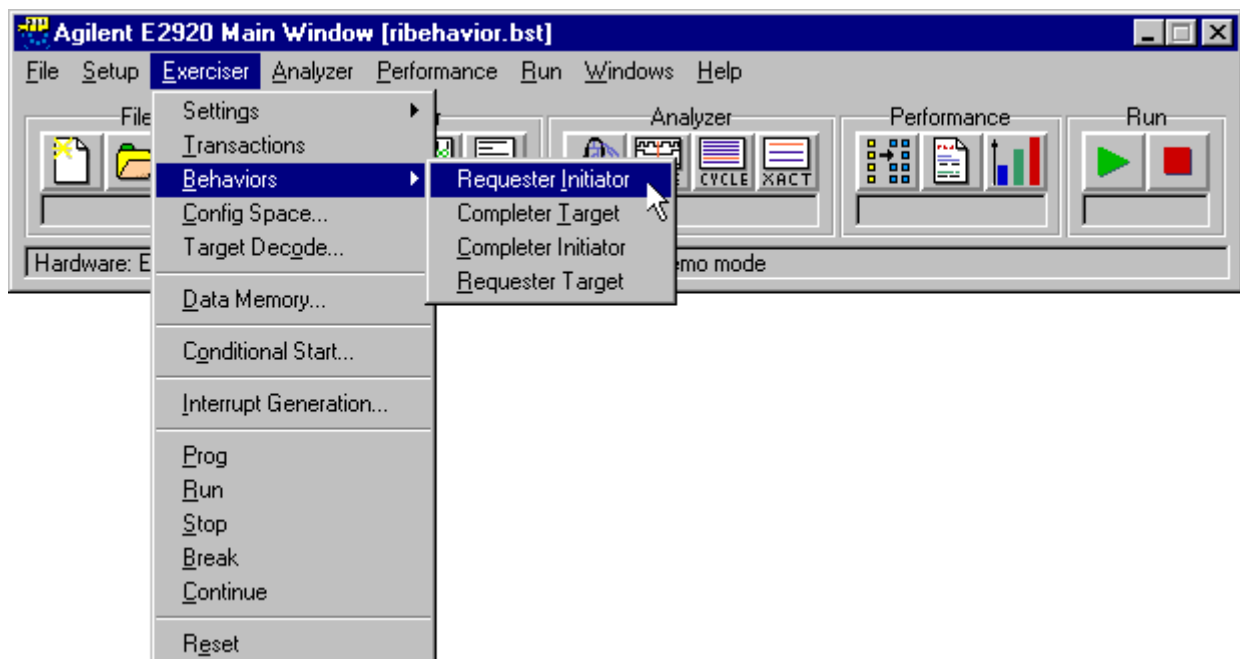
If you were connected to the testcard, the results of the test could be viewed on the screen. However, as you are working in offline mode, you need to start by loading the required files:

- 1 Load the setup file for this example (RIBehavior.bst) by selecting *Load* from the *File* menu in the main window.
- 2 Load the PCI-X signal waveform file for this example (RIBehavior.wfm) by selecting *Load from file* from the *File* menu in the Waveform Viewer window.

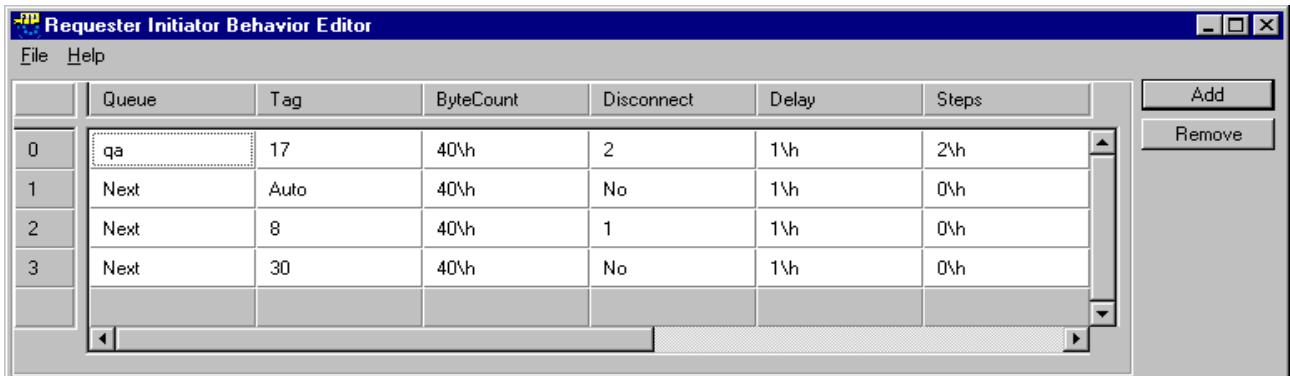
Setting up the Requester-Initiator Behaviors

Once you have loaded the setup file and the waveform file, continue by setting up the requester-initiator behavior for the test.

- ◆ Open the Requester-Initiator Behavior Editor dialog box by selecting *Exerciser -> Behaviors -> Requester Initiator*.




In the Requester-Initiator Behavior Editor you see that the behaviors for this test are already displayed. They are stored in the setup file (RIBehavior.bst) together with all the other settings.



Running the Test


If you were connected to a testcard, you would start the test:

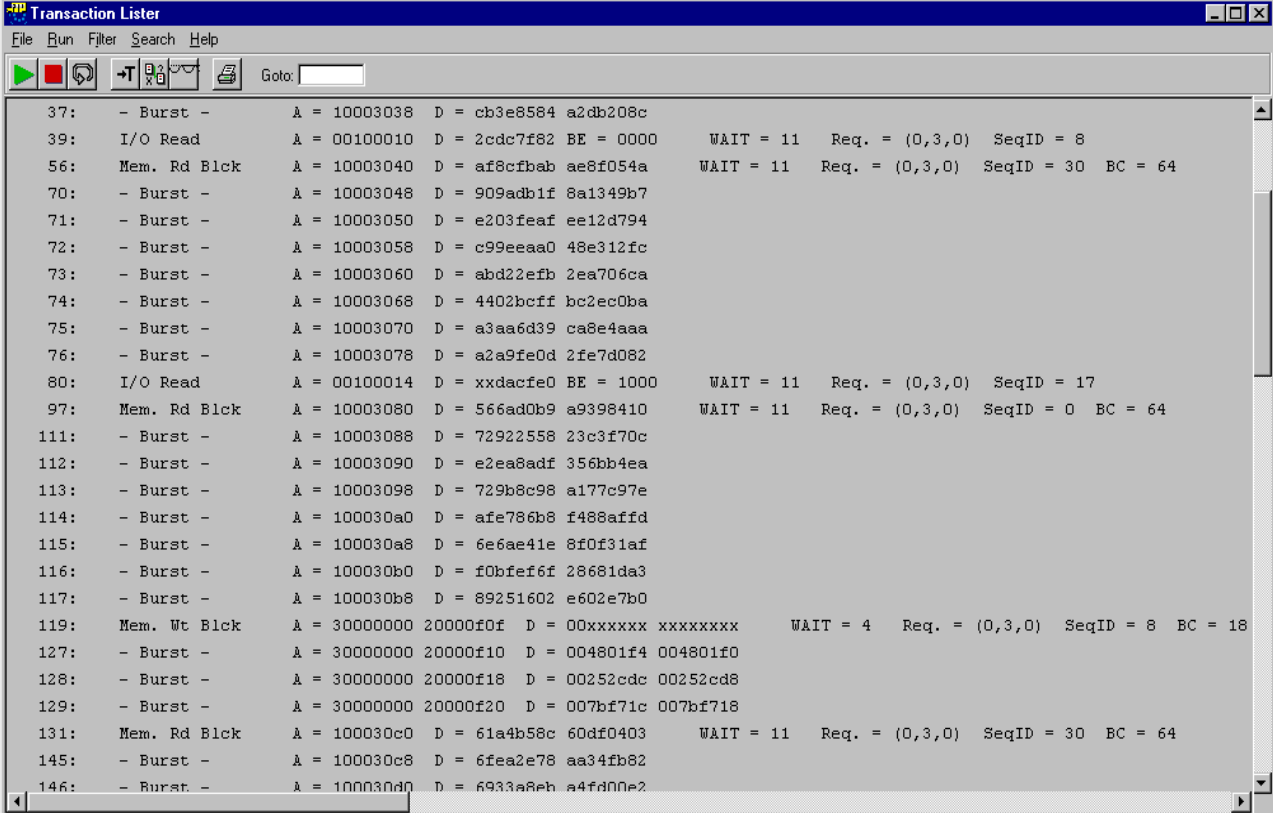
- by clicking the Run button  in the main window, or
- by choosing *Run* from the *Exerciser* menu.

Viewing the Results

Use the Analyzer's Transaction Lister to inspect the results of this test. The waveforms were loaded from the waveform file RIBehavior.wfm..

To open the transaction lister:

- ◆ Click the Transaction Lister button  in the main window (or use the *Transaction Lister* item in the *Analyzer* menu).



Transaction ID	Type	Address (A)	Data (D)	Wait	Requester (Req.)	Sequence ID (SeqID)	Byte Count (BC)
37:	- Burst -	A = 10003038	D = cb3e8584 a2db208c				
39:	I/O Read	A = 00100010	D = 2cdc7f82 BE = 0000	WAIT = 11	Req. = (0,3,0)	SeqID = 8	
56:	Mem. Rd Blck	A = 10003040	D = af8cfbab ae8f054a	WAIT = 11	Req. = (0,3,0)	SeqID = 30	BC = 64
70:	- Burst -	A = 10003048	D = 909adb1f 8a1349b7				
71:	- Burst -	A = 10003050	D = e203feaf ee12d794				
72:	- Burst -	A = 10003058	D = c99eeaa0 48e312fc				
73:	- Burst -	A = 10003060	D = abd22efb 2ea706ca				
74:	- Burst -	A = 10003068	D = 4402bcff bc2ec0ba				
75:	- Burst -	A = 10003070	D = a3aa6d39 ca8e4aaa				
76:	- Burst -	A = 10003078	D = a2a9fe0d 2fe7d082				
80:	I/O Read	A = 00100014	D = xxdacfe0 BE = 1000	WAIT = 11	Req. = (0,3,0)	SeqID = 17	
97:	Mem. Rd Blck	A = 10003080	D = 566ad0b9 a9398410	WAIT = 11	Req. = (0,3,0)	SeqID = 0	BC = 64
111:	- Burst -	A = 10003088	D = 72922558 23c3f70c				
112:	- Burst -	A = 10003090	D = e2ea8adf 356bb4ea				
113:	- Burst -	A = 10003098	D = 729b8c98 a177c97e				
114:	- Burst -	A = 100030a0	D = afe786b8 f488affd				
115:	- Burst -	A = 100030a8	D = 6e6ae41e 8f0f31af				
116:	- Burst -	A = 100030b0	D = f0bfe6f6 28681da3				
117:	- Burst -	A = 100030b8	D = 89251602 e602e7b0				
119:	Mem. Wt Blck	A = 30000000	20000f0f D = 00xxxxxx xxxxxxxx	WAIT = 4	Req. = (0,3,0)	SeqID = 8	BC = 18
127:	- Burst -	A = 30000000	20000f10 D = 004801f4 004801f0				
128:	- Burst -	A = 30000000	20000f18 D = 00252cdc 00252cd8				
129:	- Burst -	A = 30000000	20000f20 D = 007bf71c 007bf718				
131:	Mem. Rd Blck	A = 100030c0	D = 61a4b58c 60df0403	WAIT = 11	Req. = (0,3,0)	SeqID = 30	BC = 64
145:	- Burst -	A = 100030c8	D = 6fea2e78 aa34fb82				
146:	- Burst -	A = 100030d0	D = 6933a8eb a4fd00e2				

The transaction lister shows that the blocks are partitioned into sequences and that the order of blocks differs from that displayed in *"Guided Tour: Generating Requester-Initiator Block Transfer"* on page 23.

Also displayed are:

- the number of waits
- the bus number, the device number and the function of the requester (Req.)
- the tag number (SeqID)
- the byte count (BC) for every transaction

The following table gives an overview of all of the traffic generated by the exerciser:

Lines	Transfer	RI Block	RI Behavior
0	I/O Read bytes 0 ... 2	0	0
17 ... 37	Memory Read Block 64 bytes	1	1
39	I/O Read bytes 3 ... 6	0	2
56 ... 76	Memory Read Block 64 bytes	1	3
80	I/O Read bytes 7 ... 9	0	0
97 ... 117	Memory Read Block 64 bytes	1	1
129 ... 129	Memory Write Block 18 bytes	2	2
131 ... 151	Memory Read Block 64 bytes	1	3
179 ... 191	Memory Write 64 bytes	3	0
193 ... 208	Memory Read Block 17 bytes	1	1
234 ... 244	Memory Write 64 bytes	3	2 Disconnect at ADB
247 ... 252	Memory Write 13 bytes	3	2
256	Memory Write 2 bytes	3	3

Requester-initiator block execution depends not only on the programmed requester-initiator behavior, but also on the interaction with the completer-target. Thus every sequence might need several bus transactions to complete.

Guided Tour: Specifying Completer-Target Behavior

This example illustrates how to set up your Agilent PCI-X testcard as a completer-target device. The behavior of the target is fully programmable. This includes the address ranges that are decoded, how received data is handled, which data is transferred on request, and the protocol behaviors that are used during the transactions.

This guided tour gives an example of setting up the completer-target and is based on the settings of all previous guided tours.

The properties of five behaviors are programmed in the example:

- Behavior 0
 - Decode speed B is selected.
 - The target responds with retry after three wait cycles.
 - The current behavior is repeated three times.
- Behavior 1
 - Decode speed B is selected.
 - The completer-target accepts the data transfer after 3 wait cycles, but responds with a disconnect ADB after subsequent 12 cycles.
- Behavior 2
 - Decode speed B is selected.
 - The completer-target responds with a single data phase disconnect after 8 cycles.
- Behavior 3
 - Decode speed B is selected.
 - The completer-target accepts the data transfer after 3 wait cycles, but responds with a disconnect ADB after subsequent 3 cycles.
- Behavior 4
 - Decode speed C is selected.
 - The completer-target accepts the data transfer after 3 wait cycles, but responds with disconnect ADB after subsequent 3 cycles.
 - The completer-target responds with a single data phase disconnect after 3 cycles.

The following table shows a summary of the specified behaviors:

Behavior	Decode Speed	Initial	Latency	Subseq	SubseqPhase	Repeat
0	B	Retry	3	-	-	3
1	B	Accept	3	Disconnect	12	-
2	B	Single	8	-	-	-
3	B	Accept	3	Disconnect	3	-
4	C	Single	3	-	-	-

For the remaining properties, the default values are used.

Loading the Setup Files

If you were connected to the testcard, the results of the test could be viewed on the screen. However, as you are working in offline mode, you need to start by loading the required files:

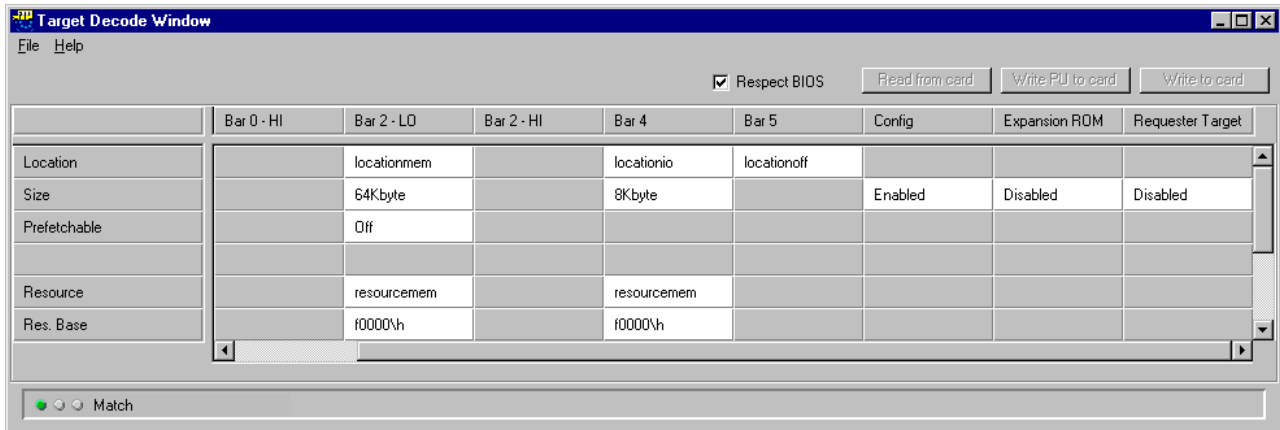
- 1** Load the setup file for this example (CTBehavior.bst) by selecting *Load* from the *File* menu in the main window.
- 2** Load the PCI-X signal waveform file for this example (CTBehavior.wfm) by selecting *Load from file* from the *File* menu in the Waveform Viewer window.

Setting up the Target Decoders

In this guided tour, you will view the prepared target decoder setups.

- ◆ To view the setup for the target decoders, choose the *Target Decode* item in the *Exerciser* menu.

In the Offline/Demo Mode the Target Decode window appears slightly different than the screenshot below.



Base Address Registers The content of the Target Decode window is mainly an interpretation of the configuration space settings. The address space type and location are set in the location field of the respective decoder.

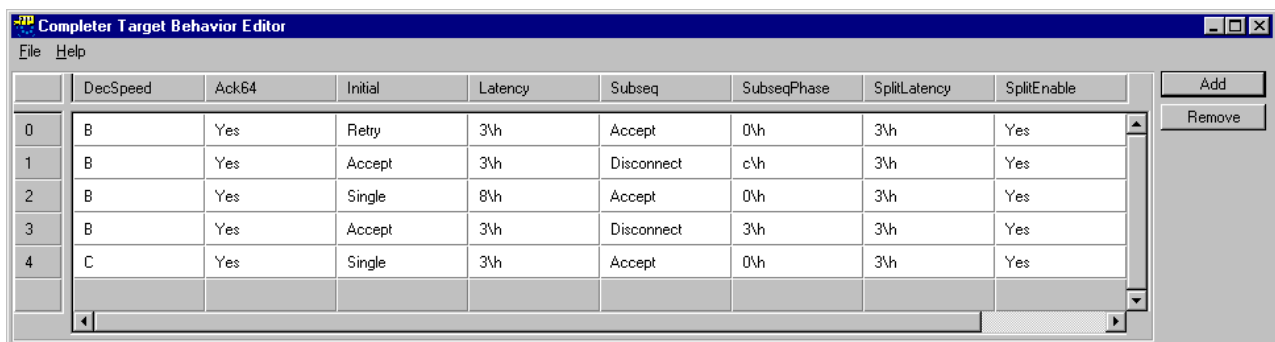
Internal Resources The three rows *Resource*, *Res. Base*, *Res. Size* determine to which internal data resource the decoders are connected to. These settings define how the received data is to be handled or which data is to be sent on request.

Specifying the Completer-Target Protocol Behavior

Once you have loaded the setup file and the waveform file, continue by setting up the completer-target behavior for the test.

- ◆ Open the Completer-Target Behavior Editor dialog box by selecting *Exerciser -> Behaviors -> Completer Target*.

In the Completer-Target Behavior Editor the behaviors for this test are already displayed. They are stored in the setup file (CTBehavior.bst) together with all the other settings.

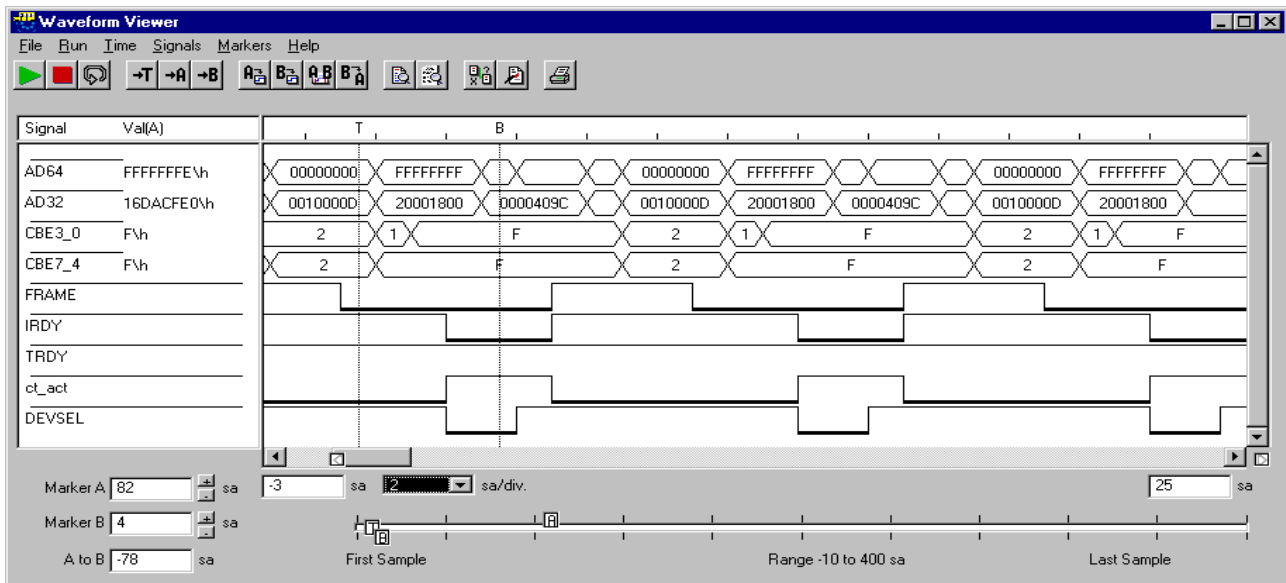


These behaviors specify what happens when the testcard's target is accessed by a requester-initiator.

Viewing the Results

Compare the settings made in the previous step with the traffic captured on the bus.

- ◆ Click the Waveform Viewer button  in the main window (or use the *Waveform Lister* item in the Analyzer menu) to open the waveform viewer.



Notice that the PCI-X Exerciser target behaves exactly as it was set up: The first data phase has 3 wait states (after the trigger point T). Marker B shows the retry.

The behavior can also be easily observed in the transaction lister.

The `ct_act` signal is always high while the testcard's completer-target is active.

The following table gives an overview of all of the traffic generated by the exerciser:

Lines	Transfer	RI Block	RI Behavior
0	I/O Read bytes 0 ... 2	0	0
10	I/O Read bytes 0 ... 2	0	0
20	I/O Read bytes 0 ... 2	0	0
30	I/O Read bytes 0 ... 2	0	1
49	I/O Read bytes 3 ... 6	0	2
68	I/O Read bytes 7 ... 9	0	3
87	Memory Read Block 273 bytes	1	4
106	Memory Read Block 265 bytes	1	0
116	Memory Read Block 265 bytes	1	0
126	Memory Read Block 265 bytes	1	0
136 ... 179	Memory Read Block 265 bytes	1	1
184	Memory Read Block 17 bytes	1	2
203 ... 217	Memory Read Block 9 bytes	1	3
222	Memory Write Block 18 bytes	2	4
234	Memory Write Block 17 bytes	2	0
245	Memory Write Block 17 bytes	2	0
256	Memory Write Block 17 bytes	2	0
267 ... 276	Memory Write Block 17 bytes	2	1
281	Memory Write 130 bytes	3	2
295 ... 312	Memory Write 127 bytes	3	3
317	Memory Write 15 bytes	3	4
330	Memory Write 7 bytes	3	0
340	Memory Write 7 bytes	3	0
350	Memory Write 7 bytes	3	0
360	Memory Write 7 bytes	3	1

Guided Tour: Defining the Transfers that will be given a Split Response

The task in this guided tour is to program the completer-target so that it signals SPLIT RESPONSE on each READ transfer. To perform this task, split decoder 1 is used.

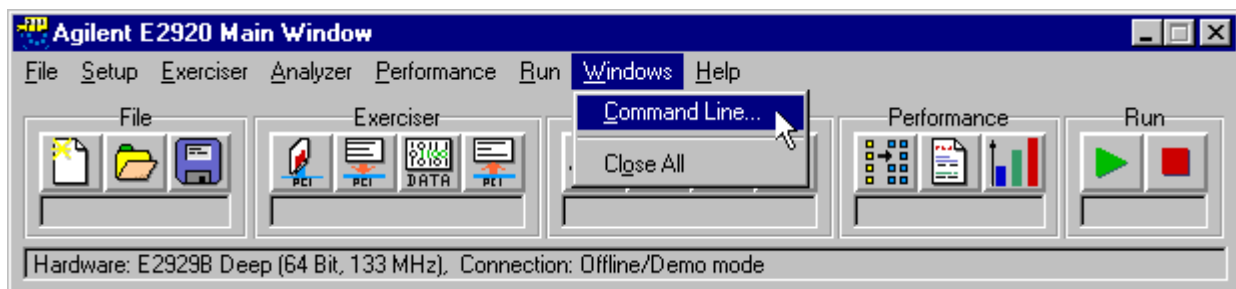
If you are working interactively, the testcard's Command Line Interface is an easy way to program these functions.

The Command Line Interface uses the CLI equivalents of the functions provided by the C-API (option 320) described in detail in the *C-API/PPR Programming Reference*.

Starting the Command Line Interface

To open the command line interface:

- ◆ Select *Command Line Interface* from the *Windows* menu.



This opens the Command Line Interface window.

Entering the Required CLI Commands

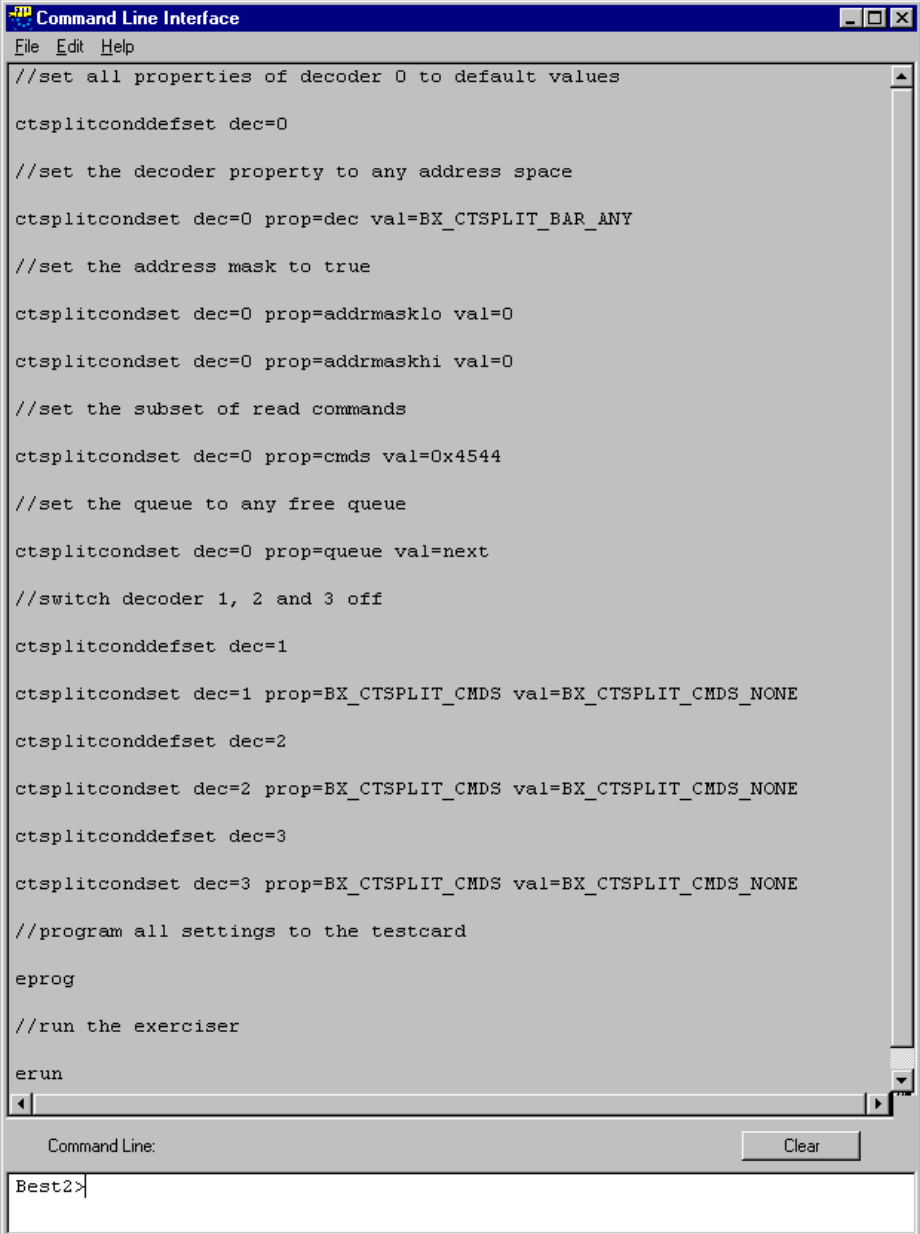
To program the completer-target such that it signals split response on each read transfer, the following steps are required:

1. The properties of split decoder 1 are set to default values.
2. The decoder must be programmed so that all read commands will be split. The read commands have the following coding:
 - 2: I/O Read
 - 6: Memory Read DWord
 - 8: Alias to Memory Read Block
 - 10: Configuration Read
 - 14: Memory Read Block

Because all available commands can be encoded in a 16-bit value, the resulting coding for all read commands is then 0x4544.

3. All conditions (command, decoder, address) must be TRUE for the split decoder to work properly:
 - The address mask must be set to don't care (always true).
 - The address range (decoder) that is being accessed must be set to *any*.
4. The request will be placed in any free queue that is automatically selected.
5. Because the testcard provides four split decoders, the remaining decoders 1, 2 and 3 must be switched off.
6. All settings must be written to the testcard.

Entering the required commands results in the following screenshot:



```
Command Line Interface
File Edit Help

//set all properties of decoder 0 to default values

ctsplitconddefset dec=0

//set the decoder property to any address space

ctsplitcondset dec=0 prop=dec val=BX_CTSPLIT_BAR_ANY

//set the address mask to true

ctsplitcondset dec=0 prop=addrmasklo val=0

ctsplitcondset dec=0 prop=addrmaskhi val=0

//set the subset of read commands

ctsplitcondset dec=0 prop=cmds val=0x4544

//set the queue to any free queue

ctsplitcondset dec=0 prop=queue val=next

//switch decoder 1, 2 and 3 off

ctsplitconddefset dec=1

ctsplitcondset dec=1 prop=BX_CTSPLIT_CMDS val=BX_CTSPLIT_CMDS_NONE

ctsplitconddefset dec=2

ctsplitcondset dec=2 prop=BX_CTSPLIT_CMDS val=BX_CTSPLIT_CMDS_NONE

ctsplitconddefset dec=3

ctsplitcondset dec=3 prop=BX_CTSPLIT_CMDS val=BX_CTSPLIT_CMDS_NONE

//program all settings to the testcard

eprog

//run the exerciser

erun

Command Line: Clear

Best2>
```

With the command line interface you can also log and run CLI scripts. Hence, you do not need to type all commands by hand for every test. For providing full flexibility and control on your system, use the C Application Programmer's Interface (C-API) (Option 320). With the C-API you can implement your own C programs to access all functions on the testcard.

Guided Tour: Specifying Completer-Initiator Behavior

For initiation of split transaction completion, completer-initiator behaviors describe **how** the data is completed.

This guided tour gives an example of setting up the completer-initiator behaviors and is based on the settings of all previous guided tours.

The properties of three behaviors are programmed as follows:

- Behavior 0
 - Two address steps between the assertion of GNT# and the assertion of FRAME# plus two clock cycles are inserted.
 - A 64-bit data transfer request is asserted.
- Behavior 1
 - Two address steps between the assertion of GNT# and the assertion of FRAME# plus two clock cycles are inserted.
 - 64 clock cycles between transactions are inserted.
- Behavior 2
 - Four address steps between the assertion of GNT# and the assertion of FRAME# plus two clock cycles are inserted.
 - 64 clock cycles between transactions are inserted.
 - A 64-bit data transfer request is asserted.

The following table shows the summary of the specified behaviors:

Behavior	Delay	Steps	Req64
0	-	2	Yes
1	64	2	No
2	64	4	Yes

Loading the Setup Files

If you were connected to the testcard, the results of the test could be viewed on the screen. However, as you are working in offline mode, you need to start by loading the required files:

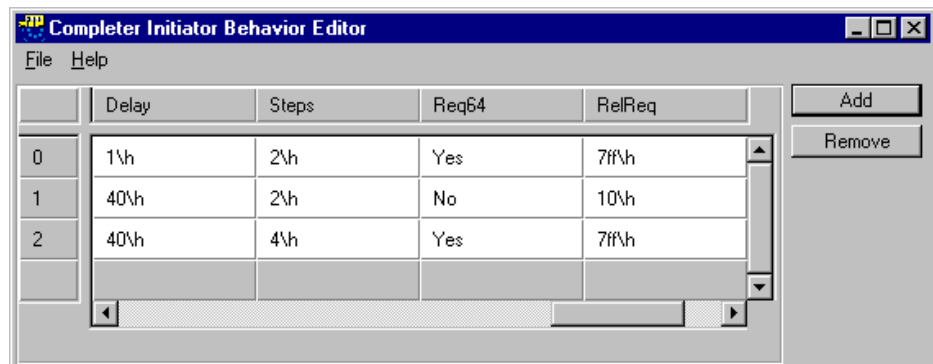
- 1 Load the setup file for this example (CIBehavior.bst) by selecting *Load* from the *File* menu in the main window.
- 2 Load the PCI-X signal waveform file for this example (CIBehavior.wfm) by selecting *Load from file* from the *File* menu in the Waveform Viewer window.

Setting up the Completer-Initiator Behaviors

Once you have loaded the setup file and the waveform file, continue with setting up the requester-initiator behavior for the test.


- ◆ Open the Completer-Initiator Behavior Editor dialog box by selecting *Exerciser -> Behaviors -> Completer Initiator*.

In the Completer-Initiator Behavior Editor you see that the behaviors for this test are already displayed. They are stored in the setup file (CIBehavior.bst) together with all the other settings.



Running the Test


If you were connected to a testcard, you would start the test:

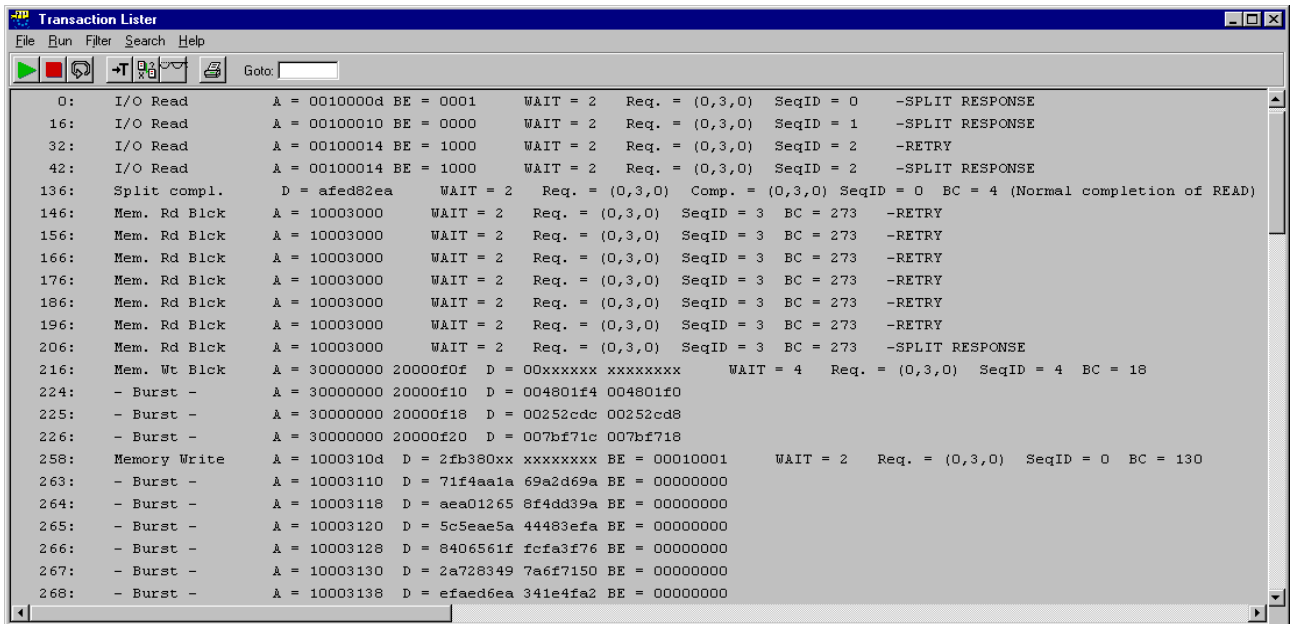
- by clicking the Run button  in the main window, or
- by choosing *Run* from the *Exerciser* menu.

Viewing the Results

Use the Analyzer's Transaction Lister to inspect the results of this test. The waveforms were loaded from the waveform file CIBehavior.wfm..

To open the Transaction Lister:

- ◆ Click the Transaction Lister button  in the main window (or use the *Transaction Lister* item in the *Analyzer* menu) to open the transaction lister.



Transaction ID	Operation	Address (A)	BE	WAIT	Req.	SeqID	BC	Notes
0:	I/O Read	A = 0010000d	BE = 0001	WAIT = 2	Req. = (0,3,0)	SeqID = 0		-SPLIT RESPONSE
16:	I/O Read	A = 00100010	BE = 0000	WAIT = 2	Req. = (0,3,0)	SeqID = 1		-SPLIT RESPONSE
32:	I/O Read	A = 00100014	BE = 1000	WAIT = 2	Req. = (0,3,0)	SeqID = 2		-RETRY
42:	I/O Read	A = 00100014	BE = 1000	WAIT = 2	Req. = (0,3,0)	SeqID = 2		-SPLIT RESPONSE
136:	Split compl.	D = afed82ea		WAIT = 2	Req. = (0,3,0)	Comp. = (0,3,0)	SeqID = 0 BC = 4	(Normal completion of READ)
146:	Mem. Rd Blck	A = 10003000		WAIT = 2	Req. = (0,3,0)	SeqID = 3	BC = 273	-RETRY
156:	Mem. Rd Blck	A = 10003000		WAIT = 2	Req. = (0,3,0)	SeqID = 3	BC = 273	-RETRY
166:	Mem. Rd Blck	A = 10003000		WAIT = 2	Req. = (0,3,0)	SeqID = 3	BC = 273	-RETRY
176:	Mem. Rd Blck	A = 10003000		WAIT = 2	Req. = (0,3,0)	SeqID = 3	BC = 273	-RETRY
186:	Mem. Rd Blck	A = 10003000		WAIT = 2	Req. = (0,3,0)	SeqID = 3	BC = 273	-RETRY
196:	Mem. Rd Blck	A = 10003000		WAIT = 2	Req. = (0,3,0)	SeqID = 3	BC = 273	-RETRY
206:	Mem. Rd Blck	A = 10003000		WAIT = 2	Req. = (0,3,0)	SeqID = 3	BC = 273	-SPLIT RESPONSE
216:	Mem. Wt Blck	A = 30000000	20000f0f	D = 00xxxxxx xxxxxxxx	WAIT = 4	Req. = (0,3,0)	SeqID = 4 BC = 18	
224:	- Burst -	A = 30000000	20000f10	D = 004801f4 004801f0				
225:	- Burst -	A = 30000000	20000f18	D = 00252cdc 00252cd8				
226:	- Burst -	A = 30000000	20000f20	D = 007bf71c 007bf718				
258:	Memory Write	A = 1000310d	D = 2fb380xx xxxxxxxx	BE = 00010001	WAIT = 2	Req. = (0,3,0)	SeqID = 0 BC = 130	
263:	- Burst -	A = 10003110	D = 71f4aa1a 69a2d69a	BE = 00000000				
264:	- Burst -	A = 10003118	D = aea01265 8f4dd39a	BE = 00000000				
265:	- Burst -	A = 10003120	D = 5c5eae5a 44483efa	BE = 00000000				
266:	- Burst -	A = 10003128	D = 8406561f fcf3f76	BE = 00000000				
267:	- Burst -	A = 10003130	D = 2a728349 7a6f7150	BE = 00000000				
268:	- Burst -	A = 10003138	D = efaed6ea 341e4fa2	BE = 00000000				

The transaction lister shows that a split response is given to all read transfers (as programmed in “*Guided Tour: Defining the Transfers that will be given a Split Response*” on page 40).

In this transaction lister, you can see that both split completion transactions and normal completion are executed. For split completion transactions, completer-initiator behaviors are used, for normal completion transactions, requester-initiator behaviors are used.

Split transaction execution depends not only on the programmed completer-initiator behavior, but also on the interaction with the requester-target, so that every sequence might need several bus transactions to complete.

The following table gives an overview of all of the traffic generated by the exerciser:

Lines	Transfer	CI Behavior
136	Split Completion 4 bytes	0
355	Split Completion 4 bytes	1
505	Split Completion 4 bytes	2
806 ... 878	Split Completion 273 bytes	0

Guided Tour: Specifying Requester-Target Behavior

This example illustrates how to set up your Agilent PCI-X testcard as a requester-target device. The behavior of the requester-target is fully programmable. This includes the address ranges that are decoded, how received data is handled, which data is transferred on request, and the protocol behaviors that are used during the transactions.

This guided tour gives an example of setting up the requester-target behaviors and is based on the settings of all previous guided tours.

The properties of four behaviors are programmed as follows:

- Behavior 0
 - Decode speed A is selected.
 - The completer-target responds with a single data phase disconnect after 8 cycles.
- Behavior 1
 - Decode speed B is selected.
 - The target responds with retry after six wait cycles.
 - The current behavior is repeated three times.
 - The 64-bit data transfer acknowledge is asserted.

- Behavior 2
 - Decode speed B is selected.
 - The requester-target accepts the data transfer after 3 wait cycles, but responds with disconnect ADB after subsequent 3 cycles.
 - The 64-bit data transfer acknowledge is asserted.
- Behavior 3
 - Decode speed A is selected.
 - The requester-target responds after 3 wait cycles.

The following table shows a summary of the specified behaviors:

Behavior	Decode Speed	Ack64	Initial	Latency	Subseq	SubseqPhase	Repeat
0	A	No	Single	8	-	-	-
1	B	Yes	Retry	6	-	-	3
2	B	Yes	Accept	3	Disconnect	3	-
3	A	No	Accept	3	-	-	-

For the remaining properties, default values are used.

Loading the Setup Files

If you were connected to the testcard, the results of the test could be viewed on the screen. However, as you are working in offline mode, you need to start by loading the required files:

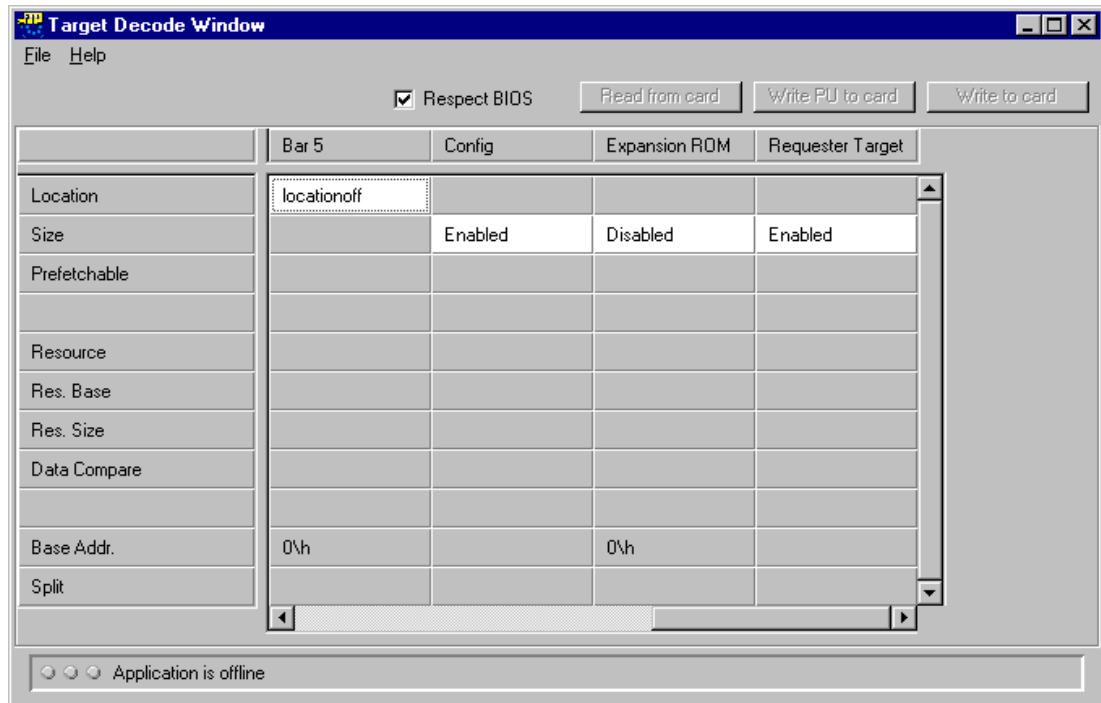
- 1** Load the setup file for this example (RTBehavior.bst) by selecting *Load* from the *File* menu in the main window.
- 2** Load the PCI-X signal waveform file for this example (RTBehavior.wfm) by selecting *Load from file* from the *File* menu in the Waveform Viewer window.

Setting up the Requester-Target Decoder

For this guided tour, you just need to enable the requester-target decoder in the Target Decode window.

- ◆ To open the Target Decode window, choose the *Target Decode* item in the *Exerciser* menu.

The prepared target decoder setups are displayed as follows.

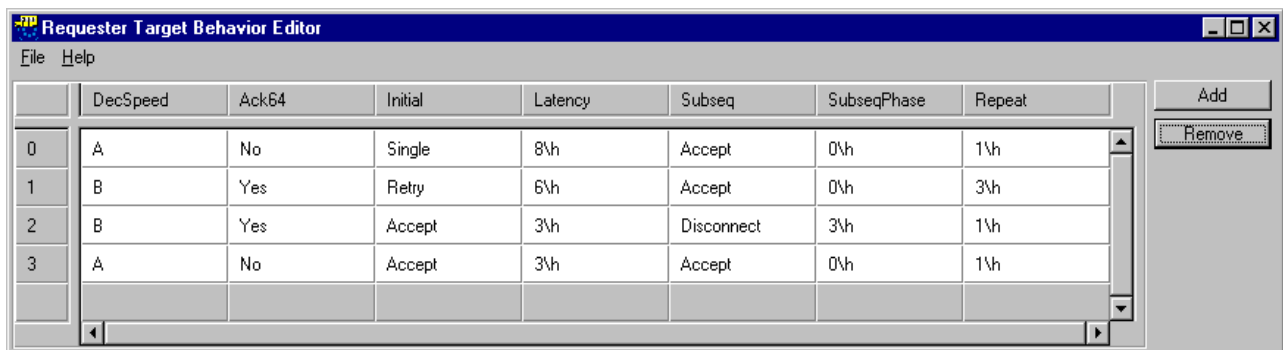


Specifying the Requester-Target Protocol Behavior

Once you have loaded the setup file and the waveform file, continue with setting up the completer-target behavior for the test.

- ◆ Open the Requester-Target Behavior Editor dialog box by selecting *Exerciser -> Behaviors -> Requester Target*.

In the Requester-Target Behavior Editor you see that the behaviors for this test are already displayed. They are stored in the setup file (RTBehavior.bst) together with all the other settings.

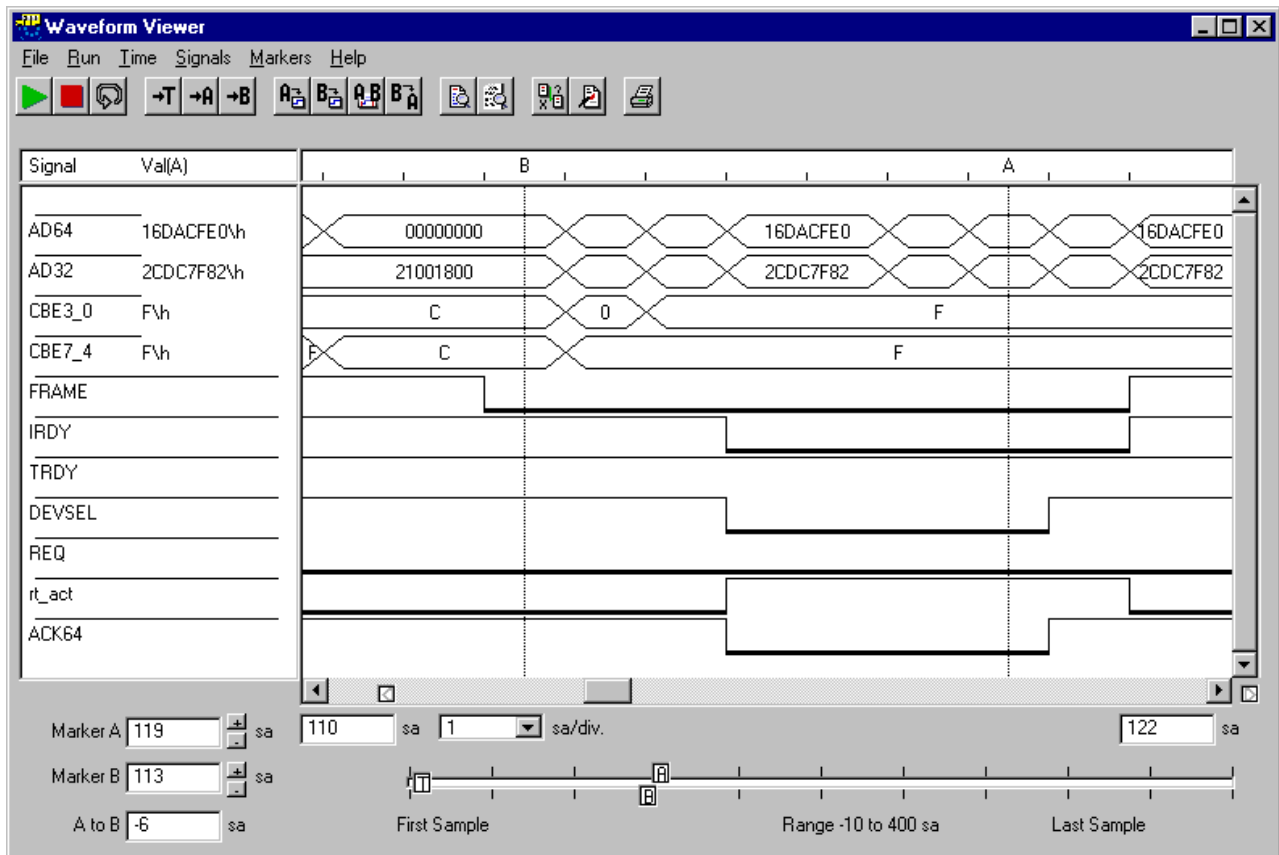


These behaviors specify what happens when the testcard's requester-target is accessed by a completer-initiator.

Viewing the Results

Use the Analyzer's Waveform Viewer to inspect the results of this test. The waveforms were loaded from the waveform file RTBehavior.wfm.

- 1 Click the Waveform Viewer button  in the main window (or use the *Waveform Lister* item in the Analyzer menu) to open the waveform viewer.



Notice that the PCI-X Exerciser target behaves exactly as it was set up, for example, take a look at behavior 1:

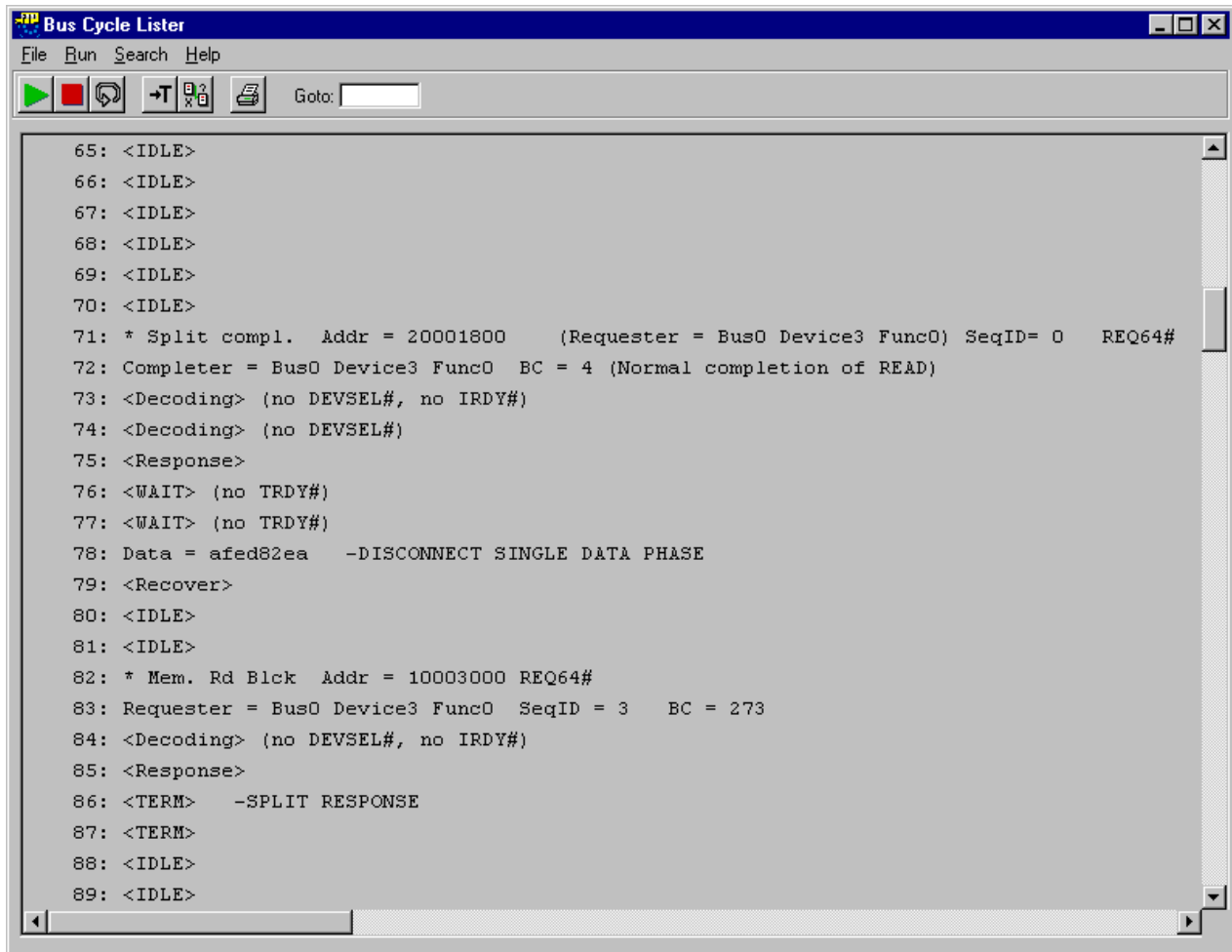
- The data phase has 6 wait states after initiation of the split completion (signed with marker B).
- The second data phase does not complete until it is retried once.

Marker A shows the retry.

The `rt_act` signal is always high while the testcard's completer-target is active.

To see the corresponding section in the Bus Cycle Lister window:

- 2 Click the Bus Cycle Lister button  in the main window (or use the *Bus Cycle Lister* item in the *Analyzer* menu) to open the bus cycle lister.



The behavior can also be easily observed in the transaction lister.

The following table gives an overview of all of the traffic generated by the exerciser:

Lines	Transfer	RT Behavior
71	Split Completion 4 bytes	0
113	Split Completion 4 bytes	1
125	Split Completion 4 bytes	1
137	Split Completion 4 bytes	1
149	Split Completion 4 bytes	2
149	Split Completion 4 bytes	3
224	Split Completion 4 bytes	0
237	Split Completion 4 bytes	1
249	Split Completion 4 bytes	1
261	Split Completion 4 bytes	1
273 ... 292	Split Completion 269 bytes	2 Disconnect at ADB
297 ... 340	Split Completion 145 bytes	3

The PCI-X Exerciser as a Requester-Initiator Device

The Exerciser of your Agilent E2929/E2930 testcard can simulate any device on the PCI-X bus under test. On the whole, there are two different types of devices—initiators and targets. Other devices (for example, network interface cards) have both initiator and target functionality.

The initiator is divided into requester-initiator and completer-initiator, the target is divided into completer-target and requester-target.

All necessary information about the testcard's **requester-initiator device** is provided here.

Programming Transactions A device on a PCI-X bus is called a requester-initiator if it requests to access the bus and performs data transfers when it has been granted access to the bus. A data transfer consists of one or more transactions. Information on the programming of **requester-initiator transactions** and their **properties** is located in “*Programming Requester-Initiator Transactions*” on page 54.

Requester-Initiator Behaviors Additionally, the Agilent PCI-X Exerciser allows to control the protocol behavior of its requester-initiator device. These protocol **behaviors** normally do not have any effect on the result of a transaction. They merely determine the behavior of the requester-initiator in terms of different types of delays that get inserted, transaction terminations, and so on. Programming these behaviors is described in “*Programming the Requester-Initiator Behaviors*” on page 63.

Run Options When the requester-initiator device is completely set up, you can run it using several options. These are explained in “*Starting the Requester-Initiator*” on page 66.

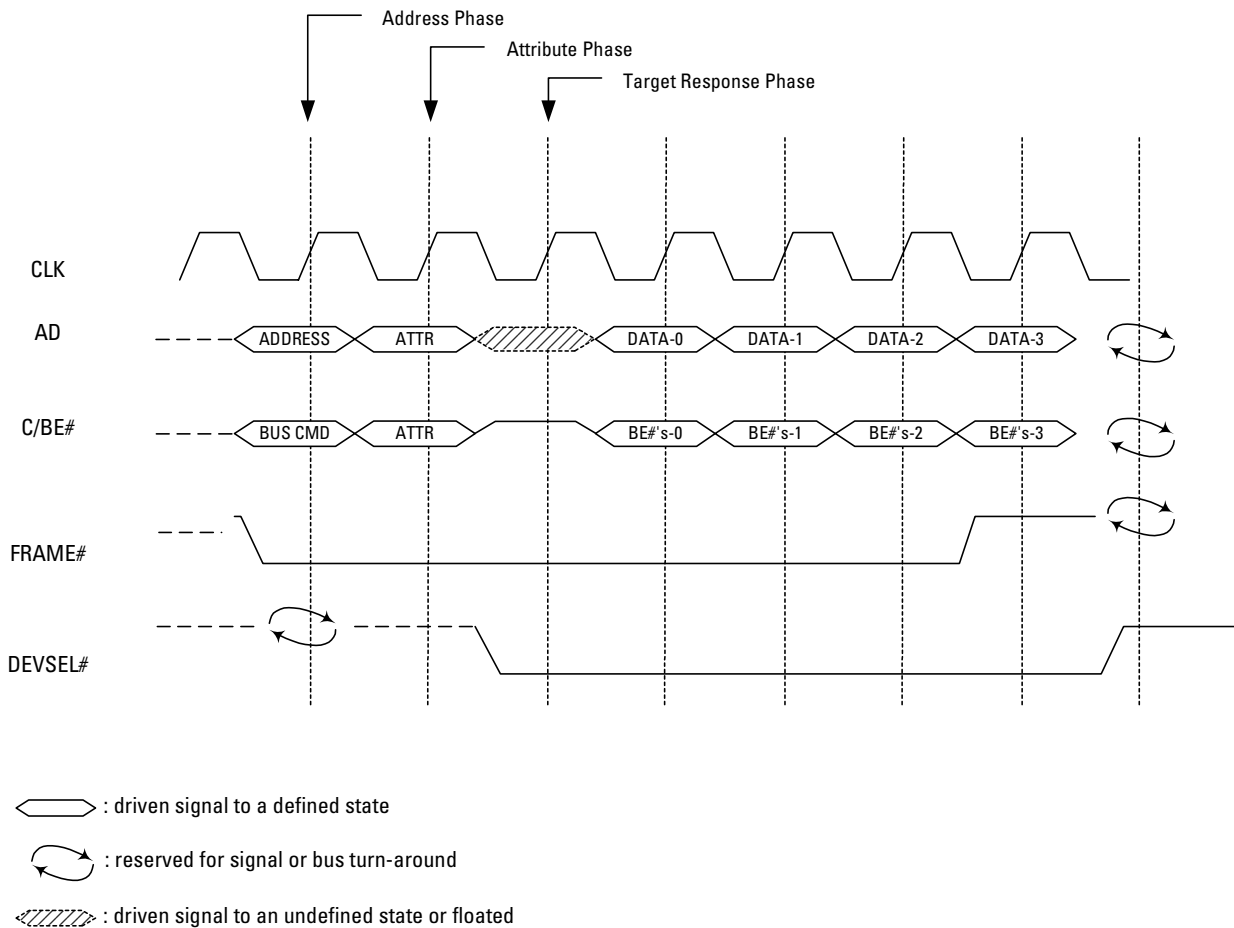
Programming Requester-Initiator Transactions

In order to set up the Agilent PCI-X Exerciser as a requester-initiator, you need to specify **requester-initiator transactions**. When the requester-initiator is started, it performs these transactions on the bus. For this purpose it requests the use of the bus from the PCI-X bus arbiter. When the bus is granted to the requester-initiator, it initiates the data transfers on the bus. For more details, refer to “*Requester-Initiator Transactions Overview*” on page 55.

For each transaction, you can specify transaction properties such as a bus command or a bus address (see “*Available Transaction Properties*” on page 59) that are valid for the whole transaction. To specify transactions, the Agilent E2929/E2930 testcard provides the Requester-Initiator Block Editor dialog box.

NOTE These transaction definitions basically define *what* is to be done by the requester-initiator, that is, they specify the data that is to be transferred from one location to the other. Compared with that, the **requester-initiator behaviors** specify *how* the transaction is to be performed (see “*Programming the Requester-Initiator Behaviors*” on page 63).

Requester-Initiator Transactions Overview



The figure shows a typical PCI-X Mode 1 write transaction, consisting of:

- One address phase.
- One attribute phase.
- One target response phase (where the signal is in an undefined state or floated).
- One or more data phases.

According to the number of data phases, two transaction types can be distinguished: **burst transactions** and **DWORD transactions** (see “*Burst and DWORD Transactions*” on page 58).

For the requester-initiator transactions, you can specify any combination of single transactions and block transactions.

Transaction Phases

Address Phases In the address phase the requester-initiator asserts the `FRAME#` signal and drives the address of the target device on the bus along with the bus command. All target devices on the bus latch and decode the address.

The bus commands are typically of one of the following types:

- Memory Read DWORD or Memory Write
- I/O Read or Write
- Configuration Read or Write
- Memory Read or Write block
- Split Completion
- Interrupt Acknowledge
- Alias to Memory Read or Write Block
- Device ID Message
- Reserved Cycle

Attribute Phase Following the address phase(s) comes the attribute phase. In the attribute phase the requester-initiator drives the attributes on the bus along with the byte count and the number of byte enables.

The following table shows the available attributes and how they can be controlled:

Attribute	Controlled By
Relaxed Ordering	Requester-initiator block property <i>RelaxOrder</i> .
No Snoop	Requester-initiator block property <i>NoSnoop</i> .
Reserved Bit 31	Requester-initiator block property <i>Reser31</i> .
Function Number	Cannot be controlled (is fixed at 0).
Device Number	Configuration space.
Bus Number	
Tag	Requester-initiator behavior property <i>Tag</i> .

The following table shows how the byte count and the number of byte enables can be controlled.

Further Transaction Characteristics	Controlled By
byte count	Requester-initiator block property <i>NumBytes</i> and requester-initiator behavior property <i>ByteCount</i> (and possible previous target terminations).
Number of byte enables	Requester-initiator block property <i>Byten</i> . Starting address and (byte count -1) are not included in the sequence (only valid for write transactions).

For more information about the block and behavior properties, refer to the *Agilent E2929/E2930 Windows and Dialog Boxes Reference*.

Target Response Phase In the target response phase, the target that detects the address as one in its own address range, replies with a device select *DEVSEL#* signal.

Data Phases After the addressed target signals that it is ready for the transaction (by asserting *TRDY#*), the first data phase takes place. In the data phases the data is transferred. If the bus command was a write command, the data is sent by the requester-initiator. In case of a read command, the data is sent by the target.

There can be more than one data phase in a transaction. Transactions with several data phases are commonly referred to as bursts. The way in which a transaction is eventually performed depends also on the protocol and the device properties.

In **PCI-X Mode 2**, initiators of burst push transactions other than Memory Write use source-synchronous clocking to reach the highest possible data rates. The initiator drives data at two times (DDR) or four times (QDR) the common clock frequency.

With source-synchronous clocking, the initiator also drives strobes to be used for latching the data at the destination. In PCI-X Mode 2, the *C/BE#* bus is used for these data strobes. Thus, it only contains valid byte enables for the data phases of Memory Write transactions.

The Agilent E2930 testcard currently supports DDR only.

Burst and DWORD Transactions

PCI-X supports transactions with one or more data phases. According to the command used in a transaction, there can be two different types of transactions:

- Burst transactions
- DWORD transactions

Burst Transactions Transactions that use memory commands (except Memory Read DWORD) are burst transactions, and are permitted to have any number of data phases.

The following commands specify a burst transaction. (The term used in the requester-initiator block editor is given in parentheses.)

- Memory Read Block (*MemReadBlock*)
- Memory Write Block (*MemWriteBlock*)
- Memory Write (*MemWrite*)
- Alias to Memory Read Block (*AliasMemReadBlock*)
- Alias to Memory Write Block (*AliasMemWriteBlock*)
- Split Completion (*Split*)
- Device ID Message (*DevIdMsg*)

DWORD Transactions Transactions that use commands (except for memory commands) are DWORD transactions.

DWORD transactions are restricted to:

- a single data phase (an aligned DWORD or less) and
- a 32-bit transaction (REQ64# must be deasserted).

The following commands specify a DWORD transaction. (The term used in the requester-initiator block editor is given in parentheses.)

- Interrupt Acknowledge (*IntAck*)
- I/O Read (*IoRead*)
- I/O Write (*IoWrite*)
- Configuration Read (*ConfigRead*)
- Configuration Write (*ConfigWrite*)
- Memory Read DWORD (*MemReadDWord*)

Available Transaction Properties

The transaction properties are stored in the requester-initiator block transfer memory on the testcard. They are also referred to as requester-initiator block properties. They hold parameters such as the bus command and the addresses from and to which data is to be sent. Some of these properties must be specified, others use default values if omitted.

Each transaction has the following programmable properties. (The syntax used in the requester-initiator block editor is given in parentheses.)

- Bus Command (*Command*)

This property is always required and specifies the PCI-X bus command used for the transaction. The possible choices are memory read or write block, memory write, alias to memory read block or write block, I/O read or write and memory read DWORD. Additionally, you can drive interrupt acknowledge, configuration read or write and a split command on the bus.

- Bus Address (*AD32, AD64*)

The bus address also is required for every transaction. It can be 32 or 64 bits. Certain bus commands require aligned PCI-X bus addresses, for example memory read DWORD (see “*Burst and DWORD Transactions*” on page 58).

- Number of Bytes (*NumBytes*)

This property specifies the amount of data that is transferred by the requester-initiator. The amount of data transferred in one block can be up to 4 GByte.

- Compare Flag (*DataCmp*)

Depending on the property *Resource*, real-time data compare can be performed on the internal memory or the data generator:

- Internal memory as resource:

If the optional compare flag is set, data read from another device is not stored in the testcard's data memory but is compared with data stored previously.

- Data generator as resource:

The generator calculates the expected data and compares it with incoming data.

For more information about the data compare feature, refer to “*Real-Time Data Compare*” on page 118.

- Conditional Start (*CondStart*)

This property defines if block execution starts unconditionally or if a conditional start pattern must have occurred:

- *No*

Unconditional start.

- *Once1*

After the exerciser has been started, block execution waits until the conditional start pattern 1 has occurred at least once.

- *Wait1*

After the end of the previous requester-initiator sequence, block execution waits until the conditional start pattern 1 has occurred.

- *Once2*

After the exerciser has been started, block execution waits until the conditional start pattern 2 has occurred at least once.

Pattern 1 and pattern 2 can be programmed with the Command Line Interface (CLI).

- Outstanding Request Completion (*Completion*)

Setting this flag causes the block execution to wait until all outstanding requests have been completed.

- Byte Enable Control (*Byten*).

With the parameter *Byten*, you can define fixed byte enable values for a transaction.

For a block that is read, this property is ignored. At the beginning and at the end of a sequence, the byte enables are masked. That means, the starting address and (byte count -1) are not included in the sequence.

Use the Command Line Interface (CLI) to store sequences of byte enable values in the byte enable memory (see “*Using the Command Line Interface*” on page 125).

- Internal Address (*IntAddr*).

This address points either to the testcard’s internal memory or to the start value of the data generator. (The data resource can be determined by the resource property.)

During write transactions, the data supplied by the memory or the data generator is sent to the target. In read transactions, the received data is stored in the internal memory. When doing a data compare (*DataCmp* flag set), this address does not have any effect.

See also “*Using Data Resources*” on page 109.

- Transfer Queue (*Queue*)

With this parameter you can determine into which queue the transaction is put: queue *A*, queue *B* and *Auto*, where *Auto* selects any free queue.

Setting this property allows you to determine if the blocks will be executed concurrently or in any combination of sequential and concurrent execution.


How the blocks leave the two different queues, is controlled with the corresponding requester-initiator behavior property (see “*Programming the Requester-Initiator Behaviors*” on page 63).

For information on the properties *RelaxOrder*, *NoSnoop* and *Reser31*, refer to *Agilent E2929/E2930 Windows and Dialog Boxes Reference*.

How to Specify Requester-Initiator Transactions

To specify transactions and properties, the Agilent PCI-X Exerciser provides the Requester-Initiator Block Editor.

To open the Requester-Initiator Block Editor:

- ◆ Click the Requester-Initiator (RI) Transactions button , or select the *Transactions* item from the *Exerciser* menu in the main window.

You can set up a maximum of 256 programmable transactions.

Add and Remove Transactions To add one more block line below the last for editing:

- ◆ Click the *Add* button.

To remove the last block line for editing:

- ◆ Click the *Remove* button.

Transaction Settings To change a setting:

- ◆ Click the corresponding field in the table.

Either a check box, a selection list or an editable text field comes up, allowing you to modify the setting.

Programming the Requester-Initiator Behaviors

While the block properties define *what* the requester-initiator does, the requester-initiator behaviors allow you to specify *how* it does it.

More specifically, the **requester-initiator behaviors** describe all properties of a bus phase during a data transfer. Examples are the number of wait states that the requester-initiator inserts into a data phase, or whether a parity error should be signaled during an address phase and so on.

The specified requester-initiator behaviors do not affect the result of the data transfer. Thus, you can repeat a test using the same requester-initiator transaction settings while varying the behavior settings, and compare the two results.

For the latter case, the Agilent PCI-X Exerciser provides a **requester-initiator behavior editor** to set up additional parameters that define the protocol behavior of the requester-initiator device.

Requester-Initiator Behavior Memory

All these behaviors are stored in the requester-initiator behavior memory. In this memory the behaviors are kept in memory lines, and every memory line is then assigned to a transaction.

You can set up a maximum of 256 programmable sequence-level behaviors, which are successively used for every new sequence.

Requester-Initiator Behavior Editor

To set up these requester-initiator behaviors, the PCI-X Exerciser provides the Requester-Initiator Behavior Editor window.

To open the Requester-Initiator Behavior Editor:

- ◆ Select *Behavior* → *Requester-Initiator* from the *Exerciser* menu in the main window.

Programming the Behaviors

Setting up and programming the behaviors in the requester-initiator behavior editor dialog box works in similar way to the transaction editor dialog box.

Available Requester-Initiator Behaviors

Each requester-initiator behavior has the following programmable properties. (The term used in the requester-initiator behavior editor is given in parentheses.)

- Select the requester-initiator queue A or B (*Queue*)
By setting this property and the corresponding block property you can determine if blocks get executed in order or if they bypass each other
- Select a fixed or a automatically tag number (*Tag*)
- Define the byte count for the sequence (1 ... 4096) (*ByteCount*)
This property partitions the value of the block property *NumBytes* into sequences of a maximum length of 4096 bytes.
- Disconnect at ADB number N (*Disconnect*)
This property allows you to break a sequence into multiple transactions. The requester-initiator disconnects at every N-th allowable disconnect boundary (ADB). Typically, the requester-initiator will not disconnect, because the data transfer will have been completed when it requests a transaction.
When the RI disconnects a sequence, it resumes the disconnected sequence. It is not possible to execute some other action in between.
- Clock delay before assertion of REQ# (*Delay*)
This property allows you to vary latencies between transactions. Sometimes the minimum achievable latency to the next RI transaction is restricted by the most recent event and sometimes by the data path configuration.
For more information about how to determine the most appropriate data path configuration, refer to “*Latencies between Requester-Initiator Transactions*” on page 65.
- Number of address steps (*Steps*)
The number of address steps is the number of clock cycles between the assertion of GNT# and the assertion of FRAME# plus two clock cycles, which are designed into the register-to-register interface of PCI-X.

- Request 64-bit wide data transfer (Y/N) (*Req64*)
- Release REQ# N clocks after the address phase (*RelReq*)
- Number of repeats (*Repeat*)
The current behavior is repeated N times before the next behavior is used.
- Usage of skip registers (*Skip*)
Use the content of a 'skip register' and add it to the address while doing repeats of this behavior (selection of 8 skip registers is possible).

For more information about the requester-initiator behaviors, refer to the *Agilent E2929/E2930 Windows and Dialog Boxes Reference*.

Latencies between Requester-Initiator Transactions

If the data path configuration is set appropriately, there is always at least one way of achieving the fastest possible time between requester-initiator transactions.

The following table helps you determine a data path configuration where *Fast* denotes the highest rate allowed by the PCI-X specification or one clock slower, and *Slow* means a rate about 10 to 20 clocks.

Minimum Idles between the Most Recent Event and the Subsequent Transaction		Subsequent RI Transaction	
		RI write	RI read
Most Recent Event	RI write	Fast	Fast
	RI read	Fast, see Table 1.	Fast
	Retry from same target	Fast	Fast
	Disconnect from target	Fast	Fast
	Write to requester-target	Fast, see Table 2.	Fast
	Write to target	Fast, see Table 2.	Fast
	Read from target	Fast, see Table 3.	Fast

Table 1 Latency for a RI Write after a RI Read

Latency for a RI Write after a RI Read		RI Write	
		Data-out = Mem	Data-out = Gen
RI read	Data-in = Mem	Slow	Fast
	Data-in = Mem Compare	Fast	Fast
	Data-in = Gen Compare	Fast	Fast

Table 2 Latency for a RI Write after a Write to the Requester-Target / Completer-Target

Latency for RI Write after Write to Requester-Target / Completer-Target		RI Write	
		Data-out = Mem	Data-out = Gen
Requester-Target / Target	Data-in = Mem	Slow	Fast
	Data-in = Mem Compare	Fast	Fast
	Data-in = Gen Compare	Fast	Fast

Table 3 Latency for a RI Write after a Write from the Completer-Target

Latency for RI Write after Write from Target		RI Write	
		Data-out = Mem	Data-out = Gen
Read from Target	Data-out = Mem	Slow	Fast
	Data-out = Gen	Fast	Fast

Starting the Requester-Initiator

When you have completely set up the requester-initiator, you can prepare to run your test:

- You can exactly specify the conditions for starting and running the requester-initiator (see “*Preparing Test Execution*” on page 66).
- You have several ways to start the requester-initiator (see “*Running the Requester-Initiator*” on page 68).
- If required, you can stop the requester-initiator manually (see “*Stopping the Requester-Initiator*” on page 73).

Preparing Test Execution

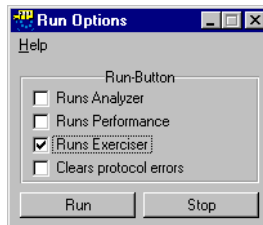
You have several possibilities to run the test:

- You can start the test manually by means of the Run button or the *Start* item from the *Run* menu. You can use the Run Options to determine the testcard components to be started (see “*Setting up the Run Options*” on page 67).
- You can specify a start condition to start the initiator automatically after a certain event (see “*Specifying a Start Condition*” on page 67).

Setting up the Run Options

First of all, you need to decide whether you want to run the Exerciser alone or in combination with other parts of the Agilent E2929/E2930 testcard. These settings are made in the Run Options window:

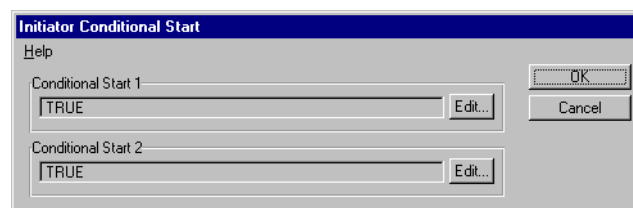
- 1 In the main window, select *Run Options* from the *Run* menu. The Run Options dialog box opens.



- 2 Select the testcard components to be started when using the Run button or the *Start* item from the *Run* menu.
Only components that are enabled with your testcard can be selected.

Specifying a Start Condition

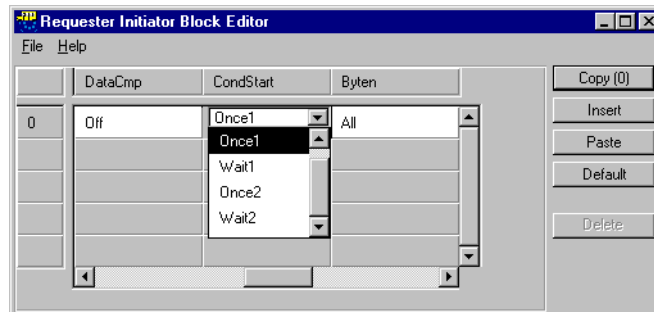
With the Initiator Conditional Start dialog box, you can specify two different start conditions for the initiator transactions.



To create the pattern expressions click the Edit button. This opens the Pattern Editor dialog box.

After you finished defining a start condition, it appears in the text field. When the defined pattern occurs (and the condition is met), the requester-initiator starts running.

These start conditions can be selected in the Requester Initiator Block Editor to start transactions not until the specified pattern occurs on the PCI-X bus (or on the external trigger input ports if specified).



Running the Requester-Initiator

When you have finished programming the requester-initiator transactions along with the requester-initiator behaviors, you can start your test.

To run the requester-initiator:

- ◆ Select *Run* from the *Exerciser* menu, or
click the Run button  in the main window.

Requester-Initiator Run Status When the requester-initiator is started, execution starts with block 0, requester-initiator behavior 0 and requester-target behavior 0. All execution status information is cleared.

Execution Order of Blocks For information on programming the execution order of blocks, see “*Programming Requester-Initiator Execution Order of Blocks*” on page 70.

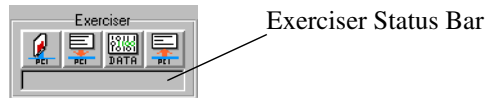
To program out-of-order block execution, use the requester-initiator queues, which can be specified by the respective block and behavior property.

Start of a Block A block is started after all the following conditions have been met:

- The wait-for-completion block property is disabled or there are no outstanding requests.
- The previous request/transaction is close to completion on the bus.
- A free tag is available with no outstanding request.

As a result, the requester-initiator requests a bus access from the bus arbiter. When the requester-initiator obtains access to the bus, it drives the transactions on the bus as specified.

Requester-Initiator Status During the requester-initiator run, the current status of the requester-initiator is displayed in the status bar. Error status messages, such as a data compare error message, will help you to identify and solve possible problems in your system under test.



The following list explains every status that can be displayed in the Exerciser status bar:

- *RUNNING*
Currently running of a test.
- *COMPLETION*
Waiting for the completion of all outstanding requests.
- *WAITING*
Waiting for the start condition.
- *STOPPED*
The exerciser has not yet started or successfully completed its tasks.
- *AB*
Execution has been stopped by an initiator abort.
- *PAUSED*
Exerciser has stopped generating new sequences; you can continue by selecting *Continue* from the *Exerciser* menu.

Furthermore, the following flags are available:

- *T*
Signals that the selected tag is busy. To specify a tag, see “*Available Requester-Initiator Behaviors*” on page 64.
- *CE*
Indicates whether a compare error has occurred.

Reaction to Terminations The reaction of the requester-initiator to terminations is determined by the following conditions:

- Whether execution is stopped immediately after an initiator or target abort can be defined in the exerciser generic settings dialog box (see “*Programming Requester-Initiator Reaction to Terminations*” on page 71.)
- If a retry is received by the requester-initiator, the same transaction will be repeated after a programmable number of clocks. A retry can be programmed by completer-target behavior *Initial* (see “*Available Completer-Target Behaviors*” on page 94).
- If a disconnect is received by the requester-initiator, the current sequence will be resumed after the delay specified by the requester-initiator behavior *Delay*. A disconnect can be programmed by completer-target behavior *Initial* (see “*Available Completer-Target Behaviors*” on page 94).

Programming Requester-Initiator Execution Order of Blocks

How the requester-initiator blocks are executed can be defined by the exerciser generic settings for the requester-initiator. To specify these settings, the Agilent E2929/E2930 Graphical User Interface provides the Exerciser Generic Settings dialog box. This dialog box consists of several pages, such as the Initiator page.

To open the Initiator page:

- ◆ Select the Initiator option from Settings, in the Exerciser menu.

To specify the execution order of blocks, use the following properties:

- *Infinite*
Execution of the programmed block lines will be infinitely repeated until the requester-initiator stop command is executed or another termination condition is met.
- *... times*
Execution of the programmed block lines will be repeated N times.

For description of all exerciser generic settings, please refer to *Agilent E2929/E2930 Windows and Dialog Boxes Reference*.

Programming Requester-Initiator Reaction to Terminations

If and when execution of the programmed requester-initiator block lines will be stopped, can be defined by the exerciser generic settings for the requester-initiator. To specify these settings, the Agilent E2929/E2930 Graphical User Interface provides the Exerciser Generic Settings dialog box. This dialog box consists of several pages, such as the Initiator page.

To open the Initiator page:

- ◆ Select the Initiator option from Settings, in the Exerciser menu.

To specify the reaction to terminations, use the following properties:

- *Stop on Master Abort*

Select this check box to stop execution immediately after an initiator abort. The execution can be resumed with the next sequence.

- *Stop on Target Abort*

Select this check box to stop execution immediately after a target abort. The execution can be resumed with the next sequence.

The reason for the termination can be determined from the exerciser status. For status information, see “*Requester-Initiator Status*” on page 69.

Programming Injection of Errors, Interrupts and Triggers

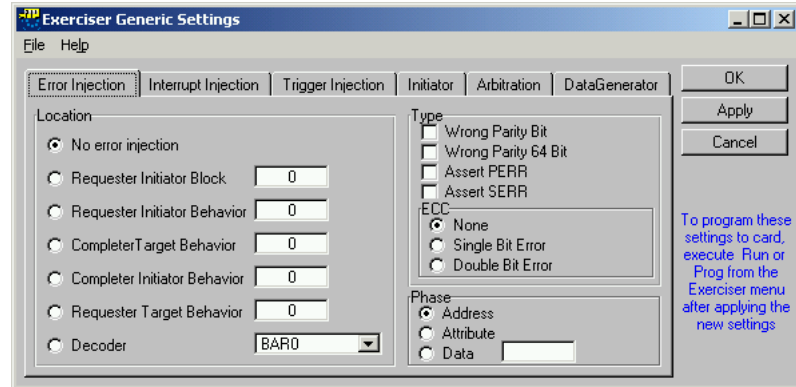
You can program the exerciser to inject errors, interrupts or triggers in combination with certain blocks or behaviors. This is possible for requester-target blocks as well as for all initiator and target behaviors and the decoders.

The Agilent E2929/E2930 Graphical User Interface provides the Exerciser Generic Settings dialog box, where you can define when and what is to be injected. This dialog box consists of several pages, such as the Error Injection, Interrupt Injection and Trigger Injection pages.

To open the Exerciser Generic Settings dialog box:

- ◆ Select the corresponding option from *Settings*, in the *Exerciser* menu.

This opens the Exerciser Generic Settings dialog with the respective page on top.



Each page contains a Location area where you can specify the block, behavior or decoder that is to inject the corresponding signal.

For a description of the specific settings available on each page, please refer to the *Agilent E2929/E2930 Windows and Dialog Boxes Reference*.

Stopping the Requester-Initiator

The requester-initiator stops automatically when it has successfully performed the specified transactions.

However, to stop the requester-initiator manually:

- ◆ Select *Stop* from the *Exerciser* menu.

(Clicking the Stop button in the main window will stop the analyzer, too.)

This guarantees that all outstanding requests get completed.

Manual Stop Required

There can be situations where the initiator does not complete the transactions, for example, if the bus hangs.

In this case, selecting *Break* from the *Exerciser* menu resets all bus state machines and forces the exerciser to release the bus.

Continuing the Initiator Operation

If the Exerciser has stopped generating new sequences (Exerciser status signals *PAUSED*), you can continue block execution by selecting *Continue* from the *Exerciser* menu.

The PCI-X Exerciser as a Completer-Target Device

The Agilent E2929/E2930 testcard can simulate a completer-target device on a PCI-X bus.

All necessary information about the testcard's **completer-target device** is found here.

If the Agilent E2929/E2930 testcard is set up as a PCI-X completer-target device, it can be used to test the functionality of a completer-target device on the bus.

Definition of a Completer-Target

A PCI-X completer-target (CT) device is the target of a transaction. It decodes all PCI-X bus transactions except split completion transactions.

Configuration Space and Target Decoders

Like every PCI-X target device, the Agilent E2929/E2930 testcard has a configuration space and target decoders. These features are described in detail in *“Configuration Space and Completer-Target Decoders” on page 76*.

For information on how to set up the target decoders and modify the configuration space header with the Agilent Graphical User Interface, see *“Target Decoder Setup” on page 84*.

When setting up the testcard you need to keep in mind that the testcard has one memory for storing information on the configuration space header and one for storing information on the target decoders (see *“Standard and Power-Up Databases” on page 91*).

Completer-Target Behaviors

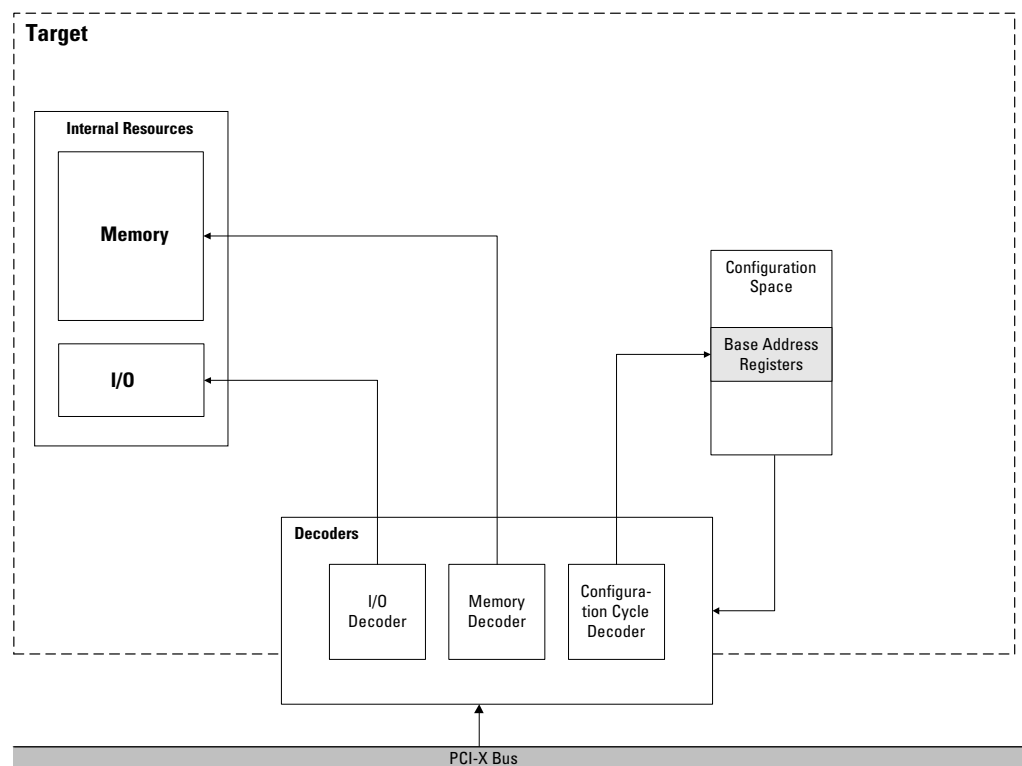
Additionally, the Agilent E2929/E2930 testcard provides full control over the behavior of the target's protocol. This means, you can determine how the completer-target is going to react to transactions in terms of inserted waits, retries, target aborts, and so forth. For more information on how to specify the completer-target behavior, see *“Programming Completer-Target Behaviors” on page 93*.

Configuration Space and Completer-Target Decoders

In general, every PCI-X target device consists of the following components:

- a **configuration space** holding configuration data about the device (see “*Configuration Space Header*” on page 77),
- a set of **decoders** for different types of accesses (I/O, memory, configuration cycles, and so on, see “*Target Decoder Properties*” on page 79),
- **internal data resources** where received data is stored or processed, or where data to be sent is taken from (see “*Data Resources*” on page 83).

The following figure gives an overview of the components of a target.



Different targets use different components to match their individual requirements. The Agilent E2929/E2930 testcard can be set up to simulate any type of target device.

Configuration Space Header

Every PCI-X device has a configuration space, which consists of 32-bit registers containing configuration data. Multi-function devices provide one configuration space per functional unit.

The configuration space of most PCI-X devices is divided into two sections, a public and a private section. The first 16 registers of the configuration space make up the public section. They are referred to as the **configuration space header**. It is accessible by the system BIOS and used for system management. The private section is optional and can be used for private, device-specific purposes.

The following figure shows the configuration space header of a PCI-X device according to the PCI-X specification.

31	16	15	00	
Device ID		Vendor ID		00
Status Register		Command Register		01
Class Code			Rev. ID	02
BIST	Header Type	Latency Timer	Cache Line Size	03
Base Address 0				04
Base Address 1				05
Base Address 2				06
Base Address 3				07
Base Address 4				08
Base Address 5				09
Card Bus CIS Pointer				10
Subsystem ID		Subsystem Vendor ID		11
Expansion ROM Base Address				12
Reserved			Capability Pointer	13
Reserved				14
Max. Latency	Min. GNT	Interrupt Pin	Interrupt Line	15

The configuration space header consists of 16 32-bit registers containing configuration data. The information stored in these registers is needed by the system to provide proper communication between the different devices on the bus. In the figure above, the registers with white background are fixed—they cannot be modified by the system. These registers contain information like the device type, latency and interrupt information, and other information. Status and command registers are partially fixed.

Base Address Registers The gray registers can be modified by the system. They hold the base addresses of the address spaces assigned to the different target decoders. The BIOS assigns these address ranges to the PCI-X devices during power up to ensure non-conflicting address ranges for all devices.

The two main types of base address registers are:

- Base Address Registers 0 to 5

These registers determine whether a decoder decodes memory or I/O transactions, the location of the address range (in 32 or 64-bit address space, or below 1 MB) and whether it is prefetchable.

- Expansion ROM Base Address Register

A PCI-X device may provide a device ROM containing a power-on self-test, BIOS or interrupt service routines. This register contains information on whether an expansion ROM exists and where its address space is located.

The completer-target needs the address range information to decide whether it has to react to transactions driven onto the bus. There are different types of decoders for different types of accesses. The decoders are closely linked to entries in configuration space.

The Agilent Testcard's Configuration Space Header

In order to simulate all the possible types of PCI-X devices, the configuration space header of the Agilent E2929/E2930 testcard is freely programmable.

The contents of every single register can be changed according to your test requirements. Furthermore, you can determine which bits of the registers are fixed and which can be changed by the BIOS during the configuration cycles.

The settings in the configuration space header of the testcard always determine the *current* behavior of the associated decoders. They can be reprogrammed after the BIOS configuration phase, so that the testcard can behave different than initially declared to the BIOS.

NOTE The PCI-X specification exactly defines the meaning of the various bits in the configuration space header. Thus, a fair amount of knowledge is needed to make proper settings and to avoid errors. The option to completely program the configuration space header is provided for very sophisticated tests, for example, to test systems without a BIOS.

All the settings regarding the base address registers and target decoders can be made in the GUI by specifying the respective properties in the Target Decode window (see “*Target Decoder Setup*” on page 84).

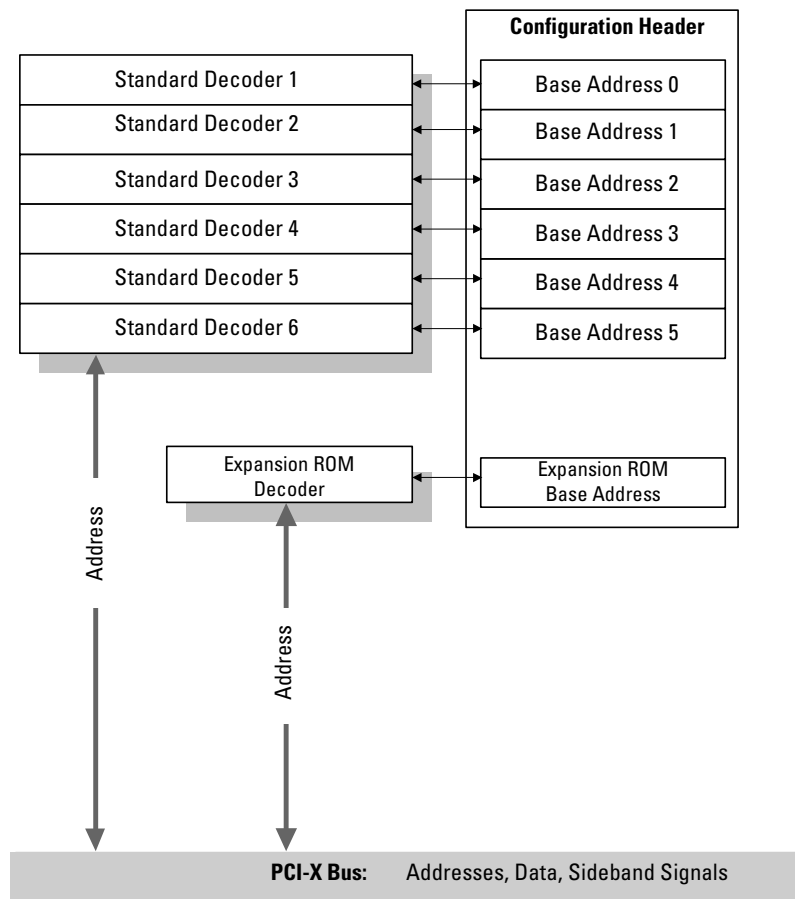
Target Decoder Properties

The Agilent E2929/E2930 testcard provides several decoders for different purposes. Depending on the contents of the base address registers in the configuration space header they claim transactions from the bus.

The following decoders of the Agilent E2929/E2930 testcard are accessible from the GUI:

- The three programmable decoders (six bars) for memory or I/O accesses
- The standard non-programmable configuration decoder
- The expansion ROM decoder
- A requester-target decoder for decoding split completion transactions. For more information on see “*The PCI-X Exerciser as a Requester-Target Device*” on page 103.

The following picture shows these decoders:



The transactions to be claimed by each decoder except the configuration and the requester-target decoder are specified by the base addresses (see “*Base Address*” on page 82).

Additionally, the programmable decoder is equipped with a set of parameters that define its properties such as whether the decoded memory space is prefetchable, or which data resource it is connected to (see “*Data Resources*” on page 83).

Standard Decoders

The standard decoders are connected to the six base address registers in the configuration space header of the testcard. They claim transactions to memory and I/O address ranges as defined by the base address entry and the programmed size.

To decode 64-bit addresses, the standard decoders *BAR 0-LO* and *BAR 0-HI* or *BAR 2-LO* and *BAR 2-HI* are used.

When programming the testcard, note that there must not be any unused decoders between used ones. For example, if you need three standard decoders, use decoders 1, 2, and 3, and not 1, 3, and 4. Failure to observe this rule renders the setup non-PCI-X compliant, which may cause the testcard to be completely disabled by the BIOS.

Expansion ROM Decoders

The Agilent E2929/E2930 testcard provides a programmable expansion ROM. Expansion ROMs typically are used as boot ROM. An expansion ROM is typically used as a boot ROM. It can contain code, for example, for own power-on-self-tests or for BIOS and interrupt service routines. These can be loaded and executed by the BIOS of the system under test by using an expansion ROM decoder.

The expansion ROM decoder behaves just like a standard decoder.

This decoder has the following programmable properties:

- Size of the address space
- Resource for the expansion ROM (data memory or flash)

If the selected resource is the flash memory, the content of the expansion ROM becomes non-volatile.

Configuration Decoder

The configuration decoder decodes configuration cycles. This decoder is non-programmable and provides a standard way of programming the testcard from PCI-X.

A dip switch allows you that any configuration cycles is not accepted and thus, you can hide the testcard from configuration space.

There is no entry in the configuration space needed, because configuration cycles aim to the configuration space itself. Whether or not the decoder claims a transaction depends on the bus commands (“config read” and “config write”), the IDSEL signal (Initialization Device Select), and the function, if it is a multi-function device.

This decoder can be used, for example, to test the start-up behavior of the system under test (BIOS tests, host bridge test, and so on).

Base Address

The information stored in the base address registers determines which transactions the decoder claims. The following parameters are coded:

- Base Address and Size.

The contents of the base address register always start with a bit sequence followed by a number of zeros. The base address is coded in the most significant bits of the registers. The number of following zeros defines the size of the decoded address range. A transaction to any address that starts with the same sequence as the base address will be decoded. As an example, if the base address is 3b4c0000, every transaction between this start address and 3b4cffff will be claimed.

- Location and Prefetchable.

There is a minimum size for address ranges, because the least significant bits are needed to code the address range location and the prefetchable flag.

The location must be considered by the BIOS when allocating the memory. The available options are in the 32-bit range, the 64-bit range, below 1 MB, and in the I/O range.

The prefetchable flag specifies whether memory is prefetchable and, therefore, whether a initiator can take advantage of optimized access to the target's memory. This must also be considered by the BIOS when allocating the memory. This property is not available in the I/O range.

Data Resources

After a decoder has claimed a transaction, data will be transferred in a certain direction (performing either a read or a write command). For this data, the testcard provides several memories as data resources.

Along with the selected resource, you need to specify an internal start address and the size for this resource. This size does not need to be equal to the size of the address range that the decoder actually decodes.

The available resources of the Agilent E2929/E2930 testcard are:

- Data Memory, Data Generator and Compare Unit

You can specify several internal memory address ranges or the start value of the data generator (with an internal base address and the size) as resources.

Incoming data can be stored in the internal memory or can be compared with reference data from the memory or the data generator.

For more information about the data resources, refer to *“Using Data Resources” on page 109*.

- Configuration Space

The complete configuration space of the Agilent E2929/E2930 testcard (public and private section) can be employed as a resource. The private section contains the interrupt status register and some registers for internal use. This can be useful, for example, to read configuration space information or the interrupt status from the I/O address space.

NOTE Overwriting the configuration space (especially the private section) can result in losing the PCI-X connection to the testcard. In such a case use the RS-232 link to communicate with the testcard.

- Expansion ROM

This address range emulates a PCI-X device’s expansion ROM, which usually contains the boot software. Its size is 64 KB and it is available via the Expansion ROM Base Address Register.

- Internal DBI Register

The decoder can also be connected to the internal DBI register of the PCI-X Asic.

Target Decoder Setup

In order to run your Agilent E2929/E2930 testcard as a target device on the PCI-X bus, you need to set it up according to your test requirements.

The Graphical User Interface of the Agilent E2929/E2930 testcard enables you to quickly view and edit the setup of the target decoders for both the completer-target and the requester-target.

In a system under test with a BIOS, this setup is mainly done automatically during power up. However, you might still be interested to view or change this setup.

Some settings, like the internal connections to data resources or the protocol behavior can be used directly after the target is initialized. Other settings concerning the BIOS and the configuration space require that the system is rebooted. For this purpose, your changes need to be stored in the power-up database which is used on restart.

Because the Agilent E2929/E2930 testcard is able to emulate any PCI-X device and to test any system, even without a BIOS, there is also the option provided to manually perform the necessary assignments (see *“Overwriting BIOS Settings” on page 90*).

The explanation of how to set up the target decoders can be found in *“Programming the Completer-Target Decoders” on page 85*.

How to do changes to the configuration space header is described in *“Modifying the Configuration Space Header” on page 87*.

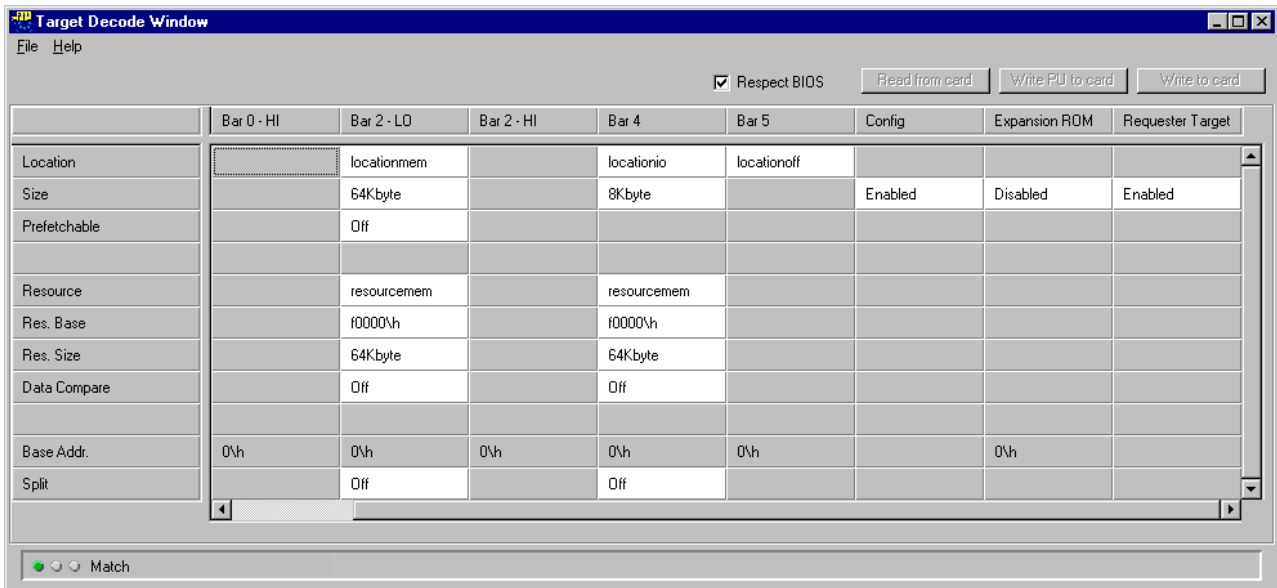
All settings that can be done with the Target Decode window of the GUI result in PCI-X specification-compliant behavior of the target. Therefore, the behavior is described as *normal*. When doing changes to the configuration space that are not compliant with the PCI-X specification, the behavior becomes *custom*.

NOTE With the C Application Programming Interface (C-API), additional options to program the decoders and their behavior are available. See the *Agilent E2929/E2930 C-API/PPR Programming Reference*.

Programming the Completer-Target Decoders

To set up the target decoder settings for a completer-target device, follow the steps below. If you only want to view the current settings, skip the respective steps.

- 1 To open the Target Decode window, select *Target Decode* from the *Exerciser* menu in the main window.



The columns of this window contain the settings for the configuration decoder, the six standard decoders and the expansion ROM decoder (from left to right).

- 2 Disable the requester-target decoder, which is only used for decoding split completion transaction.

- 3** To display the settings currently in use in the testcard's standard database, click the *Read from card* button.

The match indicator in the lower left corner of the window shows green.

- 4** For every decoder, select *Enabled* or *Disabled*. If a decoder is disabled, it does not claim any transaction, no matter what the other settings for this decoder are.

- 5** Select the *Resource*, the resource base address (*Res. Base*) and the resource size (*Res. Size*) for the decoders.

These parameters determine which data resource on the testcard the decoder is connected to internally. This resource is used to supply and/or receive data for the transactions. The available options are:

- *resourcemem/resourcegen*

This selection connects the decoder to the data memory/data generator.

For both data resources, the data compare unit can be activated with *Data Compare*.

- *resourcecfg*

This selection connects the decoder to the configuration space, both the public and the private section.

- *resourcedbi*

This selection connects the decoder to the internal DBI registers of the PCI-X Asic.

- 6** Specify the address range to be covered by the decoders (parameters *Base Addr.*, *Size*, *Location*, and *Prefetchable*).

The settings for these parameters are stored in the configuration space header of the target. By default, they are protected and cannot be changed for a running system. However, you can store them in the power-up database by clicking the *Write PU to card* button. When restarting the system, they will be considered by the BIOS.

By default, the base address is not editable in this window. BIOS normally assigns the base addresses to the various targets. However, you can decide to overwrite the BIOS settings. See “*Overwriting BIOS Settings*” on page 90 for details.

When selecting the base address manually, ensure that multiple targets do not decode overlapping address ranges.

CAUTION

Multiple devices decoding overlapping address ranges may result in hardware damage.

- 7 For *Bar 0-LO*, *Bar 2-LO* and *Bar 4*, split decoding can be switched on or off.
- 8 Write your settings to the testcard.
 - If you want to use the settings immediately, click the *Write to card* button. By default, BIOS-relevant settings will not be stored to the testcard (see “*Overwriting BIOS Settings*” on page 90).
 - If you want to use the settings after the next power up, click the *Write PU to card* button. With this option also BIOS-relevant settings can be stored and employed.
- 9 Check the match indicator in the lower left corner of the window.

If the display matches the settings in the testcard’s standard database, the indicator shows green. If the display matches the contents of the power-up database, it is yellow, and if it does not match at all, it is red. The latter occurs if the changes made to BIOS-relevant settings, such as the memory location, cannot be written to the standard database of the testcard (see “*Overwriting BIOS Settings*” on page 90).

Modifying the Configuration Space Header

With the Graphical User Interface of the Agilent E2929/E2930 testcard you can freely program every single register of the testcard’s configuration space header. Usually, all settings in the configuration space header that need to be made when starting the system are either made by the BIOS or from within the Target Decode window. The latter applies to settings concerning the base address registers.

The option to completely program the configuration space header is provided for very sophisticated tests, for example, to test systems without a BIOS or with a very rudimentary BIOS.

For every bit of the various registers in the configuration space header, you can determine whether it is fixed or programmable from outside (BIOS). For both types, values can be specified as required for BIOS configuration during system start up:

- Determine **fix** values, for example a “Vendor ID”, which is then read-only and can be evaluated by the BIOS.
- Determine **programmable** values, for example base address register entries. They can be used by the BIOS to determine the wanted size of the decoded address range and will then be overwritten with the actual base address.

NOTE The PCI-X specification exactly defines the meaning of the various bits in the configuration space header. Thus, a fair amount of knowledge is needed to make proper settings and to avoid errors.

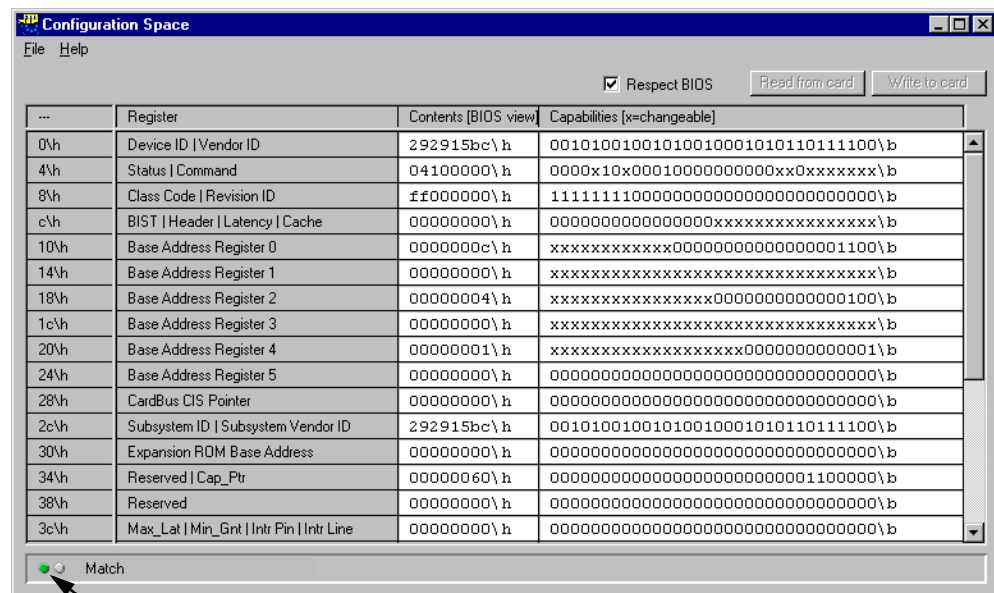
In some cases, manipulating the base address registers in the configuration space header can result in the testcard becoming inaccessible from the PCI-X interface—neither by the controller nor by the software application—especially if the improper settings are stored as power-up defaults. In this case, you can still use the RS-232 interface cable to access the testcard.

CAUTION

Setting up the PCI-X testcard to decode address ranges that are also decoded by other devices can result in hardware damage.

Edit the configuration space header of the Agilent E2929/E2930 testcard as follows:

- 1 Select *Config Space* from the *Exerciser* menu in the main window to open the Configuration Space window.



Match Indicator

- 2 Observe the *Match* indicator in the status bar at the bottom of the window.

If the indicator shows red, the current settings on the PCI-X testcard and the values presented in this window do not match. To load the current settings from the testcard into this window, click the *Read from card* button.

3 Edit the fields in the table according to your test requirements.

The table rows represent the different registers of the configuration space header. Some rows contain several registers indicated by a vertical line separator in the *Register* column. The columns of the table show from left to right:

- The hexadecimal byte address relative to the first byte of the configuration space.

- *Register*

The name(s) of the register(s) according to the PCI-X specification.

- *Contents [BIOS view]*

The current values in the register(s) as seen by the BIOS.

- *Capabilities [x=changeable]*

These fields contain a representation of the registers' programming mask.

0 or 1 means: This bit is fixed (read-only) and its value cannot be changed by the BIOS or another routine from outside.

x means: This bit is programmable by the BIOS as well as by other routines.

4 Click the *Write to card* button.

The current contents of this window will be written to the configuration space header and the mask memory of the Agilent E2929/E2930 testcard.

If a bit has a fixed value in the right column, a possibly differing value in the *Contents* column will be ignored.

5 Close the Configuration Space window.

A dialog box appears, informing you that you made changes to the settings in the configuration space header.

6 Click *Store*, if you want to make these changes persistent for the next time you power up the testcard.

Or click *Close*, if you want to use the new modifications only for the current test session.

Overwriting BIOS Settings

Usually, the system BIOS handles the assignment of the address spaces to the different target devices on the PCI-X bus. These address spaces are asserted when the system starts running and are then valid constantly.

Therefore, all settings that are relevant to this mechanism are protected from manual changes. This applies to the standard database where the settings are stored that are currently in use. However, you can write your modified settings to the power-up database which will be considered by the BIOS after restarting the system under test.

The exception to this behavior is the base address itself. For safety reasons, the fields holding the base addresses in the Target Decoder window are not editable by default. This is to avoid hardware damage, possibly caused by overlapping address ranges for different decoders.

Because the Agilent E2929/E2930 testcard is able to emulate any PCI-X device and to test any system, even without a BIOS, an option is provided to perform the necessary assignments manually.

To allow manual changes:

- ◆ Clear the *Respect Bios* check box in the Configuration Space window.

You can now make your changes to the target settings including the base addresses, and store them in the power-up database as well as in the standard database.

CAUTION

Ensure that different targets do not decode overlapping address spaces. Decoding overlapping address spaces may result in hardware damage.

Standard and Power-Up Databases

Some of the changes you do in the configuration space header and the target decoders can be used immediately in the running system. Others, however, do not come into effect until the system under test is rebooted. This applies to settings that are (at least partially) made by the BIOS during the configuration phase when starting the system.

For this purpose the Agilent E2929/E2930 testcard provides two databases to store all settings.

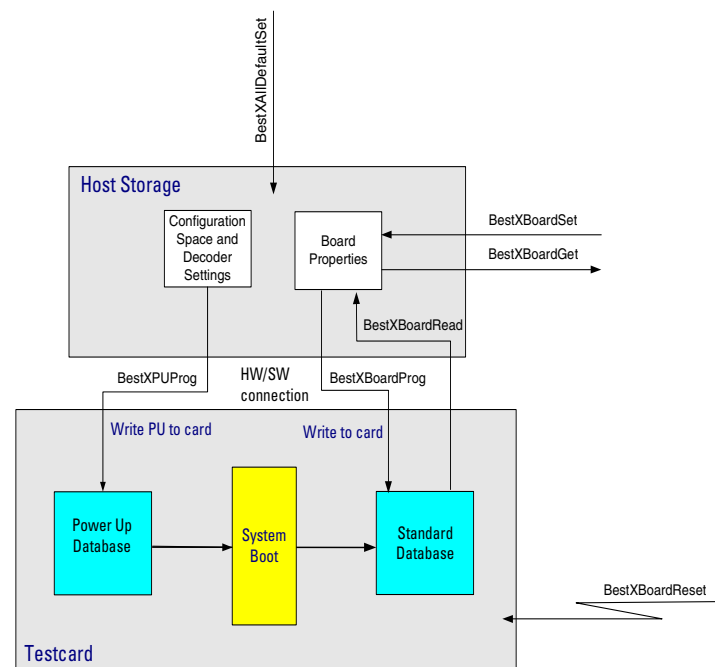
- **Standard Database.**

This memory keeps the currently used settings for the configuration space header and the decoders.

- **Power-Up Database.**

If you want to specify settings that will be used when restarting the system under test, you have to store them in the power up database. When the system is booted, the contents of the power up database are loaded to the standard database. After that, the BIOS sets all bits that it has given access to.

The figure below illustrates the two databases.



For both databases you can determine freely, which bits are fixed and which can be modified by the BIOS or other program routines. This information is stored in the masks.

This is done in the right column of the Configuration Space window of the GUI (see “*Modifying the Configuration Space Header*” on page 87). All bits that are marked with an x can be set by the BIOS. All other bits are set fixed to their values 0 or 1. The values that the BIOS has inserted in the programmable areas are shown in the middle column.

Programming Completer-Target Behaviors

In contrast to a requester-initiator, a target cannot perform a “target run” as it has a passive behavior. After setting up and enabling the decoders and base address registers, the Agilent PCI-X Exerciser is able to react to accesses from requester-initiator devices.

Besides these functional settings, the PCI-X Exerciser also allows to specify behaviors that control the completer-target regarding the PCI-X protocol. This means, for instance, that you can define the number of waits that the completer-target inserts in a data phase, or whether the completer-target signals a retry or a target abort (see “*Available Completer-Target Behaviors*” on page 94).

These behaviors do not affect the result of a data transfer. Hence, you can repeat a test with fixed target decoder settings but varying completer-target behaviors and then compare the results.

Completer-Target Behavior Memory

All these behaviors are stored in the completer-target behavior memory. In this memory, the behaviors are kept in memory lines, and every memory line is then assigned to a transaction.

You can set up to 256 programmable sequence-level behaviors, which are successively used for each request, that is, for every address phase.

Completer-Target Behavior Editor

To set up these completer-target behaviors, the PCI-X Exerciser provides the Completer-Target Behavior Editor window.

To open the Completer-Target Behavior Editor:

- ◆ Select *Behavior* → *Completer-Target* from the *Exerciser* menu in the main window.

Programming the Behaviors

Setting up and programming the behaviors in the completer-target behavior editor dialog box works similar to the transaction editor dialog box.

Available Completer-Target Behaviors

Each completer-target behavior has the following programmable properties. (The term used in the completer-target behavior editor is given in parentheses.)

- Decode speed A, B or C (*Decode*)

NOTE Decode speed A is only supported up to 66 MHz.

- Accept 64-bit wide data transfer (Y/N) (*Ack64*)

With this attribute, the target signals that it is capable of accepting 64-bit accesses.

- Initial target response (*Initial*)

This property defines how the completer-target responds after waiting the number of cycles specified under *Latency*.

If the split condition is met, this behavior is ignored and a split response is given after the number of wait cycles defined under *SplitLatency*.

For more information about split response conditions, see “*Defining a Split Response Condition*” on page 96.

- Number of initial latencies (*Latency*)

For more information on initial latencies, see “*Initial Latencies*” on page 95.

- Subsequent target response (*Subseq*)

This property specifies the target response in subsequent data phases. It comes only into effect if the corresponding *Initial* completer-target behavior was set to *Accept*.

- Subsequent data phases (*SubseqPhase*)

If the value of behavior *Subseq* is set to *Disconnect*, this property defines the subsequent data phase.

- Split response after programmable number of wait cycles (*SplitLatency*)

- Split response enable (*SplitEnable*)

This property defines if a split response is generated. You must set up the decoder in the Target Decode window accordingly for this property to have an effect.

- Number of repeats (*Repeat*)

The current behavior is repeated N times before the next behavior is used.

For more information about the properties, refer to the *Agilent E2929/E2930 Windows and Dialog Boxes Reference*.

Initial Latencies

The initial latency can be programmed with the completer-target behavior property *Initial*.

However, sometimes the actual initial latency is larger, independent of the programmed value. In particular, reads from the completer-target are slow if the data path is not prepared for the given address. When the previous address has already been read, or when a retry to the same address has been signaled, reads from completer-target can be fast.

The following table shows how the minimum initial latency depends on the most recent event.

In this table, *Fast* means as fast as the PCI-X specification allows with decode 'B' speed or one clock slower, and *Slow* stands for up to 15 initial wait cycles (within the specification).

Minimum Initial Waits of Access to Target after most Recent Events		Subsequent Target Access	
		Write to Target	Read from Target
Most recent event	RI write	Fast	Slow
	RI read	Fast	Slow
	Write to target	Fast	Slow
	Read from previous target address	Fast	Fast
	Read from other target address	Fast	Slow
	Retry from target	Fast	Fast

Defining a Split Response Condition

Split response condition properties identify a particular type of requests that shall be given a split response and determine in which request queue those requests will be put for later completion. Four different request types can be defined.

All split response properties can only be programmed by using the C-API or the command line interface (CLI).

For required commands refer to *Agilent E2929/E2930 C-API/PPR Programming Reference*.

The PCI-X Exerciser as Completer-Initiator Device

The Agilent E2929/E2930 testcard can simulate a completer-initiator device. This is used to test the functionality of a completer-initiator device on the PCI-X bus.

- Definition of a Completer-Initiator** A PCI-X completer-initiator (CI) device initiates PCI-X split completion transactions only.
- Split Completion Transaction** The completer-initiator can start a completion transactions only after the completer-target has given a split response. How transaction attributes are controlled can be found in *“Generating Split Completion Transactions” on page 102.*
- Completer-Initiator Behaviors** Additionally, the Agilent E2929/E2930 testcard provides full control of the completer-initiator’s protocol behavior. The behaviors merely determine the behavior of the completer-initiator in terms of different types of delays that get inserted, transaction terminations, and so on. For more information on how to specify the completer-initiator behavior see *“Programming Completer-Initiator Behaviors” on page 98.*

Programming Completer-Initiator Behaviors

The PCI-X Exerciser allows you to specify behaviors that control the completer-initiator with regard to split completion transactions.

Before split completion transactions are initiated, all requests can be accumulated:

- in-order with programmable partition sizes, or
- with interleaved partial responses from four different request queues, or
- out-of-order from four different queues.

The completer-initiator behaviors are used to control the execution order, the partition size and the bus behavior of successive (partial) split completion transactions. For more information on the behaviors properties see “*Available Completer-Initiator Behaviors*” on page 99.

Completer-Initiator Behavior Memory

All behaviors are stored in the completer-initiator behavior memory. In this memory the behaviors are kept in memory lines, and every memory line is then assigned to a transaction.

You can set up to 256 programmable sequence-level behaviors that are successively used for each generated completion transaction request.

Completer-Initiator Behavior Editor

To set up these completer-initiator behaviors, the PCI-X Exerciser provides the Completer-Initiator Behavior Editor window.

To open the Completer-Initiator Behavior Editor:

- ◆ Select *Behavior* → *Completer-Initiator* from the *Exerciser* menu in the main window.

Programming the Behaviors

Setting up and programming the behaviors in the completer-initiator behavior editor dialog box works similar to the transaction editor dialog box.

Available Completer-Initiator Behaviors

Each completer-initiator behavior has the following programmable properties. (The term used in the completer-initiator behavior editor is given in parentheses.)

- Select the request queue (*Queue*) from which the next completion is generated:
 - Select the next queue (*Next*)
 - Select no queue (*None*)
The requests will be accumulated for out-of-order completion.
 - Select any non-empty available queue (*qauto*)
 - Select queue A, B, C or D with the option to skip the behavior, if the currently selected queue is empty.
 - Select queue A, B, C or D with the option to wait until the currently selected queue gets a request, if this queue is empty.

NOTE

Exercise care when using this option, because requests in other queues will not be completed if the selected queue does not receive a request.

- Select the Size of the next partial completion transaction (*Partition*):
 - The full byte count is transferred without disconnecting the completion (*No*)
 - The completion is disconnected at every N-th allowable disconnect boundary after the current start address.
- State controlling of a reserved bit of AD[] bus in the respective address/attribute phase (*Addrxx/Attrxx*).

These bits are used to define split completion transactions (see “Generating Split Completion Transactions” on page 102).

- Split completion message (*ErrMsg*)

If generation of a split completion message has been selected, a user-defined split completion message is generated, as opposed to a normal split completion transaction. The latter message, which would otherwise be generated, contains the data read or the default write completion message.

For information on how to define the content of the message, see “*Defining Split Completion Messages*” on page 101.

NOTE

Only one type of split completion message can be generated per test run.

- Limited size of completion transactions (*Partition*)

This property defines whether and how the size of completion transactions is limited and thus how these transactions are partitioned.

- Wait for conditional start pattern (*CondStart*)

This property defines if the completion starts unconditionally or if a conditional start pattern must have occurred:

- *No*

Unconditional start.

- *Once1*

After the exerciser has been started, block execution waits until the conditional start pattern 1 has occurred at least once.

- *Wait1*

After the end of the previous requester-initiator sequence, block execution waits until the conditional start pattern 1 has occurred.

- *Once2*

After the exerciser has been started, block execution waits until the conditional start pattern 2 has occurred at least once.

Pattern 1 and pattern 2 can be programmed with the Command Line Interface (CLI).

- Clock delay before assertion of REQ# (*Delay*)

This property allows you to vary latencies between transactions.

Sometimes the minimum achievable latency to the next CI transaction is restricted by the most recent event and sometimes by the data path configuration.

- Number of address steps (*Steps*)

The number of address steps is the number of clock cycles between the assertion of `GNT#` and the assertion of `FRAME#` plus two clock cycles, which are designed into the register-to-register interface of PCI-X.

- Request 64-bit wide data transfer (Y/N) (*Req64*)
- Release `REQ#` N clocks after the address phase (*RelReq*)
- Number of repeats (*Repeat*)

The current behavior is repeated N times before the next behavior is used.

For more information about the completer-initiator behaviors refer to the *Agilent E2929/E2930 Windows and Dialog Boxes Reference*.

Defining Split Completion Messages

The content of the split completion message can be defined by the exerciser generic settings. To specify these settings, the Agilent E2929/E2930 Graphical User Interface provides the Exerciser Generic Settings dialog box. This dialog box consists of several pages, such as the Initiator page.

To define the content of the split completion message:

- 1 Open the Initiator page by pointing on the *Exerciser* menu to *Settings*, and then clicking the Initiator option.
- 2 In the *Completion Message* text box, enter a 32-bit value that defines the content of the split completion message as follows:
 - Bits 19 ...31 will be placed on the AD bus during the data phase.
 - Bits 0 ... 11 are determined by the remaining byte count.

Generating Split Completion Transactions

The following table shows the transaction properties and attributes and how they are controlled:

Properties and Attributes	How Controlled
Command	Block property command to "Split"
AD32	Implicitly set, based on start address and progress
Reserved AD[7] in address phase	Completer-initiator property "Addr7"
Requester function number	Repeated from request
Requester device number	
Requester bus number	
Tag	
Relaxed ordering	
No snoop	
Reserved AD[31] in address phase	Completer-initiator property "Addr31"
Byte count	First repeated from request, then updated, based on progress
Function number	From configuration space
Device number	
Bus number	
Reserved AD[28::24] in attribute phase	Completer-initiator properties "ATTR24" ... "ATTR28"
Split completion message	Completer-initiator property "ErrMsg"
Reserved AD[31::30] in attribute phase	Completer-initiator properties "ATTR30" ... "ATTR31"
Completer termination	Completer behavior property "Partition"

The PCI-X Exerciser as a Requester-Target Device

The Agilent E2929/E2930 testcard can simulate a requester-target device on a PCI-X bus. It can be used to test the functionality of a requester-target device on the bus.

Definition of a Requester-Target A PCI-X requester-target (RT) device decodes PCI-X split completion transactions only.

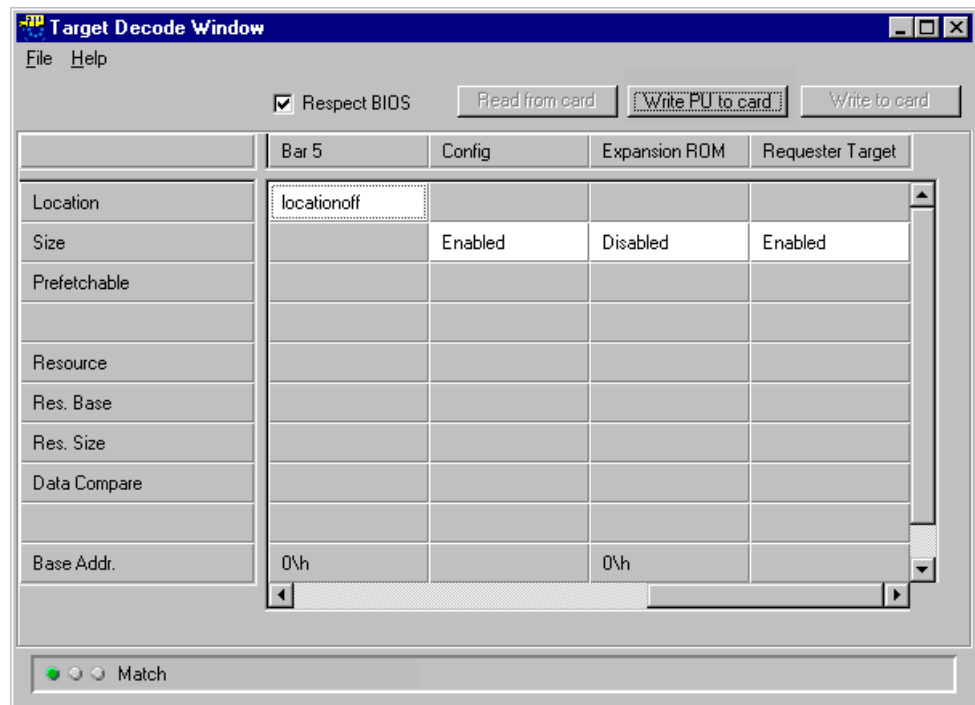
Requester-Target Decoder Like every PCI-X target device, the Agilent E2929/E2930 testcard has got a target decoder. For information on setting up the decoder for the requester-target device with the Agilent Graphical User Interface, see *“Requester-Target Decoder Setup” on page 104.*

Requester-Target Behaviors Additionally, the Agilent E2929/E2930 testcard provides full control of the requester-target’s protocol behavior. This means that you can determine how the requester-target is going to react to split completion transactions in terms of inserted latencies, retries, target aborts, and so forth. For more information on how to specify the requester-target behavior see *“Programming Requester-Target Behaviors” on page 105.*

Requester-Target Decoder Setup

To set up the target decoder settings for a completer-target device, follow the steps below. If you only want to view the current settings, skip the respective steps.

- 1 To open the Target Decode window, select *Target Decode* from the *Exerciser* menu in the main window.



The columns in this window contain the settings for the configuration decoder, the six standard decoders, the expansion ROM decoder (from left to right) and the Requester-Target decoder.

- 2 Enable the requester-target decoder, which is only used for decoding split completion transaction.

3 Write your settings to the testcard.

- If you want to use the settings immediately, click the *Write to card* button.
- If you want to use the settings after the next power up, click the *Write PU to card* button.

4 Check the match indicator in the lower left corner of the window.

If the display matches the settings in the testcard's standard database, the indicator shows green. If the display matches the contents of the power-up database, it is yellow, and if it does not match at all, it is red. The latter occurs if the changes made to BIOS-relevant settings, such as the memory location, cannot be written to the standard database of the testcard.

Programming Requester-Target Behaviors

The requester-target cannot perform a “target run” as it has a passive behavior. On enabling the decoder, the Agilent PCI-X Exerciser is able to react to accesses from requester-initiator devices.

Besides these functional settings, the PCI-X Exerciser allows you to specify behaviors that control the requester-target with regard to split completion transactions. For more information about the requester-target behaviors see “*Available Requester-Target Behaviors*” on page 106.

These behaviors do not affect the result of a data transfer. Hence you can repeat a test with fixed target decoder settings while varying the completer-target behaviors, and then compare the results obtained

Requester-Target Behavior Memory

All these behaviors are stored in the requester-target behavior memory. In this memory the behaviors are kept in memory lines, and every memory line is then assigned to a split-completion transaction.

You can set up to 256 programmable sequence-level behaviors, which are successively used for every new split-completion transaction when this is received as a result of its own request.

Requester-Target Behavior Editor To set up these requester-target behaviors, the PCI-X Exerciser provides the Requester-Target Behavior Editor window.

To open the Requester-Target Behavior Editor window:

- ◆ Select *Behavior* → *Requester-Target* from the *Exerciser* menu in the main window.

Programming the Behaviors Setting up and programming the behaviors in the requester-target behavior editor dialog box works similar to the transaction editor dialog box.

Available Requester-Target Behaviors

Each requester-target behavior has the following programmable properties. (The term used in the requester-target block editor is given in parentheses.)

- Decode speed A or B (*Decode*)

NOTE Decode speed A is only supported up to 66 MHz.

- Accept 64-bit wide data transfer (Yes/No) (*Ack64*)

With this attribute, the target signals that it is capable of accepting 64-bit accesses.

- Initial target response (*Initial*)

This property defines how the completer-target responds after waiting the number of cycles specified under *Latency*.

- Number of initial latencies (*Latency*)

This property defines the number of wait cycles.

- Subsequent target response (*Subseq*)

This property specifies the target response in subsequent data phases. It comes only into effect, if the corresponding *Initial* completer-target behavior was set to *Accept*.

- Subsequent data phases (*SubseqPhase*)

If the value of behavior *Subseq* is set to *Disconnect*, this property defines the subsequent data phase.

- Number of repeats (*Repeat*)

The current behavior is repeated N times before the next behavior is used.

Using Data Resources

The Agilent PCI-X testcard provides the following data resources:

- The data memory
- The data generator

The data resources are commonly available for the requester-initiator—as a target and as a data source for split completion transactions. Both data resources can also be used for real-time data compare.

Data Memory

The Agilent E2929/E2930 testcard provides an internal data memory that can be used as a data resource by the requester-initiator/requester-target and the completer-target/completer-initiator.

The data memory can be used to:

- Supply data that is sent to the bus.
- Store data received from the bus.
- Compare data received to reference data in the data memory (without being stored).

The data memory is optimized for high throughput and allows programmable protocol behavior. It is implemented as a wrap-around memory and can emulate a virtual data resource that is much larger than its actual size.

For information on how the data memory is built up physically, see *“Data Memory Organization” on page 110*.

Details about how to use the features of the data memory in the GUI can be found in *“How to Use the Data Memory Editor” on page 112*.

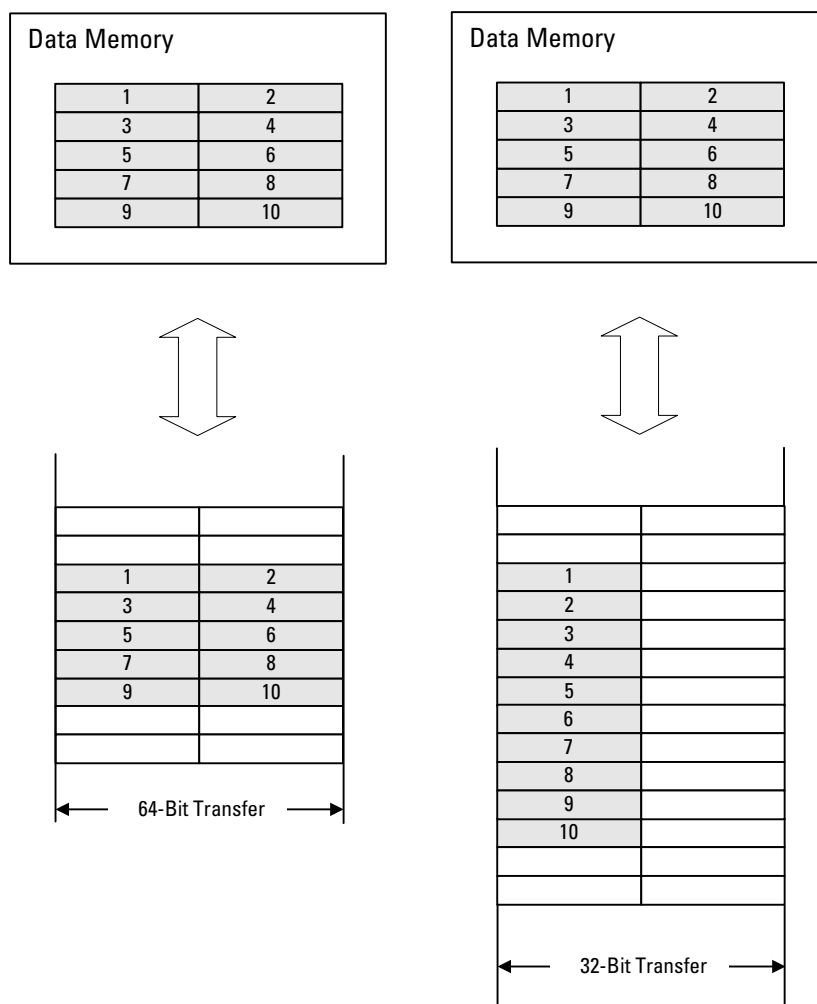
Data Memory Organization

The Agilent E2929 testcard has 1 MB ($128K \times 2$ dwords) programmable read/write data memory. The data memory of the Agilent E2930 testcard is 4 MB ($512K \times 2$ dwords).

When using the data memory as a data resource for PCI-X Mode 2 transfers, after 32 KB the memory is exhausted and the data that follows will not be transferred back-to-back.

Data Alignment For usability reasons, when transferring data between the PCI-X bus under test and the data memory, the data must be aligned with respect to the PCI-X bus width and the width of data transfer (32-bit or 64-bit).

The following figure illustrates how data that is stored in the data memory is driven onto the PCI-X bus and how received data is stored (for 64-bit and 32-bit data transfer).



To align the data correctly for data transfers to or from the PCI-X bus, follow these rules:

- For 64-bit transfers:

The specified byte addresses must be 64-bit aligned. For both the internal address and the PCI-X bus addresses, the least significant **three bits** must be 0.

- For 32-bit transfers:

The specified byte addresses must be 32-bit aligned. For both the internal address and the PCI-X bus addresses, the least significant **two bits** must be 0.

Initiator and Target Partitions

Basically, you are free to use any part of the data memory for the initiator and the target. However, if you need to use the data memory as a resource for all devices at the same time, it is recommended to define memory partitions.

This can be done by using different internal addresses for the initiator and the target. This applies to the following parameters:

- In the initiator:

The block properties *IntAddr* and *NumBytes* must be set appropriately.

- In the target:

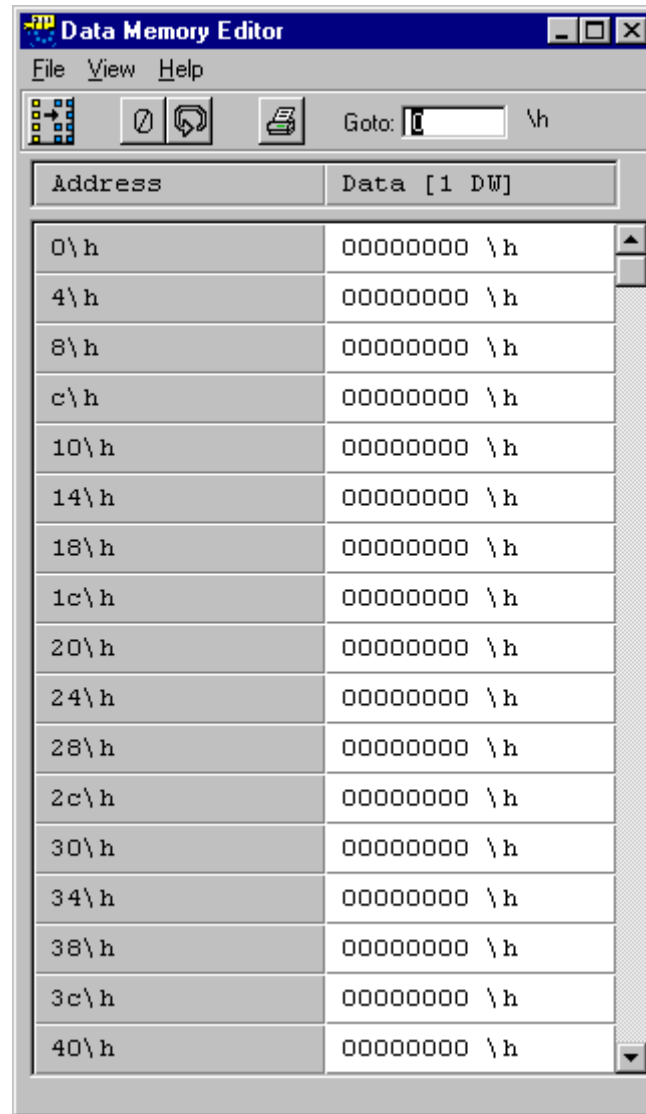
When the *Resource* property in the Target Decode window is set to *resourcemem* (data memory) or *Data Compare* is on, the properties *Res. Base* address and *Res. Size* must be set appropriately.

NOTE If the initiator of the Agilent E2929/E2930 testcard is communicating with its own target via the PCI-X bus, the target has no access to the data memory. When the initiator performs write commands, the target cannot store the received data in the data memory. Similarly, when performing read commands, the target sends dummy data.

How to Use the Data Memory Editor


To view and edit the contents of the data memory:

- ◆ Open the Data Memory Editor window by selecting *Data Memory* from the *Exerciser* menu in the main window.



Features The data memory editor lets you perform the following tasks:

- Setting the contents to zero

To set the complete data memory to zero, click the Zero button .


- Setting the search address

The Goto text field can be used to jump directly to the specified address in the data memory. Enter the address in hex format in the text field and press the Return key. Another way to locate a certain address is to use the scrollbar on the right.

- Modifying the contents

You can modify the contents of the data memory at a certain address by clicking into the respective data field (right column) and typing the new value. The changes are written to the testcard immediately after you press the Return key.


- Reloading the contents from the testcard

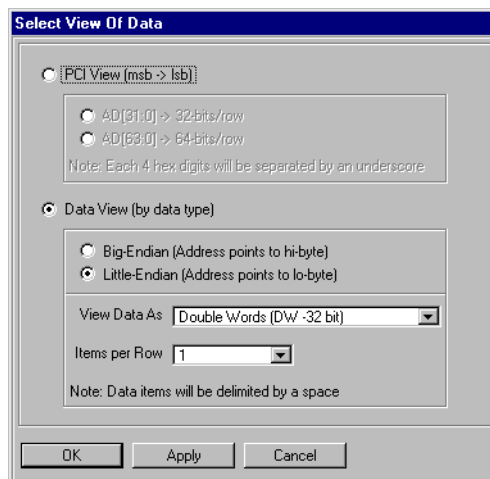
To reload the contents of the window from the testcard, click the Refresh button  or select *Refresh Data* from the *View* menu.

- Managing Files

You can save the whole memory content into a memory file (extension .mem). Previously saved files can be loaded into the editor for examination or reuse. The functions for the file actions are found in the *File* menu.

Modifying the Data View You can also modify the way the data is displayed in the data memory editor:

- ◆ Click the Data View button  or choose *Select View Of Data* from the *View* menu. This opens the Select View Of Data dialog box.



The two different views of the memory contents are:

- **PCI-X View**

The data memory contents are displayed as they are sent on the PCI-X bus. Thus, the data amount per row represents one bus cycle. Select AD[31:0] to display 32 bits/row for 32-bit transfers. Or select AD[63:0] to display 64 bits/row for 64-bit transfers.

In this view, the most significant bit is always displayed first in every row.

- **Data View**

This view displays the memory contents regarding the data type. The available options are:

- The size of the data packages: bytes, words, dwords, or qwords.
- The number of items displayed per row: the options here depend on item size.
- The byte weight within the item: little endian or big endian.

Data Generator

Features You can use the onboard data generator to supply data as an alternative to the data memory. The data generator has the following features:

- It allows the testcard to deliver fast data patterns without initial latencies.
- It generates unique data patterns (up to 2^{21} DWORDs) that allows you to deterministically link a certain address to a certain data pattern. This feature allows you a unidirectional data path verification.

Furthermore, the data generator provides:

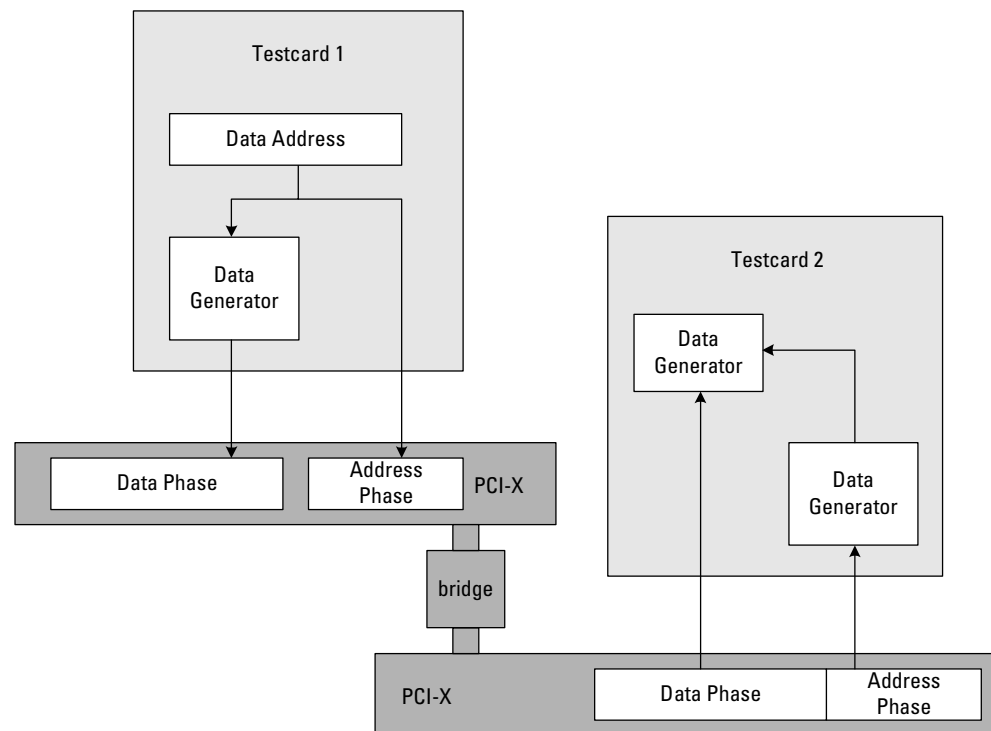
- 21-bit width of unique data. The data itself is 64 bits wide.
- A programmable start value that can be used to generate unique data patterns.
- The data pattern can be changed with every data phase.

Performing Unidirectional Data Path Verification

The feature that a certain address is bijectively linked to a certain data patterns allows you a concept of unidirectional data path verification.

In combination with a second testcard and the real-time data compare feature, extended-period load stressing on any PCI-X to PCI-X data path can be performed. Errors are detected in real time.

The following figure shows the test configuration:



The data generator implementation is identical in all testcards. Therefore, writing a certain data pattern from testcard 1 to testcard 2 does not require reading back the data and verifying it at testcard 1. Using the knowledge of the relationship between an address and the associated data, testcard 2 is able to make a comparison and check for data consistency.

Testcard 2 can also check for addressing errors that would remain uncovered with a simple write/read/compare approach.

The input for the data generator is bus address bit 3 ... 22.

How to Use the Data Generator

To use the data generator as data source for data leaving and entering the testcard, go through the following steps:

1 Open the requester-initiator block editor by selecting *Transactions* from the *Exerciser* menu in the main window.

2 Select for the transactions in the *Resource* column *DataGen*.

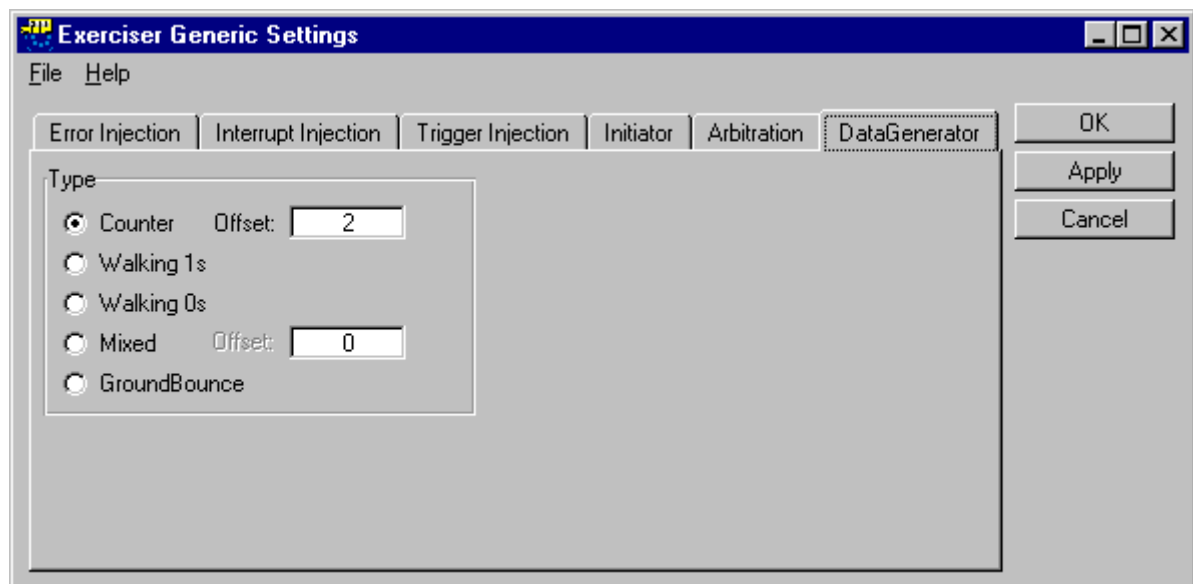
If the testcard is acting as a target, proceed with **step 3**, otherwise go to **step 5**.

3 Open the Target Decode window by selecting *Target Decode* from the *Exerciser* menu.

4 In the *Resource* row select the data source *resourcegen* for the decoders *Bar 0-LO*, *Bar 2-LO*, or *Bar 4*.

5 Open the Data Generator page by pointing on the *Exerciser* menu to *Settings* and clicking *Data Generator*

This opens the Data Generator page.



Data Generator Features The Data Generator page provides the following features:

- Generating a count-up data pattern (unique data)

To generate a count-up data pattern, select the *Counter* button and enter an offset to the bus address in the range $0 \dots (2^{20} - 1)$.

The counter creates a 64-bit wide data pattern, consisting of two count values from bit 0 ... 22 and from bit 32 ... 54. The remaining 18 bits can be preset with an arbitrary value or the identification to enable an easy identifier for any seen data in the system.

If the counter is programmed with an initiator identification, the data compare can be switched off for the 18 fixed bits to still allow unidirectional data verification.

The bit content of the counter is as follows:

63	55	54	35	34	32	31	23	22	3	2	0
					100						000
9 bits			20 bits		3 bits		9 bits		20 bits		3 bits

- Generating walking ones or zeros

To generate a pattern with walking ones or zeros, select either *Walking 1s* or *Walking 0s*.

- Generating a pseudo random pattern (unique data)

To generate a counter with shuffled bits, select the *Mixed* option and enter an offset to the bus address in the range $0 \dots (2^{20} - 1)$.

The generated data is a 64-bit wide pattern that changes with a period of 2^{21} and appears as a pseudo-random sequence.

- Generating a ground bounce pattern

To generate data where `0x00000000` and `0xffffffff` patterns alternate, select the *GroundBounce* option.

NOTE This pattern can be used for unidirectional data path verification. For more information, see “*Performing Unidirectional Data Path Verification*” on page 115.

Real-Time Data Compare

The compare unit on the Agilent E2929/E2930 testcard can be used by both the Exerciser's initiator and target. If enabled, the initiator uses it for read operations and the target decoders use it during write operations.

For the initiator, the usage of the compare unit can be enabled and disabled for every block transfer individually. For the target, single target decoders can be set up to do data comparisons. These decoders are Bar 0-LO, Bar 2-LO and Bar 4. In both cases, all devices can employ the compare unit when they receive data. Only those bytes are compared that are not disabled by byte enable settings.

Compare Unit Usage

The compare unit compares the incoming data with the reference data held in the data memory or with that coming from the data generator. It supplies a signal for the trace memory in the case of a comparison error.

If a comparison error occurs, the testcard can use the error output signal to trigger the trace memory in order to capture the data traffic preceding and following the error.

Detailed information on the circumstances of the error, such as the used bus command, the bus address, the extended command, any attributes, the actual and reference data can be obtained from the trace memory.

When a comparison error occurs, the requester-initiator completes the current iteration of blocks and stops. This allows you to run a software controlled compare over the entire specified data movement.

For more information, refer to *Capturing Data in the Trace Memory* in the *Agilent E2929/E2930 PCI-X Analyzer User's Guide*.

Reference Data

If the data memory supplies the reference data, the internal data space assigned to the initiator is protected from target access and can therefore not be overwritten by the target. However, the initiator can use data that has previously been recorded by the target.

Programmable Data Path Configurations

The data source can be selected for each block line. There is a common incoming and outgoing data source for each block line. The settings influence the start latencies of requester-initiator blocks.

In the Requester-Initiator Block Editor dialog box, the source for each block line is controlled by the block property *Resource*.

Block property *DataCmp* controls if the data is compared against the internal data compare unit or if it is transferred directly to the data source.

Outgoing data (write as initiator or read from target) can originate from:

- The data memory.
- The data generator.

Incoming data (read as initiator or write to target) can be:

- Written into the data memory.
- Compared with the content of the data memory.
- Compared with the output of the data generator.

Different settings of these requester-initiator block properties result in the following configurations:

Data Path Configurations		Resource	
		DataMem	DataGen
DataCmp	Off	Initiator reads from memory. Initiator writes data from memory. Target reads from memory. Target writes to memory.	Initiator reads and discards the data. Initiator writes data from generator. Target reads from generator. Target writes and discards the data.
	On	Initiator reads data that is compared to memory. Initiator writes from memory. Target reads from memory. Writes to the target are compared with the content of the memory.	Initiator reads data that is compared with the output of the data generator. Initiator writes from data generator. Target reads from data generator. Writes to target are compared with the output of the data generator.

Generating Interrupts

Devices that do not have REQ# and GNT# lines connected to them cannot request an access to the PCI-X bus as a requester-initiator device. However, they can generate an interrupt to request servicing from the system software.

PCI-X Interrupts The PCI-X specification defines the following four interrupt signals:

- INTA#,
- INTB#,
- INTC#,
- INTD#.

Single-function devices with only one configuration space always have to use INTA#. Multi-function devices act as several independent devices on the PCI-X bus and have one configuration space each. They may use different interrupt signals as long as they meet the following preconditions:

- INTB# may only be used, if INTA# is covered by at least one function of the multi-function device.
- INTC# may only be used, if INTB# is covered by at least one function of the multi-function device.
- INTD# may only be used, if INTC# is covered by at least one function of the multi-function device.

Apart from these restrictions, any combination of interrupts is permitted by the specification. The interrupts can also be shared by several devices.

The PCI-X specification permits a wide range of scenarios how the four PCI-X interrupts, the ISA initiator interrupt controller, the ISA slave interrupt controller, and optionally, a programmable interrupt router may be interconnected. See the PCI-X specification for details.

The interrupt features provided by the Agilent E2929/E2930 testcard are described in “*Interrupt Capabilities of the Testcard*” on page 122. “*How to Assert and Deassert Interrupts*” on page 123 shows how to use these features.

NOTE You can also program the PCI-X Exerciser to generate interrupts when certain blocks or behaviors are executed (see “*Programming Injection of Errors, Interrupts and Triggers*” on page 71).

Interrupt Capabilities of the Testcard

The Agilent E2929/E2930 testcard can generate any of the PCI-X interrupts INTA#, INTB#, INTC#, and INTD#. This functionality is essential when developing interrupt drivers for PCI-X devices. Interrupts can be asserted and deasserted by means of the Graphical User Interface.

Interrupt Status Register The current status of the interrupts—asserted or deasserted—is stored in the interrupt status register. This register is located in the private section of the configuration space header. It can be read, for example, by interrupt drivers to determine whether the testcard has generated an interrupt.

With the use of this register, any interrupt and any combination of interrupts can be generated by the testcard. Thus, you can simulate both single-function devices and multi-function devices.

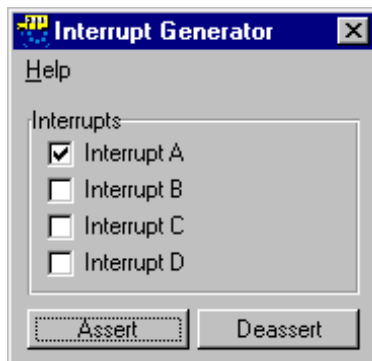
Furthermore, the testcard can act as several devices asserting interrupts at the same time. It can also behave as a multi-function device that generates an illegal combination of interrupts which is not PCI-X specification compliant.

How to Assert and Deassert Interrupts

By asserting and deasserting interrupts, you can examine the behavior of the system under test in response to the various interrupts and their combinations.

To generate the interrupts, proceed as follows:

- 1 Select *Interrupt Generation* from the *Exerciser* menu in the main window.



- 2 Select the interrupts that you want to assert or deassert.
- 3 Click on the *Assert* or *Deassert* button.

The specified interrupts will be asserted or deasserted immediately.

CAUTION

Asserting interrupts can cause the system under test to hang. If you are running the control software (GUI) on the system under test, unsaved data will be lost. Either run the GUI on a remote PC or save all your settings and test data before asserting interrupts.

Using the Command Line Interface

The Command Line Interface (CLI) provides a very convenient way to directly call the C functions that control the Agilent E2929/E2930 testcard. You simply enter the commands with their parameters into the CLI window. This way you can communicate interactively with the testcard.

NOTE It is recommended to program either only via GUI or only via CLI.

Basically, you can call nearly any of the testcard's C functions via the CLI. However, to obtain the full flexibility and control of the testcard's features, it is recommended to use the Application Programming Interface (C-API) available as option 320 for your Agilent E2929/E2930 testcard.

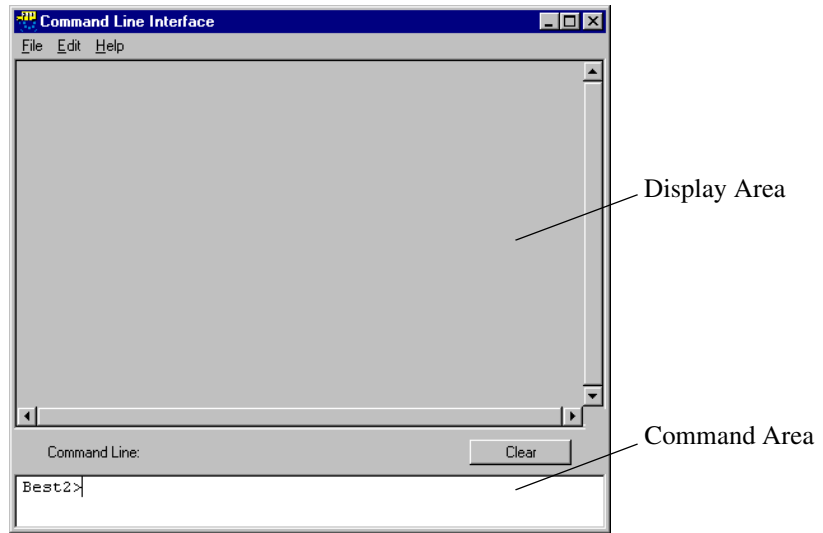
There is one CLI command for every C function, but within the GUI only those commands are supported that are part of your licensed software components. For example, if you do not have the Analyzer option enabled, you cannot call its C functions via the CLI.

A description to all CLI commands and their CLI abbreviations can be found along with the respective C-API function descriptions in the Agilent E2929/E2930 *C-API/PPR Programming Reference*.

How to Start the CLI

To open the Command Line Interface window:

- ◆ Select *Command Line* from the *Windows* menu in the main window.



The Command Line Interface window is divided into two areas: The display area and the command area. The CLI commands (or their abbreviations) are entered in the command area and executed with the return key. The commands and the system responses are then listed in the display area.

TIP Use the cursor-up and cursor-down keys in the command area to scroll through the last 50 command lines entered.

Basic CLI Command Syntax

Because calls to C functions in C programs are not very informative, a more convenient syntax is used for the CLI commands and the CLI abbreviations. The software interprets the command lines and calls the respective functions with the specified parameters, if applicable.

The syntax for the three types of CLI commands is:

- Commands without parameters.

Commands that do not require any parameters are entered as they are. The following example simply checks the connection to the testcard. As a result the LEDs on the testcard flash five times.

Example: `BestXPing`

Or the abbreviation: `ping`

- Commands that pass parameters.

Use the “equals” sign to pass a value to a command’s parameter. The example shows how to lock the onboard resource *Exerciser*.

Example: `BestXResourceLock resource=BX_RESLOCK_EXERCISER`

Or the abbreviation: `reslock res=exe`

- Commands that request parameters.

The syntax for commands that request parameters is the same as when passing parameters. For example, check whether the previous command has locked the Exerciser successfully.

Example: `BestXResourceIsLocked resource=BX_RESLOCK_EXERCISER`

Or the abbreviation: `resislock res=exe`

Assuming that a connection has been established to the testcard and no error has occurred, the system reports in the display area how often and from which port the Exerciser has been locked.

Using CLI Scripts

A CLI script is a sequence of CLI commands that can be stored in a text file. Such script files can later be loaded and executed in the Command Line Interface window. This reduces the typing effort if you want to use the same sequence of CLI command lines several times.

Generating Scripts

To generate and save a CLI script, you can:

- Use any text editor. Enter the sequence of CLI commands in the text editor and save it as a text file.
- Copy and paste the command lines from either CLI area into the text editor and save the editor file.
- Use command logging. While you are executing CLI commands, you can log this input into a script file. To start logging, select *Start Logging* from the *File* menu and specify a script file name in the file dialog box. From this moment, all the command lines that you type get logged to the script file until you select *Stop Logging* from the *File* menu.

Executing CLI Scripts

There are two different ways to run a CLI script from a file:

- Select *Run Script* from the *File* menu of the CLI window and pick the script file from the file dialog box.
- In the CLI command area, use the command `do scriptfile`, where `scriptfile` is the file name together with the absolute or relative path.

NOTE

Command logging only stores the sequence of command lines. If you want to store the complete contents of the display area for later information or comparison with other reports, select *Save As* from the *File* menu.

Index

... times (Execution of Blocks) 70

A

Add Transactions 62

Address Range 86

Agilent Testcard's Configuration Space Header 78

Architecture of the PCI-X Exerciser 12
Diagram of components 13

B

Base Address

Parameters 82

Registers 78

Basic CLI Command Syntax 127

Behavior Editor

Completer-Initiator 98

Completer-Target 93

Requester-Initiator 63

Requester-Target 106

Behavior Memory

Completer-Initiator 98

Completer-Target 93

Requester-Initiator 63

Requester-Target 105

Behaviors

Available CI Behaviors 99

Available CT Behaviors 94

Available RI Behaviors 64

Available RT Behaviors 106

Completer-Target 93

Programming CI Behaviors 98

Programming RT Behaviors 105

Requester-Initiator 63

BIOS Settings, Overwriting 90

Burst Transactions 58

Bus Address (AD32, AD64) 59

Bus Command (buscmd) 59

Byte count (RI behavior) 64

Byte Enable Control (Byten) 61

C

CLI Scripts 128

Command Line Interface

Example 40

Command Line Interface (CLI) 125

Command Line Interface (window) 126

Command Syntax 127

Scripts 128

Starting 126

Command Syntax (CLI) 127

Compare Flag (DataCmp) 60

Completer-Initiator (CI)

Available Behaviors 99

Definition 12

PCI-X Exerciser as a CI Device 97

Programming Behaviors 98

Completer-Target (CT)

Available Behaviors 94

Components Overview 76

Decoder Properties 79

Decoders 76

Definition 12

PCI-X Exerciser as a CT device 75

Programming 85

Programming Behaviors 93

Conditional Start (condstart) 60

Configuration Decoder 82

Configuration Space 76, 83

(window) 88

Header 77

Header for Agilent Testcard 78

Modifying Header 87

Configurations

Programmable Data Paths 119

D

Data Alignment 110

Data Compare 118

Data Compare Unit 118

Data Generator

Features 117

Overview 114

Use 116

Data Memory 83

Editor 112

Organization 110

Data Path

Configurations 119

Verification 115

Data Phases 57

Data resources 83

Configuration Space 83

Data Generator 114

Data Memory 83

Expansion ROM 83

Internal DBI Register 83

Use 109

Databases

Standard and Power-Up 91

Disconnect (RI behavior) 64

Documentation Overview 7

DWORD Transactions 58

E

Error Injection page 71

Execution of Blocks 70

Exerciser Generic Settings (dialog box) 70, 71

Exerciser Status Bar 69

Expansion ROM 83

Expansion ROM Decoders 81

I

Infinite (Execution of Blocks) 70

Initiator page

Execution order of blocks 70

Reaction to terminations 71

Split completion message 101

Internal Address (IntAddr) 61

Internal DBI Register 83

Interrupt 121

(De)asserting 123

Interrupt Generation (dialog box) 123

Status Register 122

Interrupt Injection page 71

L

Latencies

between Requester-Initiator

Transactions 65

Completer-Target 95

M

Manual Stop (RI) 73

Match Indicator 88

N

Number of Bytes (NumBytes) 59

O

Offline/Demo Mode (radio button) 22
 Optimization of the PCI-X Exerciser 10
 Out-of-order Block Execution 68
 Outstanding Request Completion (Completion) 60
 Overview
 Documentation 7
 PCI-X Exerciser 9
 Requester-Initiator Transactions 55

P

PCI-X Exerciser
 as a Completer-Initiator Device 97
 as a Completer-Target Device 75
 as a Requester-Initiator Device 53
 as a Requester-Target Device 103
 Overview 9
 Sample Session 19
 PCI-X Interrupts 121
 PCI-X Transactions *see* Transactions
 Phases of RI transactions 56
 Power-Up Database 91
 Programmable Data Path Configurations 119
 Programming
 Completer-Initiator Behaviors 98
 Completer-Target Behaviors 93
 Completer-Target Decoders 85
 Requester-Initiator Behaviors 63
 Requester-Initiator Transactions 54
 Requester-Target Behaviors 105
 RI Execution of Blocks 70
 RI Reaction to Terminations 71

Q

Queue (RI behavior) 64

R

Reaction to Terminations (RI) 70
 Real-Time Data Compare 118
 Reference Data for Data Compare 118
 Remove Transactions 62
 Requester-Initiator (RI)

Available Behaviors 64
 Definition 12
 How to specify transactions 62
 Latencies 65
 PCI-X Exerciser as a RI device 53
 Programming Behaviors 63
 Programming Execution Order of Blocks 70
 Programming Transactions 54
 reaction to terminations 71
 Starting 66
 Status 69
 Stopping 73
 Transactions Overview 55

Requester-Target (RT)
 Available Behaviors 106
 Decoder Setup 104
 Definition 12
 PCI-X Exerciser as a RT device 103
 Programming Behaviors 105
 Resource Base Address and Size 86
 Run Options 67
 Run Status (RI) 68

S

Sample PCI-X Exerciser Session 19
 Scripts (CLI) 128
 Select View Of Data (dialog box) 113
 Settings
 Generic Exerciser Settings 70, 71
 Transactions 62
 Setup
 PCI-X Exerciser Test 17
 Requester-Target Decoder 104
 Target Decoders 84
 Source-Synchronous Clocking 57
 Specification of RI transactions 62
 Split Completion 18
 Split Completion Transactions 102
 Split Response Condition 96
 Standard Database 91
 Standard Decoders 81
 Start Condition (Requester-Initiator Device) 67
 Starting the Requester-Initiator 66

Stopping the Requester-Initiator 73

Stops

 on Initiator Abort 71
 on Target Abort 71

T

Target Decode (Window)
 example 36
 Target Decode (window) 85, 104
 Target Decoders
 Configuration Decoder 82
 Expansion ROM Decoders 81
 Programming 85
 Properties 79
 Setup 84
 Standard Decoders 81
 Test possibilities 16
 Test Setup 17
 Testcard, Interrupt Capabilities 122
 Transaction Properties 59
 Transactions 17
 Adding 62
 Burst 58
 DWORD 58
 Latencies (RI) 65
 Phases of RI Transactions 56
 Programming RI Transactions 54
 Removing 62
 Requester-Initiator, Overview 55
 Settings 62
 Split Completion (CI) 102
 with Split Completion 18
 Transfer Queue (Queue) 61
 Trigger Injection page 71

U

Unidirectional Data Path Verification 115

V

Validation of the PCI-X Exerciser 11

W

Waveform Viewer
 example 26