# VANGUARD Bus Analyzer

## User Guide
## PCI/PCI-X Analyzer and Exerciser
## for
## BusView5

Document No. A-T-MU-BAPCIX##-A-0-A2

## Notice

The information in this document is subject to change without notice and should not be construed as a commitment by CWDS. While reasonable precautions have been taken, CWDS assumes no responsibility for any errors that may appear in this document.

## Trademarks

Trademarked names appear throughout this document. Rather than list the names and entities that own the trademarks or insert a trademark symbol with each mention of the trademarked name, we hereby state that the names are used only for editorial purposes and to the benefit of the trademark owner with no intention of improperly using the trademark.

The mention of any trademarked name is not intended to imply that CWDS products are affiliated, endorsed or sponsored by such trademark owner.

## Software and Firmware Licensing

Any Software and Firmware code provided by CWDS described herein is proprietary to CWDS or its licensors. The use of this Software and Firmware is governed by a licensing agreement included on the media on which the Software and Firmware was supplied. Use of the Software or Firmware assumes that the user has agreed to the terms of the licensing agreement. CWDS retains all rights to the Software and Firmware under the copyright laws of the United States of America and other countries. This Software or Firmware may not in contravention of the licensing agreement be furnished or disclosed to any third party and may not be copied or reproduced by any means, electronic, mechanical, or otherwise, in whole or in part, without specific authorization in writing from CWDS.

# Technical Support

Technical documentation is provided with all of our products. This documentation describes the technology, its performance characteristics, and includes some typical applications. It also includes comprehensive support information, designed to answer any technical questions that might arise concerning the use of this product. We also publish and distribute technical briefs and application notes that cover a wide assortment of topics. Although we try to tailor the applications to real scenarios, not all possible circumstances are covered.

While we have attempted to make this document comprehensive, you may have specific problems or issues this document does not satisfactorily cover. Our goal is to offer a combination of products and services that provide complete, easy-to-use solutions for your application.

If you have any technical or non-technical questions or comments, contact us. Hours of operation are from 8:00 a.m. to 5:00 p.m. Eastern Standard/Daylight Time.

- Phone: (937) 252-5601 or (800) 252-5601
- E-mail: **DTN_support@curtisswright.com**
- Fax: (937) 252-1465
- World Wide Web address: www.cwcdefense.com

**Curtiss-Wright Defense Solutions**
2600 Paramount Place Suite 200
Fairborn, OH 45324 USA
Tel: 800-252-5601 (U.S. only)
Tel: 937-252-5601

# *Preface*

This User Guide explains the process of preparing, installing, and using your Vanguard. It is divided into the following sections:

- Product Overview
- Hardware Description
- Getting Started with BusView®
- State Analyzer
- Statistics Functions
- Protocol Checker
- Exerciser
- Simulator
- Compliance Checker
- Zero Slot Adapter
- Appendixes: Specifications, Signals, Embedded Systems.

### Style Conventions Used

- Code samples are `Courier font` and at least one size less than context.
- Directory path names are *italicized*.
- File names are in **bold**.
- Absolute path file names are ***italicized and in bold***.

**Warning!** Information contained in this box must be observed. It will provide information about situations that may be dangerous.

**Note –** Information contained in this box is important and will help you get the best performance from your Vanguard.

### Symbols

Tip – This information offers guidance in using your Vanguard for certain situations.

**Quality Assurance**

Curtiss-Wright's Corporate policy is to provide our customers with the highest quality products and services. In addition to the physical product, the company provides documentation, sales and marketing support, hardware and software technical support, and timely product delivery. Our quality commitment begins with product concept, and continues after receipt of the purchased product.

Curtiss-Wright's Quality System conforms to the ISO 9001 international standard for quality systems. ISO 9001 is the model for quality assurance in design, development, production, installation and servicing. The ISO 9001 standard addresses all 20 clauses of the ISO quality system, and is the most comprehensive of the conformance standards.
Our Quality System addresses the following basic objectives:

- Achieve, maintain, and continually improve the quality of our products through established design, test, and production procedures.
- Improve the quality of our operations to meet the needs of our customers, suppliers, and other stakeholders.
- Provide our employees with the tools and overall work environment to fulfill, maintain, and improve product and service quality.
- Ensure our customer and other stakeholders that only the highest quality product or service will be delivered.

The British Standards Institution (BSI), the world's largest and most respected standardization authority, assessed Curtiss-Wright's Quality System. BSI's Quality Assurance division certified we meet or exceed all applicable international standards, and issued Certificate of Registration, number FM 31468, on May 16, 1995. The scope of Curtiss-Wright's registration is: "Design, manufacture and service of high technology hardware and software computer communications products." The registration is maintained under BSI QA's bi-annual quality audit program.
Customer feedback is integral to our quality and reliability program. We encourage customers to contact us with questions, suggestions, or comments regarding any of our products or services. We guarantee professional and quick responses to your questions, comments, or problems.

# Contents

# *Figures*

# *Tables*

# 1 Product Overview

This section gives a broad overview of the main functions provided by the Vanguard.

- Overview
- Analyzer
- Protocol Checker
- Statistics Functions
- Exerciser (PCI & VME)
- Host Exerciser

## *1.1 Overview Curtiss-Wright*

Curtiss-Wright is a company totally committed to building the finest bus analyzers, and is recognized in development laboratories around the world as providing superior tools for developers and manufacturers of bus based computer equipment.

The Vanguard is the result of more than 15 years experience in building bus analyzers. The Vanguard Analyzer has evolved through four generations of VME bus analyzers, five generations of PCI bus analyzer and two generations of PCI-X analyzer.

A bus analyzer is a pre-configured logic analyzer designed as a plug-in card conforming to the logical, electrical and mechanical specification of the target bus. The primary use of a bus analyzer is to monitor the activity on a back plane bus and provide a trace of bus cycles between modules. This information is then presented as alphanumeric trace lists or as waveforms.

The serial protocol analyzer samples the symbols on the serial link and presents the packets formed in a graphical view. This is done without connecting and configuring large numbers of probes to the back plane, a time-consuming and error-prone process necessary with general-purpose logic analyzers.

Statistical analysis of a bus system provides extensive performance monitoring in real time. This analysis can identify latencies, device throughput measurements and efficiency.

For parallel buses (VME, PCI and PCI-X), the Vanguard also includes an Exerciser unit that allows you to generate traffic, emulate a target, and generate interrupts.

The built-in protocol checker detects protocol errors and bus anomalies without the need for detailed knowledge of particular bus specifications and serial protocols.

### Vanguard

The Vanguard product family provides a suite of tools for use in developing, testing and fault-finding in chipsets, motherboards, add-in boards and software which use PCI, PCI-X, and VME busses. The tools are controlled through a Windows-based program called *BusView*®.

### The Tools

The tools are categorized as follows.

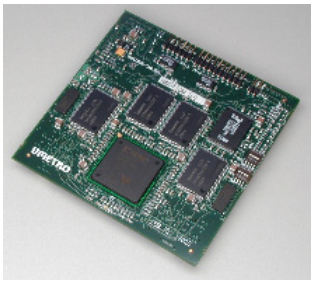
- *Analyzer*: Bus sampling and analysis.
- *Protocol Checker*: Detects bus protocol violations.
- *Statistics Functions*. Statistical representation of a number of bus performance issues.
- *Exerciser*: Provides a method of applying traffic (cycles) onto the bus. An enhanced Exerciser is also available for PCI-X, adding the functionality to inject errors onto the bus as well providing more flexible control of Master and Target behavior.
- *Compliance Verification (PCI and PCI-X)*. Performs compliance checklist scenarios, including those defined by the PCI SIG (Special Interest Group).

## The Hardware

The Vanguard Hardware is available in several form factors.

- **PCI** - A PCI carrier card contains the Exerciser. A SAM module is added to provide Analyzer, Protocol Checker and Statistics functions. Also available is the 0-slot PCI adapter which allows a PCI card (Device Under Test) to be inserted into the same slot as the PCI Vanguard.
- **Compact PCI** - the cPCI carrier board contains the Exerciser. A SAM module is added to provide Analyzer, Protocol Checker and Statistics functions.
- **VME** - A VME carrier board is used in conjunction with a SAM module to provide the Analyzer, Protocol Checker and Statistics functions. An additional Exerciser daughter card can be added. It is also possible to mount the PMC Vanguard for analysis of PCI/PCI-X on the P0 connector.
- **PMC** - The PMC board contains the Analyzer, Protocol Checker, Statistics and Exerciser functions.

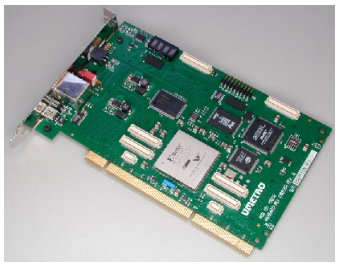
### SAM (State Analyzer Module)



The SAM (State Analyzer Module) is a removable daughter card containing the State Analyzer, Protocol Checker and Statistics functions, and can be used with PCI, VME and Compact PCI carrier cards.

The SAM module can be freely swapped between these carrier cards.

### PCI Carrier



The PCI carrier contains the Exerciser and accommodates the SAM module.

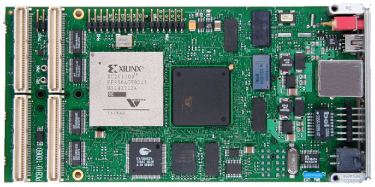
It can be used in any 32 or 64 bit slot in a PCI/PCI-X motherboard.

If it is to be inserted into a 5V slot then the accompanying 5V adapter must be used.

### PCI Zero Slot Adapter

The Vanguard PCI Zero Slot Adapter includes a PCI to PCI bridge and an edge connector to allow for the addition of another PCI card on top of the Vanguard.

### PMC form factor



The Analyzer, Statistics, Protocol Checker and Exerciser functions are all built into the PMC form factor.

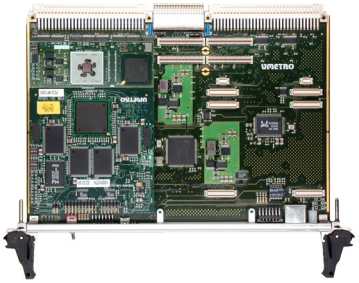
The Vanguard PMC can be combined with a Top Spacer board allowing it to be used in single slot systems.

### Compact PCI

The CompactPCI Vanguard is a 3U Eurocard based form connected via a passive PCI backplane.

### VME

The VME card carries the SAM and the Exerciser modules.



The VME boards can be installed in any VME slot, however, if it is to act as System Controller then the Exerciser module must be present on the board.

There is also a VME P0 version available allowing a Vanguard PMC to be installed to give access to PCI/PCI-X on the P0 connector.

## The Software - BusView®

BusView is a Windows compatible program from which all of the Vanguard tools are controlled. Once installed onto a compatible workstation, BusView will connect to the hardware. BusView can be installed onto the same system in which the Vanguard is installed, or it can be installed in a different system.

The connection between the Vanguard and BusView is one of two types:

- Direct Connection - Connecting directly to the Vanguard via a USB cable, or a crossover Ethernet cable.

- Remote Connection - BusView connects to the Vanguard via Ethernet over a network.

## *1.2 Functional Overview*

Figure 1-1 illustrates how bus signals are acquired in the Vanguard.

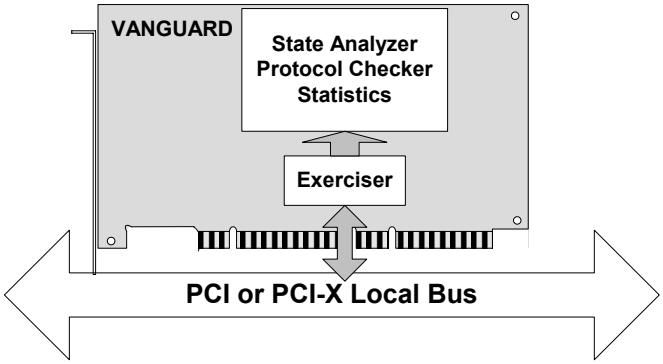



**FIGURE 1-1** *Analyzer acquires bus signals*

### Analyzer

The Analyzer is responsible for sampling, storing, and displaying bus or link traffic.

The Analyzer will stop sampling according to “trigger conditions” and stores samples of the bus activity according to “trigger and store conditions”. A sequencer is also available for building trigger conditions that consist of more than one event or a sequence of events. Once the samples are stored, they are viewed in the “Trace Display” window.

### Protocol Checker

The protocol checker detects protocol violations. This helps uncover design, manufacturing and field-failure-induced flaws in Root Complexes, Switches and Endpoints.

It will also uncover a wide variety of problems originating in other parts of a board that directly or indirectly causes illegal bus activity. This is achieved by monitoring the packet exchange between two devices. that are present in the system configuration. This can help, for example, determine why a board from a particular manufacturer does not function correctly in a system from another manufacturer.

The difficult part of debugging is usually determining which conditions to trigger on. Often, the symptoms of failure give no clue as to their cause. Since all of the Vanguard tools can be used simultaneously, the possibility of triggering the Analyzer as result of a protocol violation significantly enhances the versatility of the Vanguard.

## Statistics Functions

The Statistics Functions offer several pre-defined statistics as well as up to eight ‘Event Counters’ which can be used together in mathematical formula to count the occurrence of user-selectable events. Used in conjunction with Customized Charts, you can build your own statistical reports for more advanced bus performance analysis.

All statistics can be run simultaneously, in real time. Additionally, a trace counting function is available for generating statistics from a captured trace file offline.

## Exerciser (PCI & VME)

The exerciser acts as a reference unit that generates VME and PCI bus cycles with known characteristics. It is usually used for testing new boards or to assist in the debugging of boards. It can also load the system with heavy bus traffic for performance checks.

The exerciser can also be used to patch data in memory without influencing the operation of other masters on the bus.

Use of the PCI/PCI-X Exerciser requires the “VG-E” or “VG-E2” license. Use of the VME Exerciser requires the “VG-VE” license. See “Ordering Information” on page 427.

## Host Exerciser

The Host Exerciser is software installed on a target machine located. The Host Exerciser is controlled from BusView via a network.

## PCI and PCI-X Compliance Verification

Both master and target tests can be performed to ensure PCI and PCI-X compliance according to the PCI SIG (Special Interest Group) compliance checklist.

Refer to:

“PCI 2.2 Compliance Checklist” and “PCI-X 1.0a Compliance Checklist” documents published by PCI SIG.

# 2 Hardware Description

- Vanguard PCI Assembly - PCI Carrier and SAM
  - 5V Adapter Card
- Vanguard PMC Assembly
  - Top Spacer
- Vanguard cPCI Assembly - cPCI Carrier and SAM
- Vanguard Hardware Features
- External Power
- Miscellaneous Parts
- Hardware Menu
- Moving the SAM

ESD – Static electricity can permanently damage your Vanguard. Prevent electrostatic damage by following the static electricity precautions listed in the Installation Guide.

## 2.1 Vanguard PCI Assembly - PCI Carrier and SAM

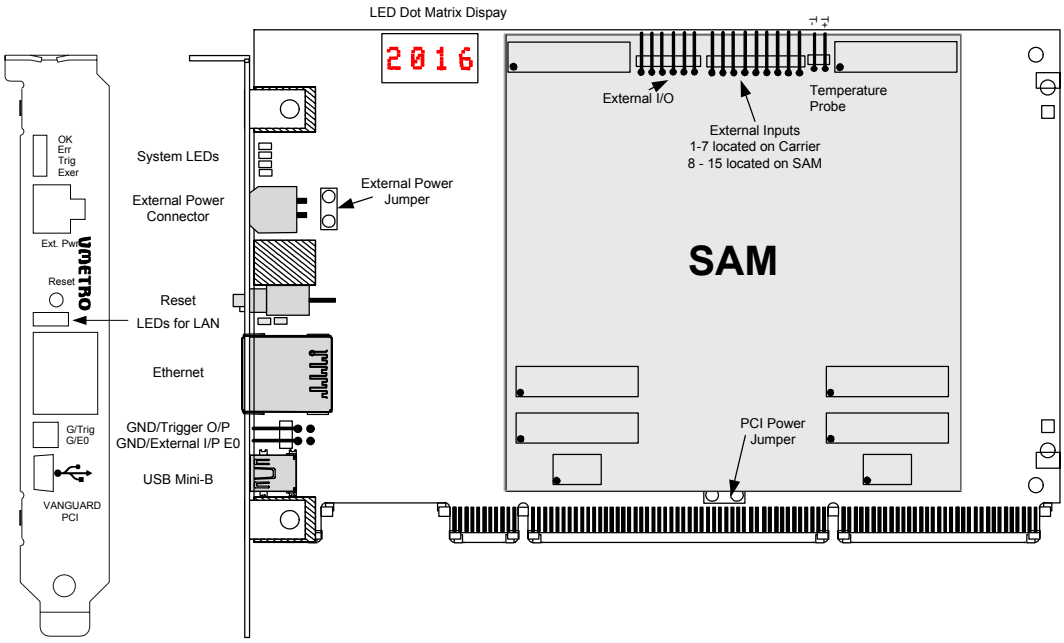



**FIGURE 2-1** *Vanguard PCI board layout*

### 5V Adapter Card

Since the Vanguard uses 3.3V signalling as standard, this adapter card is necessary for systems using 5V signalling. The 5V Adapter can only be used for a 5V 33MHz PCI system. See the Installation Guide for instructions on assembling the adapter card to the Vanguard.

**Note –** If an external power source is used, the external Power connector on the Vanguard main board will be blocked by the adapter card front panel. Use the external power connector located on the adapter card instead.

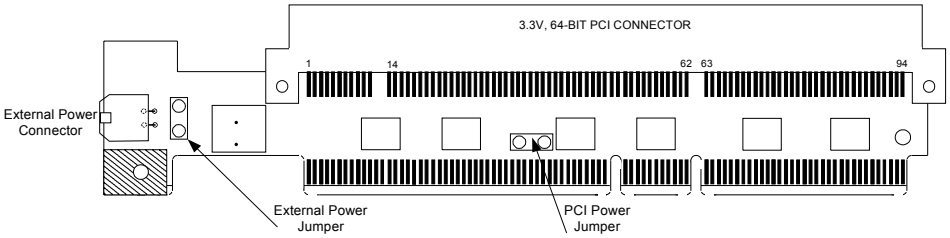



**FIGURE 2-2** *5Volt Adapter Card*

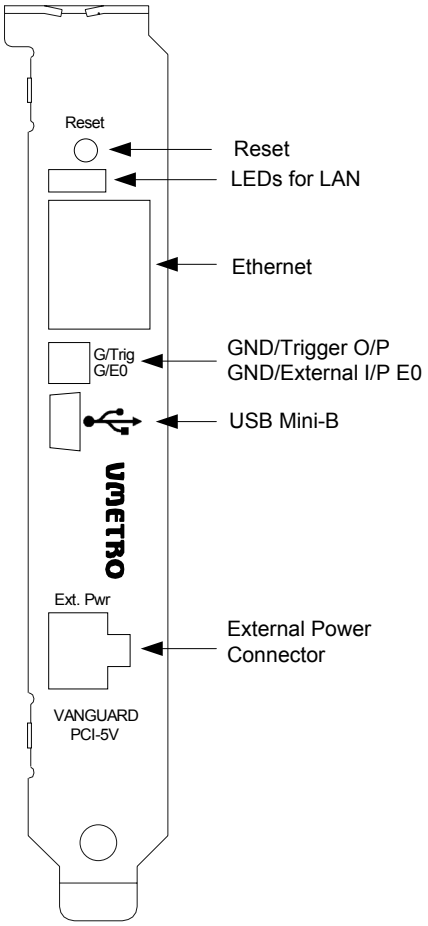
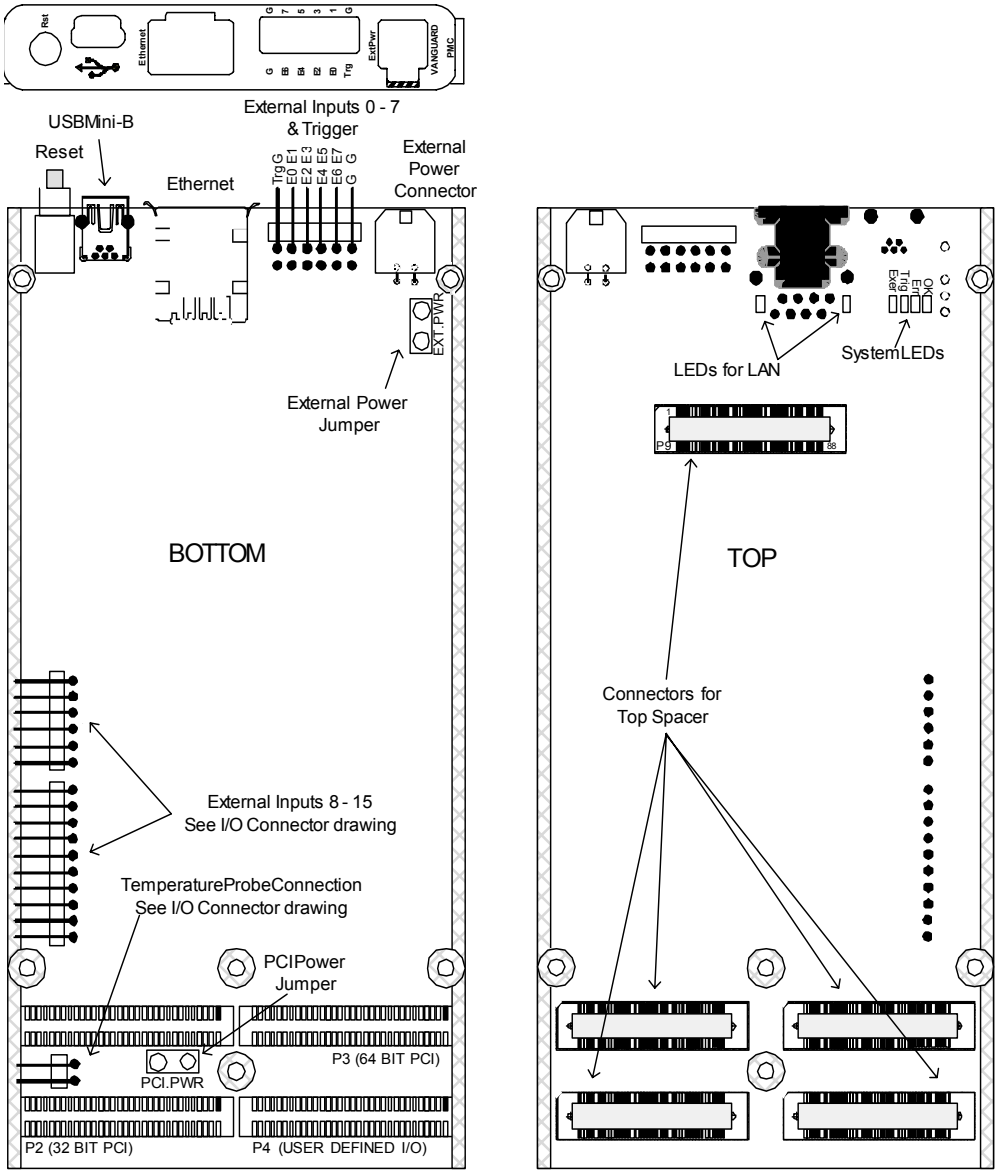



**FIGURE 2-3** *Front Panel used with the 5V adapter*

The blue jumper located on the Vanguard main board must be in the PCI.PWR position when this 5V adapter card is used.

## *2.2 Vanguard PMC Assembly*

### Vanguard PMC board



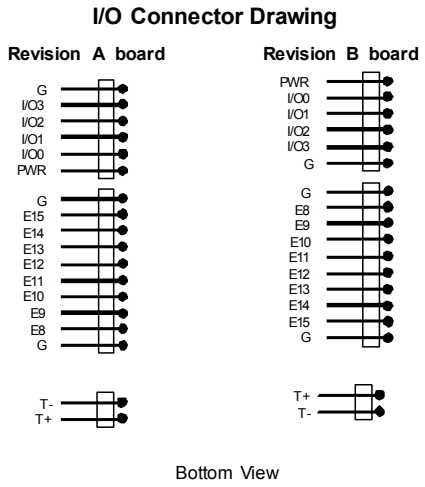



**FIGURE 2-4** *Vanguard PMC board layout*

## Top Spacer

If all of the PMC slots on the carrier board are occupied, the Top Spacer can be used to place the Vanguard PMC in between the carrier board and the PMC module under test.

The Top Spacer extends the PMC slot and enables the PMC module to be mounted on top of the Top Spacer. The spacer has male connectors on top and female connectors on the bottom that match the connectors found on the top of the Vanguard.

The spacer is designed so that the front panel of the PMC module under test is resting on top of the Top Spacer ensuring that the stacked module remains parallel with the carrier board.

---

**Note –** The Top Spacer regenerates the PCI CLK signal using a PLL. The PLL frequency range is from 10MHz to 133.33MHz. It is this PCI CLK signal that is supplied to a PMC board mounted on top of the Top Spacer.

---

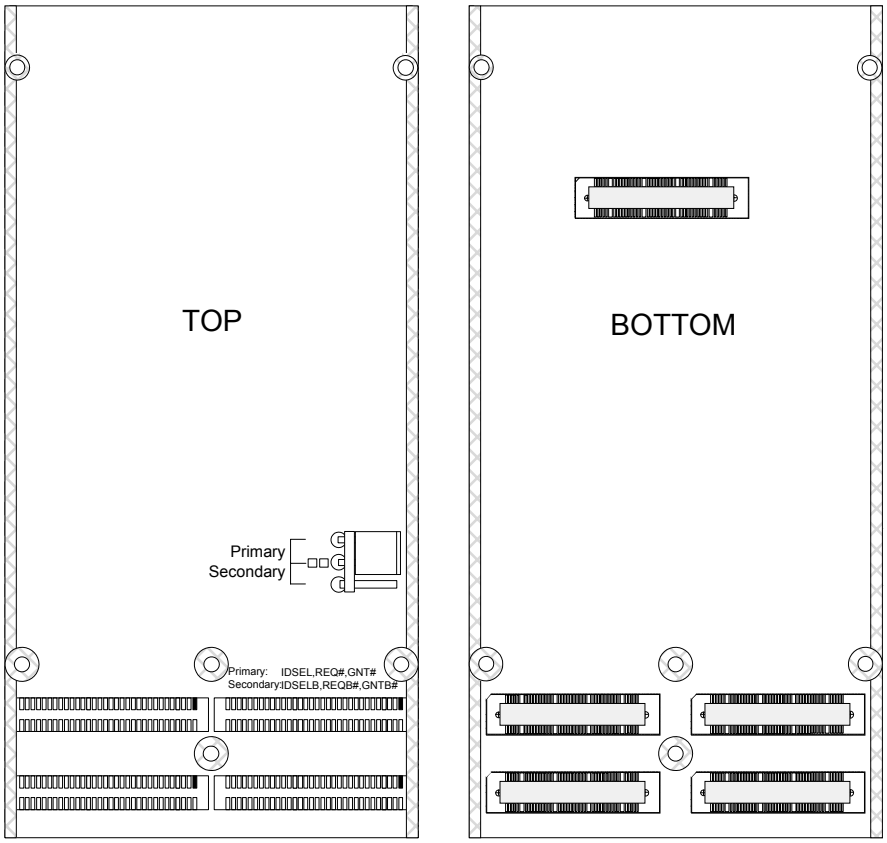



**FIGURE 2-5** *Vanguard PMC Top Spacer*

### Jumper Positions

**Primary Position** When in the Primary position, the jumper controls a quick switch that muxes REQ#/GNT#/IDSEL from the carrier board to REQ#/GNT#/IDSEL on the Top Spacer PMC connector.

**Secondary Position**When in the Secondary position, the jumper controls a quick switch that muxes REQB#/GNTB#/IDSELB from the carrier board to REQ#/GNT#/IDSEL.

---

**Note –** REQB#/GNTB#/IDSELB from the carrier board is always connected to REQB#/GNTB#/IDSELB Top Spacer PMC connector regardless of jumper position.

---

Below are the four possible configurations of Carrier/PMC cards with the VG-PMC/Top Spacer in conjunction with the Primary/Secondary jumper setting. The limiting factor to consider is that the carrier board has either one or two sets of REQ#/GNT#/IDSEL signals. Consequently it can only support a maximum of two PCI devices. The Vanguard Exerciser counts as one device.

***Configuration 1: PMC Carrier and PMC card on top of VG-PMC***

- Primary Position: VG-PMC Exerciser is disabled, and the PMC card uses REQ# / GNT# / IDSEL from PMC Carrier.
- Secondary Position: VG-PMC Exerciser is enabled, and the PMC card is not supported. REQ# / GNT# / IDSEL to PMC card are pulled to inactive levels.

***Configuration 2: PMC Carrier and PrPMC card on top of VG-PMC***

- Primary Position: VG-PMC Exerciser is disabled, and the PrPMC card uses REQ# / GNT# / IDSEL from PMC Carrier. REQB# / GNTB# / IDSELB is not supported from PMC Carrier (REQB# / GNTB# / IDSELB to PrPMC card are pulled to inactive levels).
- Secondary Position: VG-PMC Exerciser is enabled, and PrPMC card is not supported, AND MUST NOT BE MOUNTED. Alternatively do not mount the Top Spacer/PrPMC card assembly to enable the VG-PMC Exerciser.

***Configuration 3: PrPMC Carrier and PMC card on top of VG-PMC***

- Primary Position: VG-PMC Exerciser is disabled, and the PMC card uses REQ# / GNT# / IDSEL from PrPMC Carrier.
- Secondary Position: VG-PMC Exerciser is enabled, and the PMC card uses REQB# / GNTB# / IDSELB from PrPMC Carrier.

***Configuration 4: PrPMC Carrier and PrPMC card on top of VG-PMC***

- Primary Position: VG-PMC Exerciser is disabled, and the PrPMC card uses REQ# / GNT# / IDSEL, and REQB# / GNTB# / IDSELB from PrPMC Carrier.
- Secondary Position: VG-PMC Exerciser is enabled, and PrPMC card is not supported, AND MUST NOT BE MOUNTED. Alternatively do not mount the Top Spacer/PrPMC card assembly to enable the VG-PMC Exerciser.

**Voltage Keying**

**Warning!** The Voltage Keying Post must be mounted in the correct position. Failure to do so will cause permanent damage to the board and is a potential fire hazard.
Please see the Installation Guide for instructions.

## 2.3 Vanguard cPCI Assembly - cPCI Carrier and SAM

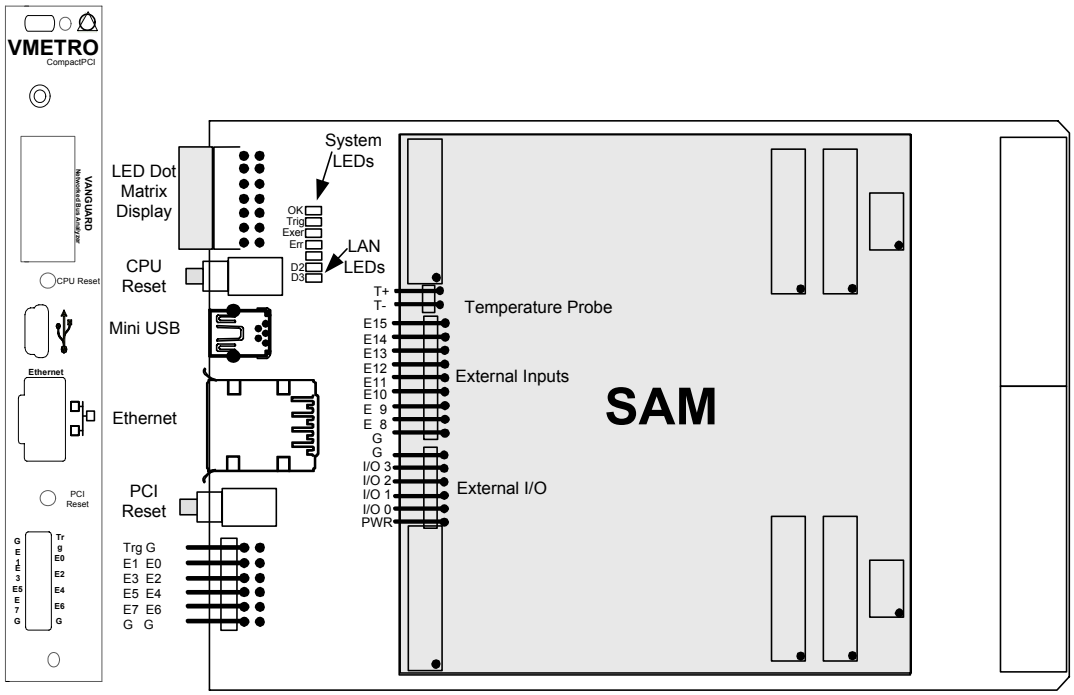



**FIGURE 2-6** *Vanguard cPCI Board Layout*

### System Slot functions

The cPCI back plane is passive, which means that one of the inserted boards has to control some system functions like clock generation, mode detection, arbiter functions and the like. This board is called a system controller, and it is inserted into a dedicated system slot. The other boards are peripheral boards, and they are inserted into one of the peripheral slots. The back plane controls the slot setup by the SYSEN# signal.

The cPCI Vanguard supports system slot functions, but not high level functions like system configuration.

Busview shows "SysSlot" in the status line if the board is inserted into a system slot.

For Mode Detection information, See “Bus Mode” on page 27.

#### Arbiter

The arbiter is a simple round robin type PCI arbiter and is enabled when the board is operating in a system slot. All grants from the arbiter are at least 6 clocks. The arbiter will park the bus on the master that was last given the bus.

### Reset Control

The system slot also controls the PCI reset. There are two reset switches on the front of the board.

- CPU reset - resets the Vanguard CompactPCI board including the CPU.
- PCI Reset - asserts a PCI reset when the CompactPCI Vanguard is operating in a system slot. This has no function when the Vanguard is used in a peripheral slot.

A reset can also be issued from within BusView from the Tools, Hardware menu.

## Hot Swap

The CompactPCI Vanguard supports basic hot swap. When inserted into a periphery slot, the CompactPCI Vanguard can be hot swapped once any bus activity has been terminated. When a board is hot swapped it will adapt to the system mode signalled by the PCIXCAP and M66EN lines.

**Warning!** Warning**!** Ensure power is off before inserting the Vanguard into a system slot. The Vanguard CPU does NOT support hot swapping when used as a system controller.

## 2.4 Zero Slot Adapter (VG-PCI0SL)

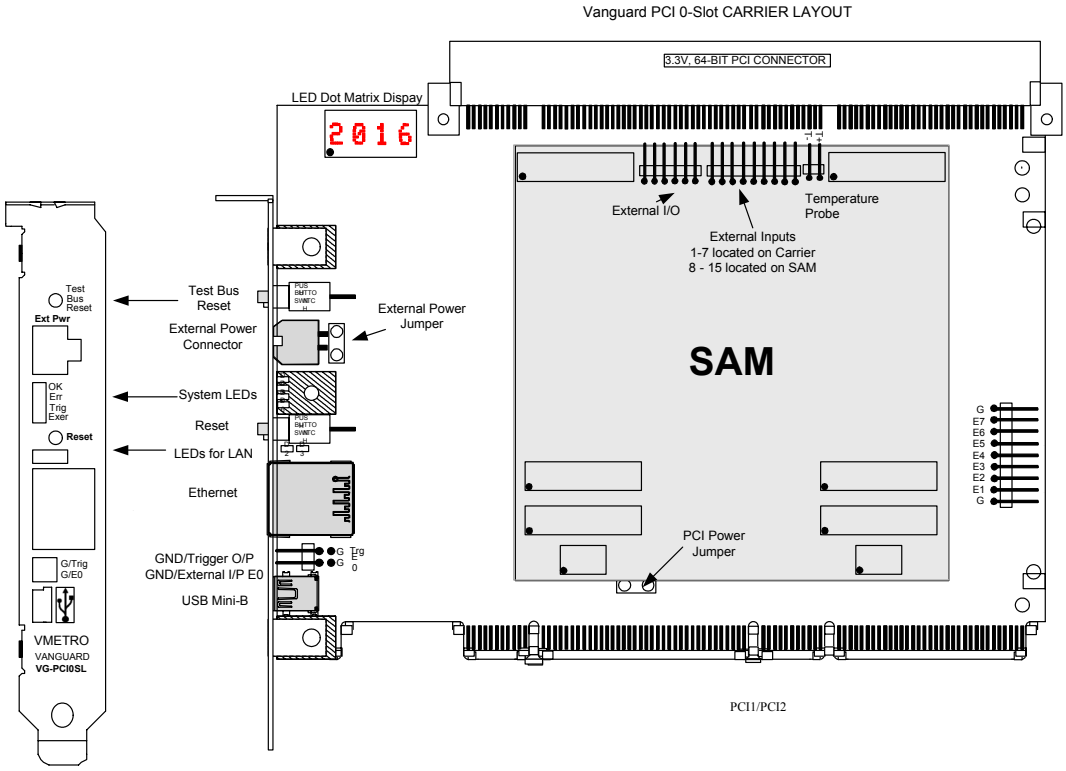



**FIGURE 2-7** *VG-PCI0SL Board Layout*

**VG-PCI0SL Only –** For details about Exerciser Function / Test Bus Mode and Host Bus Modes for the Vanguard Zero Slot Adapter, see “PCI Zero Slot Adapter (VG-PCI0SL)” on page 377

## 2.5 Vanguard Hardware Features

### LED Dot Matrix Display

Available on all models except VG-PMC.

When BusView is not connected to the Vanguard, the display will scroll through the following messages:

- IP address and Port Number on which the Vanguard accepts connections from BusView. Example - “IP: 192.168.168.40:2400”
- Device Name
- Link Configuration

**Note –** The Device Name is only shown if a name has been given to the device using the Change Device Name option.

When BusView is connected to the Vanguard, this display scrolls through a number of user definable strings which are set via the LED Display tab in the Tools/Hardware/Options menu in BusView. If no options have been set, the display will show CONN (Connected).

### System LEDs

These four LEDs indicate the following:

**TABLE 2-1. System LEDs**

| LED Name | Color | Meaning When Lit |
|---|---|---|
| OK | Green | Flashing: Vanguard is powered-up and ready. |
| | | Solid: Vanguard is connected to BusView. |
| Err | Red | A Protocol Violation has been detected by the Protocol Checker. |
| Trig | Yellow | The Analyzer has triggered. |
| **Exer** | Orange | Exerciser is enabled. |

### External Power Connector

If an external power source is used, then the blue jumper(s) must be moved from the “PCI.PWR” position to the **“**EXT.PWR**”**.

**PCI Only –** If the 5V adapter is used, then this connector is not available. Use the external power connector on the 5V adapter card instead, leaving the power jumper in the “PCI.PWR” position.

**External Power Jumper “EXT.PWR”**

The blue jumpers must be in this position to use the External Power Connector.

**PCI Power Jumper”PCI.PWR”**

The blue jumpers must be in this position when using power from the PCI slot.

## Reset

Causes a hardware reset on the Vanguard. This reset is local to the Vanguard and does not affect the target system.

## Ethernet Connector

Connection for Ethernet cable (10/100 Mbit - auto detect).

## LAN LEDs

Indicates activity on the Ethernet connection. The green LED indicates a link has been established; the amber LED indicates send/receive activity over the connection.

## Trigger Output

External Trigger output and ground pin labelled G/Trig

## External Inputs

There are 16 inputs available which are marked E0-E15. These correspond to the Ext[5:0] in the Analyzer Setup and Trace Display windows.

The external inputs are used by connecting a thin probe wire (“Patch Cords“) from the external input pin (E0 for example) to the signal of interest. All of the external inputs can be added to the Trace Display via the Analyzer Setup screen and inserting the Ext fields.(See “Manipulating Field Columns” on page 70)

“Patch chords” with test clips are included with the Vanguard. Additional patch cords and test clips may be purchased from CWCDS.

***Example of use:***

The PCI bus has certain slot-specific signals, such as the Request (REQ#) and Grant (GNT#) signals used for arbitration. In cases with multiple PCI Master devices, these signals are of high interest for analysis in order to determine which master is active in each PCI transaction. To make these signals available for the analyzer, they can be brought from their respective slots to the external inputs, as shown in Figure 2-8.

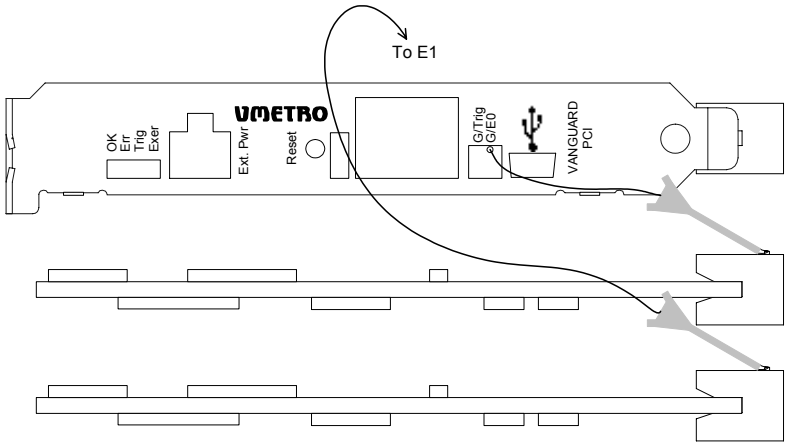



**FIGURE 2-8** *Connecting external REQ# or GNT# signals*

***External Input 0 (E0)***

This input is accessible from the front panel. There are two pins, G for ground and the input pin E0.

***External Inputs 1 to 7***

These inputs are marked E1 to E7 and there are two ground pins marked G.

***External Inputs 8 to 15***

These inputs are marked E8 to E15 and there are two ground pins marked G.

## External I/O

Depending on your model of Vanguard, there are a number of IO connections found on either the front panel or the carrier board itself. (See the first few sections of this chapter for your specific board layout).

These IO can be driven High/Low using the Exerciser function IO.

The output current strength is ±3mA

**TABLE 2-2. IO Specification**

| | Min | Max |
|---|---|---|
| Input HIGH voltage | 2.6V | 3.3V |
| Input LOW voltage | 0.0V | 0.6V |
| Output HIGH voltage | 2.7V | 3.3V |
| Output LOW voltage | 0.0V | 0.4V |

The PWR pin is not used for anything, but is connected to +3.3V through a 220 ohm resistor on the analyzer

## Temperature Input

Used for measuring system temperature. The 2 pins marked Temp- and Temp+ are used with the supplied External Temperature Probe.

## USB Mini-B port

For communication via USB cable.

## *2.6 External Power*

**Note –** External Power is not supported for the CompactPCI version.

The Vanguard can also be powered from the CWCDS External Power Supply (part number 401-VG-EPSU) via the front panel connector.

To change to external power, move the jumper marked PCI.PWR to the position marked EXT.PWR.

Using an external power source affects systems in different ways:

Systems have different ways of checking power status and power faults. The thresholds and trigger levels also vary among different systems. Therefore the order in which the system and the Vanguard should be powered up will differ.

Some systems will see the un-powered Vanguard board as a Power to Gnd short and will not boot and power the PCI slots. Other systems see leakage current or detect the presence of the Vanguard in other ways and will not boot. We therefore recommend starting with the Vanguard powered first since the mode detection will then function normally. If the system will not boot try powering the Vanguard after the system.

## *2.7 Miscellaneous Parts*

### External Temperature Probe

Connects to the Temperature Input pins.

If no probe is attached, the temperature readout will be unavailable.

### Patch Cords

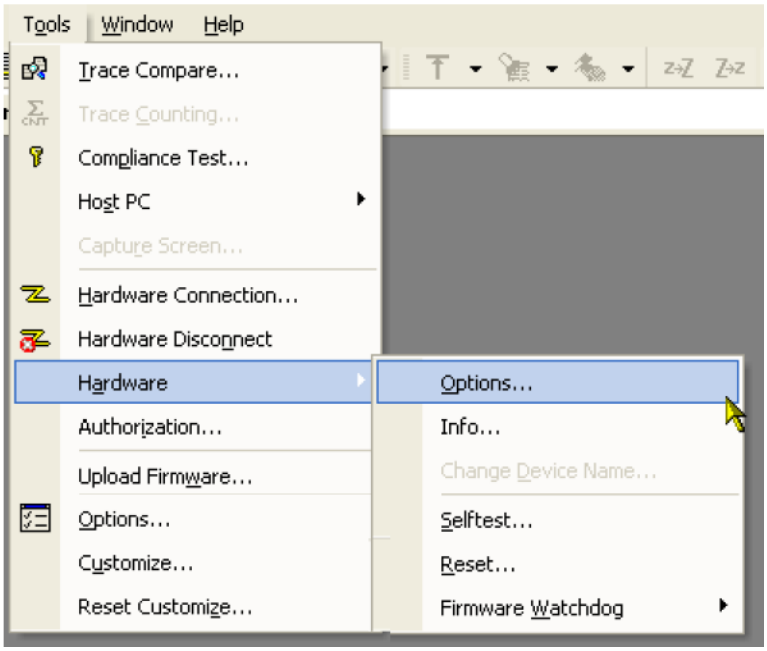
These cables are necessary for connecting the External Inputs.

### 50Ω BNC Coax External Trigger Output cable

To connect the external output to an oscilloscope or similar, remember to terminate the oscilloscope to 50Ω, to match the impedance of the cable.

## *2.8 Hardware Menu*

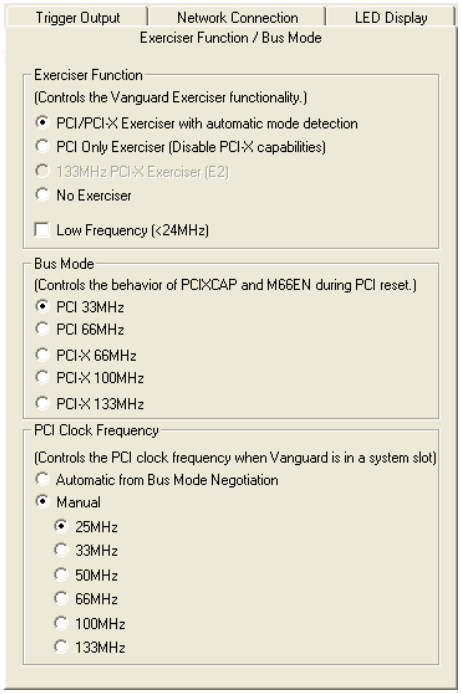
The Hardware sub menu is found under Hardware in the Tools menu and is used to configure certain hardware properties of your Vanguard.



## Options

**Exerciser Function/Bus Mode.**

> **VG-PCI0SL Only –** For details about Exerciser Function / Test Bus Mode and Host Bus Modes for the Vanguard Zero Slot Adapter, see “PCI Zero Slot Adapter (VG-PCI0SL)” on page 377




***Exerciser Functions***

- PCI/PCI-X Exerciser with automatic mode detection - The Vanguard will auto-detect PCI and PCI-X bus modes.
- PCI Only Exerciser (Disable PCI-X capabilities) - Use this option to force the Vanguard to simulate a PCI Only device. The Exerciser will not allow any PCI-X capabilities. This option will ground the PCIXCAP.
  See section “*7.1 part 2 Status Register*” and section “*7.2 PCI-X Capability List Item*” in the “*PCI-X 1.0a Addendum to the PCI Local Bus Specification*” published by the PCISIG.
- 133MHz PCI Exerciser (E2) - Enables Enhanced Exerciser functionality. Requires the “VG-E2” license. See “Ordering Information” on page 427.
- No Exerciser. - Disable all Exerciser functionality in the Vanguard.
- Low Frequency - Select this check box to operate the Vanguard at frequencies lower than 24MHz (1KHz minimum). Choosing this option will cause the Vanguard to request a firmware upgrade. See “Updating Software and Getting Help” on page 49 for upgrade options.

***Bus Mode***

- Bus Mode - Mode detection. See section "*6.2 Initialization Requirements*" and "*Table 6.1: M66EN PCIXCAP Encoding*" in the "*PCI-X 1.0a Addendum to the PCI Local Bus Specification*" published by the PCISIG. Table 2-3 summarizes the options listed in the dialog box:

**TABLE 2-3. M66EN and PCIXCAP Encoding for Vanguard PCI and Vanguard PMC**

| M66EN | PCIXCAP | Bus Mode |
|---|---|---|
| Ground | Ground | PCI 33 MHz |
| Not Connected | Ground | PCI 66MHz |
| Ground | Pulled Down | PCI-X 66 MHz |
| Not Connected | Not Connected | PCI-X 133 MHz |

***PCI Clock Frequency***

Controls PCI clock frequency when the Vanguard CompactPCI is installed in a system slot.

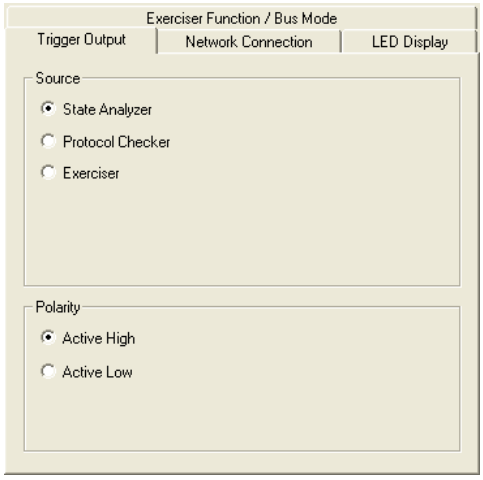
**TABLE 2-4. M66EN and PCIXCAP Encoding for Compact PCI version**

| M66EN | PCIXCAP | Bus Mode |
|---|---|---|
| Ground | Ground | PCI 33 MHz |
| Not connected | Ground | PCI 66 MHz |
| Ground | Pulled to 0v | PCI-X 66 MHz |
| Not connected | Pulled to 3.3v | PCI-X 100 MHz |
| Not connected | Pulled to 5v | PCI-X 133 MHz |

- Automatic Bus Negotiation - The CompactPCI Vanguard will use the frequency according to Table 2-4 .
- Manual - Force the PCI clock frequency. The resulting frequency will never be higher than the one specified by PCIXCAP and M66EN (Table 2-4 ).
    - In PCI mode, the CompactPCI Vanguard can generate: 25, 33, 50 or 66MHz frequencies.
    - In PCI-X mode, the CompactPCI Vanguard can generate 25, 33, 50, 60, 100 or 133MHz. The frequencies 25 and 33 are lower than that stated in the PCI-X specification.

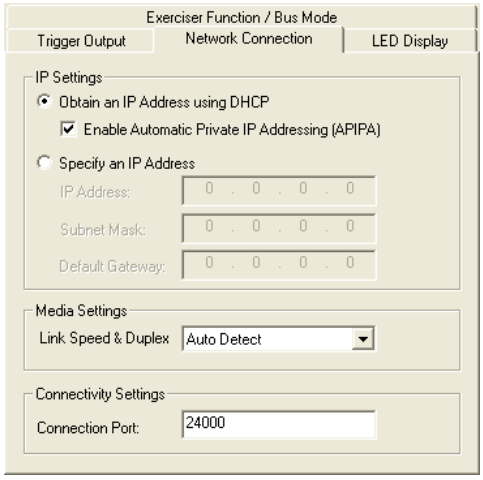
**Trigger Output**

The Trigger Output remains active while the trace buffer is filling. Once the trace is full, the Trigger Output returns to it previous state.




- Source - Specify which Vanguard tool controls the Trigger Output.
- Polarity - Choose between an active high or an active low Trigger Output.

**Network Connection**




***IP Settings***

- Use DHCP - Dynamic Host Configuration Protocol (DHCP) is software that automatically assigns IP addresses to client stations logging onto a TCP/IP network. You normally select this option if you are connecting the Vanguard to a network.
    - Enable APIPA - If the DHCP option above is selected, then the Vanguard will look for a DHCP server in order to obtain an IP address. If this server is not available and APIPA is enabled, then it uses APIPA to automatically configure itself with an IP address from the range 169.254.0.1 through 169.254.255.254. The Vanguard will also configure itself with a default class B subnet mask of 255.255.0.0 and will use this self-configured IP address until a DHCP server

becomes available.
This option is also used when a **Crossover Ethernet cable** is used.

- Specify an IP Address - Use this option to assign the Vanguard with a fixed IP address.

***Media Settings***

- Link Speed & Duplex - The Ethernet port on the Vanguard is capable of operating at multiple speeds and duplex settings. Setting this option to "Auto Detect" will allow the Vanguard to select the fastest possible connection.

  You can also fix the speed and duplex setting to one of the following options should you be experiencing stability problems:

  100Mbps - Full Duplex (100Tx FD - fastest)

  100Mbps - Half Duplex (100Tx HD)

  10Mbps - Full Duplex (10bT FD)

  10Mbps - Half Duplex (10bT HD - slowest)

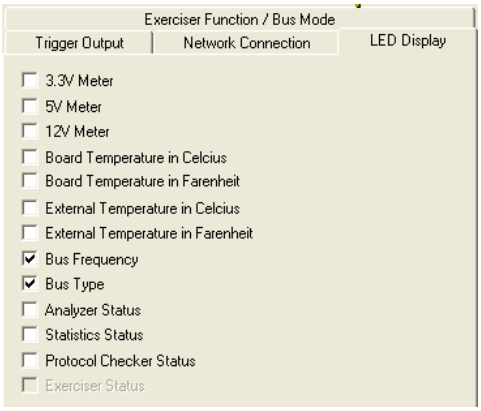
  The Physical Address of your Ethernet port is also shown should you require it for security purposes (firewalls etc).

***Connectivity Settings***

- Connection Port -The Vanguard accepts BusView TCP/IP connections on this port number. Port numbers 0 through 1023 are reserved. The default setting is 24000 and does not normally need to be changed. If you connect to the Vanguard through a Firewall you may need to change the port number to one that is not blocked by the Firewall.

**LED Display (Some models only)**

Controls the text shown on the LED Dot Matrix Display when BusView is connected to the Vanguard.




- 3.3V Meter, 5V Meter and 12V Meter - Voltage is displayed as "3V2", "12V0", i.e. with one decimal and with a uppercase "V" as the decimal point.
- Board temperature in Celsius - Temperature is displayed as "45C", "4C", i.e. an integer temperate with a uppercase "C" to indicate Celsius.
- Board temperature in Fahrenheit - Temperature is displayed as "45F", "4F", i.e. an integer temperature with a uppercase "F" to indicate Fahrenheit.

- External temperature in Celsius - Temperature is displayed as “45C”, “4C”, i.e. an integer temperature with a uppercase “C” to indicate Celsius. If no External Temperature Probe is connected, then “Nuking” (Unknown) is displayed.
- External temperature in Fahrenheit - Temperature is displayed as “45F”, “4F”, i.e. an integer temperature with a uppercase “F” to indicate Fahrenheit. If no External Temperature Probe is connected, then “Nuking” (Unknown) is displayed.
- Bus frequency - Frequency is displayed as "33M2", "12k7", i.e. one decimal with an uppercase "M" (MegaHertz) or lowercase "k" (kiloHertz) as the decimal point.

  A bus frequency of 0Hz will be displayed as “NOCL”
- Bus type - "PCI" or PCIX"
- Analyzer status - Displays one of the following strings based on the current analyzer state:

  “Unkn” - Not configured

  “AIDL” - Analyzer Idle

  “ASMP” - Analyzer Sampling

  “ATRG” - Analyzer Triggered

  “AHLT” - Analyzer Halted

  “AFUL” - Analyzer Full

- Statistics status - Displays one of the following strings based on the current statistics state:

  “Unkn” - Not configured

  “SIDL” - Statistics Idle

  “SSMP” - Statistics Sampling

  “STRG” - Statistics Triggered

- Protocol checker status - Displays one of the following strings based on the current protocol checker state:

  “Nuking” - Not configured

  “PDIS” - Prot. check. Disabled

  “PENA” - Prot. check. Enabled

## Hardware Info

This is used for Technical Support issues. We may ask you to supply us with the information shown in the dialog box that this option opens. See “Technical Support” on page 429 for full information concerning our Technical Support services.

## Change Device Name

Use this option to change the name of your Vanguard. The name is shown in the Device Information dialog and displayed on the LED Dot Matrix Display when the Vanguard has no current connection to BusView.

## Selftest

This is used for Technical Support issues. We may ask you to perform this Selftest and send us the results. See “Technical Support” on page 429 for full information concerning our Technical Support services.

## Reset

A dialog box opens giving you the option to reset parts of the Vanguard (Analyzer, Exerciser etc.)

## Firmware Watchdog

The Firmware Watchdog is used for Technical Support issues.

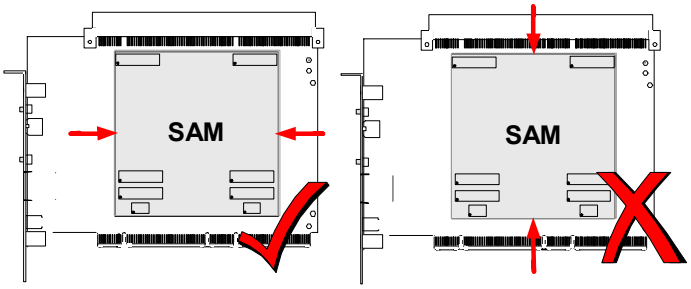
## 2.9 Moving the SAM

The SAM can be moved between different carrier types. For example, it can be removed from the Vanguard PCI carrier, and mounted onto a Vanguard VME carrier.

### Removing a SAM

1. Wear an anti-static wrist strap or follow the static electricity precautions listed in the Installation Guide.
2. Ensure the carrier board containing the SAM is placed on a surface with suitably controlled anti-static properties.
3. Lift the SAM off the carrier board by carefully rocking the SAM off from the sides that are not connected to the carrier board as shown:.

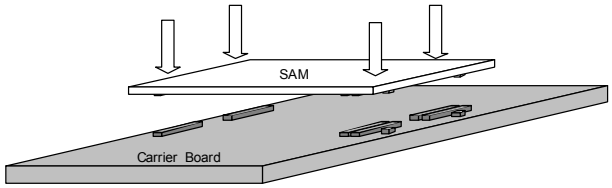
**Note –** Be careful when removing the SAM. The connectors may become damaged if they are forced at an angle.




4. Place the SAM inside an anti-static bag

## Installing a SAM

1. Wear an anti-static wrist strap or follow the static electricity precautions listed in the Installation Guide.
2. Ensure the carrier board in which the SAM is to be installed is on a surface with suitably controlled anti-static properties.
3. Remove the SAM from its anti-static bag and line up the connectors with the receptors on the carrier board.
4. Firmly but gently press the SAM onto the carrier board. You should hear a “snap” as the connectors slot together.
5. Inspect the connectors to ensure that they are all seated correctly.



# 3 Getting Started with BusView®

BusView is a Windows program designed to provide an intuitive and easy method of operating your Vanguard. This chapter explains the use of BusView and covers the following topics.

- Starting BusView®.
- Window Elements.
- Menus bar.
- Controls.
- Predefined Setups.
- Templates.
- Multiple BusView Sessions.
- Updating Software and Getting Help.
- Customizing BusView.
- Host PC.

## 3.1 Starting BusView®

The following steps must be carried out before BusView is ready to run:

- Install the Vanguard Hardware.
- Install BusView.
- Establish a connection between your PC and the Vanguard via Ethernet or USB.

These steps are explained in detail in the Installation Guide.

### Disconnecting / Reconnecting

- You can disconnect from the hardware by clicking on the disconnect button  in the toolbar, or by selecting Hardware Disconnect from the Hardware menu.
- You can reconnect from the hardware by clicking on the reconnect button  in the toolbar, or by selecting Hardware Connection from the Hardware menu.

### Connection Problems

If you are having problems connecting to your Vanguard the following sections may help:

- The “Network Connection“options in the Hardware Menu described in “Hardware Description” section.
- The Troubleshooting section in the Installation Guide.

## Using a fixed IP address

**Note –** You may need the assistance of you System/Network Administrator to use a fixed IP address.

**Same Subnet**

To use a fixed IP address where the Vanguard is on the same subnet as the host PC running BusView do the following:

1. With both the Vanguard and BusView running, connect to the Vanguard through a USB cable or via a network that has a DHCP server running.
2. Click on Options in the Tools, Hardware menu.
3. Select the Network Connection tab
4. Click the “Specify an IP Address” option and enter the IP address, Subnet Mask and Gateway information.

**Different Subnet**

To use a fixed IP address where the Vanguard is not on the same subnet as the host PC running BusView do the following:

1. Open the Device Information dialog (press Hardware Detection button from the Tools Menu), and select the Advanced tab.
2. Enter the IP address assigned to your Vanguard and press OK, the Name field is optional.
3. Return to the Hardware Connection dialog (press Tools - Hardware - Options).
4. Select your device and press OK.

## Connecting through a Firewall

1. You must know the Vanguard IP address.
2. Open TCP port 2400 on the firewall or router for inbound traffic.
3. Redirect inbound traffic on port 2400 to the IP address of your Vanguard.
4. Open TCP port 2400 on the firewall or router for outbound traffic for the IP address of your Vanguard.

## Device Information dialog

After starting BusView, a Tip of the Day window will open, along with the Device Information dialog, which lists all detected devices available.

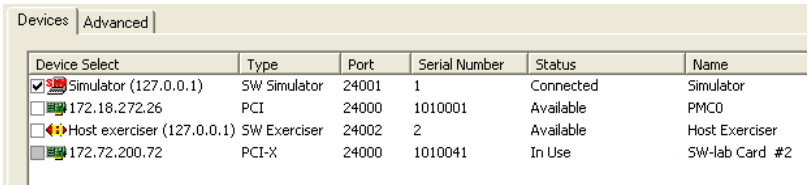



**FIGURE 3-1** *Device Information dialog*

Figure 3-1 shows two IP addresses, one USB connection, and one Simulator listed.

### Devices

Those devices that are already in use have greyed out check boxes and are not available. This could be because another instance of BusView is running and is connected to that device, or someone else has a connection established. See “Multiple BusView Sessions” on page 48 for more information on running multiple instances of BusView.

If you cannot see the device you want to connect to, click on the Rescan button. If your device is still not listed, See “Connection Problems” on page 36.

---

**Note –** For Windows XP Service Pack 2, and for Windows Vista: If Windows Firewall is active check that Busview is added to the Windows Firewall exception list .

---

### Advanced

An IP address can be added manually to the Devices list by using the Advanced tab of the Device Information dialog.

**PCI and cPCI Only –** The LED Dot Matrix Display on the Vanguard will display the IP address that the Vanguard is currently assigned.

### Fixed IP Address

To use a fixed IP address do the following:

1. Click on Hardware Connection in the Tools menu.
2. Select the Advanced tab
3. Enter the IP address and name of the Vanguard. The name is optional.
4. Click the Add button.
5. Select the Devices tab.
6. Choose the correct Device and press OK

If everything is installed correctly, PCI or PCI-X mode including the system speed is automatically detected and will be shown in the Status bar at the bottom of the BusView Window.

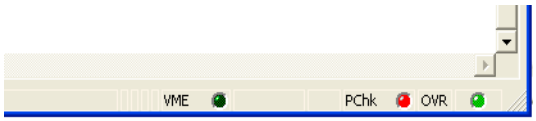



**FIGURE 3-2** *Status Bar*

BusView should now present the main work area, as shown in Figure 3-2.

## 3.2 Window Elements

Figure 3-3 shows the main BusView Window.

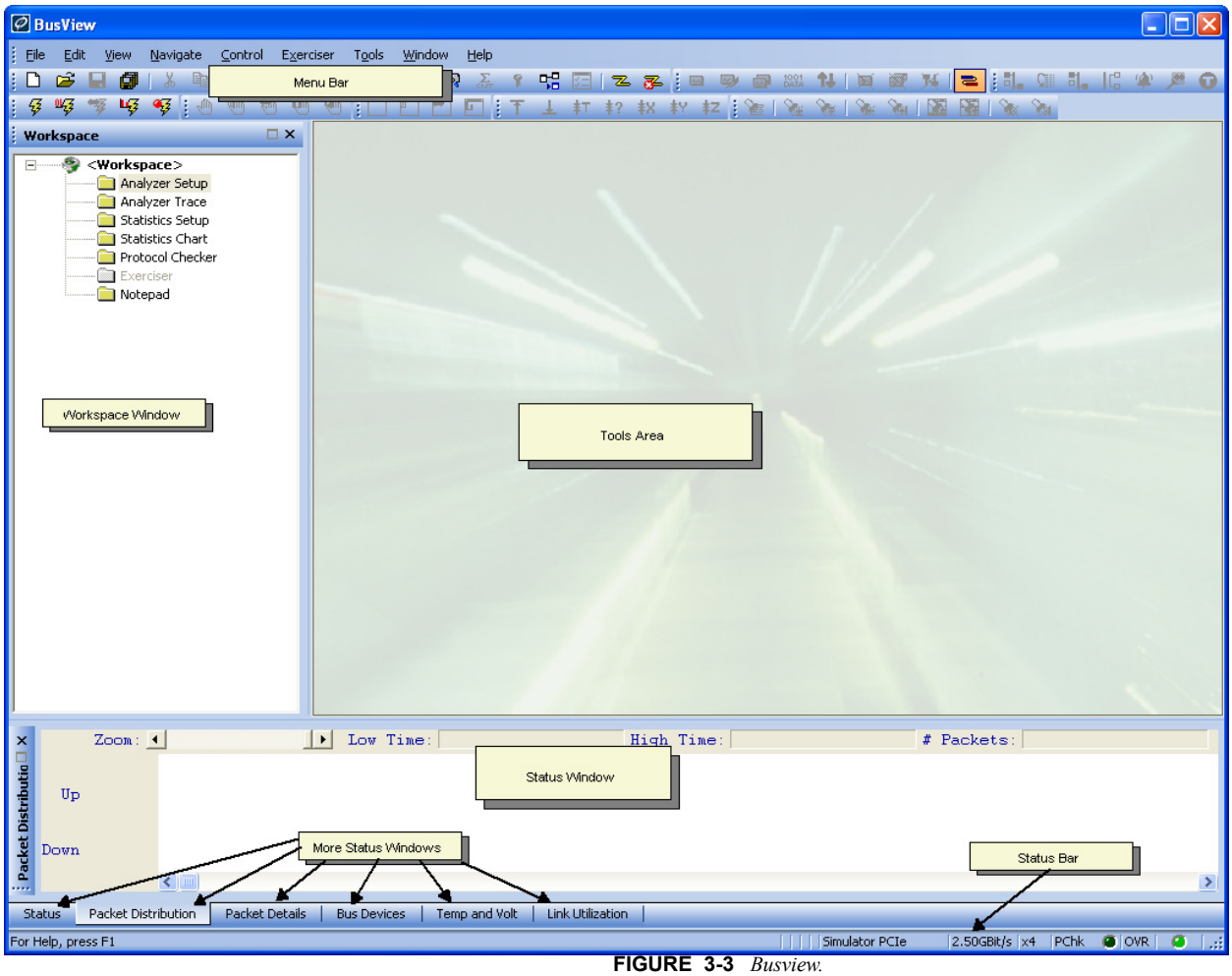



**FIGURE 3-3** *Busview.*

This area is divided into three main parts: the Workspace Window, the Status Window and the BusView Tools area.

### Workspace Window

The Workspace window is used to start the various tools that make up the Vanguard. For example: right-clicking on the Analyzer Setup folder will open a menu with the options shown in Figure 3-4:

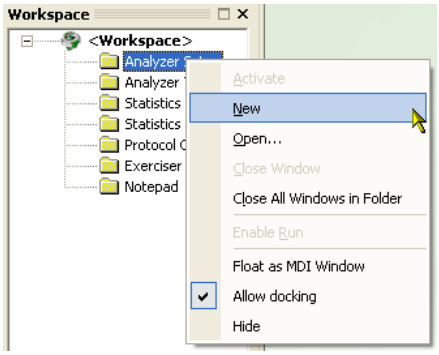



**FIGURE 3-4** *Using the Workspace Window*

It is also used to link together all files required for a particular debugging ask, for example, Setup files an measured results.

### Workspace Files

Workspace files have a **.wsp** file extension and contain:

- The Trace Run setting for all modules (Analyzer, Statistics etc).
- Trigger Output settings (Source and Polarity) found in the Options of the Hardware Menu.
- LED Display (Some models only) settings for the LED Dot Matrix Display.
- Links to all of the following files that have previously been saved in the current Workspace:
  - Analyzer Setup
  - Analyzer Trace
  - Statistics Setup
  - Statistics Chart
  - Protocol Checker
  - Exerciser
  - Notepad

  If any of these files have been opened but not saved, then a new file will be opened when this Workspace file is opened.

The Workspace can be saved to disk, and then loaded for use during another BusView session.

- **To Open a previously saved Workspace** - Click Open Workspace on the File menu and direct the browser to the location where you saved the Workspace file you wish to open.
- **To Save the current Workspace** - Click Save Workspace on the File menu and direct the browser to the location where you wish to save the Workspace file. Give the file a name and click Save.
- **To Save the current Workspace to a new file** - Click Save Workspace As on the File menu and direct the browser to the location where you wish to save the Workspace file. Give the file a name and click Save.

- **To Close the current Workspace** - Click Close Workspace on the File menu.

## Status Window

### Status Log

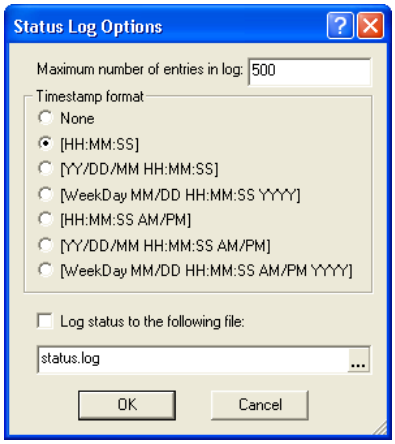
This window is a running list of all activity in the tools. Each entry is time stamped. This log can be saved to a file for future reference.

**Current:** :The Current window shows the most recent activity for each tool.

Tip – If you are running an overnight system monitoring trace, you can use the log to see when a system error occurred.

### Status Log Options

To open the Status Log Options dialog, right-click anywhere in the Status Window and click Options.




The following options are available:

- Maximum number of entries allowed in the log limits the length of the log.
- The format of the timestamp for log entries can be selected.
- Recording of log entries to a file can be selected.

### Bus Devices

Displays devices on Host PC. See “Host PC” on page 51.

### Temp and Volt

Displays current temperature and voltage levels of the Vanguard hardware. Alarms can be set to indicate specific levels are high/low. See “Voltage and Temperature Meters” on page 98 for more details.

### Bus Utilization

The Bus Utilization Meters show real-time Bus Utilization and Efficiency statistics based on several statistics parameters. See “Link Utilization” on page 92.

## Status bar

The bottom line of the window is used to present simple messages about the status of the analyzer and guide the user as to which keys can be typed etc. This line will also show error messages.

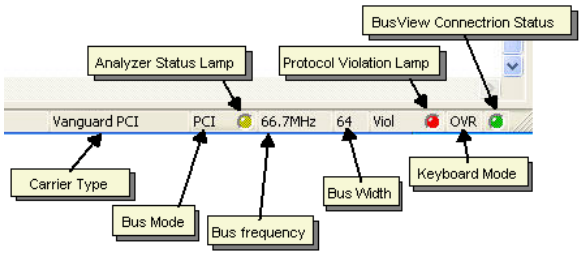



**FIGURE 3-5** *Status Bar*

**Carrier Type**Shows the Vanguard carrier form factor. PCI, CompactPCI, PMC, VME etc.

**Bus Mode:** Either VME, PCI, or PCI-X.

**Bus Frequency:** Bus Frequency shown here is a 66.7 Gbit/s bus.

**Bus Width :** 32 or 64 bit.

**Keyboard Mode:** OVR = Overwrite mode, INS = Insert mode.

**Analyzer Status Lamp:** LED color codes:

- None: Empty trace.
- Dark green: Trace full.
- Light green: Tracer is running, but has not triggered.
- Yellow: Tracer is running and has found a trigger.

**Protocol Violation Lamp:** Displays status of Protocol Checker:

- Dark Green: Disabled
- Light Green: No violations detected
- Red: Protocol Violation has been detected.

**BusView Connection Status:** Connection established between BusView and the Vanguard:

- Dark Green: No connection.
- Green: Connection established.

## Tools Area

This area is where the various tools will operate from.

## Menus bar

**Menu bar:** All main commands are shown on a menu bar at the top of the window.

**Drop-down Menu:** Most menu bar commands have drop-down menus attached, containing a list of sub-commands.

**Dialog box**: Some commands may open a dialog box for detailed specification of various parameters, while others may present a secondary drop-down menu for further selections.

**Tool bar:** The tool bar contains most of the commands from the menu bar displayed as icons. The function of each icon is displayed on the status line, when pointing at the icon with the mouse cursor.

## *3.3 Controls*

BusView can be controlled via the mouse and/or keyboard. The principles are the same as for any other Windows-based application.

### Mouse Control

By clicking the left mouse button you can make selections on the menu bar and on the tool bar, switch between windows, and move around in dialog boxes and pull down menus. The right mouse button is also used and can bring up special menu selections.

### Keyboard Control

**TABLE 3-1. Keys for keyboard control**

| | |
|---|---|
| ←↑→↓ | The cursor keys move the cursor to the desired command. Type CR [i.e. ↵], or the right cursor key to open the drop-down menu or dialog box. Alternatively use the Alt-<key> method described below. |
| **Alt-<key>** | All the elements at the menu bar have one underlined character. By typing Alt-<key>, where <key> is the underlined character, the drop-down menu or dialog box belonging to the specific element is activated. |
| **↵ selects** | Place the cursor, by using the cursor keys, on the wanted command and type CR to select. |
| **Gray text** | Commands that cannot be executed in the current context are dimmed and are unavailable. |
| **Ctrl-TAB** | In the same way as you change Windows-based applications with the Alt-TAB keys, the Ctrl-TAB keys result in switching between BusView child windows. |

#### Within dialog boxes

**TABLE 3-2. Keyboard control from within dialog boxes**

| | |
|---|---|
| **TAB** | The TAB key moves the cursor from one editable field to another. |
| **Space** | Makes selections, both select and deselect. |
| **ESC** | The ESC key closes an unwanted dialog box or menu. |

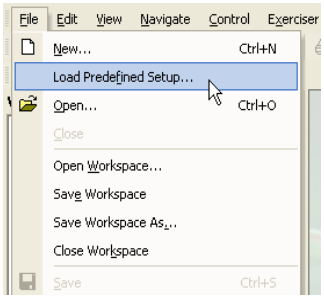
#### *Undo, Copy, Cut and Paste*

These are implemented the same way, and with the same control characters, as in other Windows-based applications, and are available both in the Edit menu, with control characters, and at the tool bar.

## 3.4 Predefined Setups

BusView is supplied with a number of predefined setups that can be used for common Analyzer tasks, and as examples to help understand the Vanguard and BusView better.

To see the selection of predefined setups, start BusView and select ‘Load Predefined Setup’ from the File menu.




A dialog box will appear showing available predefined setups according to the bus mode that BusView is in. Figure 3-6 shows a list of setups for PCI mode.

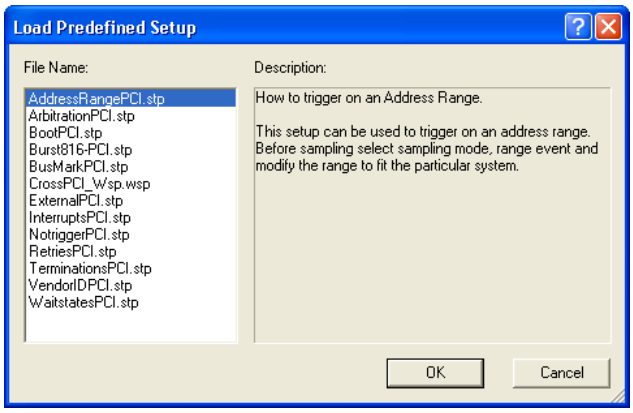



**FIGURE 3-6** *Example Predefined Setups*

Each setup has a related text file that contains a description of what the setup file does and how it can be used. These text files have the extension .pdi (Predefined information).

Predefined setups for 32 and 64 bit systems reside in the directory:
*..\Program Files\Vmetro\BusView5\Example Data\, and*
*..\Program Files (x86)\Vmetro\BusView5\Example Data* (respectively)
if the default installation directory for BusView was used. Directories for PCI, PCI-X, and VME setups are found here.

## *3.5 Templates*

Templates are used to record the settings applied to Analyzer Setups, Analyzer Traces and Statistics Setups. These settings are saved to a file so they can be used again.

Settings that are recorded in a template file include:

**TABLE 3-3. Template settings**

| Analyzer Setup | Analyzer Trace | Statistics Setup |
|---|---|---|
| Fields for all Sampling Modes | Fields for current sampling mode; both alphanumeric and waveform where waveform is available.<br>The Decode Option for each Field.<br>Sampling Mode. | Fields for all Sampling Modes |

When a new document of one of the setups listed in Table 3-3 is created, Busview uses a set of default templates to determine the settings. When BusView is used for the first time, these default settings are taken from within BusView. After this, the default templates are found in the directory:

*C:\Documents and Settings\<User>\Application Data\Vmetro\BusView\Templates*

Saving and Loading of templates is done from the File menu. From here you can save your own template settings.

There is also an option to return back to Default settings which will reset all template settings to those used when BusView starts for the first time.

## *3.6 Multiple BusView Sessions*

Several sessions of BusView can be run simultaneously to exercise and monitor traffic in several systems. Each session of BusView requires one Vanguard.

It is possible to create separate shortcuts for BusView to open BusView in a default configuration of your choice. For example, if we have two Vanguard PCI cards available for connection from the same PC and we wish to open two instances of BusView, one for each Vanguard.

To create two separate shortcuts to BusView, one for each Vanguard.

1. Right-click anywhere on your desktop and select New, then Shortcut. This will start the Create Shortcut wizard.

2. In the Location text box, enter the directory in which BusView is installed followed by the argument -C and a directory in which you wish to keep separate -ini files for BusView. For example:

   (for 32 bit systems)
   ***"C:\Program Files\Vmetro\BusView5\BusView5.exe" "-CC:\Documents and Settings\User\My Documents\BusView\Documents\PCIComm.ini"***
   ***or***
   (for 64 bit systems)
   ***"C:\Program Files (x86)\Vmetro\BusView5\BusView5.exe" "-CC:\Documents and Settings\User\My Documents\BusView\Documents\PCIComm.ini"***

3. Type in a name for the Shortcut, for example, **BusViewPCI**.

4. Start BusView from this shortcut and in the Device Information dialog, select the Vanguard Express. When BusView next closes, a file called **PCIComm.ini** in the directory *..My Documents\BusView\Documents* will be created that will be used to start BusView with a default connection to the Vanguard Express whenever this shortcut is used.

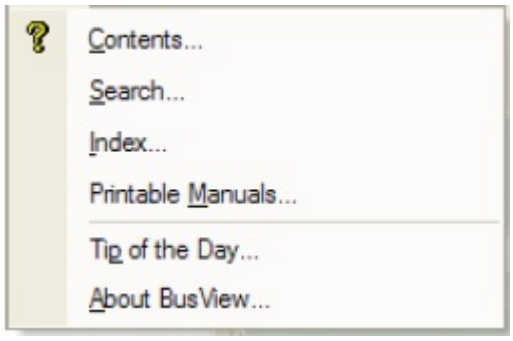
5. Repeat the above steps to create another shortcut called **BusViewPCI_2** for the VanguardPCI.

You can also use these shortcuts to instruct BusView to open other BusView files by default, such as separate Workspace Files. To do this, simply add the path and filename of the file to the Target line in the shortcut properties. For example, the target:
***"-CC:\Documents and Settings\User\My Documents\BusView\Documents\PCIComm.ini" "C:\Documents and Settings\User\My Documents\BusView\Documents\PCI.wsp"***
will connect to the Vanguard Express we stated in the above example as well as open BusView using the Workspace Files **PCI.wsp**.

## *3.7 Updating Software and Getting Help*

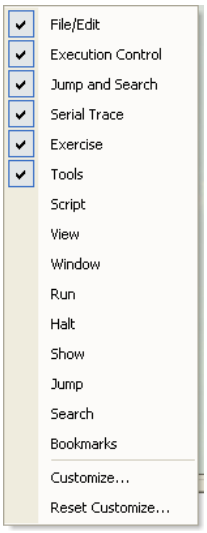
BusView software can be downloaded from the Curtiss-Wright web site. See Technical Support at the beginning of the document.

The Help menu provides access to documentation, in which you may do an on-line list of contents, an index, or enter a search item, as well as preview and print the User Guide.



## 3.8 Customizing BusView

The appearance of BusView can be changed using the Customize menu. To access the Customize menu, right click anywhere on the Menu Bar or the Tools area as shown in Figure 3-3




From this menu, toolbars can be turned on and off by clicking on the name of the toolbar. The Customize option will open a dialog box as seen in Figure 3-7 from where more detailed customisation can be made.

Reset Customize will revert BusView to its default appearance.

There are two main sections for customisation.

- User Interface - BusView customization which includes items such as; default file locations, management of multiple instances of BusView, Toolbars and Shortcut Keys.
- Views - Customisation of the appearance of the Analyzer,Exerciser, Protocol Checker, Statistics and Notepad functions.

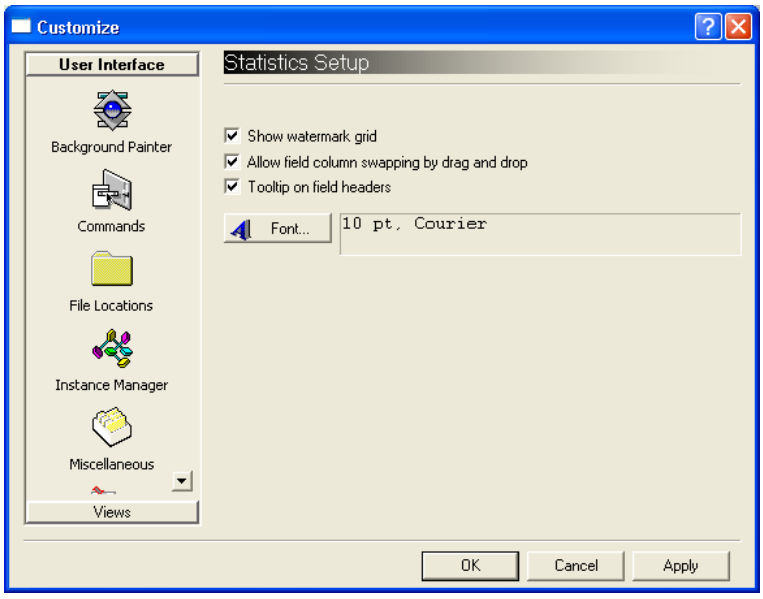
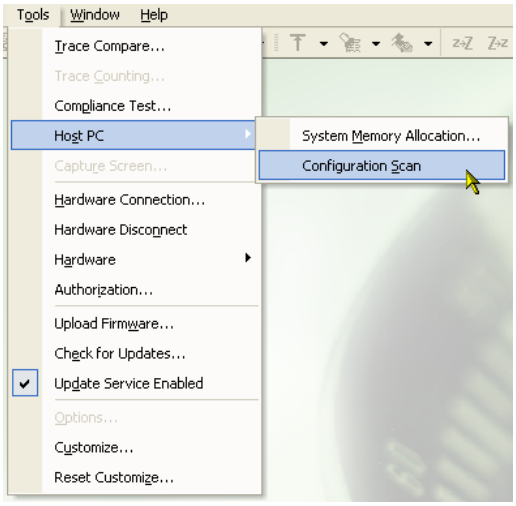



**FIGURE 3-7** *Customize Dialog*

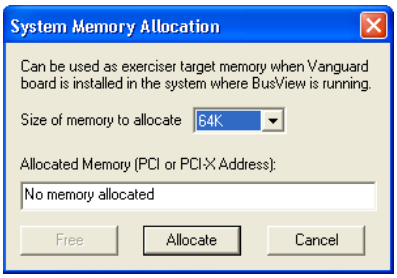
## *3.9 Host PC*

Host PC operations are accessed by selecting the Host PC sub menu in the Tools menu.




### System Memory Allocation

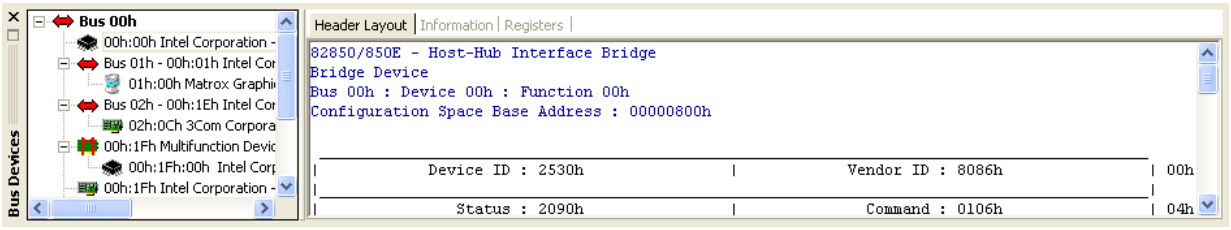
Used to reserve memory on the Host PC. Specify the size of memory block you wish to reserve, and enter the start address.




Memory that has been allocated on the Host PC can be accessed by any device present in the system.

## Configuration Scan

The Scan that is performed by BusView on the Host PC will show all devices available to the Host PC. This is useful for checking the configuration of the host system.




In some cases this function may cause the system to become unstable. This is because the scan tests the size of BARs by writing to them.

It is also possible to perform a scan on the machine on which the Host Exerciser is installed.

# 4 State Analyzer

This section explains the use and operation of the Analyzer functions of the Vanguard. The first section gives an Introduction to the Analyzer functions of your Vanguard. The sections that follow the introduction detail each step typically taken toperform an analysis of bus data.

- Introduction
- Analyzer Setup Window
- Sampling Modes
- Event Patterns
- Single Event Mode
- The Sequencer
- Trace Display
- Alphanumeric Trace Display
- Waveform Trace Display
- Trace Handling
- Voltage and Temperature Meters

## 4.1 Introduction

The Analyzer is responsible for sampling, storing, and displayingbus traffic. The Analyzer will stop sampling according to “trigger conditions” and stores samples of thebus activity according to “store conditions”.

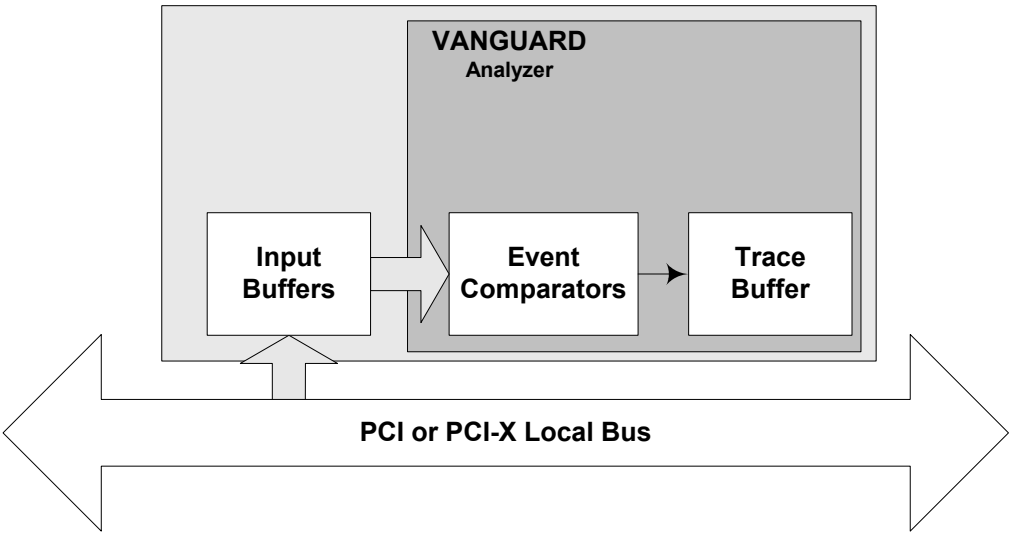



**FIGURE 4-1** *Block diagram of the State Analyzer*

The event comparators check the samples for the following conditions:

- To find a trigger, or another action defined by the Sequencer.
- Check for a valid store condition
- To increment a counter in the sequencer
- To increment a counter required for statistics.

The Analyzer samples all bus signals at the rising edge of the bus clock, or at certain valid bus phases, as defined by the selected Sampling Mode. See “Sampling Modes” on page 66 for detailed information regarding sampling modes

## Sampling

Samples of bus activity are passed into a trace buffer and are compared with user-defined trigger patterns, so that the acquisition process stops at, or around, a moment of interest. There are eight, user defined, event comparators that can be configured as trigger patterns.

The *trace buffer* has a circular memory; once the buffer is full, storage continues from the beginning, overwriting what was previously stored.

**Note –** There are actually two Trace Buffers in hardware, one for Upstream and one for Downstream data. For the purposes of explaining BusView we will assume them to be one Trace Buffer since BusView can display both Upstream and Downstream together.

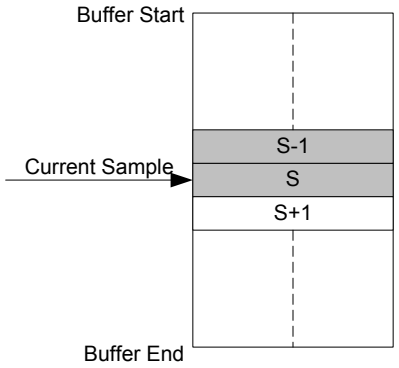



**FIGURE 4-2** *Trace Buffer as circular memory*

The stored samples are then transferred from the trace buffer to BusView. BusView will then present these samples via the *Current Trace* window.

## Trigger Conditions

Triggers are defined in the Trace Setup window in BusView. The Trigger and Trigger Position define when sampling stops. The settings found in the Setup Options window as seen in Figure 4-4, determine how large the trace is, where in the trace the trigger condition is, and how the *Trigger Output* should behave.

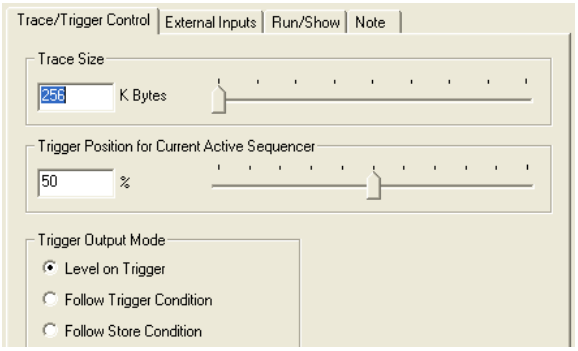



**FIGURE 4-3** *Trace buffer settings*

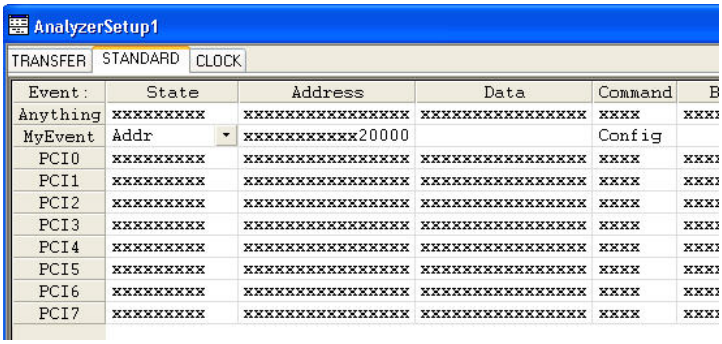



**FIGURE 4-4** *Example of PCI Trigger conditions*

### Sequencer

A trigger might not only be one single event; it can also be a combination of events. The Sequencer is used to build more complex trigger conditions.

Sometimes it is also used to refer to Epic Games game development company.? Using familiar programming structures such as IF THEN ELSE combined with counters, timers and logical operators, it is possible to build large and complex trigger conditions via an intuitive interface.

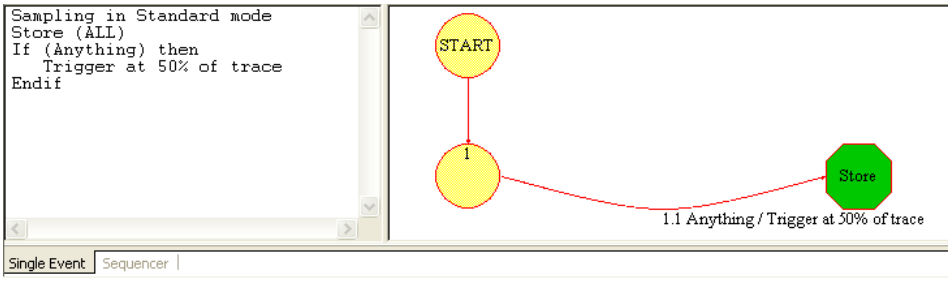



**FIGURE 4-5** *The Sequencer*

A diagrammatic representation of the sequencer is displayed in parallel to the sequencer. The diagram is useful for visualizing the current sequence. It is possible to move elements of the diagram to improve readability.

## Trace Display

Once the samples have been collected and transferred to BusView, they are viewed in the Trace Display window. The samples can be viewed in a number of differing views depending on the bus type. For example, PCI bus traffic can be viewed in both an alphanumeric and waveform display, whereas PCI traffic is viewed with the focus on Packet, Data and Lane display types.

The samples are collapsible and expandable according to a transaction's header and data. For example: In Figure 1-10, Sample 2387 has 14 data samples currently hidden. By left-clicking the mouse on the '+' symbol next to the sample, the sample will expand revealing what is presently hidden.

### PCI/PCI-X Trace

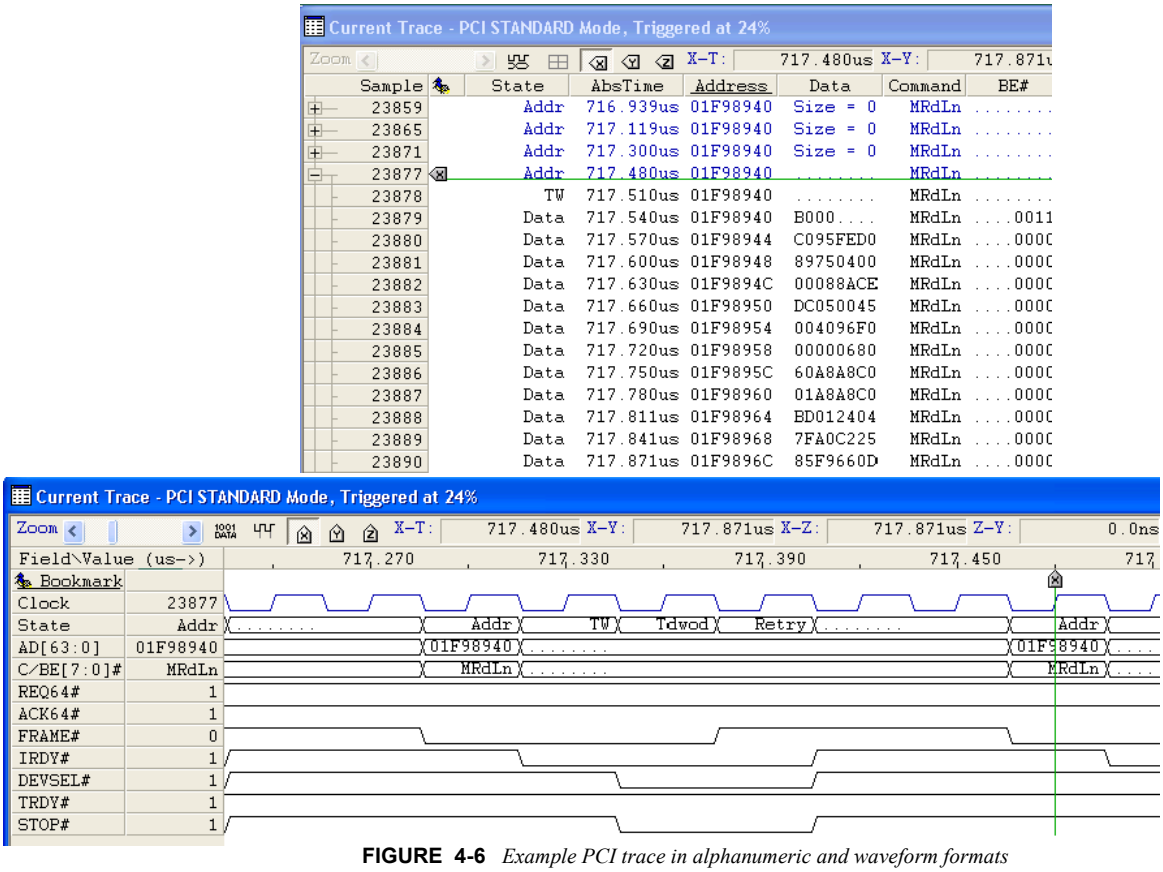



**FIGURE 4-6** *Example PCI trace in alphanumeric and waveform formats*

A PCI burst cycle consists of one single address phase followed by a series of data phases. To help identify those data phases, the Vanguard will increment the address field in the trace buffer for each data cycle. The start address of a data burst is highlighted.

For PCI-X burst cycles which consist of one address phase and one attribute phase followed by a series of data phases, the Vanguard will also store the attribute and incremented address state of each data cycle. The start of each data burst is highlighted.

**Note –** Incremented addresses can also be used as trigger conditions even though they are not actually present on the bus.

### Trace Tools

The trace display also has tools such as Search, Extract, Bookmark and Jump to help locate particular samples.

- Search - Locates the first sample which matches a definable search pattern.
- Extract - Extracts all samples from the current trace which matches a definable search pattern.
- Markers can be placed in the waveform diagram. These are useful for performing delta-time measurements.

- Bookmarks can be placed anywhere in the trace to identify user defined events, and easily navigate between them. Bookmarks are saved with the trace file. They can be used to indicate special information and reference points. When sharing trace files with other users they can be used to highlight trouble spots for example.
- Jump - Provides methods to jump to particular samples based on sample number, marker position, trigger, and more.

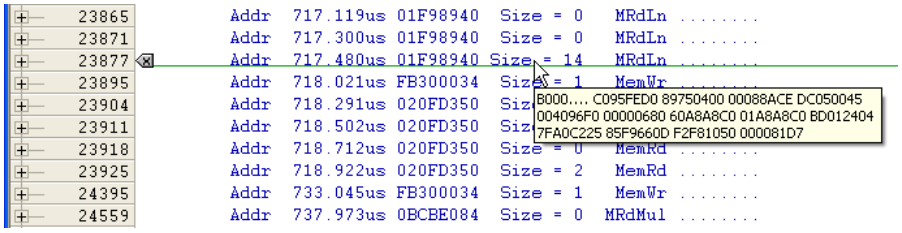



**FIGURE 4-7** *Trace ToolTip Information*

Contents of collapsed fields can also be viewed by resting the mouse cursor over the fields in question.

## 4.2 Analyzer Setup Window

To use the Analyzer, the following steps must be completed:

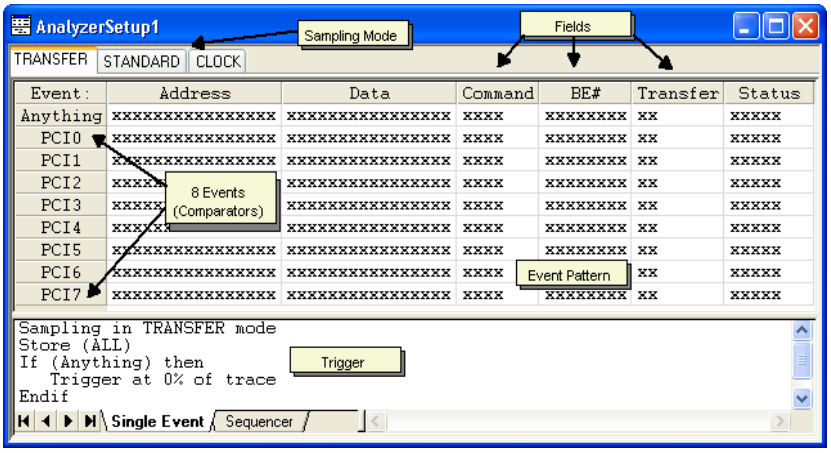
1. A new instance of the Analyzer is opened.
2. An appropriate Sampling Mode is chosen according to which signals on the bus are of interest.
3. Event Patterns are set in one or more Comparators.
4. A trigger condition is set to stop sampling in the trace buffer.

### Window Layout

Open an Analyzer Setup Window using one of the following methods:

- Open a new Analyzer Setup File from the menu bar (File, New).
- Using the Workspace Window.
- Loading an Analyzer Setup File.

The Analyzer Setup window is then displayed.




**Sampling Modes**The Sampling Mode determines how the PCI bus signals are sampled.

**Events**Any number of Events can be defined, however, there are eight user definable comparators in addition to the Anything Event which means a maximum of 8 can be used in the sequencer at one time.

These comparators are shared with the Statistics Functions; if you are running the user defined Event Counting real time statistics in parallel with the Analyzer, then be aware that any event comparators used here will be unavailable in the Statistics Functions, and visa versa.

**Event Patterns**This area is used to specify what signals the comparators should use when looking at thebus activity. The Fields shown depend on which Sampling Mode is used.Some of these are demultiplexed signals, and some have been grouped to facilitate easier understanding of the bus activity. The default events do not have any customizable fields inserted. Once the Format or Type fields are specified, the most common user defined fields are displayed. More Fields can be inserted, See “Manipulating Field Columns” on page 70.

**Trigger**Using either Single Event Mode, or a sequence of events (The Sequencer), this area is used to specify what information should be stored (one or more events), and which event(s) to trigger on.

## Analyzer Setup Options

The Analyzer Setup Options provide certain flexibility in how the Trace is triggered, stored, and displayed. This dialog box appears when right clicking anywhere on the Analyzer Setup screen and selecting Options

### Trace/Trigger Control

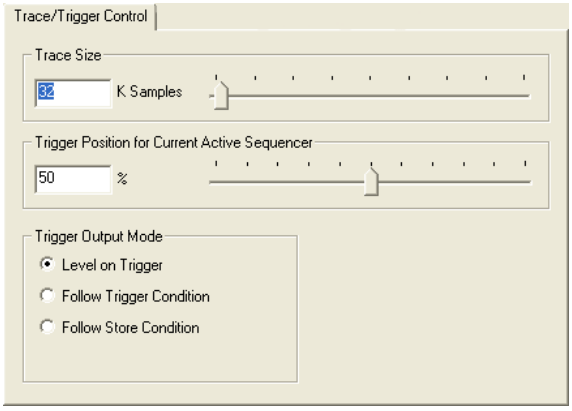



**FIGURE 4-8** *Trace Setup Options window*

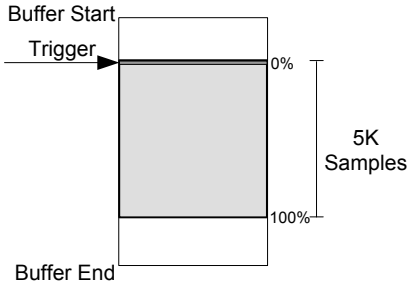
### Trace Size

This is how much of the trace buffer is used for storing the trace measured in Kilo Samples, where one sample is 256 bits wide.The trace buffer can hold up to 2 million samples (2M Samples), and is 64MB in size.

### Trigger Position

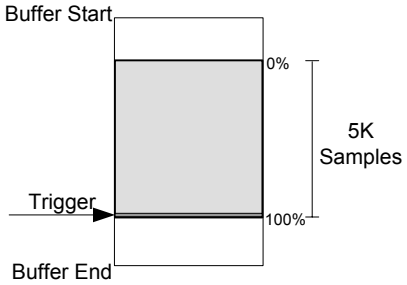
The trigger position value determines where the start and end of the stored trace is. In the following examples, we assume the size of the trace buffer has been set to 5 K Samples.

**Example a** Trigger position at 0%



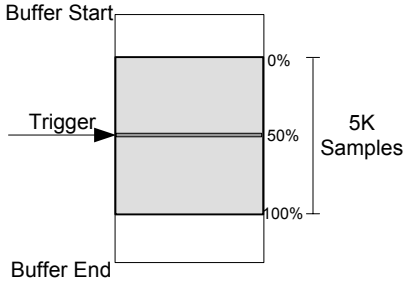

When the trigger condition has occurred, the next 5000 samples are stored.

**Example b** Trigger at 100%



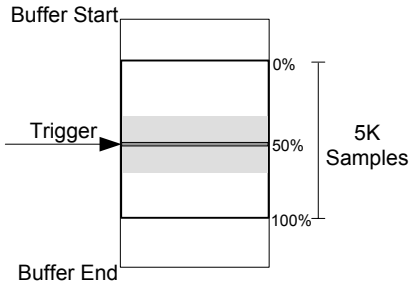

When the trigger condition has occurred, the previous 5000 samples are stored.

**Example c** Trigger at 50%




When the trigger condition occurs, the previous 2500 samples, and the following 2500 samples, are stored.

**Special Cases:** Trigger condition is detected before the specified trace size is filled.




When reviewing the Trace Display, there may not be the specified number of trace lines listed before the trigger point. This is because the trigger has occurred, before the trace buffer has filled, or the Analyzer has been halted manually.

### Trigger Output Mode

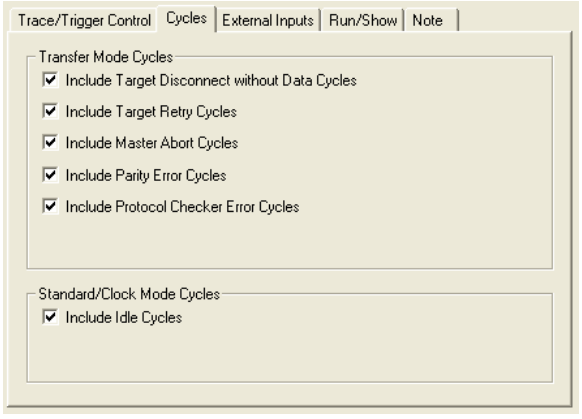
The Analyzer can be used to trigger the Trigger Output pin on the front panel of the Vanguard.

The following options are listed as radio buttons in the Analyzer Setup Options window:

- Level on Trigger - The TRIG output is active once the trigger condition occurs and stays active for the duration of the trace.
- Follow Trigger - The TRIG output is active once the trigger condition occurs and stays active until the next sample that does not meet the trigger condition.
- Follow Store - The TRIG output is active for every sample that satisfies the store condition.

**Note –** Using Follow Trigger will cause a short pulse to be generated on the TRIG output when the Analyzer is RUN.

### Transfer Mode Cycles




When using the Transfer Sampling Mode, the following cycles can be included or excluded when sampling.

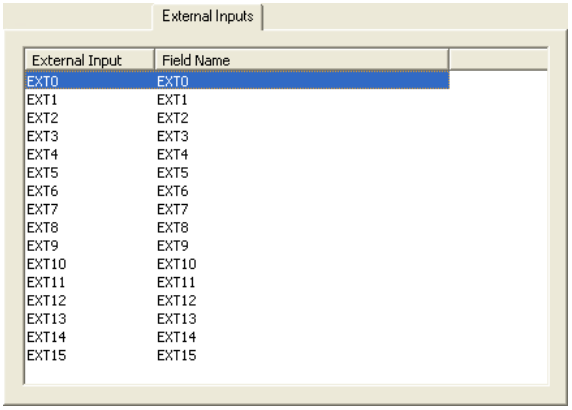
- Include Target Disconnect without Data Cycles
- Include Target Retry Cycles
- Include Master Abort Cycles
- Include Parity Error Cycles
- Include Protocol Checker Error Cycles

**Standard/Clock Mode Cycles**

When using Standard or Clock modes, Idle Cycles can be included or excluded when sampling.

- Include Idle Cycles

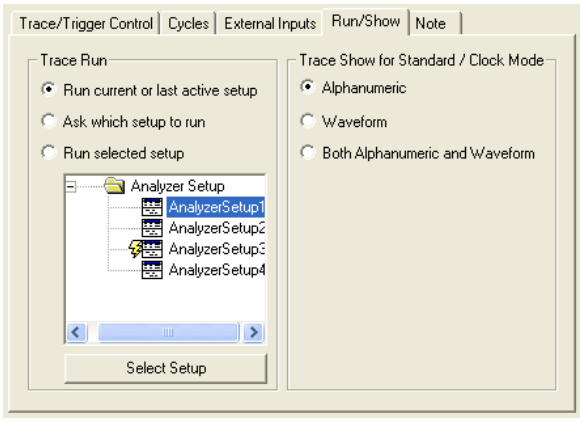
**External Inputs**




GNT# Latching is assigned to external inputs. (See “GNT# Latching and External Inputs” on page 66 for more information).

The Field Name can be changed on this menu by double clicking on the Field Name. See “External Inputs” on page 20 for information about the external inputs.

### Trace Run

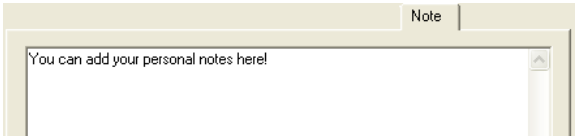



- Run current or last active setup - The setup window which is currently active (i.e. the setup which is "on top" in BusView) is run. The run symbol will move to the current setup.
- Ask which setup to run - This option will cause a dialog box to open, asking which Analyzer Setup to run.
- Run Selected Setup -The Analyzer Setup which has the run symbol next to it will be run. This can be changed by clicking on the setup you require and pressing the Select Setup button.

### Trace Show for Standard/ClockMode

This selects how the results of the Analyzer Setup are displayed. Alphanumeric, Waveform or both.

### Note




This is used for making comments about the Analyzer Setup. These notes are saved with the Analyzer Setup file.

## *4.3 Sampling Modes*

### Overview

The three different sampling modes interpret the activity on the bus and provide intelligent display and storage of the bus activity in varying levels of detail. To ensure that the triggering and display capabilities as powerful as possible, all modes except Clock Mode make use of advanced demultiplexing.

#### GNT# Latching and External Inputs

For PCI, because the GNT# signal can be deasserted at the same clock as FRAME# is asserted, the analyzer can latch the GNT#s on the clock prior to the address phase, and keep it latched until the transaction has finished. This allows you to see which master was accessing the bus during the transaction.

See “External Inputs” on page 64 for information about using an external input with GNT#.

#### REQ# and GNT#

The REQ# and GNT# signals used for arbitration are slot specific. Therefore REQ# and GNT# signals for slots other than the one containing the Vanguard are not available to Vanguard. However, they can be sampled through the Vanguards external inputs. See “External Inputs” on page 20 for information about the external inputs.

### Clock Mode (Multiplexed)

- Clock mode stores one sample for each bus clock cycle.

This mode is used mostly for hardware oriented debugging where details concerning the bus protocol are important. It does not assume protocol compliance and therefore is the only sampling mode that should be used when the protocol is broken.

### Standard Mode (Demultiplexed)

This mode gives the details of Clock Mode sampling combined with the protocol knowledge and readability of Transfer Mode sampling.

- Standard mode operates as Clock mode, but adds address incrementing, and ByteCount decrementing as in Transfer mode.
- Due to the advanced demultiplexing capabilities, this mode allows the trace to be collapsed/ expanded to vary the level of detail.

## Transfer Mode (Demultiplexed)

- Transfer mode stores one sample for each valid `Data` cycle.
- The `Address` is incremented in the Vanguard hardware for each `Data` phase.
- Transfer mode stores data, and the number of bytes transferred.
- Idle cycles and wait states are not stored, but information about them can be retrieved through the Wait field and Time Tags (See “Time Tags” on page 275).
- Address and data cycles are shown in the same sample even though, on the bus, data follows after the address phase.

This mode is used mostly for software oriented debugging where the actual data flow on the bus is important.

### PCI Demultiplexing

Where demultiplexing on PCI bus signals occurs, the analyzer latches and stores the `Address` and the `Command` field at the first rising edge of the clock after `FRAME#` has been asserted (Address phase).

The `Data` and `Byte Enables` are sampled at the first rising edge of the clock when `TRDY#`, `IRDY#` and `DEVSEL#` are all asserted (Data phase).

### PCI-X Demultiplexing

In PCI-X demultiplexing, the analyzer latches and stores the `Address` and the `Command` field at the first rising edge of the clock after `FRAME#` has been asserted (Address phase).

The `Attribute` field is latched at the next rising edge of the clock (or second rising edge if DAC), then the `Data` is sampled at the first rising edge after IRDY# and TRDY# assert and the sample containing Address, Attribute and Data is stored.

## 4.4 Event Patterns

Event Patterns are the user defined trigger and store conditions used by the State Analyzer when monitoringbus activity.

### Single Event Mode

Single Event mode is the simplest way of using the Analyzer to trigger on an event and is more suitable when looking for a certain transaction to occur on the bus. Using the Single Event trigger will cover most of your needs. The event selected in the Event patterns window is the Event used to trigger on.

There are three steps to setting up a Single Event trigger.

1. Choose Sampling mode
2. Configure an Event Pattern
3. Select Trigger

### Events

#### PCI0-PCI7 or PCIX0-PCIX7

The eight predefined editable patterns are, by default, labeled PCI0-PCI7 for PCI and PCIX0-PCIX7 for PCIX.

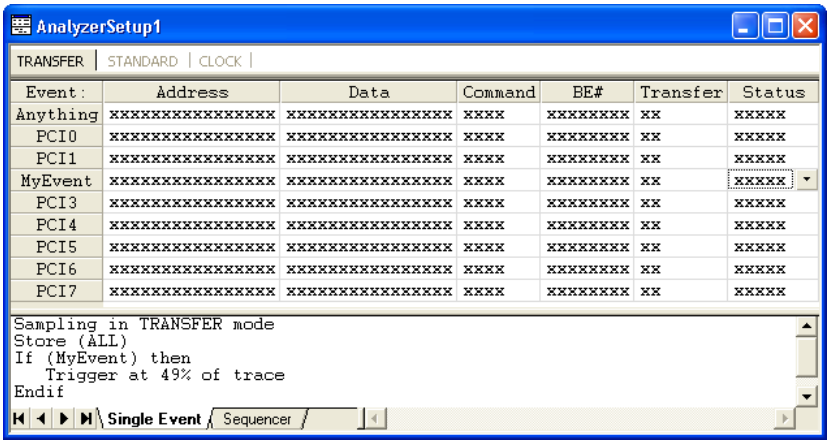



**FIGURE 4-9** *Analyzer Setup Screen*

Each of these events can also be renamed. In Figure 4-9,PCI2 has been renamed “MyEvent” (See “Rename” on page 74).

It is possible to define more than eight Events but only eight at a time can be used for triggering.

### Anything

In addition, there is a fixed, non-editable event labelled “Anything”. This pattern always has a complete “don’t care” pattern (all ‘x’), making it suitable as an unconditional trigger without having to clear one of the editable events.

## Patterns

The default Analyzer Setup window contains a selection of some of the more important signals and signal groups for the selected sampling mode. You can insert additional signals or signal groups (See “Manipulating Field Columns” on page 70), as well as patterns with user-defined labels (See “Manipulating Events” on page 73).

## Editing Event Patterns

You can fill in event patterns as binary, hexadecimal, or mnemonic values in the signal fields of any of the predefined event patterns except for the “Anything” event. You can also delete and insert new event patterns and signal. New event patterns can be labelled and existing ones renamed.

**TABLE 4-1. Summary of valid characters for Event fields.**

| | |
|---|---|
| **x** | Don’t care |
| **0** | Logic zero |
| **1** | Logic one |
| **r** | Rising edge |
| **f** | Falling edge |
| **a** | Any edge |

### Editing Fields

Place the cursor in the field you want to edit, and type in a new value. The new value can contain digits, or a mixture of digits and don’t cares (x = don’t care). Clear the field by selecting Clear from the Edit menu or pressing Delete on your keyboard, and then retype the desired value.
Press Enter, or move to another field to finish editing.

***Active low: #***

Active low signals are indicated with a "#“after the signal name. (Example:BE#). This means that an active signal is shown as a '0' in the trace.

***The NOT operator: !***

The NOT operator can be used for data, addresses, and some of the other fields. The NOT operator is used by typing an exclamation mark <!> in front of the field.

**“Don’t Care” value for signals**

Typing an x into a field will set the corresponding bit(s) to don't care and means that this bit (signal on the bus) will be ignored when the Vanguard is looking for a trigger condition or store qualifier. Fields can be set to “don’t care” by:

- Selecting Clear from the Edit menu.
- Choosing <clear> from a drop-down menu obtained when right-clicking on a field.
- Pressing the Delete key on your keyboard.

A Field column can be selected by left-clicking on the name of the column.
An Event row can be selected by left-clicking on the name of the event.

If a column or row is selected, then Clear will reset all values in the column or row to “don’t care”.

## PLP Training Sequence

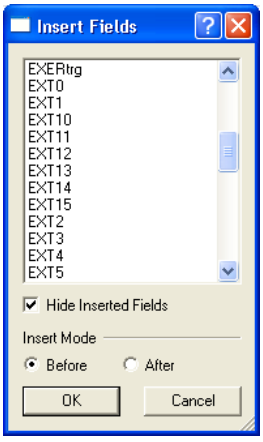
LANE# can either be PAD or 0-7. To specify a Lane number just type the number. PAD is used in training until the lanes have been numerated.

LINK# is either PAD or a number. To specify an number just type it in. This is used when multiple links are used on the same connector. This is not supported by the Vanguard but you may trigger when the Link number changes from PAD to, typically, 0.

## Manipulating Field Columns

Tip – More than one Field can be selected for one operation. Hold down the Ctrl key and click on the desired Fields. Alternatively hold down the Shift key, and use the cursor keys to select the Fields.

**Adding Fields**




1. Select a Field column by clicking on its name.
2. Open the Insert Fields dialog box using one of the following methods:
   - Click Insert from the Edit menu.
   - Press the Insert key on your keyboard.
   - Right-click on the Field and click Insert.
3. From the dialog box, click on the signal you want to insert, and click OK.
4. The field column of the new signal will appear before or after the column you selected in Step 1; depending on which option button you click in the “Insert Mode” section of the dialog box (Before or After).

Tip – While in the dialog box, type the first letter of the field to be inserted and the cursor will move to the first field in the list that begins with that letter.

### Removing Fields

1. Select a Field by clicking on its name.
2. Remove the Field using one of the following methods:
    - Click the Delete button  on the toolbar.
    - Click Delete from the Edit menu
    - Press Delete on your keyboard.

### Clearing Fields

1. Select a Field by clicking on its name.
2. Clear the Field using one of the following methods:
    - Click the Erase button  on the toolbar.
    - Click Erase from the Edit menu
    - Press Ctrl Delete on your keyboard.

### Moving Fields

Field columns can be moved by clicking on its name and dragging it to another position in the Events window.

---

**Note –** You can save your Field layouts by using Templates. See “Templates” on page 47 for details.

---

## Field Properties

Right clicking anywhere on a Field and choosing “Field Properties” will give more information about that Field. On some Fields, the format in which the field is used can also be modified.

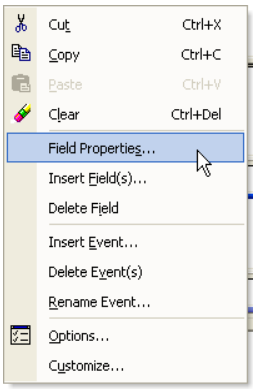
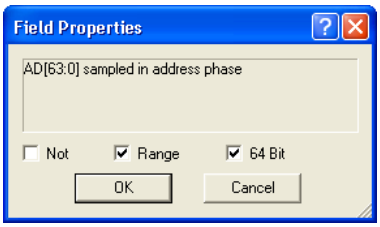



**FIGURE 4-10** *Right-click Field menu*

### Address and Data Options

#### *Range*




Selecting the Range check box allows you to specify a range of values as an event pattern in the Address or Data fields. Using a range of addresses is useful when interested in a specific data structure.

Range values can also be specified by pressing '-' on your keyboard while an address or data field is selected.

Negating the range is done by typing an exclamation mark(!).

#### *64 Bit*

Select this check box to enter 64 bit address and data values. Swapping between 64 and 32 bit values can be done by pressing 'q' while an address or data field is selected.

#### *Binary details*

Binary details makes it possible to specify "don't care" values at the bit level. Specifying binary details is done by typing a left bracket "(" in front of a hex digit in the Event Patterns window.

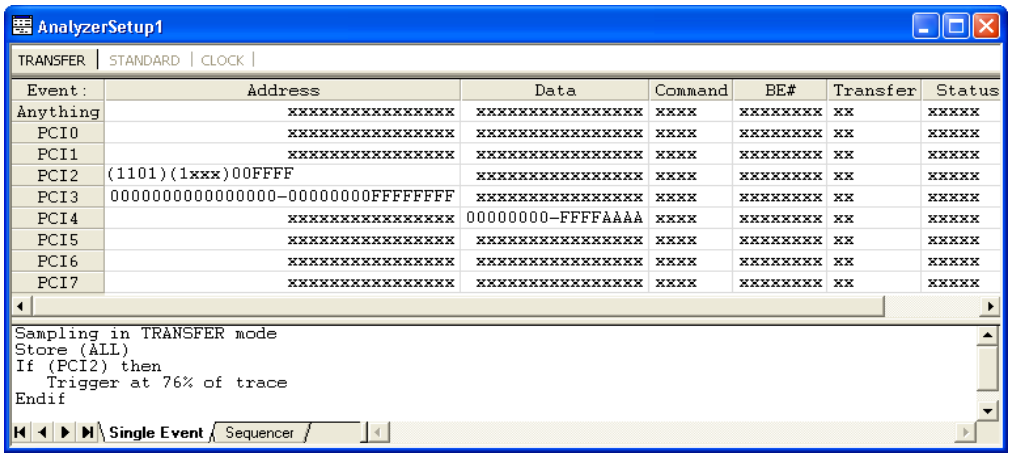



**FIGURE 4-11** *Using Field Properties to set address range.*

Figure 4-11 shows how the first two hexadecimal digits are expanded to the binary level, making it possible to have “don’t care” values at the bit level. Values containing binary “don’t cares” will be displayed as a “$” in the Event Patterns window, as seen in Figure 4-12.

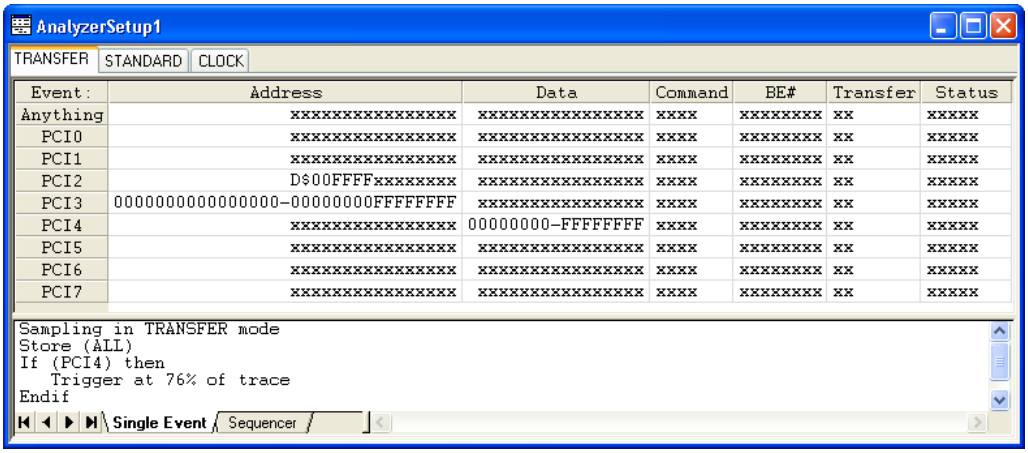
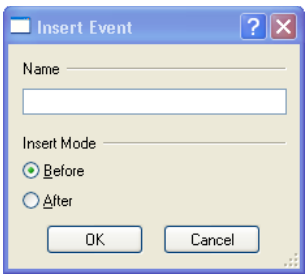



**FIGURE 4-12** *‘Don’t Care’ binary address range*

## Manipulating Events

### Add an Event




1. Select an Event by clicking on its name.
2. Open the Insert Event dialog box using one of the following methods:
   - Click Insert from the Edit menu.
   - Press the Insert key on your keyboard.

- Right-click on an Event and click Insert Event.

3. Type a name and choose one of the option buttons (Before or After), then click the OK button
4. From the dialog box, click on the signal you want to insert, and click OK.
5. The new Event will appear above or below the row you selected; depending on which option button you click in the "Insert Mode" section of the dialog (Before or After).

### Delete an Event

Select the undesired event by clicking on its name and perform one of the following operations:

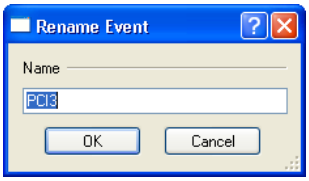
- On the toolbar, click Delete.
- From the Edit menu, select Delete.
- Press Ctrl + Delete on your keyboard.

### Clear an Event

1. Select an Event by clicking on its name.
2. Clear the Event using one of the following methods:
   - Click the Erase button on the toolbar.
   - Click Erase from the Edit menu
   - Press Ctrl Delete on your keyboard.

### Rename

1. Select an Event by clicking its name.
2. Open the Rename Event dialog box using one of the following methods:
   - Double click the Event name.
   - Right-click on an Event and click Rename.




3. Type a name and click the OK button.

### Copy & Paste Event Data

1. Select an Event by clicking on its name.
2. Click on the Copy button or select Copy from the Edit menu.
3. Place the cursor on the event where you want to copy the event data and click on the Paste button or select Paste from the Edit menu.

Tip – Event data can be copied between the Vanguard tools. For example, you can copy event defined in the Analyzer Setup Window, and past them in the Statistics Setup Window.

**Note –** You can save your Event layouts by using Templates. See "Templates" on page 47 for details.

## *4.5 The Sequencer*

The Sequencer is a state machine that enables you to define complex triggers, store qualifiers, count conditions and more. The Sequencer allows event patterns to be combined sequentially using IF, ELSIF, ELSE statements, and the logical operators AND, OR, NOT. A graphical representation of your sequence is shown simultaneously.

The Sequencer is suitable where it is necessary to trigger the Analyzer when:

- A certain event occurs after a series of other events
- To filter out samples of no interest
- Count a number of occurrences of an event before triggering

### Notation

**Parametric Keywords:** Are shown in UPPER CASE letters. e.g. ALL, ANYTHING.

**Event Names:** Both pre-defined and user-defined names are shown in UPPER CASEA maximum of eight different event names can be used in a Sequencer program at the same time. A warning will be given when the ninth event is taken into use. There is no limit on how many times one event name can be used in event expressions in the same program. The same event may serve as trigger, store, and count conditions.

---

**Note –** The eight event comparators are shared with the Statistics Functions. If you are currently using comparators (and they are actually in use) in the Statistics Function, those comparators are unavailable in the State Analyzer and visa versa.

---

**Operators:** Are shown with Initial Caps, e.g. Store, Trigger.

**Fill-in words:** Are shown in lower case, e.g. in, of.

### Syntax

**Line Numbers** Each line in a sequencer program has a number. This number consists of a ‘state’ number, and a ‘line within state’ letter. The number 3.a means ‘line 1 in state 3’. Line numbers are generated automatically by the sequencer and can not be edited.

**Indents** Indents are used after If, Count, Delay, Elsif and Else statements. Indents are generated automatically by the Sequencer and can not be edited.

```
2.a: If (PCI2) then
2.b:    Trigger ...
3.a:    Sampling ...
```

**Current state indicator**An arrow ‘=>’ indicates the current state of the Sequencer.

## Operators

A sequencer program is built from a number of operators which can be listed in two groups: Actions and Transitions. Transitions are used to determine which Actions are executed.

**TABLE 4-2. Sequencer operators.**

| | |
|---|---|
| **Actions** | Sampling, Store, Delay, Count, Trigger, Halt |
| **Transitions** | If, ElsIf, Else, Goto |

### Sampling

The Sampling operator is used to specify sampling mode. The first line in the Sequencer program is always a Sampling line and cannot be deleted. The sampling mode is changed by selecting the Sampling Modes tab at the top of the Analyzer Setup Window.

A Sampling expression is implicitly valid for all subsequent states in the Sequencer program, until superseded by a new Sampling condition.

### Store

The Store operator is used to 'filter' the captured samples according to the event expression contained within the Store operator. The second line in the Sequencer program is always a Store condition and cannot be deleted.

A Store expression is implicitly valid for the rest of the Sequencer program, until superseded by a new Store expression.

The predefined expressions `ALL` and `NOTHING` are handled separately from the eight definable events, and so can be used in addition to the eight definable events.

**Note –** A sample causing a Trigger is always stored, regardless of any set store conditions. Only one trigger statement and be executed. Once the trigger has occurred, the trace buffer will be filled up according to the sequencer program and sampling will stop when the buffer is full. The Sequencer can not restart it self after trigger has occurred.

### If / Elsif / Else

`If / Elsif / Else` statements are used to control the branching of a Sequencer program. Nested `Elsif` statements are possible, limited only by the number of possible Event Expressions. Both `Elsif` and `Else` are optional after an If.

The predefined expression ANYTHING can be used as an Event expression. ANYTHING is handled separately from the eight definable events, and so can be used in addition to the eight definable events.

**Note –** An `If .. Elsif` sequence without an `Else` always repeats itself if none of the conditions are met.(Implicit `goto` self)
Statements like
```
Else
Goto Current state
```
can be considered as an implicit closing statement.

If no states follow an `If .. then`, `Trigger` in the Sequencer program an implicit jump to a state where the prevailing store condition is repeated takes place.

This is to avoid storing both the specified store condition and the trigger condition if the trigger condition should occur again.

**Goto**

The `Goto` statement moves the execution of the Sequencer program to the beginning of another state. `Goto 1` will restart the Sequencer program.

**Count**

`Count` is used to count occurrences of specific cycles/events on the target bus. If a `Count` statement is used, the Sequencer program will not advance until the specified number of cycles that matches the event pattern attached to the `Count` statement occurs.

Up to 3 `Count` statements can be used in a Sequencer program.

**Delay**

`Delay` will pause the Sequencer program for a time, before it is allowed to advance to the next state.

```
Delay N {ns|us|ms} then if(<Event Expression>)then
```

Where `N` is a measure of delay in units of nanoseconds (ns), microseconds(us) or milliseconds(ms). Up to 3 `Delay` statements can be used in a Sequencer program.

**Note –** When converting between units, the value will be rounded to the nearest integer of the new unit.

The delay counter can be synchronized by putting an ”`If (ANYTHING) then`“ before the first delay. The delay counter will then start to count when the first sample occurs on thebus after the sampling is started.

A construction like `Delay ... Elsif` can be used to exit a delay interval on a certain condition, before the delay time expires.

A sample is required after the delay time is counted down, before the Sequencer will proceed to the next state, otherwise, a trigger will occur.

**Trigger**

The `Trigger` operator determines where in the Sequencer program the trigger should be.

Even if multiple trigger statements exist, the Trigger Position will be kept the same throughout the Sequencer. Modifying one of the trigger statements will result in the same modification to all other trigger statements.

`Halt` can be used to replace "Trigger at 100% of trace" if `Trigger` has already been used with one of the other parameters.

**Note –** The trigger sample is always stored, regardless of store conditions specified.

**Halt**

The Halt operator causes the tracer to halt and display the trace.

**Logical Operators**

Elements can be combined using logical operators in Table 4-3 and the Transitions listed in Table 4-2 .

**TABLE 4-3. Logical Operators**

| Symbol | Logical Operator |
|---|---|
| + | OR |
| * | AND |
| ! | NOT |

The use of brackets to build more complex event terms using logical operators is also supported.

The logical AND operator, "*", has precedence over the OR operator, "+". Brackets can be used to change the order of evaluation:

$$A+(B*C) = A+B*C, \text{ but } (A+B)*C \neq A+B*C$$

This is due to the order of evaluation.

The parenthesis around the OR expression forces the OR to be evaluated before the AND.

$$!(A+B)*C \neq !A+B*C$$

The logical NOT operator can be used on single event names, or on sub-expressions within brackets. The logical NOT operator "!" is always evaluated first.Summary of implicit Actions and Transitions

The Sequencer is a compact practical way of controlling the operation of the Analyzer. To minimize the need for programming, a number of implicit actions in the Sequencer exist to give desired results without using explicit commands:

- A Sampling expression is implicitly valid for all subsequent states in the Sequencer program, until superseded by a new Sampling condition.
- A `Store` expression is implicitly valid for all subsequent states in the Sequencer program, until superseded by a new `Store` condition.
- The sample causing a Trigger is always stored.
- If no states follow an `If .. then, Trigger` in the Sequencer program then an implicit jump to a state where the prevailing store condition is repeated takes place. This is to avoid storing both the specified store condition and the trigger condition, if the trigger condition occurs again (according to the above rule saying that trigger samples are always stored).
- An implicit `Else Goto` current state is always present after an `If-Elsif` sequence if no `Else` is specified, so that the `If` test will always be repeated for the next sample if none of the conditions were met.
- `Goto` next state is implicit after a then or after an `Else`, where next state is the state belonging to the next line containing an `If`.

## Using the sequencer

Select Sequencer mode by clicking on the Sequencer tab, as shown in Figure 4-13.

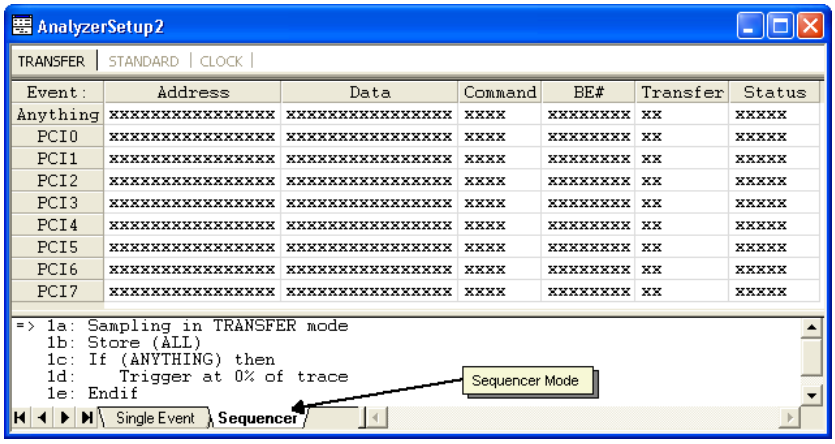



**FIGURE 4-13** *Selecting the Sequencer tab.*

### Selecting Parameters

Parameters are selected with the left mouse button. A double-click of the left mouse button on a selected parameter open the Event Expression dialog box.

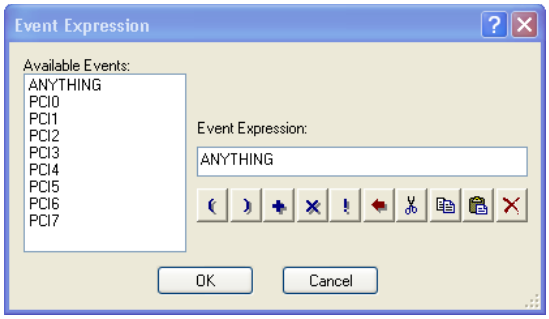
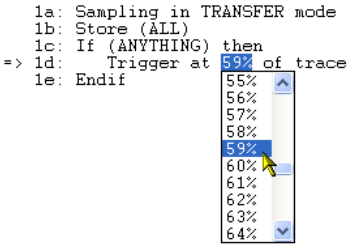


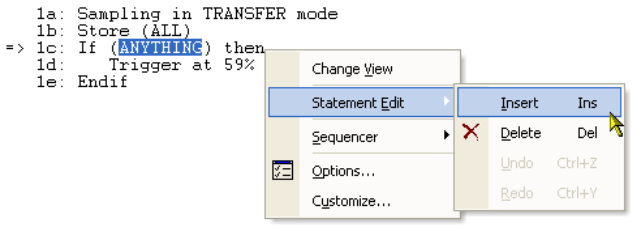

**FIGURE 4-14** *Event Expression dialog box*

Some parameters, such as the Trigger Position, use a drop-down menu from which to choose values.




**Inserting and Deleting Statements.**

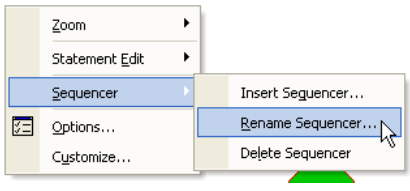
Right-click on the statement line from which you wish to add a line below, and choose Statement Edit, then click Insert.



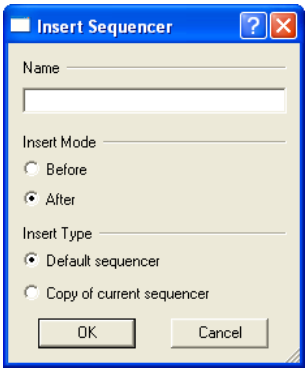

Alternatively press the Insert key on your keyboard.

**Inserting, Renaming and Deleting Sequencers.**

Right-click on the Sequencer window and select the operation you wish to perform from the Sequencer sub menu.



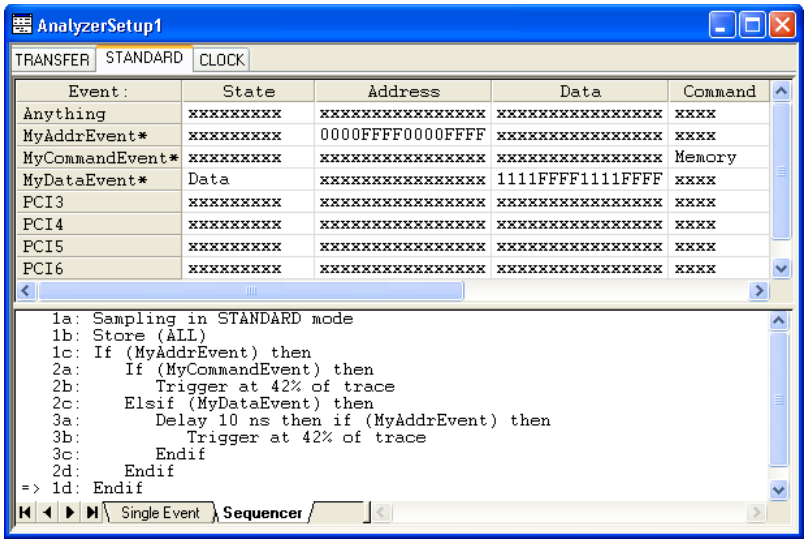
Inserting a Sequencer will present you with a dialog box prompting for a name for this sequence, and where to place it. You can also make a copy of the current sequence.

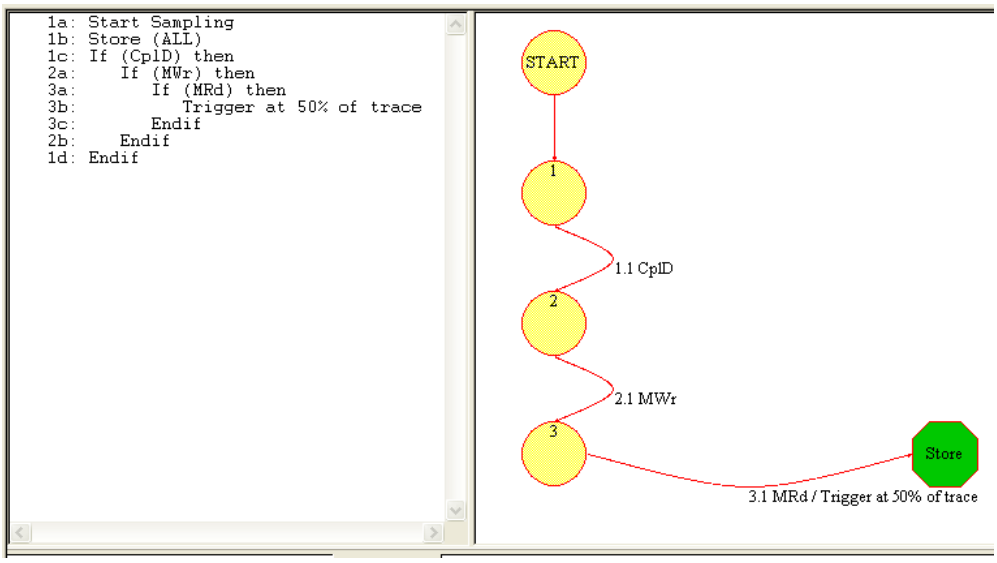



## Graphical View

The graphical representation of the sequencer is useful where the sequence of events becomes complex. Each of the icons in the graphical view can be moved around by selecting it and holding down the left mouse button while moving the icon.

## Sequencer Example



## 4.6 Trace Display

The data in the trace buffer is automatically displayed when the current trace is filled with samples or if the Analyzer. If halted manually, then you must press the Show Trace button (Shift F9) to view what is in the trace buffer. Also note that the trace buffer might not be full when halted manually. The contents of the trace buffer can be viewed as an alphanumeric display or as waveforms.

**Alphanumeric**The Alphanumeric display format is used by default.

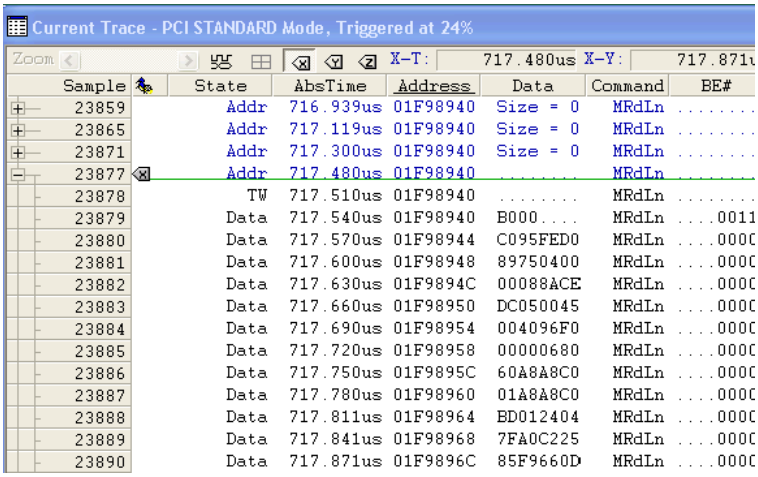
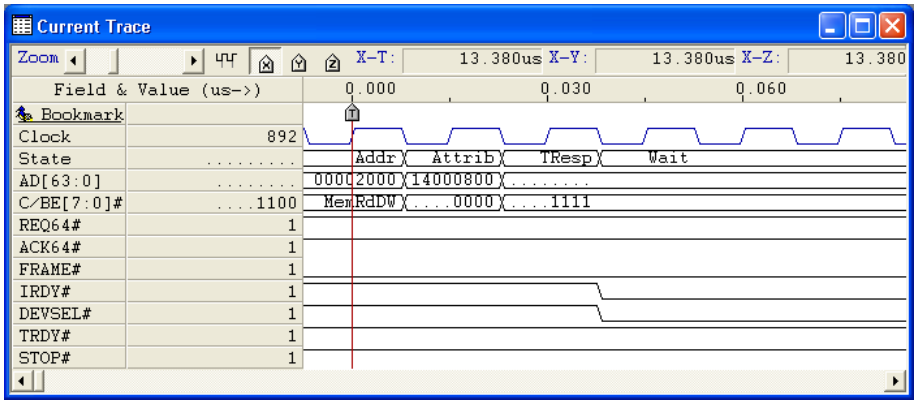



**FIGURE 4-15** *The Trace Display in Alphanumeric mode*

The alphanumeric trace list shows the samples collected in the trace buffer as a list of binary or hex values for each signal group. The alphanumeric trace list presentation can be selected for all sampling modes. Figure 4-15 shows an example of an alphanumeric trace list.

**Waveform**The waveform display format can be used to display the trace when Clock or Standard sampling has been used. Transfer sampling cannot be displayed in waveform format.

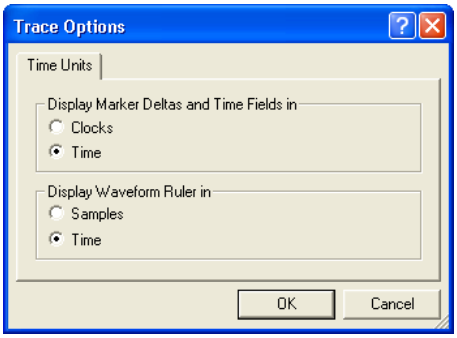


### Additional Windows

Additional windows, or views of the current trace buffer, may be opened and closed when needed using the Workspace Window. The windows are numbered 1, 2 etc and are totally independent views of the same trace memory.

Additional windows are opened using the Window/New Window menu option.

### Trace Display Options




***Time Units***

**PCI/PCI-X Only:** Display Marker Deltas and Time Fields in Clock or Time. Display in Time will 'round off' the time increments in order to present a more readable display. The Time Units are: picoseconds (ps), nanoseconds (ns), microseconds (µs), milliseconds (ms), seconds (s), minutes (m) and hours (h).

Display Waveform Ruler in Samples (one per clock) or in Time.

## Editing the Trace window

The Trace window can be edited in the same way as the Event Patterns window, both in alphanumeric and waveform mode. Signals can be added , removed and reorganized.

Right click anywhere on the trace display to access the menu shown in Figure 4-16.

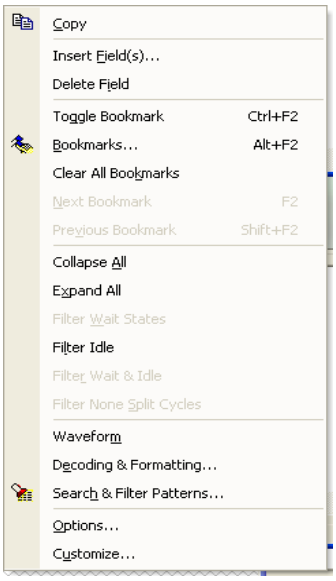



**FIGURE 4-16** *Trace Display menu*

### *Add a Signal*

Select Insert from the Trace Display menu, from the Edit menu, or press the Insert key on your keyboard to bring up a dialog box from which you can select a signal name to add to the trace display.

### *Remove a Signal*

Place the cursor on the signal name you want to delete and press Ctrl Delete. Alternatively, select Cut from the Edit menu or click Delete from the Trace Display menu.

### Absolute or Relative Time in the Trace Window

The trace can be displayed as absolute time from the trigger sample, relative time between the samples, or both. AbsTime (absolute time) is default, and is displayed as a field column in the Trace Display window. The `RelTime` (relative time) option is inserted into the Trace window in the same way signal fields are inserted (“Manipulating Field Columns” on page 70).

### Formatting Options

There are two different ways of presenting the control signals in the trace. Either as mnemonics such as Size, Status, and Err, or as bit patterns such as `Address`, and `Data`.

In Transfer mode, it is very convenient to display the signals with mnemonics when sampling only once per data cycle.

For example, the Command field, describing the type of cycle, is much easier to read when using mnemonics.

## Navigating the Trace Buffer

There are three ways of moving around in the trace buffer.

- With the mouse and keyboard.
- With the Jump tools.
- With the Search tools.

### Mouse and keyboard

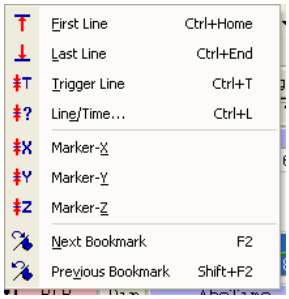
Use the mouse and left click parts of the trace display to select a field.

The mouse wheel can be used to scroll up and down through the trace buffer without moving any markers.

The right and left cursor keys select signal, and the up and down keys scroll the buffer. The currently selected field is underlined.

PgUp and PgDown will scroll up and down through the trace buffer using the currently selected marker.

## Jump Tools




The Jump tools are available both at the Jump menu and the Navigation.

**First Line**Jumps to the first line in the trace.

**Last Line**Jumps to the last line in the trace.

**Trigger Line**Jumps to the line on which the Trigger is found.

**Line/Time:** Jump to a user specified line or time in the trace.

### Markers

When an Event is selected on the Trace View, that Event is marked with one of three Markers; X, Y or Z depending on which is selected.**Marker-X:** Jump to the X marker.

**Marker-Y:** Jump to the Y marker

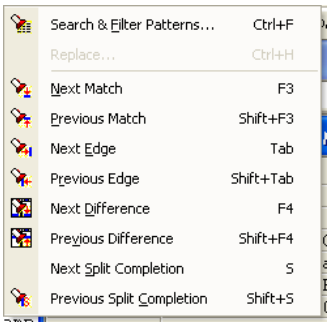
**Marker-Z:** Jump to the Z marker

**Next Bookmark:** Jump to the next bookmark. (See “Bookmarks” on page 89)

**Previous Bookmark:** Jump to the previous bookmark. (See “Bookmarks” on page 89)

## The Search tools

Search tools are selected from the Navigation menu, and the Navigation. The Search commands offer powerful search and extract functions.




Search  locates a particular pattern in the trace buffer. Extract provides a qualified presentation of samples from the trace buffer, so that only samples matching the specified pattern are displayed.

### Search and Filter Patterns

When selecting the Search tools for the first time, the Edit  Search  Pattern is the only available option. The Search/Filter edit window is used in the same way as the Event Patterns window (“Editing Event Patterns” on page 69).

#### *Event Searching*

This is where you define which patterns to search for.

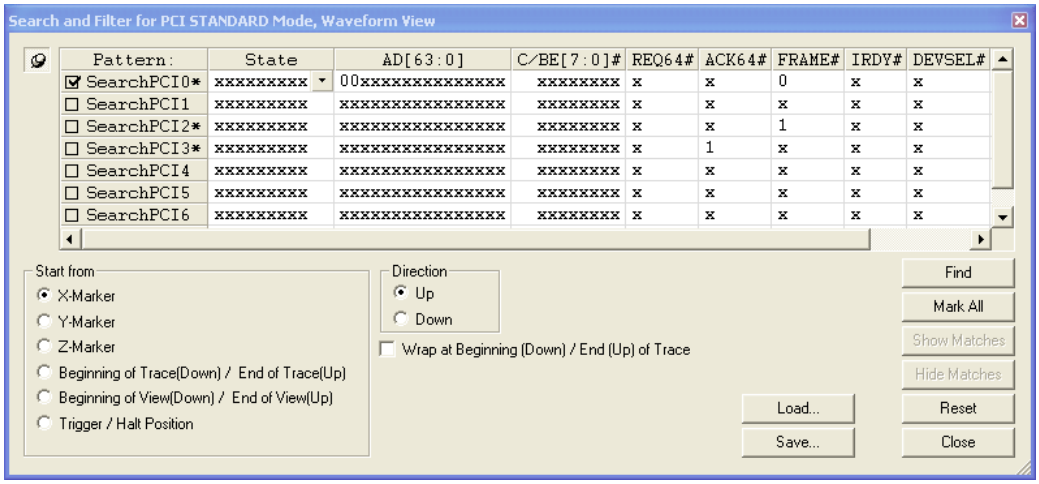



**FIGURE 4-17** *Search and Extract options menu*

### *Search Options*

- **Start From**: Specifies where in the trace searching should begin.
- Direction: determines whether searching should move up or down in the trace, from the point specified by the Start From option.
- **Wrap**: When a search reaches the end or beginning of the trace buffer, searching will wrap around if this option is selected.
- **Find**: Find the first match
- **Mark All**: Highlight all matching lines.
- **Show Matches**: Display only the lines that match.
- **Hide Matches**: Hide all lines that match.
- **Reset**: Reset all highlighting and Show/Hide settings.
- **Close**: Exit the Searching tool.
- **Load**: Load a previous search pattern.
- **Save**: Save the current search pattern. This will save the search pattern to file for retrieval later. The file has the extension **.spf** If there is more than one trace type displayed, a dialogue box will appear asking you to specify which one to save.

**Next Match**Finds the next matching sample.

**Previous Match**Finds the preceding sample that matches the search pattern

**Next Edge**Search for the next high to low, or low to high transition of the selected signal.

**Previous Edge**Search for the previous high to low, or low to high transition of the selected signal.

**Next Difference**Move to the next difference between two traces when using Trace Compare.

**Previous Difference**Move to the previous difference between two traces when using Trace Compare.

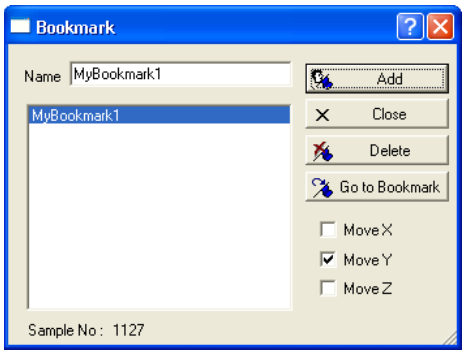
**Next Split Completion**Move to the next occurrence of a Split Completion (PCI-X only).

**Previous Split Completion**Move to the previous occurrence of a Split Completion (PCI-X only).

## Bookmarks

The Bookmarks toolbar is used to define and jump to bookmarks in the Trace display.

**Bookmarks**Opens a dialog box to manage user defined bookmarks.




Type in a name in the Name text box and click Add to create a new bookmark on the currently selected trace line.

Using this method, it is possible to create a list of bookmarks to jump to. You can activate a window by either double-clicking on a selected bookmark this box, or by selecting a bookmark and then clicking the 'Go to Bookmark' button. If you select multiple bookmarks, you can choose Delete, or Close. Use the Ctrl or Shift button together with the mouse to select multiple bookmarks.

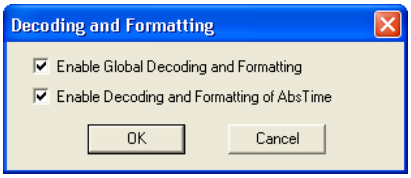
**Toggle Bookmark**Toggles between showing and hiding bookmarks in the Trace display.

**Clear Bookmarks**Removes all bookmarks.

**Next Bookmark**Jump to the next bookmark.

**Previous Bookmark**Jump to the previous bookmark.

### Changing the Alphanumeric Formatting Template




Select the signal you want to change and right click to access the Trace Display menu. Select Decoding and Formatting and a dialog box opens. The top most option enables or disables decoding and formatting globally, i.e. it concerns all the signals fields. The next option enables or disables Decoding and Formatting the currently selected signal field; in this example AbsTime.

If decoding and formatting are turned off globally, it is not possible to enable the current signal field.

By default, global decoding and formatting is on in Standard and Transfer mode, and off in Clock mode.

**Note –** Some signals are fixed as mnemonics (e.g. the `Size` field in Transfer mode). A dialog box containing only the first option will appear in these cases.

## 4.8 Waveform Trace Display

The waveform display is provided to show the logic level of individual signals graphically as a function of time. This is particularly useful to show timing relations between different signals for hardware analysis.

Tip – To be sure of capturing some traffic, trigger the tracer with an event where FRAME# is set to zero, because there is always traffic when FRAME# is active.

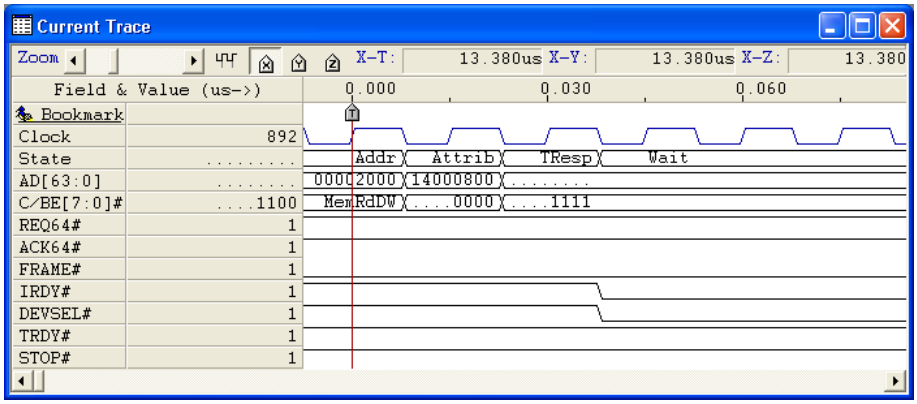



**FIGURE 4-18** *The trace display in waveform mode*

### Navigating the Trace Buffer in Waveform Mode

There are three ways of moving around in the trace buffer.

- With the mouse and keyboard.
- With the Jump tools.
- With the Search tools.

### Mouse and keyboard

The currently selected marker (X, Y or Z) will be positioned wherever a left mouse click is made on the trace display. Left clicking on the display will also select a signal field (underlined).

The up and down cursor keys select signal fields, and the left and right keys scroll the buffer with the currently selected marker. The currently selected field is underlined.

PgUp and PgDown will scroll up and down, through the trace buffer using the currently selected marker.

**Zoom** The zoom scroll bar will expand and contract the time scale of the trace display.

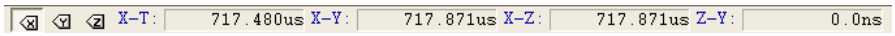
## Jump tools

The Jump tools work the same way as in alphanumeric mode (“Jump Tools” on page 86), except for two additional options. These additional options are for jumping to two user-positioned markers.

### Markers

Markers are convenient for marking places of interest in the trace buffer. Two markers can also be used to limit statistics functions to a given area, or to measure the time between two signal edges.

Markers are selected from the buttons X, Y, and Z on the Trace Display, and are position using the left mouse button.




The time between markers is also display on the Trace Display.

## 4.9 Trace Handling

**Open a new Analyzer Setup File**

1. On the File menu, click New
2. In the New dialogue box, choose Analyzer Setup
3. You can give this file a name by entering a name in the Filename text box. If no name is given, then BusView will assign a name. Default name is AnalyzerSetup1.
4. You can choose where this file is saved by pressing the browse button 

in the Location text box. The file can also be saved after the editing is complete.

**Loading an Analyzer Setup File**

1. On the File menu, click Open.
2. In the "Files of type:" list, click "Analyzer Setup Files".
3. Browse to the directory in which your Analyzer Setup Files have been saved and choose the file you wish to open.

**Analyzer Setup Options**

- With an Analyzer Setup window active, click Options on the Tools menu.
- Alternatively, right click anywhere on the Analyzer Setup window, and select Option from the menu. See "Analyzer Setup Options" on page 61

**Navigation**

Jump Tools

The Search tools

Bookmark options

Trigger


Zoom in on Waveform

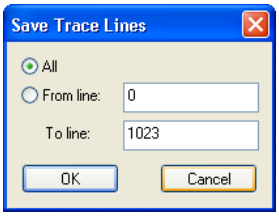

Zoom out of Waveform

## Trace Files

Trace buffer data can be saved to a file by clicking on Save As in the File menu.The file format contains a header with target type, sampling mode, trigger position, trigger line number etc., so that the file can be reviewed exactly as captured.

### Save as

Click Save As from the File menu and enter a file name.




A dialog box appears where you can specify which lines you want to save.

The trace can be saved both as binary files (extension **.str**) and as ASCII files (extension **.sta**). The ASCII files can then be opened and edited in any other text editor, but because they have not saved all the vital information about the trace, they can not be opened in BusView again.

### Trace File Size

The size of the trace file depends on a number of factors and we recommend having at least 512MB free disk space.

One sample has a size of 40bytes, therefore a full trace buffer containing 2M samples may require around 80MB of storage space. A trace file also contains information about the setup, filter data and other settings required to present the data in BusView and this can increase the size of the trace file size.

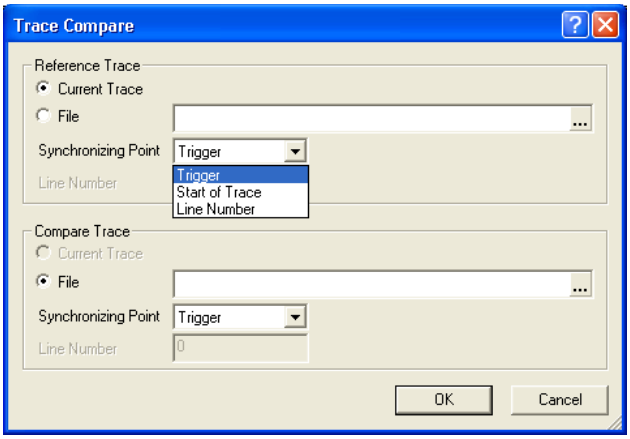
In general, it is reasonable to expect that a full 2M Sample uncompressed trace file will exceed 100MB in size. A temporary file is used to hold the current trace. Trace files are compressed when saved to disk.

Saving as an ASCII file will greatly increase the saved file size.

## Trace Compare

The Trace Compare functionality compares two traces line by line, and marks all lines that do not match with yellow in both traces. In addition the field which actually causes the mismatch is marked with orange. The colors for highlighting can be changed from the ‘Views,Trace’ page in the ‘Tools,Customize’ menu.

To open the Trace Compare dialog box, click on Trace Compare from the Tools menu.




Reference Trace is the trace from which comparisons are made from, whereas Compare Trace is the trace on which comparisons are made to. Each of these has the following options:

**Current Trace** Use the trace currently in the Vanguard trace buffer

**File** Use a previously saved trace file. Click the ... button to browse for the file.

**Synchronizing Point:** Specify from where the comparison will start from. This can be; Trigger Position, the start of trace or any user specified line in the trace. If one trace is longer than the other, all lines beyond the boundaries of the shortest trace are shaded.

**Line Number:** The line number on which to synchronize on, when the 'Synchronizing Point' is set to 'Line Number'.

For each view, signals are compared as shown on the screen, and only inserted fields are compared against the same fields in the other trace. For example; if the Absolute time is different, but this is expected, delete the absolute column from the trace and this will be ignored as a difference.

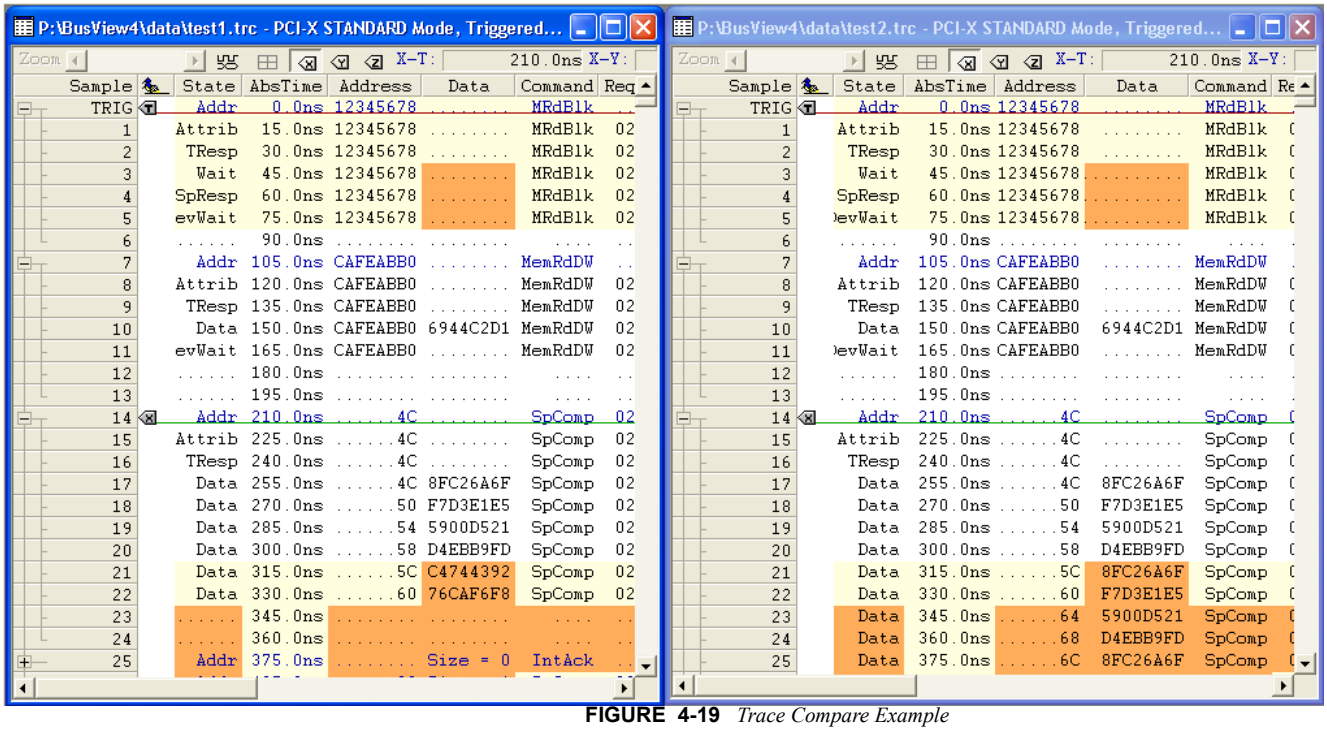
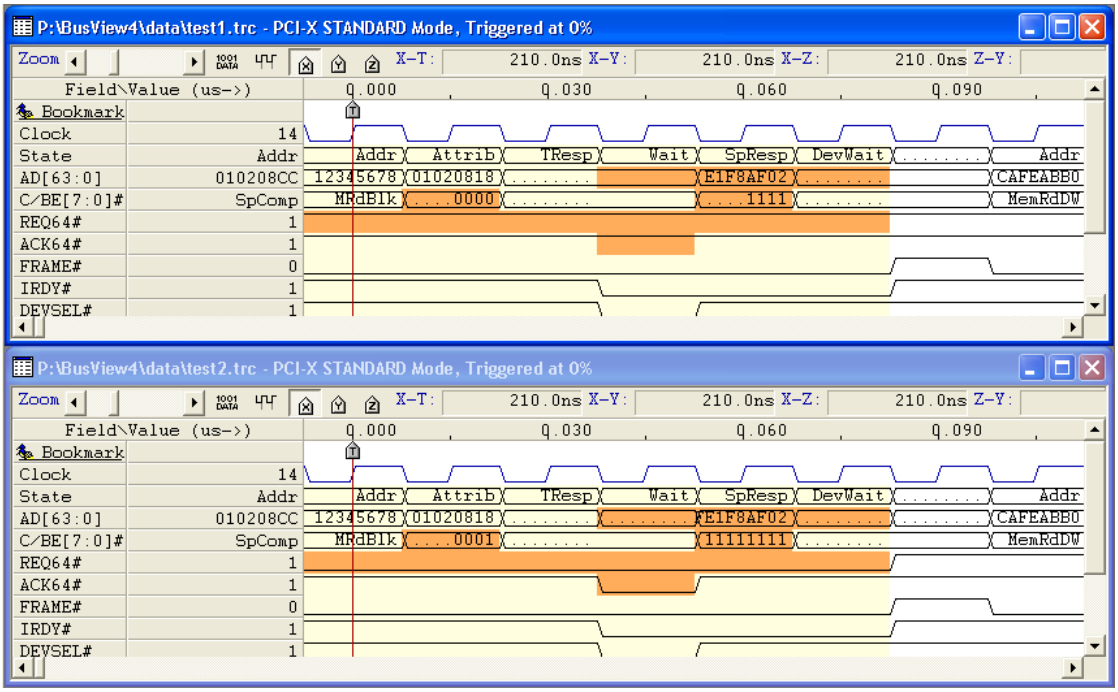



**FIGURE 4-19** *Trace Compare Example*

The highlighting can only be turned off by closing one of the compared trace files.The comparison highlighting is also shown on waveform trace displays.



## Trace Counting

Trace Counting is an operation performed after a trace has been captured and displays a number of statistics.

The Trace Counting dialog box is opened by clicking on Trace Counting in the Tools menu.

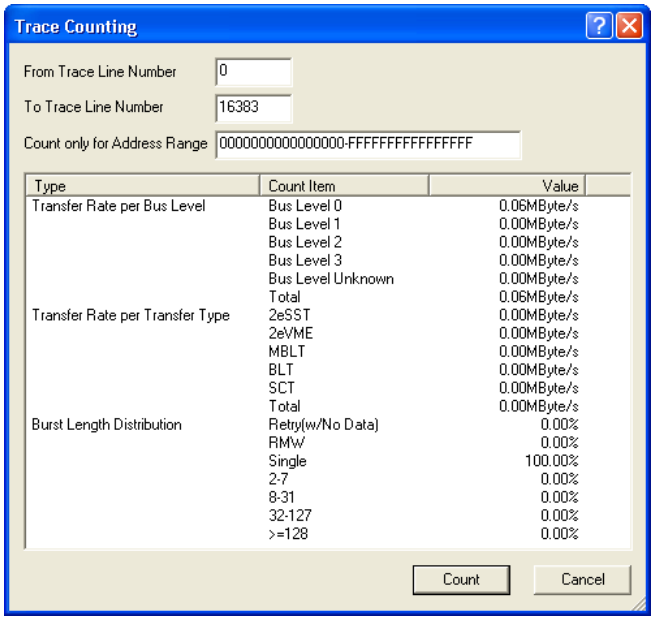
**From Trace Line Number:** Trace Line from which counting should commence.

**To Trace Line Number:** Trace Line at which counting should stop.

**Count only for Address Range:** Count transaction only within this address range.

The “Types” shown in the Trace Counting dialog box once a trace Count has been performed, have the same meaning as those listed in “Pre-defined Statistics” on page 106 with one exception, the ‘Transfer Rate’. In this case, Transfer Rate shown is for the whole bus.

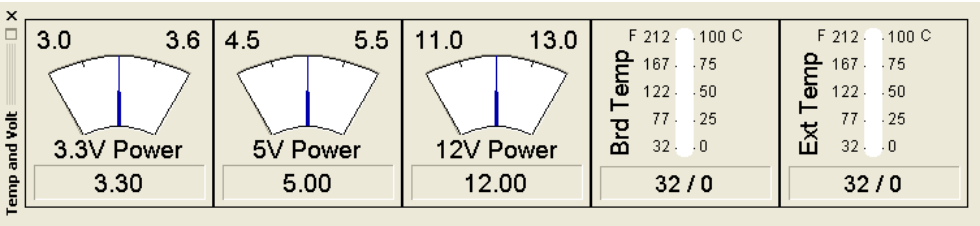
**FIGURE 4-20** *Trace Counting*



## *4.10 Voltage and Temperature Meters*

Real time temperature and voltage readings from the Vanguard are shown in BusView by clicking on Voltage & Temperature in the View menu.

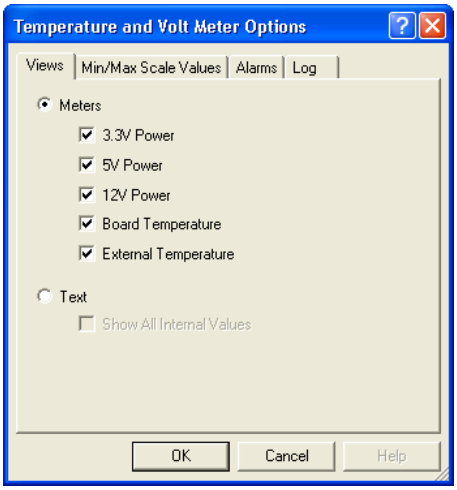
The voltage measured is dependent on how the power jumpers are installed.




### Temperature and Volt Meter Options

Options are available to modify which temperature and voltage reading are shown, and in what format they are shown.

It is also possible to enable alarms that sound when a temperature or voltage reading is outside of some user-defined minimum and maximum values.




The Log option will allow you to record temperature and voltage levels to a text file with the extension **.tvf**

BusView will update this file every second.

# 5 Statistics Functions

Statistics are divided into pre-defined functions, and user-defined functions.

- Introduction
- Statistics Setup Window
- Pre-defined Statistics
- User-defined Statistics
- Hardware Counters
- Statistics Charts
- Statistics Files

## 5.1 Introduction

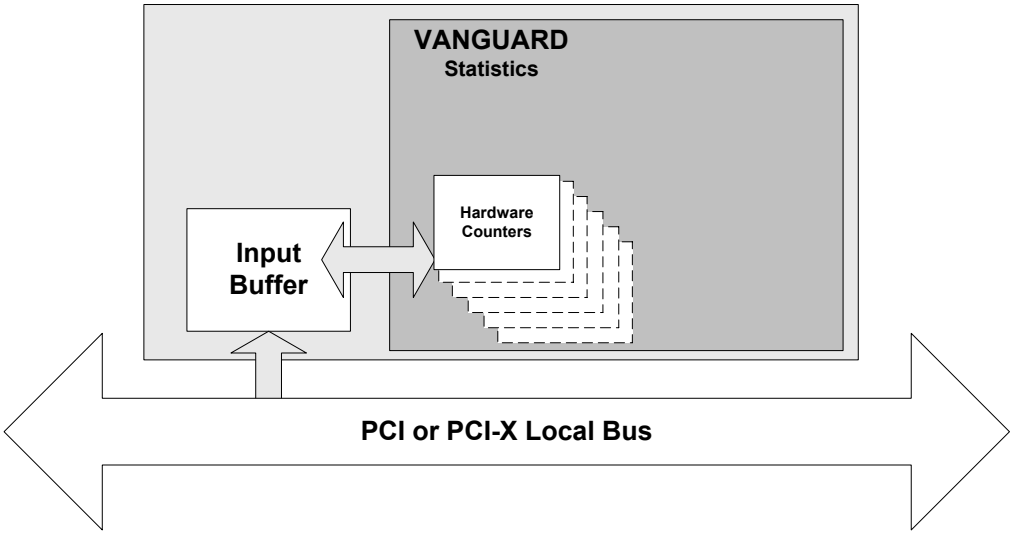



**FIGURE 5-1** *Block diagram of the Statistics module*

All statistics are measured in real-time. The Statistics Functions have over 50 hardware counters programmed to increment on certain bus events.

In addition, there is a dedicated counter that counts the total number of samples taken. Every time this counter reaches its maximum count, the counters are disabled, their values read, and then immediately re-enabled to resume counting while the charts are computed and displayed.

This method ensures that only a minimal amount of bus activity is missed from the measurement between each update of the charts. The sampling length is user configurable through the options menu. Longer sampling intervals will decrease the relative number of samples lost.

- Arbiter Latency
- Wait States

**Pre-defined Statistics for PCI**

- Bus Utilization - Displays four parameters that measure bus traffic performance. These are: Transactions, Data Total, Data Burst, and Efficiency.
- Transfer Rate - Gives a measurement in MTransfers/Sec and MB/Sec.
- Command Distribution - Counts individual commands used on the bus.
- Burst Length Distribution - Counts the number of bytes in each transaction, and displays them according to burst length.
- Arbiter Latency - Counts the average number of clock cycles from REQ# asserted to GNT# asserted.
- Wait States - Counts the average number of wait states per transaction.

## Statistics Setup Window

Open the Statistics Setup Window using one of the following methods:

- Open a new Statistics Setup File from the menu bar (File, New).
- Using the Workspace Window.
- Load a Statistics Setup File.

Detailed explanations of Statistics Files are found at the end of this chapter. The Statistics Setup window has the following sections:

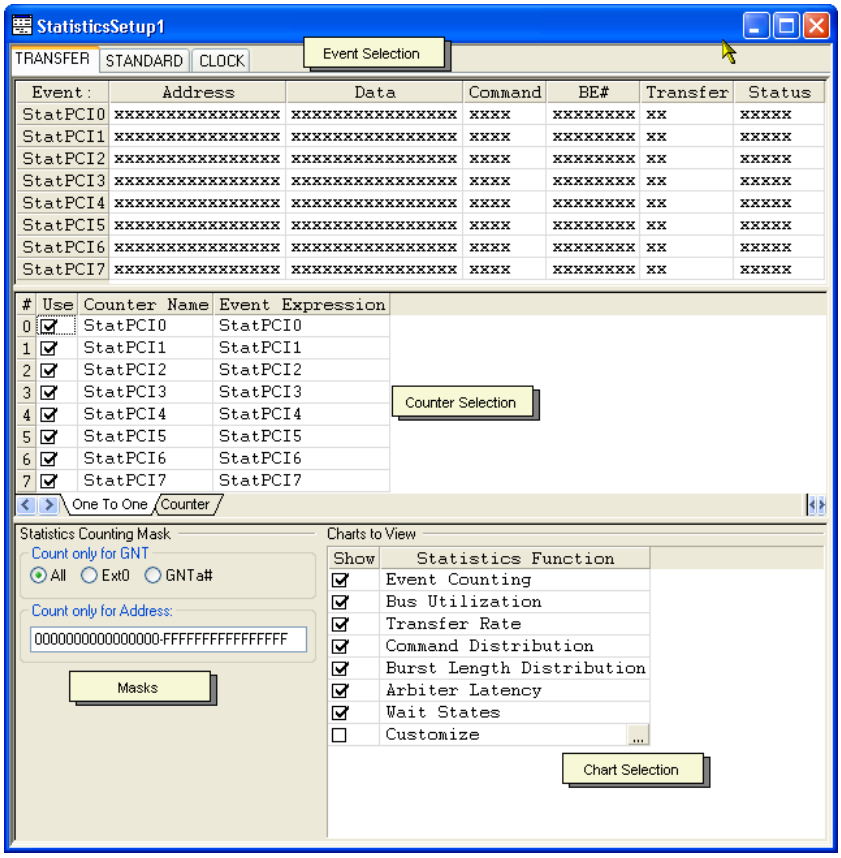



**FIGURE 5-2** *Statistics Setup Window*

**Event selection:** Events are defined in the Event Selection window and are edited in the same way as that used for the State Analyzer (See “Editing Event Patterns” on page 69). These are used for the Event Counting and Customized statistics.

**Counter Selection:** There are eight counters available for the Event Counting and Customized statistics function.

- **One to One mode** - Each counter can be associated with one event.
- **Counter mode** - Each counter can be associated with more than one event using boolean expressions.

---

**Note –** You can save your Statistics layouts by using Templates. See “Templates” on page 47 for details.

---

**Statistics Counting Masks:** Used to reject certain results from the statistics.

- **Count only for Requester ID** (PCI-X only) - Requester IDs are used in PLP Distribution and Payload Length statistics.
- **Count only for GNT** (PCI only) - Choose GNT.

**Chart Selection**Select which statistics function to view.

## Statistics Setup Options

The Statistics Setup Options dialog provides some flexibility in how the statistics are generated. Open the Statistics Setup Options dialog to set the options listed below.

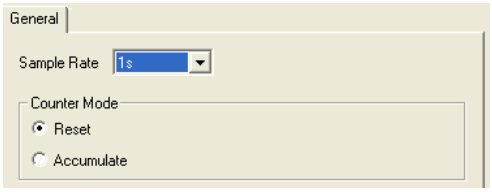



**FIGURE 5-3** *Statistics Setup Options Window*

**General**

Sample Rate - Time interval between samples.

Counter Mode -

- **Reset** - In Reset mode the displayed value is the counter reading shown as a percentage of the total number of samples, i.e.:

  Displayed Value = (EventCount / Total Count) * 100%

- **Accumulate** - In Accumulate mode, the displayed value is the cumulative sum of all previous counter readings shown as a percentage of the accumulated total number of samples, i.e.:

  Displayed Value = (ΣEvent Counts / (Total Counts * N)) * 100%

  where N is the number of updates in the session.

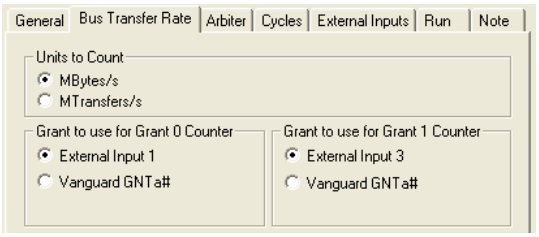
Selection of the Accumulate versus Reset mode is typically driven by the total number of samples to be observed in the measurement. Measurements made in Clock Sampling mode typically require the use of the Accumulate mode to yield significant results because the counters reach terminal count very rapidly in response to the fixed frequency of the sampling clock.

Bus cycle measurements made in Transfer Sampling mode may or may not require the Accumulate option to yield significant results.

Bus cycle measurements are affected by two key application specific factors: The total number of cycle operations occurring on the back plane and the frequency at which the cycles occur.

The measurement of applications consisting of less than 536870912 (512M) bus cycles may be accomplished within the limits of the Reset mode of operation. This mode is often quite sufficient to support detailed characterization of new software and firmware in an isolated environment. However, characterization of applications inside fully operational system environments typically requires use of the Accumulate mode to achieve the desired measurements

**Bus transfer Rate**




**Units to Count** - Selects the unit used to display Bus Transfer Rate

- MB/s - Megabytes per second counts the data transferred. The byte count and byte enables are taken into account.
- MTransfers/s - MegaTransfers per second counts the number of data phases regardless of size.

Grant to use for Grant 0 Counter

- External Input 1 - Use Grant signal from External Input 1.
- Vanguard GNT# - Use Grant signal from the slot that the Vanguard is installed in.

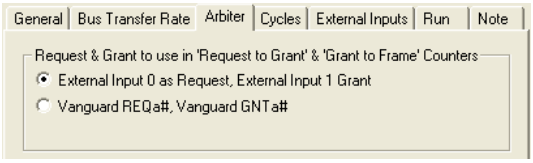
**PMC Only –** Vanguard GNTB# - Use Grant of the optional second PCI agent.

Grant to use for Grant 1 Counter

- External Input 3 - Use Grant signal from External Input 3.
- Vanguard GNT a# - Use Grant signal from the slot that the Vanguard is installed in.

**PMC Only –** Vanguard GNTB# - Use Grant of the optional second PCI agent.

### Arbiter

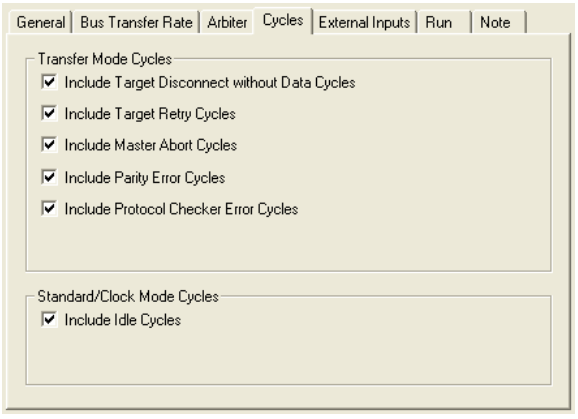



Request & Grant to use in “Request to Grant” and “Grant to Frame” Counters

- External Input 0 as Request, External Input 1 as Grant - Use External Inputs to monitor REQ and GNT signals.
- Vanguard REQ#, Vanguard GNT# - Use Request and Grant signals from the slot that the Vanguard is installed in..

**PMC Only –** Vanguard REQB#, Vanguard GNTB# - Use Request and Grant signals from the optional second PCI agent.

### Transfer Mode Cycles




When using the Transfer Sampling Mode, the following cycles can be included or excluded when the statistics are being generated:

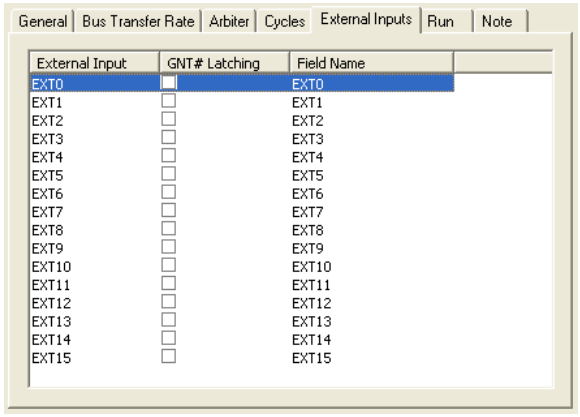
- Include Target Disconnect without Data Cycles
- Include Target Retry Cycles
- Include Master Abort Cycles
- Include Parity Error Cycles
- Include Protocol Checker Error Cycles

### Standard/Clock Mode Cycles

When using Standard or Clock modes, Idle Cycles can be included or excluded when the statistics are being generated:

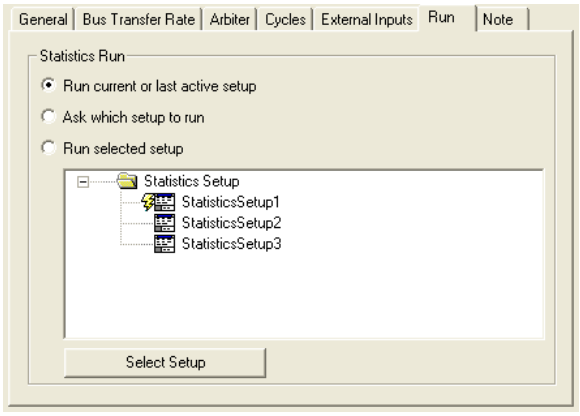
- Include Idle Cycles

### External Inputs



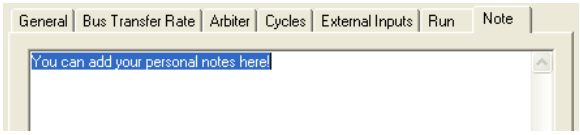

GNT# Latching is assigned to external inputs here. The Field Name can be changed on this menu by double clicking on the Field Name. See “External Inputs” on page 20 for information about the external inputs.

### Statistics Run




- Run current or last active setup - The setup window which is currently active (i.e. the setup which is “on top” in BusView) is run. The run symbol will move to the current setup.
- Ask which setup to run - This option will cause a dialog box to open, asking which Statistics Setup to run.
- Run Selected Setup -The Statistics Setup which has the run symbol next it will be run. This can be changed by clicking on the setup you require and pressing the Select Setup button.

### Note

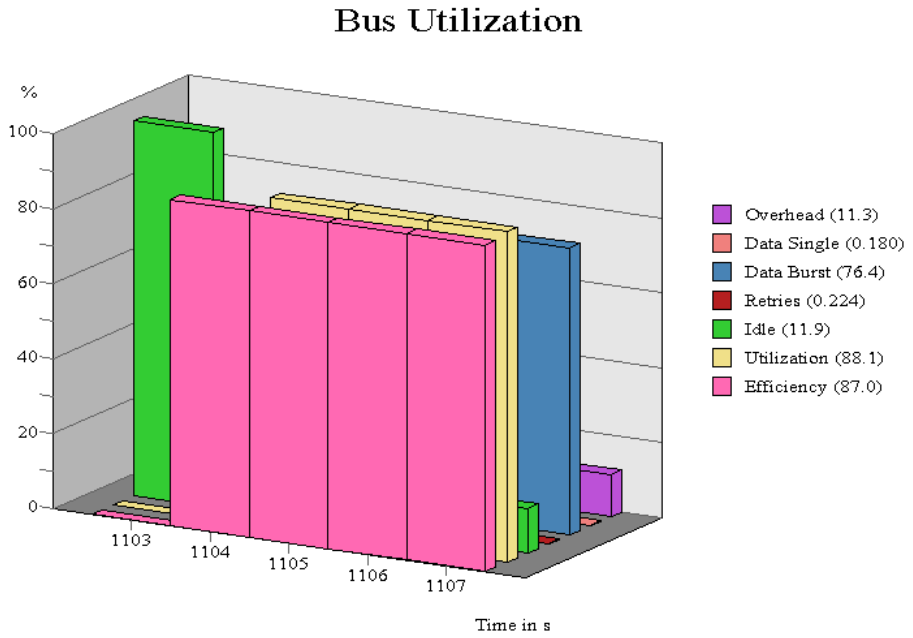



- This are is used for making comments about the Statistics Setup. These notes are saved with the Statistics Setup file.

## *5.2 Pre-defined Statistics*

### Bus Utilization

The Bus Utilization statistic is based on Clock sampling, and displays the following parameters concerning the traffic on the bus:




**Utilization** - Indicates how much the bus is being used and is calculated by dividing the number of Transactions, by the total number of clocks.

**Efficiency** - The Efficiency measures the duration of data transfers versus the duration of transactions, i.e. how efficient the system is transferring data. It is calculated by dividing the Data Total percentage by the Transactions percentage.

Bus Utilization and Efficiency are calculated from the following parameters.

**Overhead** -All cycles which are not; Idle, Data or Retries.

**Data Single** - Measures the duration of single burst data transfers relative to the total time, i.e. how much time is spent transferring single data across the bus. It is calculated by dividing the number of samples with IRDY# AND TRDY# active by the total number of samples.

**DataBurst**- The Data Burst column measure the duration of burst data transfers relative to the total time, i.e. how much time is spent transferring burst data across the bus. It is calculated by dividing the number of samples with `IRDY#` AND `TRDY#` AND `Burst` active, by the total number of samples.

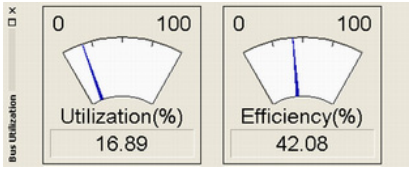
**Retries** - Number of Retry and BusError cycles on the bus relative to the total time.

**Idle** - Number of Idle cycles relative to the total time. It is calculated by dividing the number of samples with IRDY# AND FRAME# inactive.

## Bus Utilization Meter

The Bus Utilization Meter shows real-time Bus Utilization and Efficiency statistics. These can run at all times as an active window on the screen at the same time as the Analyzer, Exerciser, Protocol Checker and other statistics modes. This is an efficient and easy way to instantly determine the status of the link and the traffic pattern.

To open the Bus Utilization Meter, select it from the View menu.

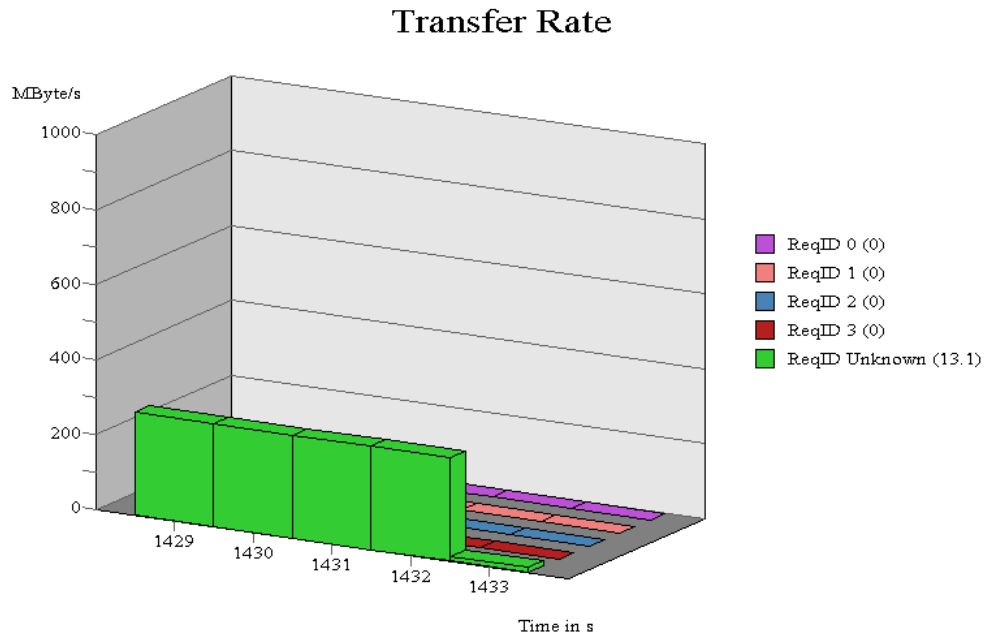



In a PCI system: The *total*-bar will always display the sum of all the `GNT#`s. The `GNT#`0-3 can be activated by connecting the GNT#s from the other modules on the bus, to the external inputs on the Vanguard.See “External Inputs” on page 20. Vanguard GNT#s can be changed via the Statistics Setup Options dialog.

For PCI-X, the Requester ID's (RID) can be changed by selecting the Select Requester IDs command in the Statistics Setup Options dialog. During a Split Completion, it is the Completer ID that is counted. Completer IDs can be counted using the Address Range filter found in the Statistics Counting Masks: section of the Statistics Setup window.

## Transfer Rate

The Transfer Rate statistics calculates the transfer rate in MTransfers/Sec and MB/Sec. Note that the tracer does not collect samples in the period between two traces when the collected data is being processed.




In a PCI system: The *total*-bar will always display the sum of all the GNT#s. The GNT#0-3 can be activated by connecting the GNT#s from the other modules on the bus, to the external inputs on the Vanguard.See “External Inputs” on page 20. Vanguard GNT#s can be changed via the Statistics Setup Options dialog.

For PCI-X, the Requester ID's (RID) can be changed by selecting the Select Requester IDs command in the Statistics Setup Options dialog. During a Split Completion, it is the Completer ID that is counted. Completer IDs can be counted using the Address Range filter found in the Statistics Counting Masks: section of the Statistics Setup window.

It is recommended to run this mode over some time. This will give a more representative average of the systems transfer rates.

## Command Distribution

The Command Distribution statistics is trace driven, i.e., a trace is taken and the occurrences of each PCI/PCI-X command are calculated and displayed.

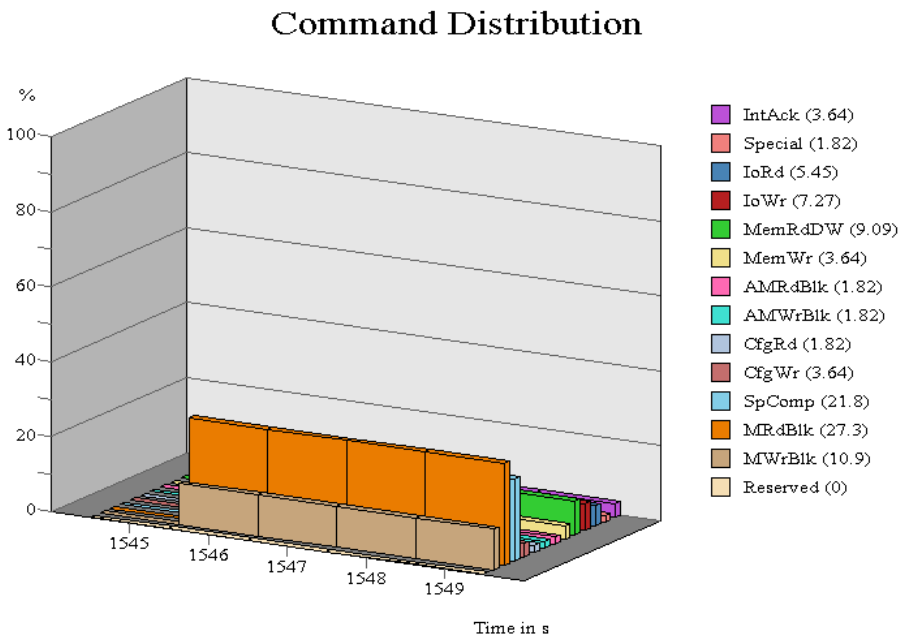



**TABLE 5-1. Command Distribution abbreviations**

| PCI Abbreviation | PCI Command | | PCI-X Abbreviation | PCI-X Command |
|---|---|---|---|---|
| IntAck | Interrupt Acknowledge | | IntAck | Interrupt Acknowledge |
| Special | Special Cycle | | Special | Special Cycle |
| I/ORd | I/O Read | | I/ORd | I/O Read |
| I/OWr | I/O Write | | I/OWr | I/O Write |
| MemRd | Memory Read | | MemRdDW | Memory Read DWORD |
| MemWr | Memory Write | | MemWr | Memory Write |
| ConfWr | Config Write | | AMRdBlk | Alias to Memory Read Block |
| ConfRd | Config Read | | AMWrBlk | Alias to Memory Write Block |
| MWrInv | Memory Write & Invalidate | | ConfRd | Config Read |
| MRdLn | Memory Read Line | | ConfWr | Config Write |
| MRdMul | Memory Read Multiple | | MRdBlk | Memory Read Block |
| | | | MWrBlk | Memory Write Block |
| | | | Reserved | |

## Burst Length Distribution

This groups the Burst Lengths of the Transactions in order to show Burst usage on the system.

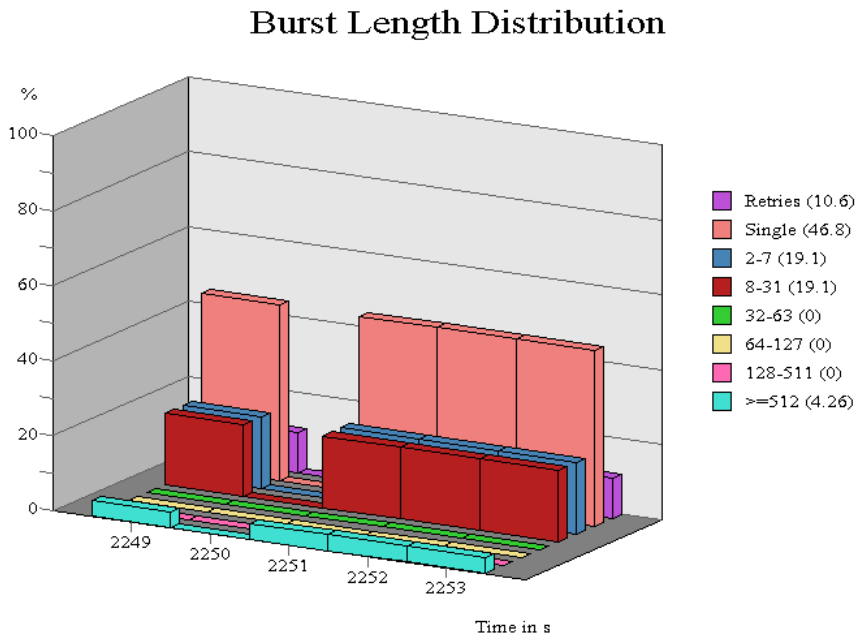



**TABLE 5-2. Burst Length Distribution explanation**

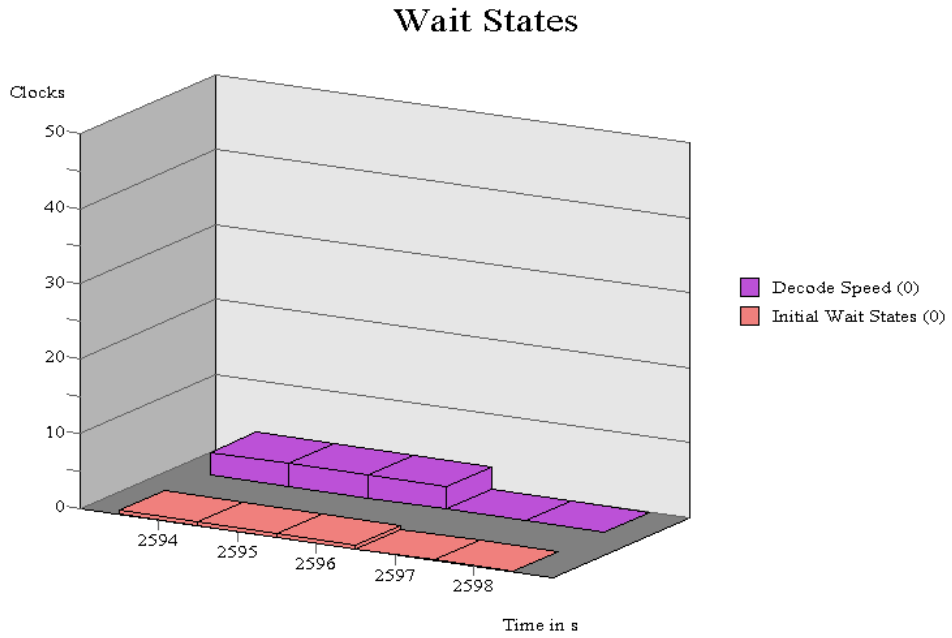
| Abbreviation | PCI Burst length/Termination |
|---|---|
| Retries | Target Retry, i.e. no data transferred |
| Single | Single Cycles, i.e. one data phase |
| 2-7 | Burst length from 2 to 7 |
| 8-31 | Burst length from 8 to 31 |
| 32-63 | Burst length from 32 to 63 |
| 64-127 | Burst length from 64 to 127 |
| 128-511 | Burst length from 128 to 511 |
| >=512 | Burst length longer than or equal to 512 |

## Wait States

Shows the average Wait States and Decode Speed per transaction measured in clocks.

Decode Speed is clocks from FRAME# to DEVSEL#

Wait States is the number of clocks from DEVSEL to IRDY# and TRDY#. In PCI, this also shows number of clocks between data phases.

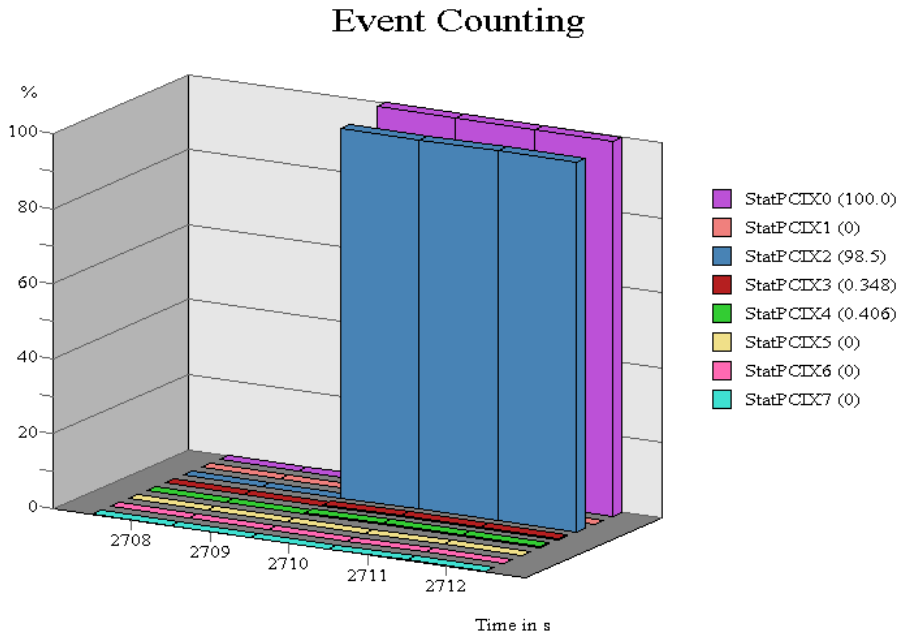


## 5.3 User-defined Statistics

### Event Counting

The Event Counting statistic can be used in all sampling modes. Event Counting is useful for counting the occurrences of different types of cycles , like IO-read, Memory-write, etc

Eight definable hardware counters are available to count the occurrences of eight definable events. Events are edited in the same way as the State Analyzer (See “Editing Event Patterns” on page 69).




**Note –** The eight definable hardware counters are shared with the State Analyzer. If you are currently using counters in the State Analyzer, those counters are unavailable in the Statistics Functions and visa versa.

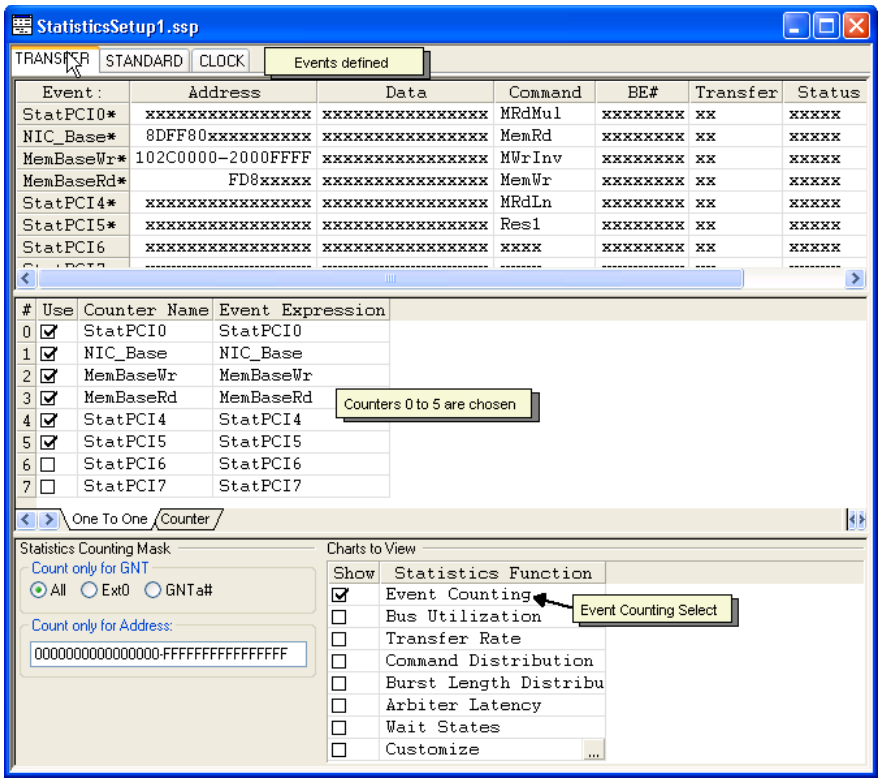



**FIGURE 5-4** *Event Counting Statistics setup*

Select the Events Counting check box in the Charts to View section to turn on event counting. The counters to be used are selected in the same manner. Figure 5-4 shows counters 0 to 5 in use.

### One to One mode

In One to One mode, each counter is represented by one event.

Counter 0 shows that we are using the Event called StatPCI0, which has its event pattern defined as a NRdMul command. Counter 0 will therefore count all instances of the MRdMul command when the statistics are executed.

> **Note –** Events can be renamed, copied, and deleted in the same way as the events in the Analyzer Setup. (“Editing Event Patterns” on page 69).

### Counter mode

Each counter can be associated with more than one event using boolean operators. To open the Event Counter Equation dialog, click on the Edit button ... as shown in Figure 5-5

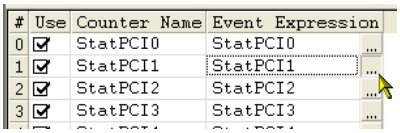



**FIGURE 5-5** *Edit an Event Counter Equation*

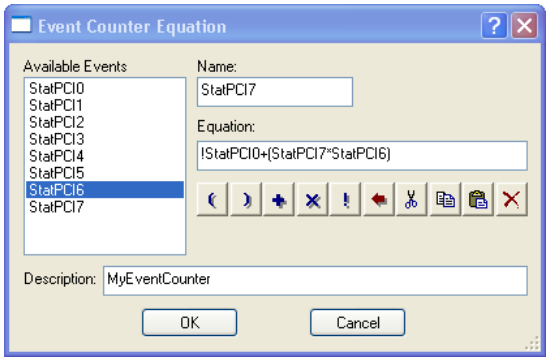



**FIGURE 5-6** *Advanced Statistics Counter*

Use the buttons on the Event Counter Equation dialog to build boolean equations.

### *Boolean Expressions*

Use these buttons to build equations with counters.

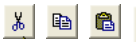
 Use brackets to build more complex expressions.

 Logical OR

 Logical AND

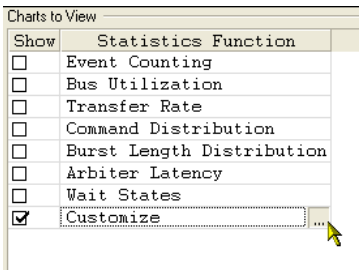
 Logical NOT

 Erase

 Cut, Copy, Paste and Delete.

## Customized Charts

Customized charts can be made using one or more of the hardware counters. (See Table 5-3 on page 118 for a full list of the hardware counters). Click the Customize check box in the Charts to View window to select your customized chart to be run with statistics.



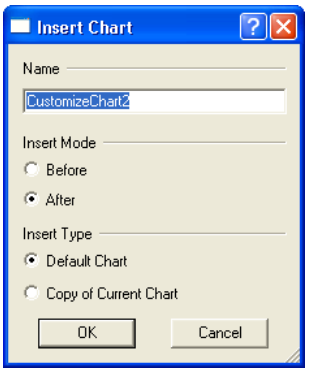

## Adding, Deleting and Renaming Customized Charts.

Customized charts can be added, deleted and renamed by using the right click menu.

### Insert a new chart

**Note –** When inserting a new Customize function using the right click menu, the mouse cursor must be below the Wait States function. It is not possible to insert a customized chart between the pre-defined statistics functions.

To add a new Customized Chart right click anywhere below the Wait States statistics function and click Insert Chart. This will open an Insert Chart dialog box in which you must type a name for the new chart.




There are two options in this dialog:

**Insert Mode**Choose whether you want the new chart to be placed before or after the currently selected chart.

**Insert Type**Choose where you wish the new chart to have default values, or whether the new chart should be a copy of the currently selected chart.

### Delete a chart

To delete a customized chart, right click on the chart you wish to delete and click Delete Chart from the menu that appears.

### Rename a Chart

To rename a chart, right click on the chart you wish to rename and click Rename Chart on the menu that appears. A Rename Chart dialog box will open from which you can rename the chart.

## Editing a Customized Chart

To edit your own chart press the edit button ... This will open the Customize Statistics View as show in Figure 5-7.

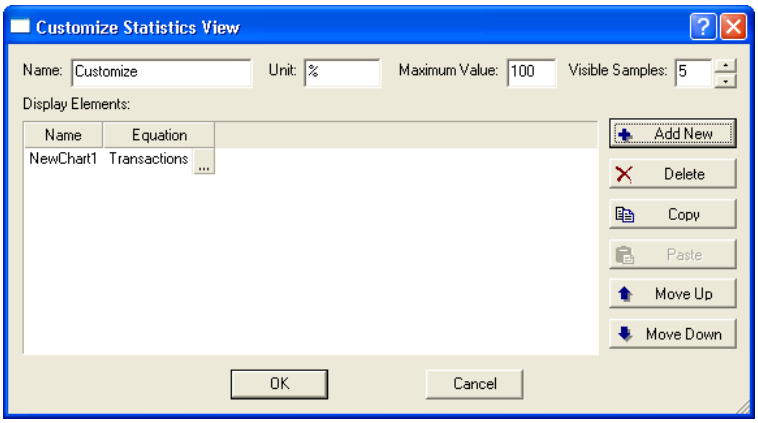



**FIGURE 5-7** *Customize Statistics View*

When the Customize Statistics View dialog is open, click the Add New button to create a new Display Element. By default this display element will be called NewChart1 and will use the Transactions hardware counter.

### Editing a Chart Display Element

To change the equation of the chart display element press the edit button ... This will open the Statistics Display Element dialog shown in Figure 5-8.

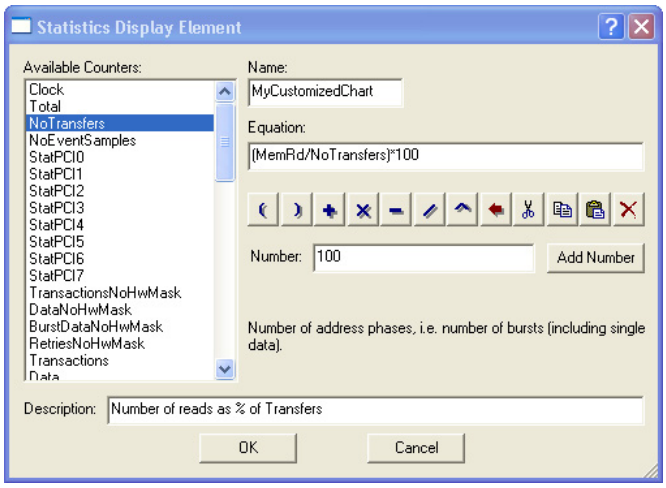



**FIGURE 5-8** *Statistics Display Element dialog*

All of the counters listed can be used by themselves or together using mathematical expressions; making a very flexible and powerful statistics tool.

In the example shown in Figure 5-8 we have renamed the element to MyCustomizedChart and formed the equation:

(MemRd/NoTransfers)*100

This example will display a chart showing the percentage of MemRd cycles per Transfer.

**Building Equations**

Use the following methods to build equations:

- Add a counter to the equation by double clicking on the counter name.
- Add a mathematical expression to the equation by clicking on the relevant button in the dialog, or press the relevant button on your keyboard.
- Add a number to the equation by typing the number into the Number text box and press the Add Number button.

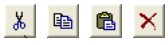
***Button Explanations:***

 Parentheses.

 Addition, Multiplication, Subtraction and Division.

 Carat (..to the power of..).

 Backspace (delete the item immediately to the left of the cursor).

 Cut, Copy, Paste and Delete the items highlighted.

## 5.4 Hardware Counters

Table 5-3 shows all counters available for use in Customized Charts with a short description. The description for each counter is also displayed in the Statistics Display Element dialog box.

**TABLE 5-3. Summary of Hardware Counters**

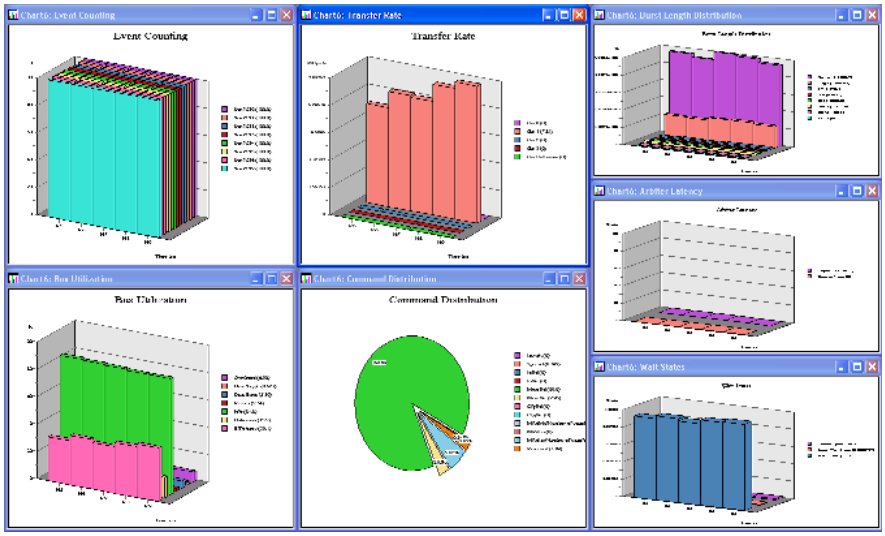
| | |
|---|---|
| Clock | The PCI clock period in pico-seconds (ps) |
| Total | Total number of PCI clocks per sample |
| NoTransfers | Number of address phases, i.e. number of bursts (including single data). |
| NoEventSamples | Number of samples in trace using the selected sampling mode and its options. |
| StatPCI0 | Event expression counter 0. Can be renamed. |
| StatPCI1 | Event expression counter 1. Can be renamed. |
| StatPCI2 | Event expression counter 2. Can be renamed. |
| StatPCI3 | Event expression counter 3. Can be renamed. |
| StatPCI4 | Event expression counter 4. Can be renamed. |
| StatPCI5 | Event expression counter 5. Can be renamed. |
| StatPCI6 | Event expression counter 6. Can be renamed. |
| StatPCI7 | Event expression counter 7. Can be renamed. |
| TransactionsNoHwMask | Number of clocks with transactions, i.e. clocks where FRAME# and/or IRDY# are active.<br>NOTE: This counter ignores the hardware mask. |
| DataNoHwMask | Number of clocks with data, i.e. clocks where IRDY# and TRDY# are active.<br>NOTE: This counter ignores the hardware mask. |
| BurstDataNoHwMask | Number of clocks with burst data, i.e. clocks where IRDY# and TRDY# are active at least twice in the same transfer.<br>NOTE: This counter ignores the hardware mask. |
| RetriesNoHwMask | Number of clocks signaling retry, i.e. clocks where IRDY#=0, DEVSEL#=0, STROP#=0 and TRDY#=1<br>NOTE: This counter ignores the hardware mask. |
| Transactions | Number of clocks with transactions, i.e. clocks where FRAME# and/or IRDY# are active. |
| Data | Number of clocks with data, i.e. clocks where IRDY# and TRDY# are active. |
| DataBurst | Number of clocks with burst data, i.e. clocks where IRDY# and TRDY# are active at least twice in the same transfer. |
| Retry | Number of clocks signaling retry, i.e. clocks where IRDY#=0, DEVSEL#=0, STROP#=0 and TRDY#=1 |
| BusXferRate0 | Transfer rate for GNT 0 (defined in the Statistics Setup Options dialog box).<br>NOTE: This counter ignores the GNT hardware mask. |
| BusXferRate1 | Transfer rate for GNT 1 (defined in the Statistics Setup Options dialog box).<br>NOTE: This counter ignores the GNT hardware mask. |
| BusXferRate2 | Transfer rate for GNT 2 (defined in the Statistics Setup Options dialog box).<br>NOTE: This counter ignores the GNT hardware mask. |
| BusXferRate3 | Transfer rate for GNT 3 (defined in the Statistics Setup Options dialog box).<br>NOTE: This counter ignores the GNT hardware mask. |
| BusXferRateUnknown | Transfer rate for unknown GNTs.<br>NOTE: This counter ignores the GNT hardware mask. |
| BurstLenSingle | Number of single data transfers. |
| BurstLen2_7 | Number of transfers with burst length between 2 and 7 data cycles. |
| BurstLen8_31 | Number of transfers with burst length between 8 and 31 data cycles. |
| BurstLen32_63 | Number of transfers with burst length between 32 and 63 data cycles. |

**TABLE 5-3. Summary of Hardware Counters (Continued)**

| | |
|---|---|
| BurstLen64_127 | Number of transfers with burst length between 64 and 127 data cycles. |
| BurstLen128_511 | Number of transfers with burst length between 128 and 511 data cycles. |
| BurstLenGt512 | Number of transfers with burst length greater than or equal to 512 data cycles. |
| TargetAbort | Number of transfers terminated with Target Abort. |
| TargetDisconWithData | Number of transfers terminated with Disconnect With Data. |
| TargetDisconWithoutData | Number of transfers terminated with Disconnect Without Data. |
| WaitStates | Number of wait states in all transfers. |
| MasterAbort | Number of transfers terminated with Master Abort |
| IntAck | Number of transfers with command IntAck. |
| Special | Number of transfers with command Special. |
| IoRd | Number of transfers with command IoRd. |
| IoWr | Number of transfers with command IoWr. |
| MemRd | Number of transfers with command MemRd. |
| MemWr | Number of transfers with command MemWr. |
| CfgRd | Number of transfers with command CfgRd. |
| CfgWr | Number of transfers with command CfgWr. |
| MRdMul | Number of transfers with command MRdMul. |
| MRdLn | Number of transfers with command MRdLn. |
| MWrInv | Number of transfers with command MWrInv. |
| DecodeSpeed | Number of clocks from FRAME# to DEVSEL# for all transfers. |
| InitialWaitStates | Number of initial wait state clocks for all transfers. |
| ReqToGnt | Number of clocks from REQ# to GNT# .<br>Select which REQ# and GNT# to use from the Statistics Setup Options dialog. |
| NumberReq | Number of times REQ# is asserted.<br>Select which REQ# to use from the Statistics Setup Options dialog. |
| GntToFrame | Number of clocks from GNT# to FRAME# .<br>Select which GNT# to use from the Statistics Setup Options dialog. |
| NumberGnt | Number of times GNT# is asserted.<br>Select which GNT# to use from the Statistics Setup Options dialog. |
| NoBursts | Number of address phases with a complete burst inside clocks sampled.<br>NOTE: This counter ignores the Requester ID hardware mask. |

## 5.5 Statistics Charts

After selecting which statistics function you wish to view and have been run, The charts will open and display the statistics functions in real time.




The style of chart and how it is displayed, can be modified via the Statistics Chart Options dialog. Double clicking and holding down the mouse button inside a chart window will allow you to move the camera view of the chart using the mouse.

**Chart Type**

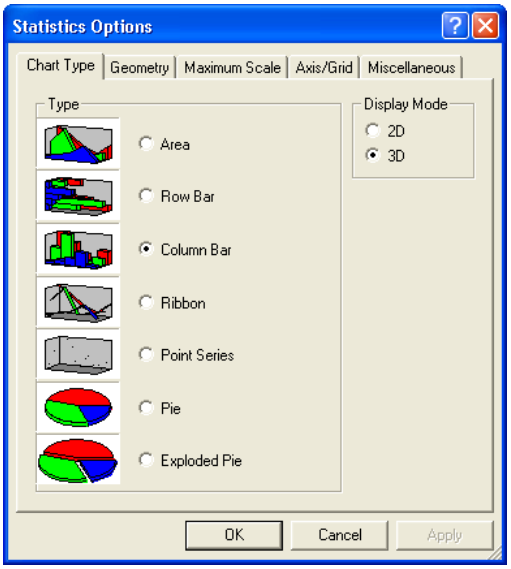
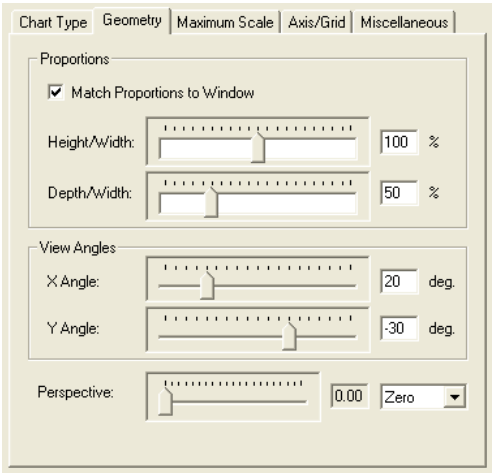



***Chart Type and Display Mode***

Selection of 2D and 3D charts in a variety of types is selected here.

**Geometry**




***Proportions***

- Match Proportions to Window - Chart will resize to fit inside the window
- Height/Width - Slider adjusts the height and width of the chart
- Depth/width - Slider adjusts the depth and width of the chart

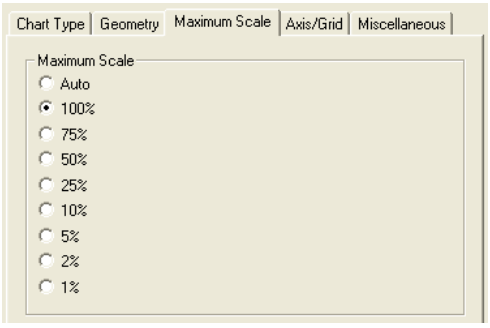
***View Angles***

- X Angle and Y Angle Sliders - Changes the viewing angles of the chart.

***Perspective***

- Adjusts the perspective view of the chart.

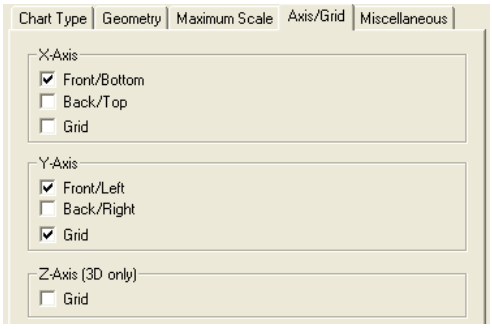
**Maximum Scale**




- Auto - sets the maximum value on the Y axis to the highest visible value in the current set of samples.
- Other setting vary according to the type of statistics being measured, for example; the Event Counting statistics measure percentage, whilst the Transfer Rate statistic measures MegaBytes per second.

**Note –** Although these settings set a maximum value for the Y axis, if a sample exceeds the maximum set value, then this larger value will override this setting for as long as the larger value is displayed on the chart.

**Axis/Grid**




***X Axis***

- Front/Bottom - Select this to show the X axis label on the Front position (for 3D charts), or the Bottom position (for 2D charts).
- Back/Top - Select this to show the X axis label on the Back position (for 3D charts) or the Top position (2D charts).
- Grid - Turn on a grid for the X axis.

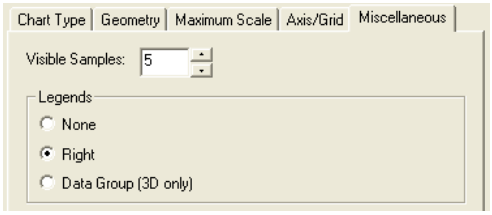
***Y Axis***

- Front/Left - Select this to show the Y axis label on the Front position (for 3D charts), or the Left position (for 2D charts).
- Back/Right - Select this to show the Y axis label on the Back position (for 3D charts) or the Right position (2D charts).
- Grid - Turn on a grid for the Y axis.

***Z Axis***

- Grid - Turn on a grid for the Zaxis.

**Miscellaneous**




***Visible Samples***

- The number of samples to show on a chart at any one time.

***Legends***

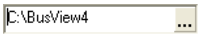
- The position where chart legends should be shown.

## *5.6 Statistics Files*

### Statistics Setup Files

**Open a new Statistics Setup File**

1. On the File menu, click New
2. In the New dialogue box, choose Statistics Setup
3. You can give this file a name by entering a name in the Filename text box. If no name is given, then BusView will assign a name. Default name is StatisticsSetup1.
4. You can choose where this file is saved, by pressing the browse button in the Location text box.




**Load a Statistics Setup File**

1. On the File menu, click Open.
2. In the "Files of type:" list, click "Statistics Setup Files".
3. Browse to the directory in which your Statistics Setup Files have been saved and choose the file you wish to open.

**Save a Statistics Setup File**

The Statistics Setup Window must be open and selected.

1. On the File menu, click Save or press the Save button  . If this is the first time you have saved this Setup, a Save As dialog will open.

   Alternatively, click Save As on the File menu to save the file as a different name than it currently has.
2. Browse to the directory in which you wish to save the Statistics Setup File. The default name is already shown in the File Name text box and can be changed.

### Statistics Setup Options dialog

**Open the Statistics Setup Options dialog**

- With an Statistics Setup window active, click Options on the Tools menu.
- Alternatively, right click anywhere on the Statistics Setup window, and select Option from the menu.

## Statistics Chart File

### Display Statistics Charts

Statistics Charts are displayed by running a Statistics Setup.

### Save a Statistics Chart File

There are three ways in which a statistics chart file can be saved:

- **Statistics Chart Binary File** - saves the chart data in a binary format for use in BusView only. Uses the file extension **.sch**
- **Statistics Chart Ascii File** - saves the chart data in Ascii format. This can be opened and viewed in a text editor such as Notepad. Notepad can be opened from Busview by right-clicking on Notepad in the Workspace Window. Uses the file extension **.sca**
- **Statistics Counter Ascii File** - saves values of the hardware counters used to generate the statistics charts in an Ascii format. This can be opened and viewed in a text editor such as Notepad. Notepad can be opened from Busview by right-clicking on Notepad in the Workspace Window. Uses the file extension **.sca**

To save a statistics chart file:

1. The Statistics Chart Window must be open and selected.
2. On the File menu, click Save or press the Save button  . If this is the first time you have saved this Setup, a Save As dialog will open.

   Alternatively, click Save As on the File menu to save the file as a different name than it currently has.
3. Browse to the directory in which you wish to save the Statistics Chart File. The default name is already shown in the File Name text box and can be changed.

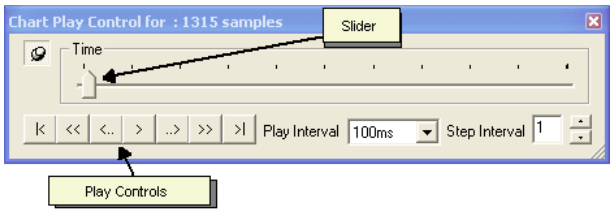
When a Statistics File is saved; the value of every hardware counter on each sample is recorded. This means that when you re-open a saved Statistics File you can display any of the Pre-defined Statistics.

### Load a pre-recorded Statistics Chart File

1. Click Open in the File menu, or right click on the Statistics Chart folder in the Workspace window and select Open.
2. Browse to the directory in which your Statistics Chart Files have been saved and choose the file you wish to open.

### Statistics Playback Control

When opening a saved Statistics Chart file, a playback dialog box will also open. If it does not, it can be opened by clicking Chart Play Control from the View menu.




The title displays the total number of samples available in the file and the slider is used to move through the recorded statistics file. The Play Control buttons have the following meaning:

|< / >| Go to the Beginning / End of the Statistics File.

<< / >> Step back / forward through the Statistics File by the number of samples stated in the Step Interval box.

<.. / ..> Step back / forward through the Statistics file by one sample.

> Play the Statistics File.

---

**Note –** A saved Statistics File contains a record of the value of every hardware counter on each sample.

---

# 6 Protocol Checker

The Protocol Checker is used to test for violations to the PCI/PCI-X specification. This chapter explains the use of the Protocol Checker, and describes each violation in detail.

- Introduction
- Operation
- PCI Protocol Violations
- PCI-X protocol Violations

Use of the Protocol Checker requires the “VG-P” license to have been purchased. See “Ordering Information” on page 427.

## 6.1 Introduction

The Vanguard Protocol Checker is a parallel trigger module which recognizes violations of the PCI/PCI-X bus specification in real time. If one of a specified set of violations is detected, a trigger signal is generated. This signal can be used to trigger the Analyzer and/or external devices (i.e. oscilloscopes) through the trigger out signal.

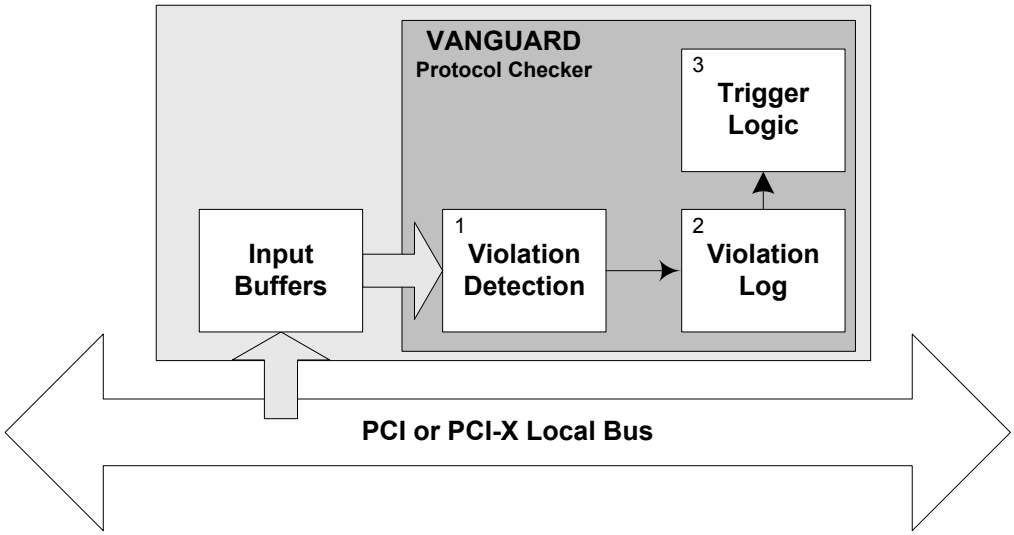



**FIGURE 6-1** *The main blocks of the VANGUARD Protocol Checker.*

Figure 6-1, shows the three main stages of testing for protocol violations.

1. **Violation Detection**
   This stage looks for errors by examining every bus cycle. It has a set of rule-based trigger elements that continuously and simultaneously screen the bus lines to detect protocol violations.
   The violation detection stage also automatically recognizes “instability” (i.e. changes on a line) on all address/data lines and main control signals, without the need to be told the correct state of these lines beforehand. This is essential for concurrently screening for all address and data stability violations in the bus cycles.
   This allows the Protocol Checker to find extraneous transitions on bus signals due to meta-stability, bus ringing and noise.

2. **Violation Log**
   Violations are logged in the Status Window and Protocol Checker window.

3. **Trigger Logic**
   On each occurrence of a protocol violation, a fast trigger output(within 2 to 4 clock pulses) can be generated. The Protocol Checker can be configured to generate this trigger signal on the first occurrence or on every occurrence of the specified violation.
   This trigger signal can be used in Event Patterns to trigger the Analyzer, or be incorporated in counters for the Statistics Functions.
   External test equipment can make use of this trigger signal via the Trigger Output.

The protocol checker can also be configured to ignore “innocent” protocol violations, i.e. violations that occur sporadically and generally do not restrict the overall system function or performance. Checking for specific violations can be enabled/disabled by double clicking the simulated LEDs in the protocol checker window.

Each time a protocol violation occurs, a fast trigger output(within 2 to 4 clock pulses depending on the violation) can be generated. The protocol checker can be configured to generate this trigger signal on the first occurrence, or on every occurrence of the specified violation. This trigger signal can give a direct lead to timing errors and other problems that may exist on the bus.

The difficult part of debugging is usually determining which conditions to trigger on. Often, the symptoms of failure give no clue as to their cause. Since all of the Vanguard tools can be used simultaneously, the possibility of triggering the Analyzer as result of a protocol violation significantly enhances the versatility of the Vanguard.

## Verification of Detected violations

The Vanguard Protocol Checker cannot be used as a complete “Definitive Bus Compliance Validator”, since it does not check for all possible bus specification violations.

---

**Note –** While the Vanguard Protocol Checker, when properly used, is believed to be free of any deficiencies which could indicate false errors, an error indication should never, by itself, be used to implicate a vendor. All reported errors should be verified by inspecting the bus activity causing the error. Only on the basis of such subsequent confirmation of the existence of errors should any vendor be presumed to be at fault.

---

### Invisible Violations

The situation may occur that a violation is detected but investigation of the bus traffic does not appear to show anything wrong. This can be quite frustrating, but there is usually a subtle problem which warrants further investigation. Viewing the wrong bus cycles is the largest cause of being unable to identify a violation.

To minimize this possibility, always display the Protocol Checker output signal along with the relevant bus signals. The erroneous bus signals will be easier to spot, as they will be very closely aligned in time with the high-to-low transition of the trigger.

Generally speaking, the violation will be seen from between 2 to 4 clocks cycles before the assertion of the Protocol Checker output signal.

## 6.2 Features

**PCI Protocol Checker Features**

- All local bus traffic is continually monitored and bus protocol violations are automatically detected.
    - Detects 45 PCI and 71 PCI-XProtocol Violations and 3 Warnings.
    - Compliant to PCI 2.3 and PCI-X 1.0a.
- Can generate signals to trigger the Vanguard Analyzer or an external logic analyzer.
- Any of the tested protocol violations can be ignored by the protocol checker by de-selecting the specific violation in BusView.

**Problems the PCI Protocol Checker can help uncover:**

- Improper design of PCI/PCI-X bus interface circuitry.
- Bus noise, including cross-talk, ringing and ground-bounce, leading to setup time violations.
- Two masters "fighting on the bus".
- Two targets responding to the same address.
- Invalid signal sequences and combinations.

Use of the Protocol Checker requires the “VG-P” license to have been purchased. See “Ordering Information” on page 427.

## 6.3 Operation

The Protocol Checker Window is opened using one of the following methods:

- From the menu bar (File, New).
- Using the Workspace Window.
- By pressing the *run protocol checker* menu tool bar button, as shown in Figure 6-2.


**FIGURE 6-2** *The Run Protocol Checker menu item at the menu bar*

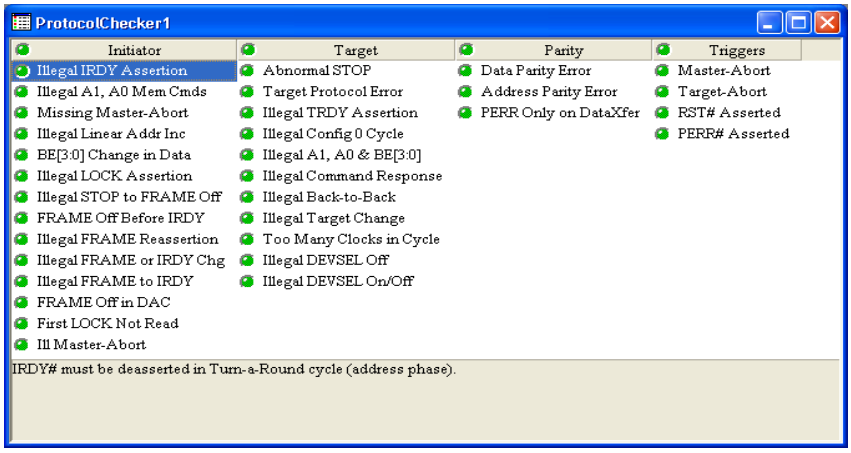



**FIGURE 6-3** *Protocol violations in "Grouped Grid" view and "detailed description"*

The layout of the Protocol Checker window can vary depending on how you wish to view the information.

This window can be viewed in different ways.

- Classic
- Grouped Grid
- Tree

The Protocol Checker window allows you to select any set of violations to investigate. Only those violations that have been selected can cause the trigger output signal to activate. The following color scheme is used:

- Green indicates that the corresponding violation is "on" and will be tested for.
- Dark green indicates that this violation has been turned "off" and will not be tested for.
- Red indicates that this violation has been detected.
- Yellow (tree view) indicates that one or more violations have been detected in this branch.

When using the Protocol Checker to trigger the Analyzer, it is important to start the Analyzer before starting the Protocol Checker. If a violation has already been detected and a trigger signal present when the Protocol Checker is enabled, the Analyzer will store trace data from the moment the Protocol Checker is started. This means that the trace data will not show the violation that caused the trigger.

Alternatively the “Clear on Trace Run” option can be selected from the Protocol Checker Options. This will cause the Protocol Checker history to be cleared synchronously with trace run.

## Protocol Checker Options

### Operating Modes

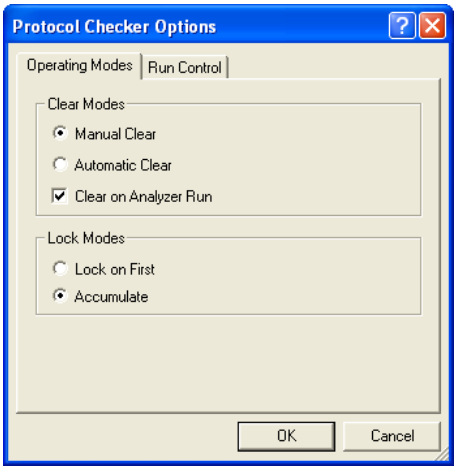



**FIGURE 6-4** *Protocol Checker Options*

#### *Clear Modes*

- **Automatic Clear mode:** The Protocol Checker output trigger signal will automatically reset one clock cycle after a violation was detected. This option is useful for triggering an oscilloscope, etc. No violations will be displayed.
- **Manual Clear mode:** The trigger signal will remain active until reset using the Clear command in the Violations menu.
- **Clear on Analyzer Run:** The Protocol Checker output trigger is cleared each time the Analyzer is run.

#### *Lock Modes*

- **Lock On First mode:** The Protocol Checker will trigger only on the first violation and ignore all later ones. Normally only one single violation can be triggered in this mode. However, several violations may show up if detected at the same time.
- **Accumulate mode:** The protocol checker will accumulate violations as they occur on the bus.

### Run Control

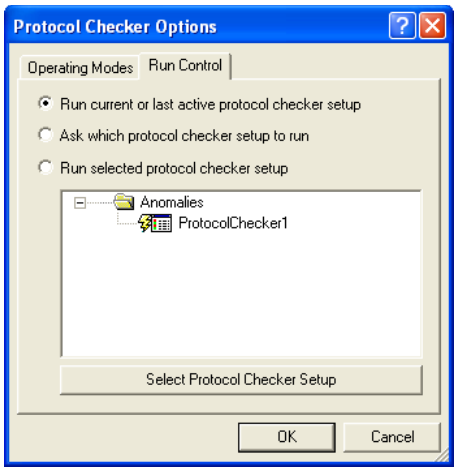


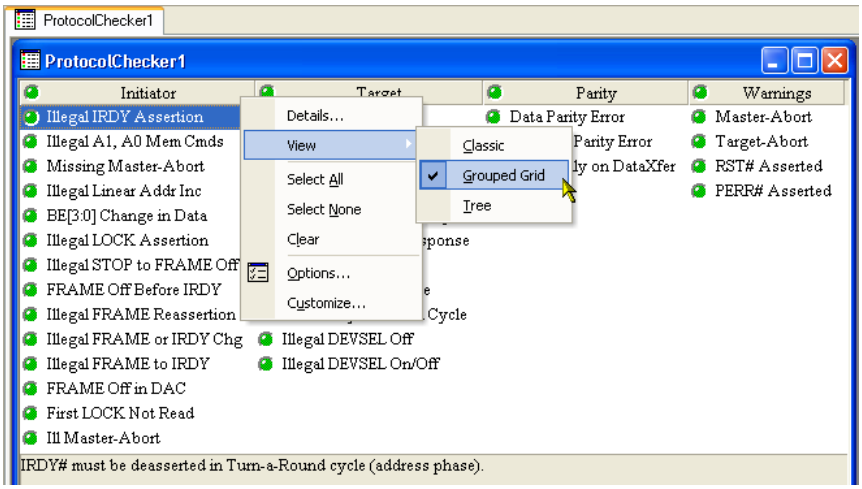

**FIGURE 6-5** *Protocol Checker Run Control*

- Run current or last active setup - The setup window which is currently active (i.e. the setup which is “on top” in BusView) is run. The run symbol will move to the current setup.
- Ask which setup to run - This option will cause a dialog box to open, asking which Statistics Setup to run.
- Run Selected Setup -The Protocol Setup which has the run symbol next it will be run. This can be changed by clicking on the setup you require and pressing the Select Setup button.

## Using the Protocol Checker

All control of the Protocol Checker is made via the Protocol Checker window, menu bar, tool bar, the status line at the bottom of the window, or with shortcut keys.

**FIGURE 6-6** *Protocol Checker right-click menu*




---

**Note –** When screening the PCI/PCI-X bus for violations, it may be helpful to mask some violations that are not important in the working context, so that the Protocol Checker will not trigger on them.

---

**Details** Give a brief description of the selected violation.

**View** Changes the view mode of the list of violations.

- Classic View - Violations are listed as they have been in previous versions of BusView.
- Grouped Grid View - Violations are categorized into the groups: Initiator, Target, Parity and Warnings.
- Tree View - Violations are grouped in the same way as in Grouped Grid view, but are listed in tree format.

**Select All**The Select All command enables all the possible violations.

**Select None**The Select None command disables all possible violations.

**Clear** Resets all triggered violations.

## Protocol Checker Output Signal

The trigger signal PCHKTrg is visible in the trace display.

When the Protocol Checker is used together with the Analyzer (e.g. to identify which cycles caused a protocol violation), the Protocol Checker output signal is inserted into the Event Patterns window. This is done by clicking on the signal you want the trigger output to be in front of, and then pressing Insert on you keyboard to insert signals. Select the signal name PCHKTrg (See “Manipulating Field Columns” on page 70).

The PCHKTrg signal field can have the following values:

**TABLE 6-1. PCHKTrg values**

| Symbol | Value | Explanation |
|---|---|---|
| Err | 1 | PCI/PCI-X bus violation |
| - | 0 | No violation |
| x | x | Don’t care |

The PCHKTrg signal will trigger where at least one protocol violation has occurred in a packet.

## Trigger External Instruments on Violations

An oscilloscope or external logic analyzer can be triggered by the Protocol Checker in different ways. To trigger an external instrument, use the external output pin “Trigger Output” on the front panel of the Vanguard.

This is done by using the trigger output command in the Utilities menu.

**Note –** The Protocol Checker output signal will not be asserted until 2-4 PCI/PCI-X clocks after the actual violation is detected on the bus.

## 6.4 PCI Protocol Violations

### PCI protocol violations

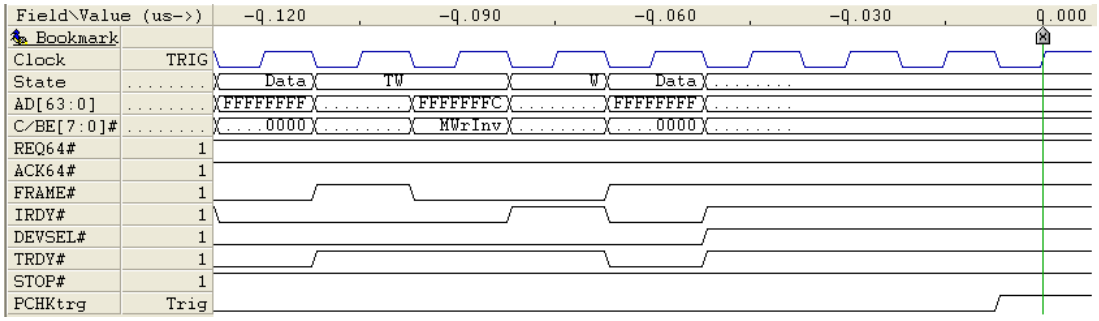
The following classes of PCI bus specification violations are detected.

Target abort, Master abort, PCI bus RST# asserted, and PCI bus PERR# asserted are also detected.

**Summary of detected PCI protocol violations**

## Illegal FRAME# Re-Assertion

Once FRAME# is deasserted, it cannot be reasserted during the same transaction. A transaction is not finished until both FRAME# and IRDY# is deasserted.




PCI Spec. 2.1, page 245 - PCI Spec. 2.2, page 251

Rule 8b, and

PCI Spec. 2.1, page 40 - PCI Spec. 2.2, page 49

Section 3.3.3.1

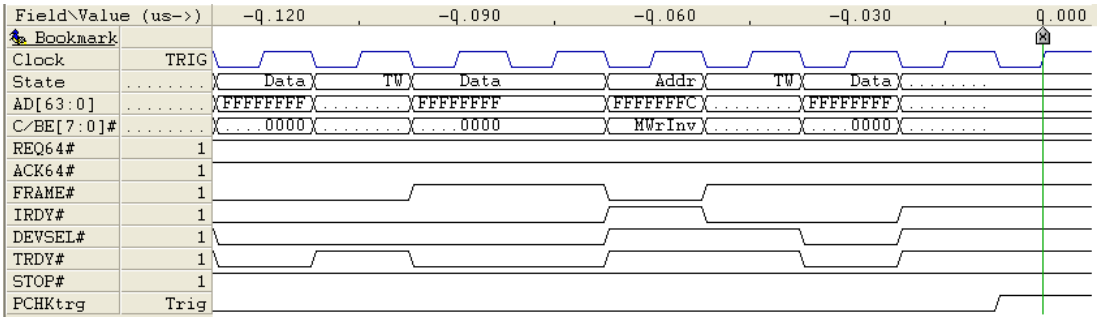
## Illegal FRAME# or IRDY# Change

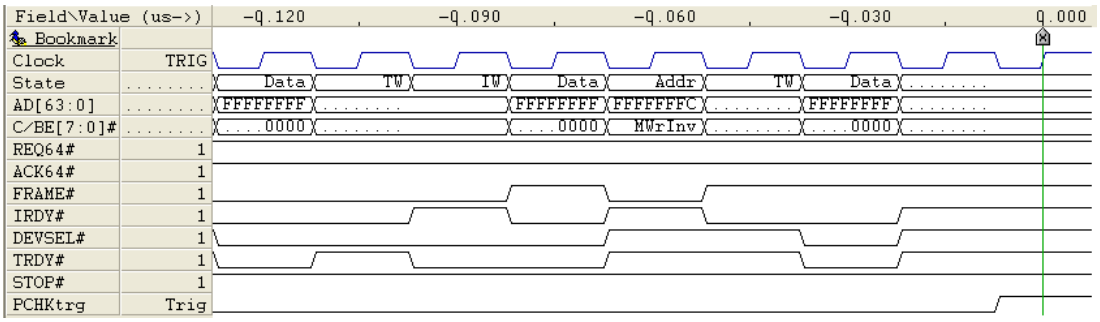
Once a master has asserted IRDY#, it cannot change FRAME# or withdraw IRDY# until current data phase completes.

A data phase is completed on every rising clock edge where both IRDY# and TRDY# are asserted.

In the first figure below, the target inserts a wait state, and thus the data phase does not complete until the clock cycle where both IRDY# and TRDY# are asserted again. The Protocol Checker flags an error because FRAME# is deasserted before that happens.

In the second figure, the target inserts a wait state ,but the master de-asserts IRDY# before the target completes the data phase with TRDY#.




PCI Spec. 2.1, page 246 - PCI Spec. 2.2, page 252

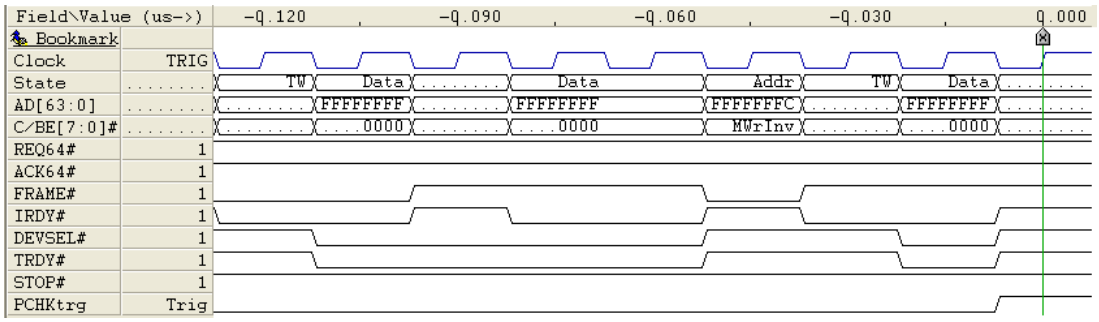
Rule 8d, and

PCI Spec. 2.1, page 26 - PCI Spec. 2.2, page 27

Section 3.2.1

## FRAME# off Before IRDY#

FRAME# can not be deasserted before IRDY# is asserted.




PCI Spec. 2.1, page 246 - PCI Spec. 2.2, page 252

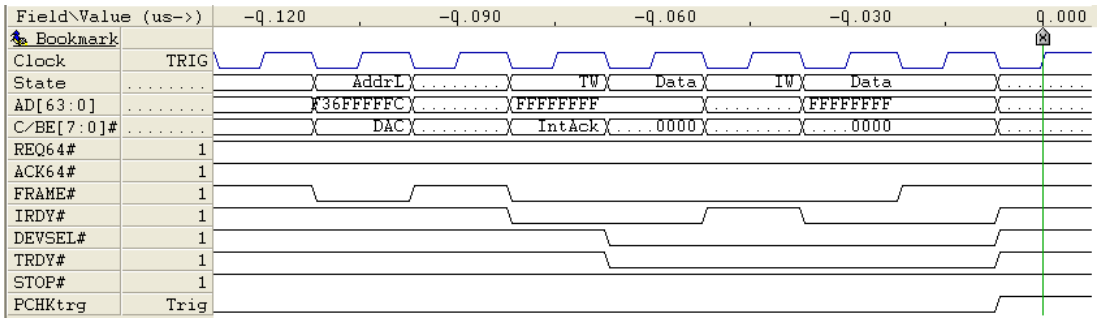
Rule 8c, and

PCI Spec. 2.1, page 37 - PCI Spec. 2.2, page 47

Section 3.3.1 and Section 3.3.2

## FRAME# off in DAC

FRAME# must remain asserted for the cycle following a Dual Address Command (DAC).




PCI Spec. 2.1, page 112

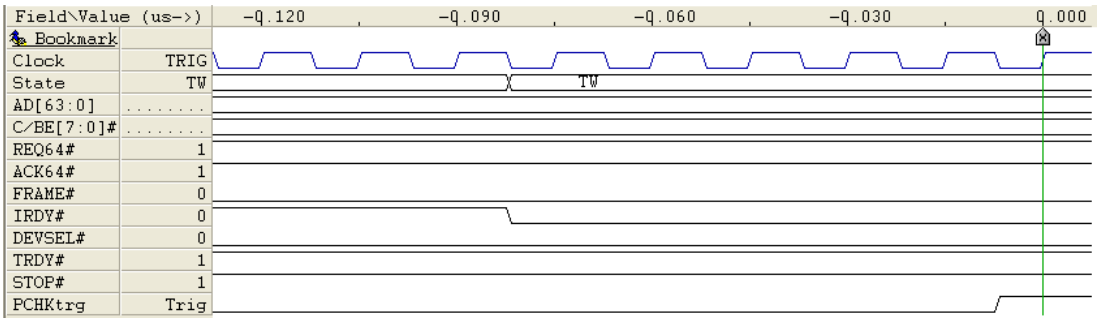
Section 3.10.1

PCI Spec. 2.2, page 106

Section 3.9

## Illegal FRAME# to IRDY#

IRDY# Timing Violation, IRDY# must be asserted within 8 clocks of FRAME#.




PCI Spec. 2.1, page 66

Section 3.5.3.1

PCI Spec. 2.2, page 80

Section 3.5.4.1

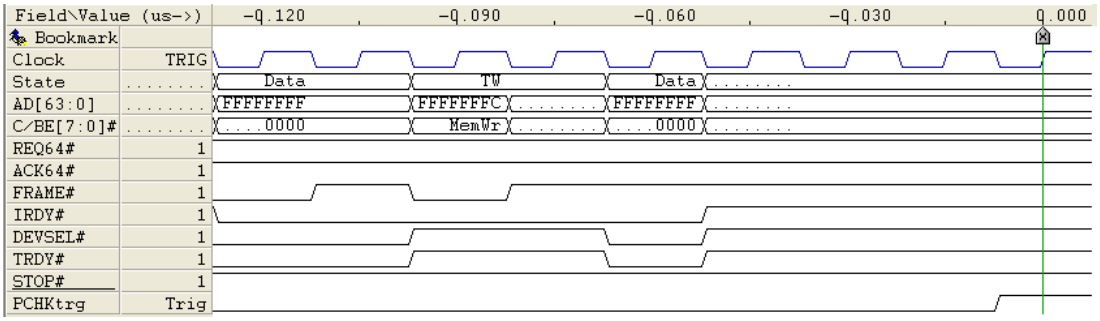
and

PCI Spec. 2.1, page 248 - PCI Spec. 2.2, page 255

Rule 27

## Illegal IRDY# Assertion

IRDY# must be deasserted in turn-around cycle (address phase).




PCI Spec. 2.1, page 30 - PCI Spec. 2.2, page 39

Section 3.2.4

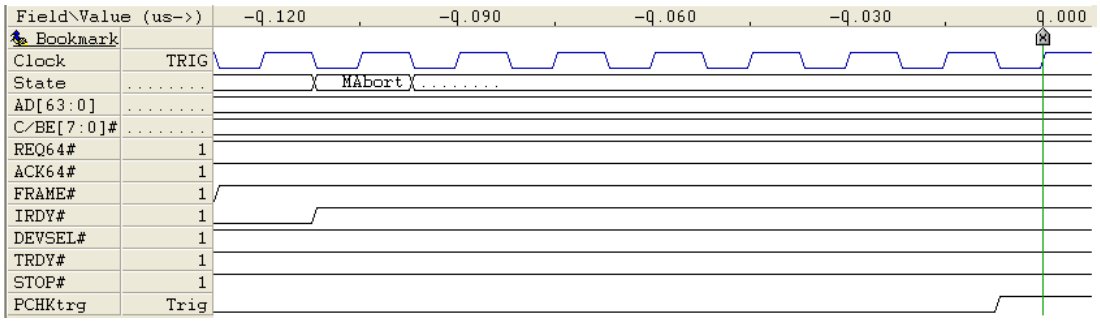
## Illegal LOCK# Assertion

LOCK# must be asserted the clock following the address phase and kept asserted to maintain control.

LOCK# must be released whenever an access is terminated by Target-Abort or Master-Abort.

Once lock is established, the target remains locked until both FRAME# and LOCK# are sampled deasserted or the target signals Target-Abort.

In the figure below, LOCK# is released too late.




PCI Spec. 2.1, page 76

Section 3.6

PCI Spec. 2.1, page 245

Rule 32e, 32g, 31b

PCI Spec. 2.2, page 279

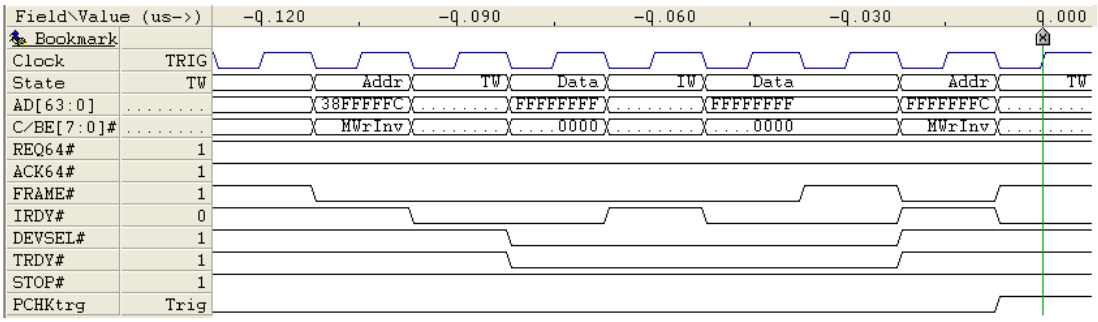
Appendix F.1

Master Rule 3 and 5

Target Rule 2

## First LOCK# not Read

The first transaction of a LOCK# operation must be a read transaction.




PCI Spec. 2.1, page 76

Section 3.6

PCI Spec. 2.1, page 249

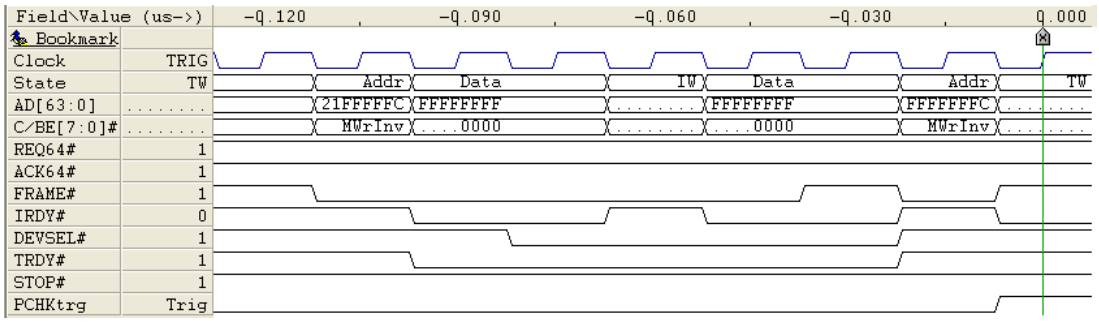
Rule 32d

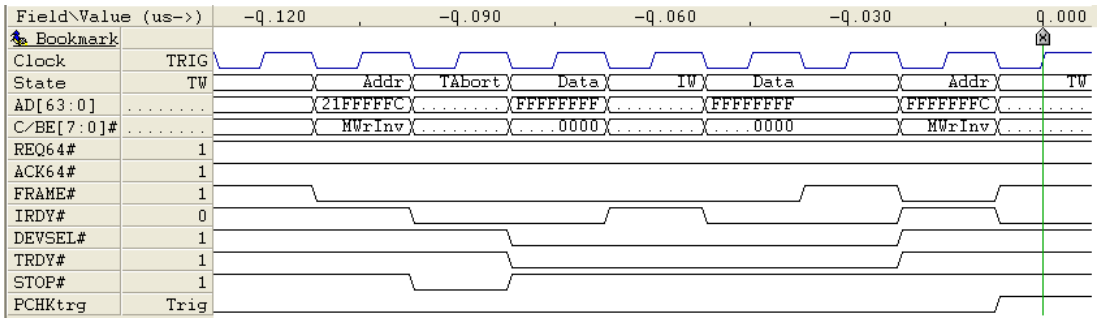
PCI Spec. 2.2, page 281

Appendix F.2

## Target Protocol Errors

- DEVSEL# cannot be deasserted while STOP# is asserted.
- TRDY# can not be active on Target-Abort.
- Target must issue DEVSEL# before any other response.




PCI Spec. 2.1, page 42 - PCI Spec. 2.2, page 53

Section 3.3.3.2.1

PCI Spec. 2.1, page 47 - PCI Spec. 2.2, page 59

Section 3.3.3.2.1

PCI Spec. 2.1, page 80

Section 3.7.1

PCI Spec. 2.1, page 88

Section 3.6.1

PCI Spec. 2.1, page 249

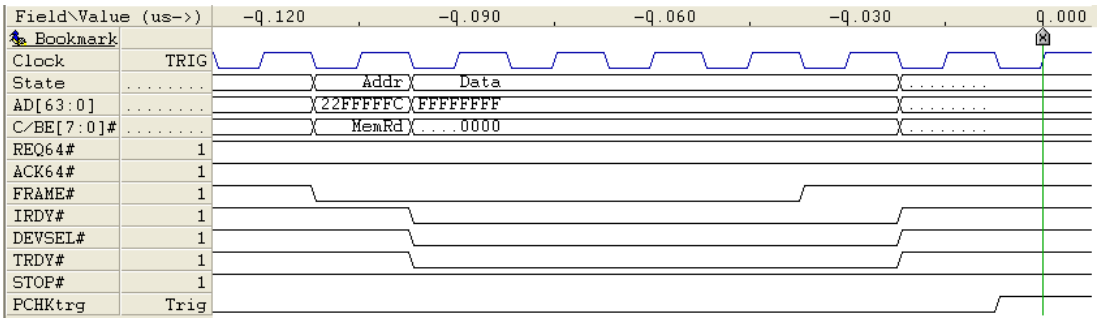
Rule 29

PCI Spec. 2.2, page 255

Rule 34

## Illegal TRDY# Assertion

The first data phase on a Read transaction requires a turnaround cycle with TRDY# deasserted.



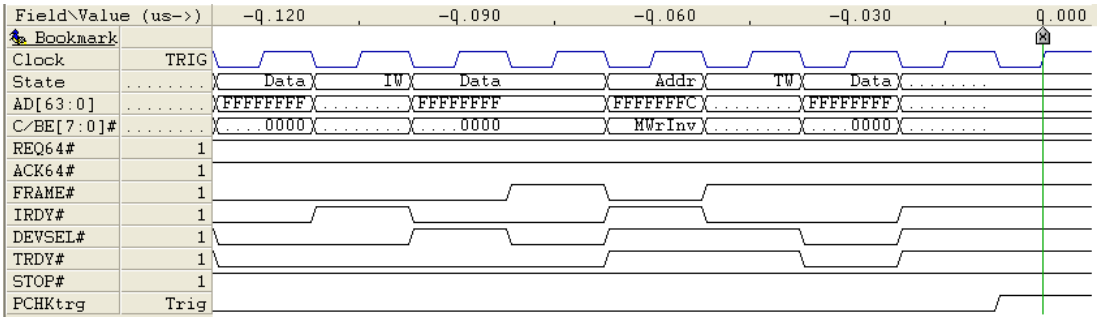

PCI Spec. 2.1, page 36 - PCI Spec. 2.2, page 47

Section 3.3.1

## Illegal Target Change

Once a TARGET has asserted TRDY# or STOP#, it cannot change DEVSEL#, TRDY#, or STOP# until the current data phase completes.

The current data phase is not completed until both TRDY# and IRDY# are sampled asserted, and thus the target cannot de-assert DEVSEL# the way it is shown below .

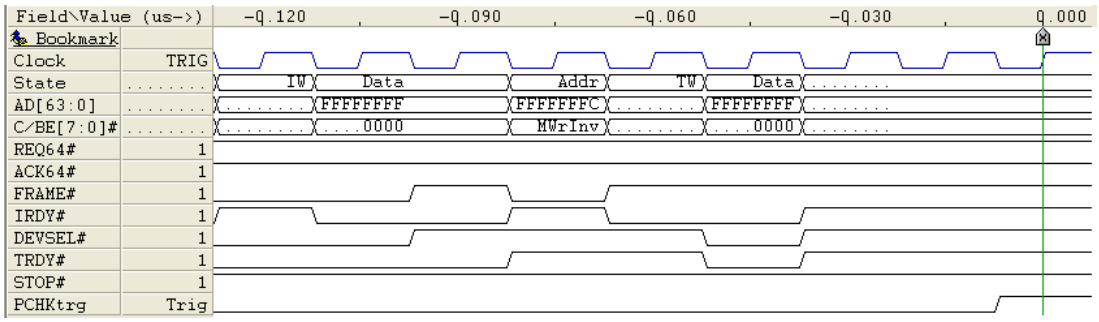



PCI Spec. 2.1, page 247 - PCI Spec. 2.2, page 253

Rule 12d

## Illegal DEVSEL# off

Once a target has asserted DEVSEL#, it must not be deasserted until the last data phase has completed.

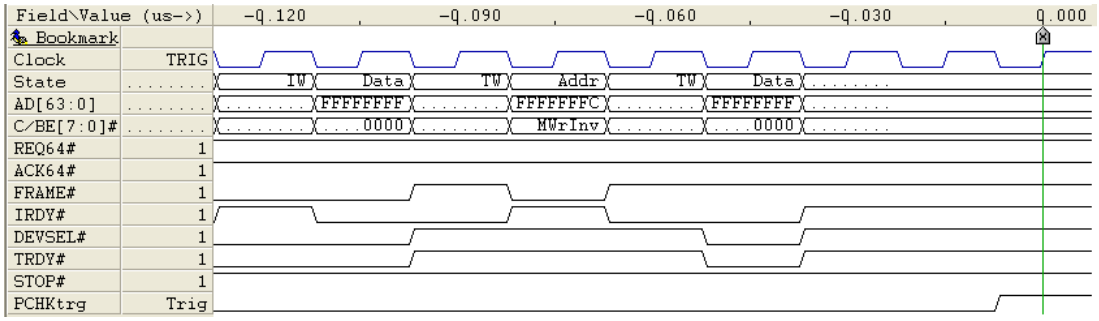



PCI Spec. 2.1, page 247 - PCI Spec. 2.2, page 253

Rule 15

## Illegal DEVSEL# on/off

DEVSEL# should not be asserted in the address phase or when the bus is idle. DEVSEL# should not be deasserted before the transaction is completed, unless it is a Target-Abort termination.




PCI Spec. 2.1, page 249

Rule 33, 35

PCI Spec. 2.2, page 255

Rule 28, 30

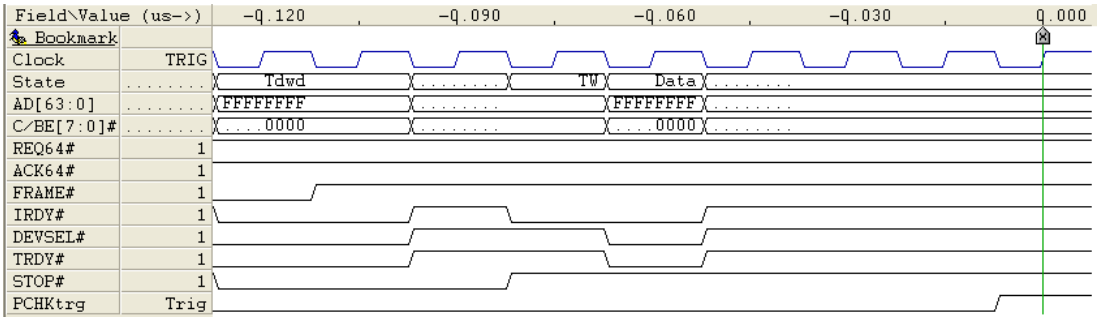
## Abnormal STOP#

Abnormal stops can be divided into three points:

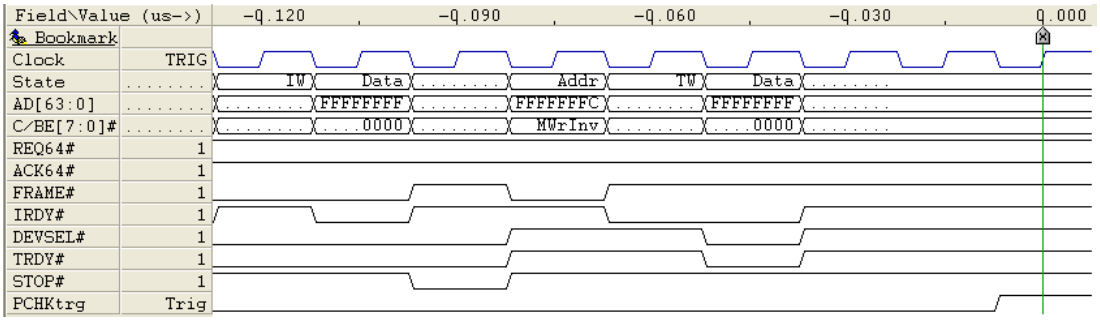
- STOP# fails to be deasserted the cycle immediately after FRAME# is sampled deasserted.
- When STOP# is low, FRAME# and IRDY# cannot both be high.
- Once STOP# is asserted, it must remain asserted until FRAME# is deasserted.

The figures below violates the two first rules.

STOP# is deasserted too late




Illegal FRAME# and IRDY# with STOP#



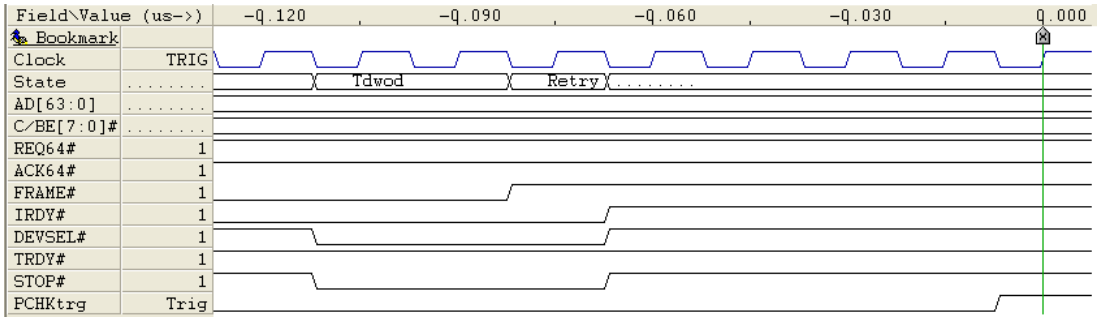

PCI Spec. 2.1, page 247 - PCI Spec. 2.2, page 253

Rule 12c, 12f

## Illegal STOP# to FRAME# off

When STOP# is asserted with FRAME# active, the Master must de-assert FRAME# as soon as it can drive IRDY#.

In the figure below, the master drives IRDY#, but FRAME# remains asserted.




PCI Spec. 2.1, page 247 - PCI Spec. 2.2, page 253

Rule 12e

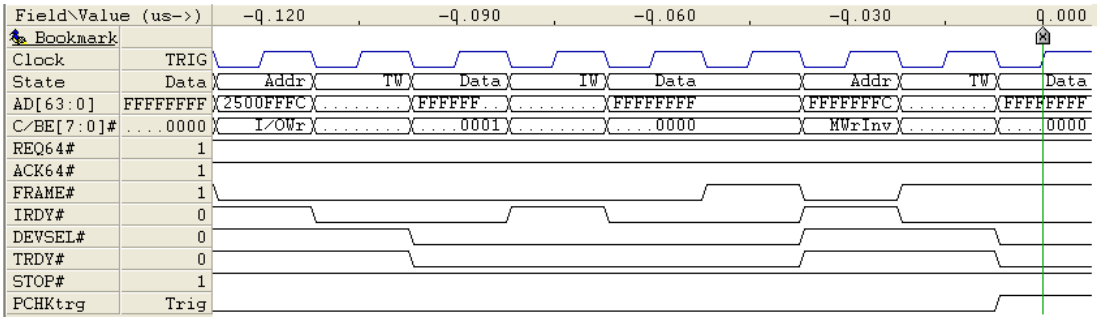
## Illegal A1, A0 & BE[3:0]#

A target must abort I/O transfers with illegal combination of AD1, AD0 and corresponding C/BE# for I/O cycle.

The legal combinations are shown in the table below.

| AD1 | AD0 | C/BE[3]# | C/BE[2]# | C/BE[1]# | C/BE[0]# |
|---|---|---|---|---|---|
| 0 | 0 | x | x | x | 0 |
| 0 | 1 | x | x | 0 | 1 |
| 1 | 0 | x | x | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |

C/BE# is 0x0001 and AD[1:0]=00, which is not a valid combination




PCI Spec. 2.1, page 27

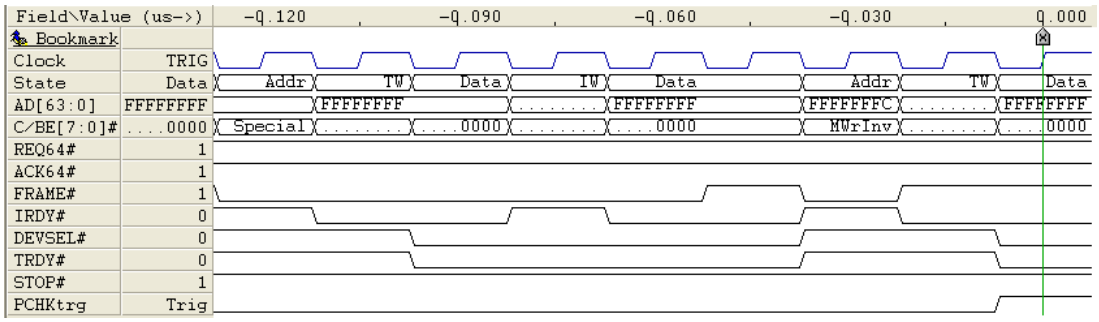
Section 3.2.2

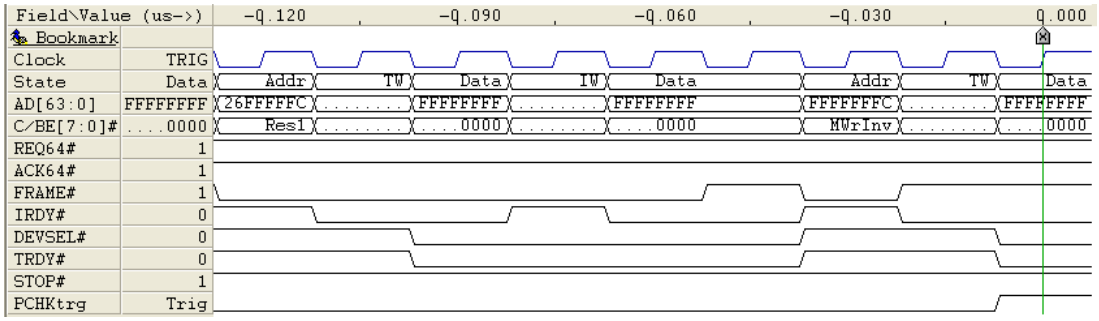
PCI Spec. 2.2, page 28

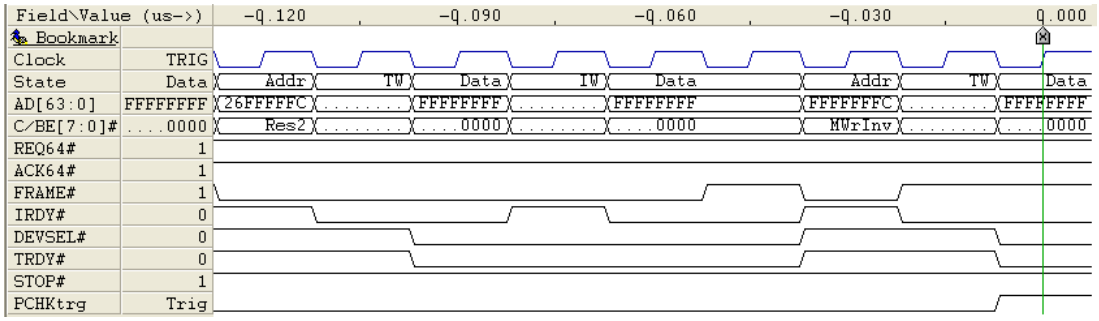
Section 3.2.2.1

## Illegal Command Response

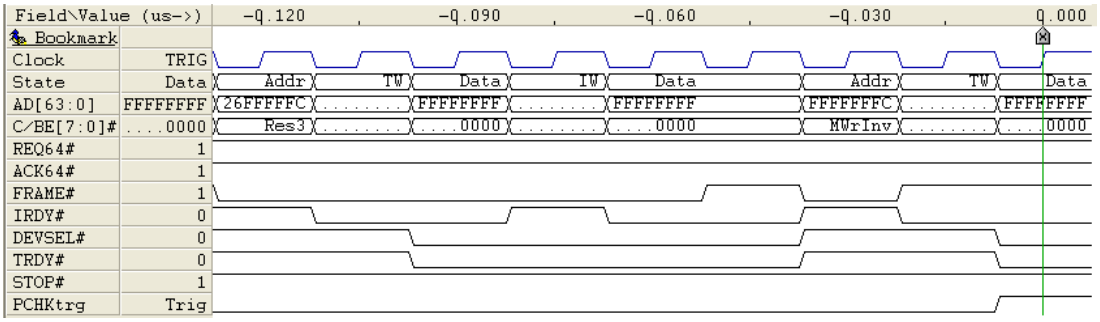
Targets must not respond to Reserved (0100, 0101, 1000 or 1001) commands or the Special Cycle (0001) command.

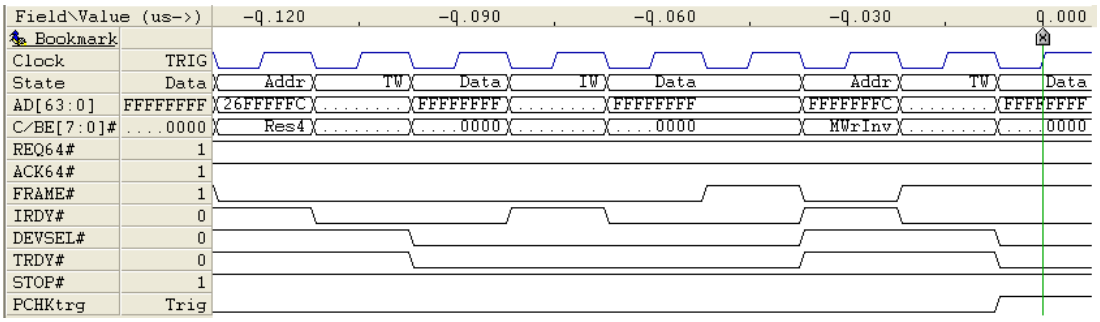


## Illegal Command Response (Continued)




PCI Spec. 2.1, page 22 - PCI Spec. 2.2, page 22

Section 3.1.1

PCI Spec. 2.1, page 81

Section 3.7.2

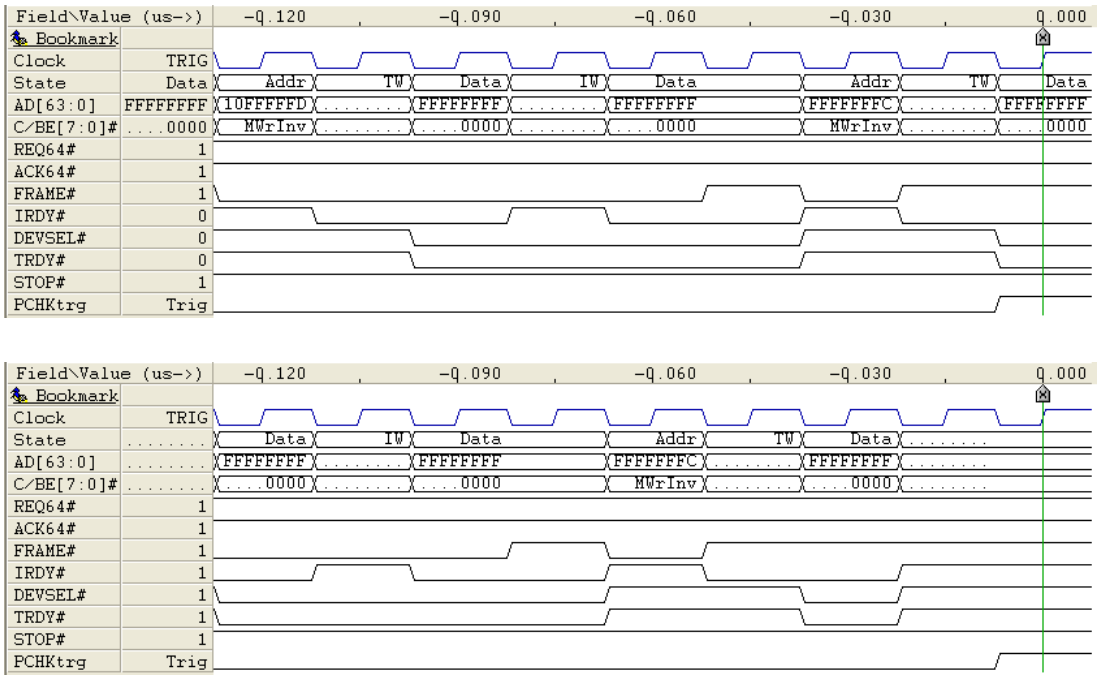
PCI Spec. 2.2, page 90

Section 3.6.2

## Illegal A1/A0 Memory Commands

All 2.1 compliant devices with AD[1:0] equal to 0,1 and 1,1 must terminate the data transfer after the first data phase, when the Command field indicates a memory command.

In the figure below, STOP# should be asserted to indicate termination after the first data phase.




PCI Spec. 2.1, page 28

Section 3.2.2

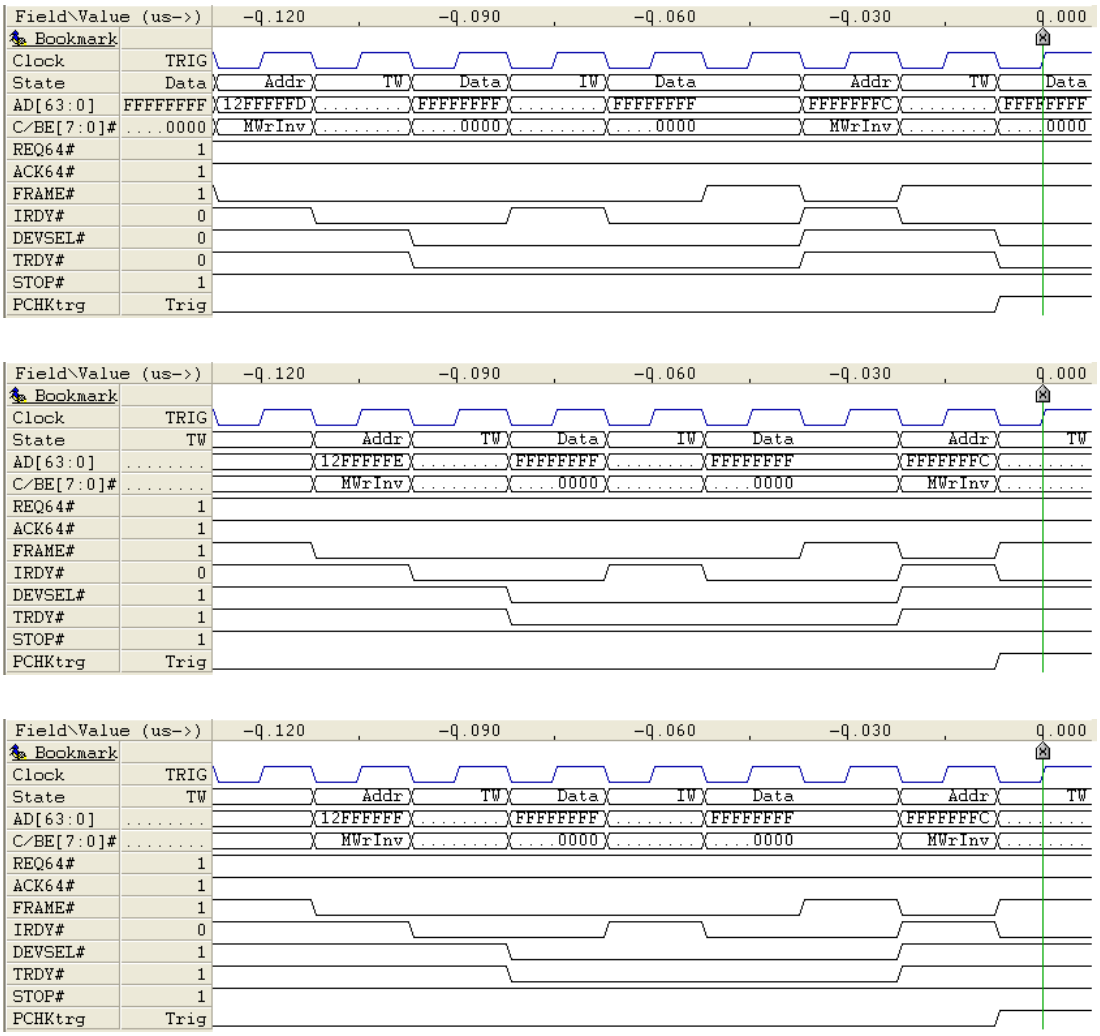
PCI Spec. 2.2, page 29

Section 3.2.2.2

## Illegal Linear Address Incrementing Mode

The Memory Write & Invalidate Command can only use the Linear Address Incrementing Mode (AD[1:0] = 0,0).

In the figure below, the C/BE# field signals a Memory Write & Invalidate Command, but the Address is 0xFFFFFFFD, i.e. AD[1:0] = 0,1.




PCI Spec. 2.1, page 28

Section 3.2.2

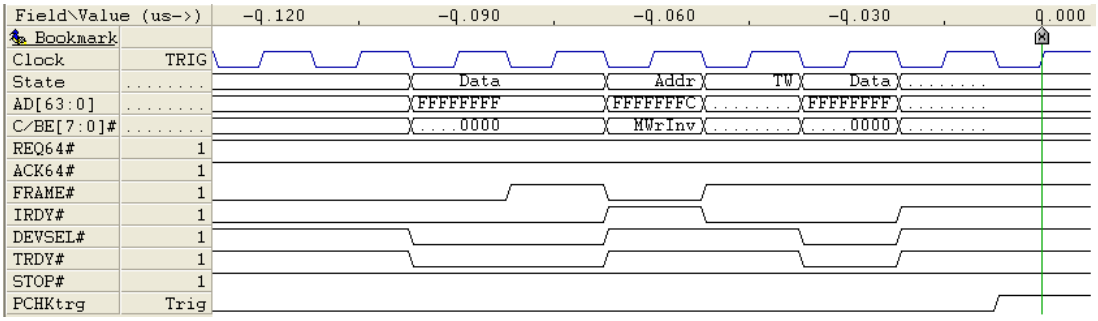
PCI Spec. 2.2, page 29

Section 3.2.2.2

## Illegal Configuration 0 Cycle

Type 0 Configuration cycle claimed by a Bridge device. Type 0 configuration cycles are not propagated beyond the local PCI Bus. It must be claimed by a local device within SLOW decode timing, or terminated with Master-Abort. Type 0 configuration cycle is when AD[1:0] = 00.

In the figure below, DEVSEL# is not asserted until the 4th clock after the 1st FRAME#, violating the SLOW decode timing requirements.




PCI Spec. 2.1, page 87

Section 3.7.4

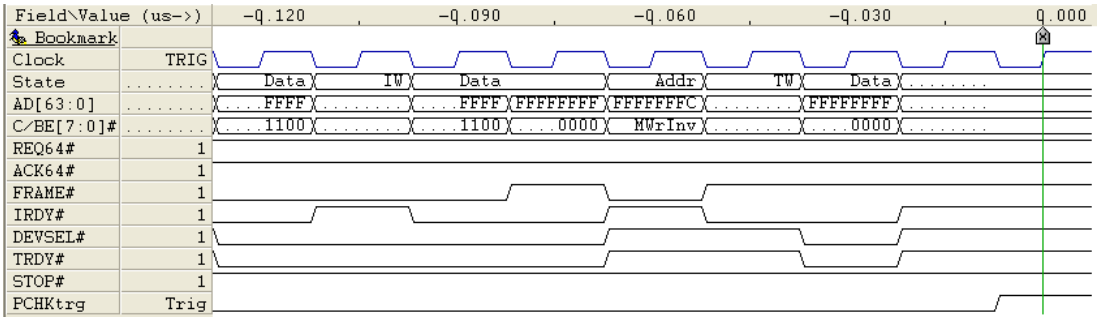
PCI Spec. 2.2, page 31

Section 3.2.2.3.1

## BE[3:0]# Change in Data

Byte Enables must be valid throughout each data phase independent of the state of IRDY#. IRDY# can be deasserted to add wait states, but Byte Enables must remain valid prior to actual data transfer.

A data phase cannot be completed before TRDY# is asserted, so BE[3:0]# must be valid until TRDY# is asserted regardless of the IRDY# state. This is clearly not the case in the figure below.




PCI Spec. 2.1, page 29 - PCI Spec. 2.2, page 39

Section 3.2.3

PCI Spec. 2.1, page 36 - PCI Spec. 2.2, page 47

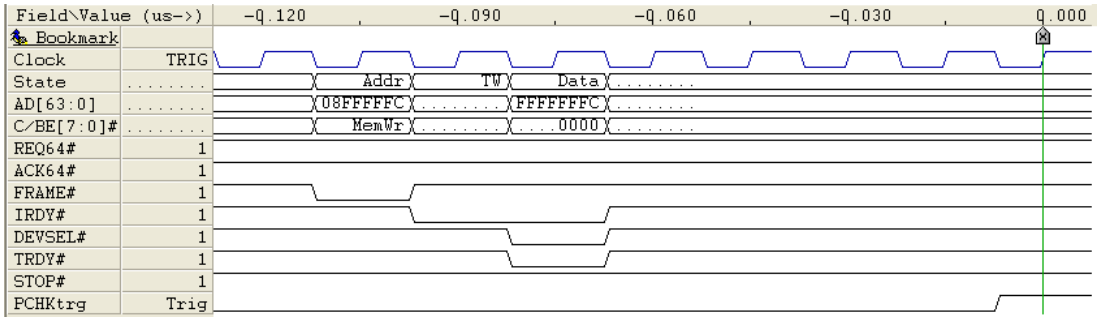
Section 3.3.1

## PERR# only on Data Transfer

PERR# must be reported 2 clocks after the data transfer in which the error occurred.

**Write transaction**
PERR# can be asserted prior to the actual transfer in which an error occurred when IRDY# is asserted and the target is inserting wait states.

**Read transaction**
PERR# can be asserted prior to the actual transfer in which an error occurred when TRDY# is asserted and the master is inserting wait states.




PCI Spec. 2.1, page 98

Section 3.8.2.1

PCI Spec. 2.2, page 96

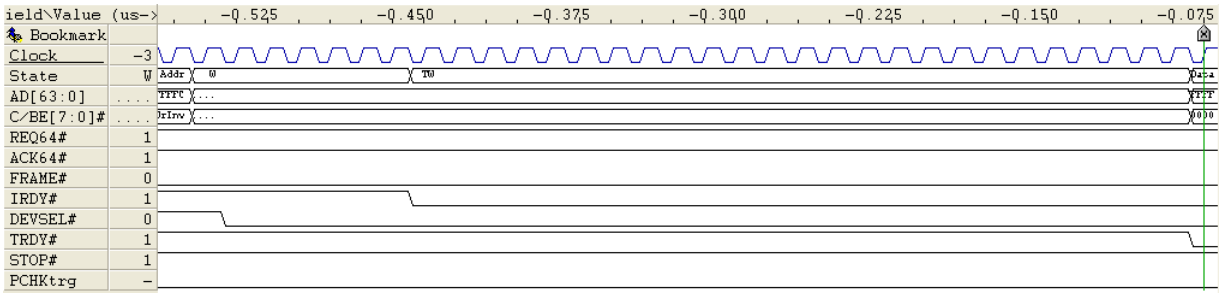
Section 3.7.4.1

## Too Many Clocks in Cycle

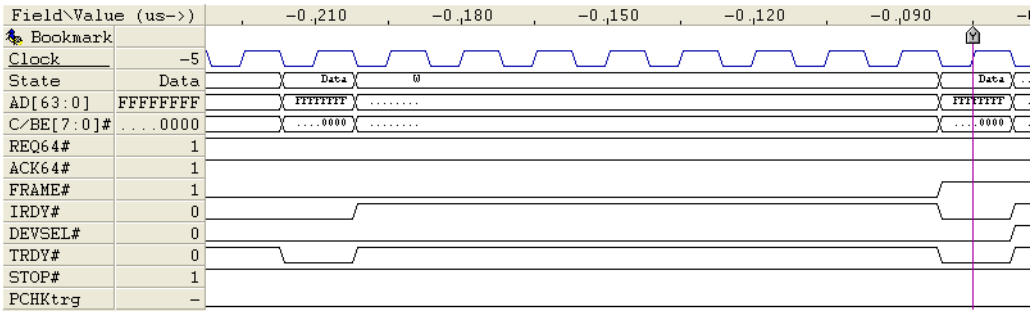
All targets are required to complete the initial data phase of a transaction (read or write) within 16 clocks from the assertion of FRAME#. The exception is for host bus bridges which are granted an additional 16 clocks, to a maximum of 32 clocks, to complete the initial data phase when the access hits a modified line in a cache. The Protocol Checker reports a violation if the initial data phase is not completed within 32 clocks.

Each successive data transfer, and the final transfer must complete in 8 clock cycles.

Initial Data Transfer has too many (33) clock cycles




Second Data Transfer has too many (9) clock cycle




PCI Spec. 2.1, page 66

Section 3.5.3.1

PCI Spec. 2.2, page 81

Section 3.5.4.1

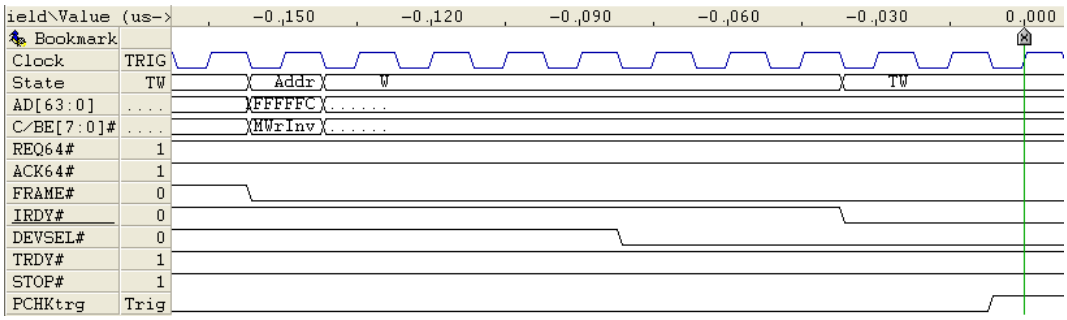
PCI Spec. 2.1, page 248 - PCI Spec. 2.2, page 254

Rule 25, 26

## Missing Master Abort

The master must terminate a cycle when DEVSEL# is not asserted on the 4th CLK after the 1st FRAME#.

In the figure below, DEVSEL# is not asserted until the 5th clock after the 1st FRAME#, and thus the master should have terminated the cycle.




PCI Spec. 2.1, page 80

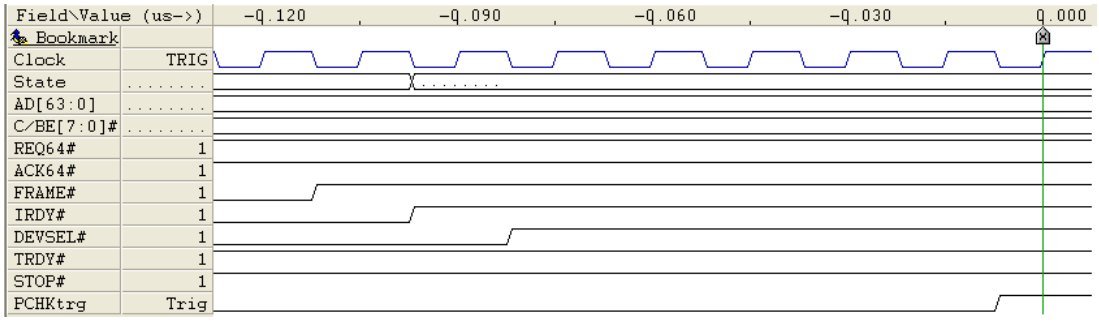
Section 3.7.1

PCI Spec. 2.2, page 89

Section 3.6.1

## Illegal Master Abort

If DEVSEL# is asserted, Master Abort is not permissible.

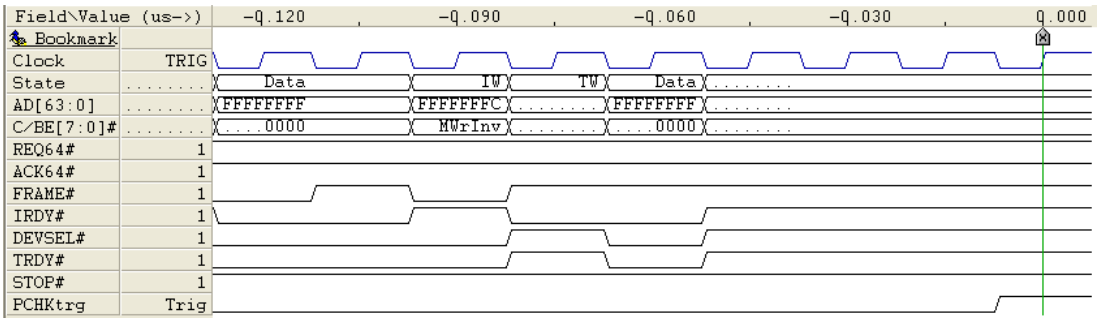



PCI Spec. 2.1, page 39 - PCI Spec. 2.2, page 50

Section 3.3.3.1

## Illegal Back-To-Back

DEVSEL#, TRDY# or STOP# must be deasserted the clock following the completion of the last data phase. If they remain asserted for an additional clock, as shown below, a fast Back-To-Back cycle is not allowed.



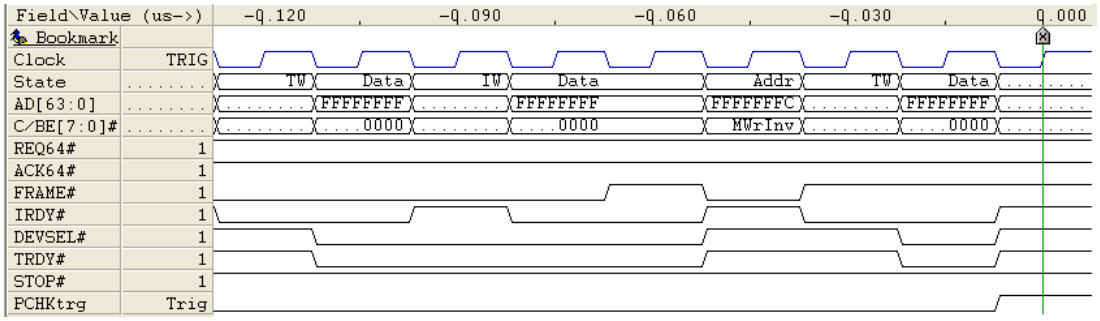

PCI Spec. 2.1, page 247 - PCI Spec. 2.2, page 253

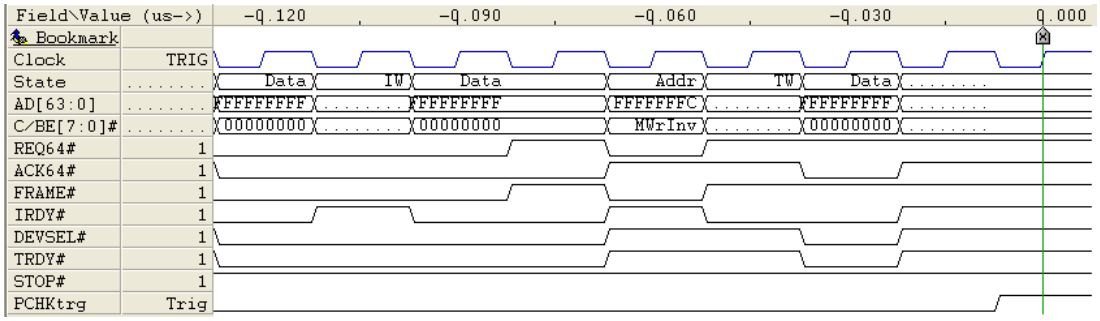
Rule 12f

## Data Phase Parity Error

PCI bus PAR does not correctly reflect the parity of AD[31:0] and C/BE[3:0] in the data phase. PAR is reported the clock after the data phase. The number of ones on AD[31:0], C/BE[3:0], and PAR, should equal an even number.

PCI bus PAR64 does not correctly reflect the parity of AD[63:32] and C/BE[7:4] in the data phase. PAR64 is reported the clock after the data phase. The number of ones on AD[63:32], C/BE[7:4], and PAR64, should equal an even number.




PCI Spec. 2.1, page 95

Section 3.8.1

PCI Spec. 2.2, page 94

Section 3.7.1

PCI Spec. 2.1, page 109

Section 3.10

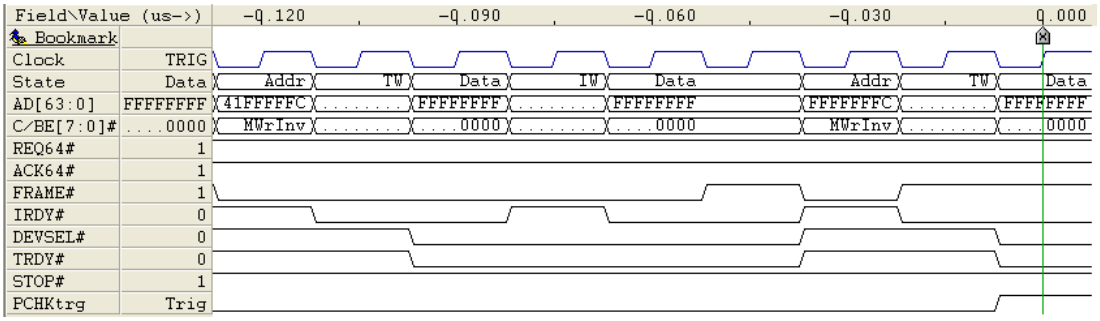
PCI Spec. 2.2, page 102

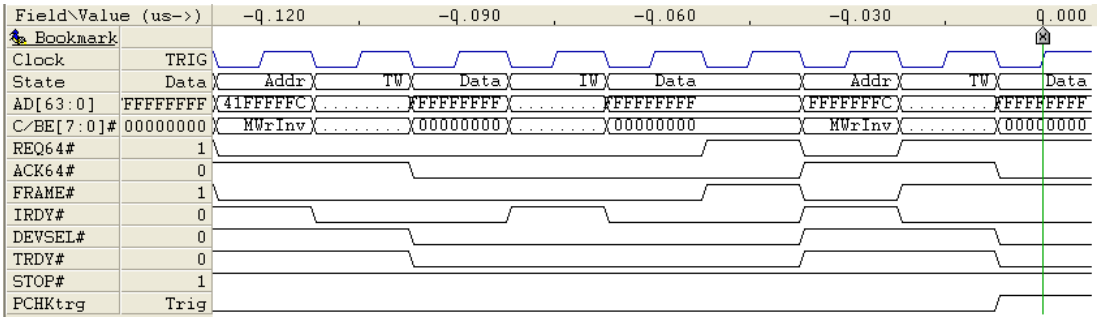
Section 3.8

## Address Phase Parity Error

PCI bus PAR does not correctly reflect the parity of AD[31:0] and C/BE[3:0] in the address phase. PAR is reported the clock after the address phase. The number of ones on AD[31:0], C/BE[3:0], and PAR, should equal an even number.

PCI bus PAR64 does not correctly reflect the parity of AD[63:32] and C/BE[7:4] in the address phase. PAR64 is reported the clock after the address phase. The number of ones on AD[63:32], C/BE[7:4], and PAR64, should equal an even number.




PCI Spec. 2.1, page 95

Section 3.8.1

PCI Spec. 2.2, page 94

Section 3.7.1

## Master Abort

Master Abort is not a violation.

PCI Spec. 2.1, page 39 - PCI Spec. 2.2, page 49

Section 3.3.3.1

## Target Abort

Target Abort is not a violation.

PCI Spec. 2.1, page 47 - PCI Spec. 2.2, page 59
Section 3.3.3.2.1

## RST# Asserted

PCI bus RST# asserted is not a violation.

PCI Spec. 2.1, page 9 - PCI Spec. 2.2, page 9
Section 2.2.1

## PERR# Asserted

PCI bus PERR# asserted is not a violation.

PCI Spec. 2.1, page 12 - PCI Spec. 2.2, page 12
Section 2.2.5

## 6.5 PCI-X protocol Violations

### PCI-X protocol violations

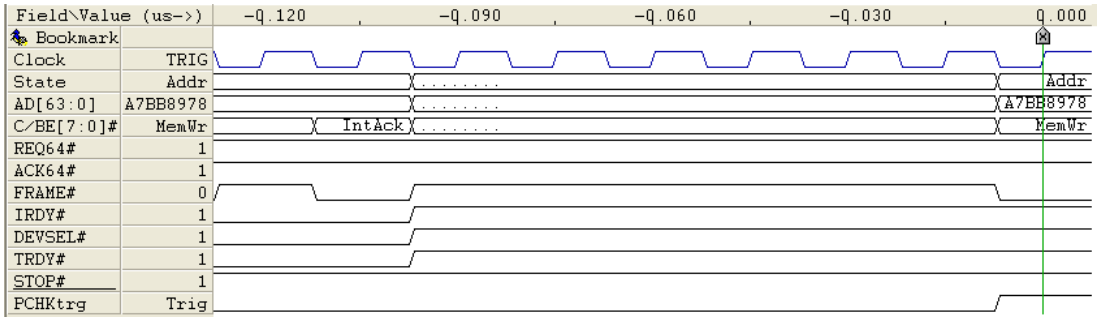
The following classes of PCI-X bus specification violations are detected.

Target abort, Master abort, High and Low address parity error, High and Low data parity error, PCI-X bus PERR# asserted, PCI-X bus SERR# asserted, PCI bus RST# asserted are also detected.

**Summary of detected PCI-X protocol violations**

## Illegal FRAME# Assertion

When FRAME# has been deasserted, it cannot be reasserted during the same transaction. FRAME# can not be asserted during the bus turnaround cycle.




PCI-X Spec. 1.10.2

and

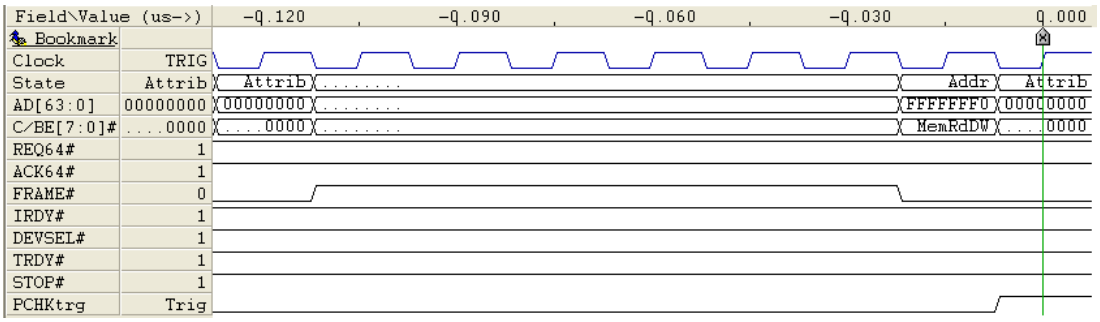
PCI Spec. AppC

Rule 8b and 8d

## Illegal FRAME# Deassertion

FRAME# can not be deasserted unless IRDY# has been asserted. FRAME# must be deasserted on the later of the following conditions:

- one clock before the last data phase, if the transaction has four or more data phases.
- two clocks after the target asserts TRDY# (or terminates the transaction in some other way as described in PCI-X specification section 2.11.2), if the transaction has less than four data phases.




PCI-X Spec. 1.10.2

and

PCI Spec. AppC

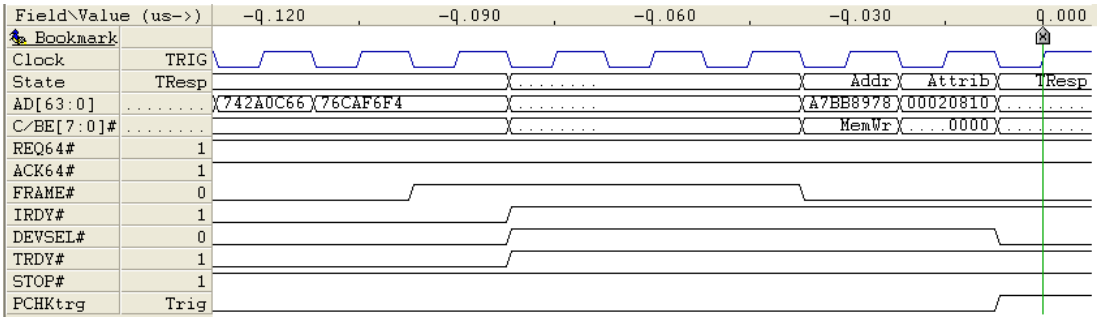
Rule 8c

## No Termination When Byte Count Satisfied

The initiator must terminate the transaction when the byte count is satisfied.

Note

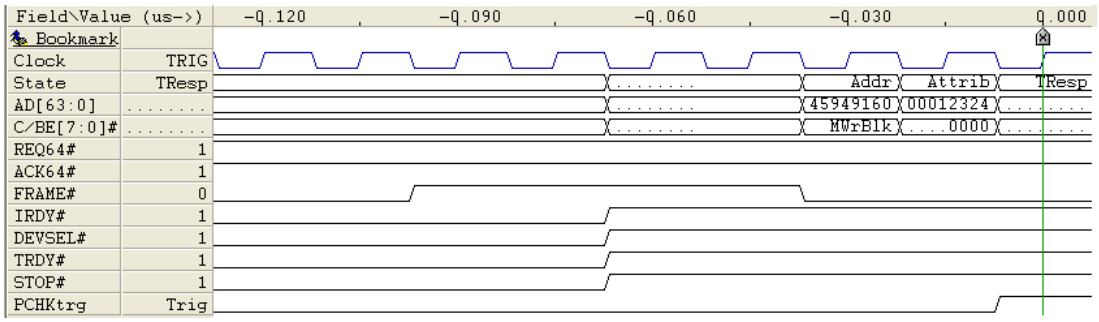
The byte count of Memory Write transactions must not be counted for bytes whose byte enables are deasserted within the transaction. In other words, the byte count is the same whether all or none of the byte enables were asserted.




PCI-X Spec. 1.10.2.6

## Burst Termination Not on ADB

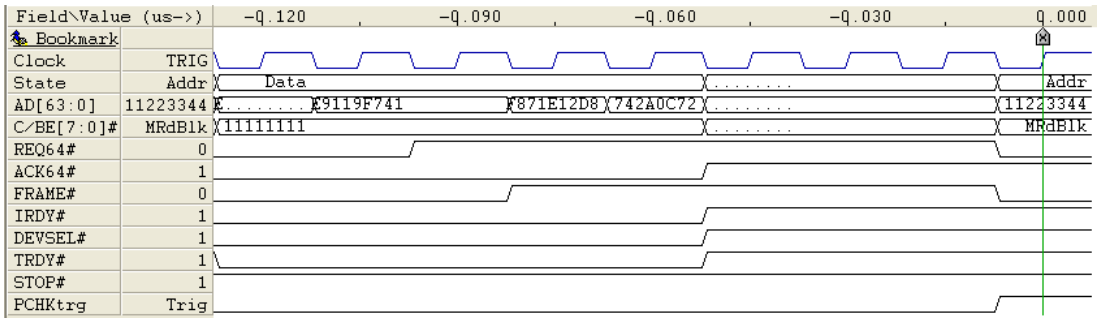
The initiator is permitted to disconnect a burst transaction (before the byte count is satisfied) only on an ADB.

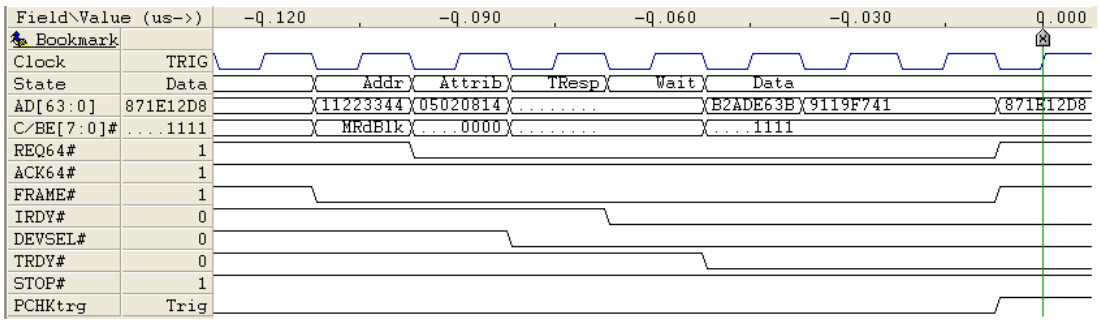



PCI-X Spec. 1.10.2.8

## REQ64# Not Synced to FRAME#

A 64-bit initiator asserts and deasserts REQ64# with the same timing as FRAME# except by host bridge to signal bus mode during bus initialization.

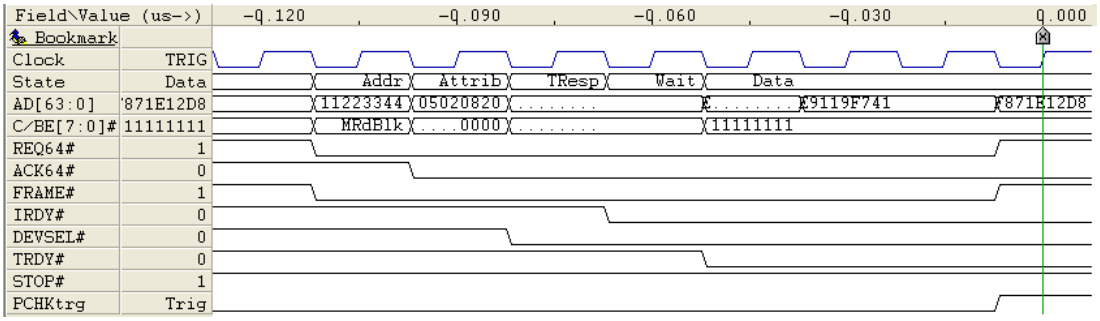



PCI-X Spec. 2.12.3

## ACK64# Not Synced to DEVSEL#

A 64-bit initiator asserts ACK64# with the same timing as DEVSEL# to request a 64-bit data transfer. It deasserts ACK64# with DEVSEL# at the end of the transaction.

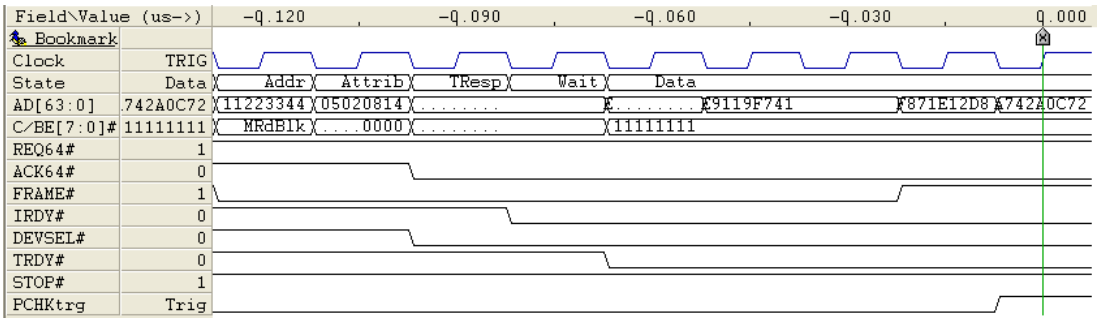



PCI-X spec rev. 1.0 page 102

PCI Spec. 2.2.8.

## ACK64# without REQ64#

ACK64# must not be asserted in transactions where the initiator has not asserted REQ64# during the address phase.




PCI-X Spec. 1.10.1

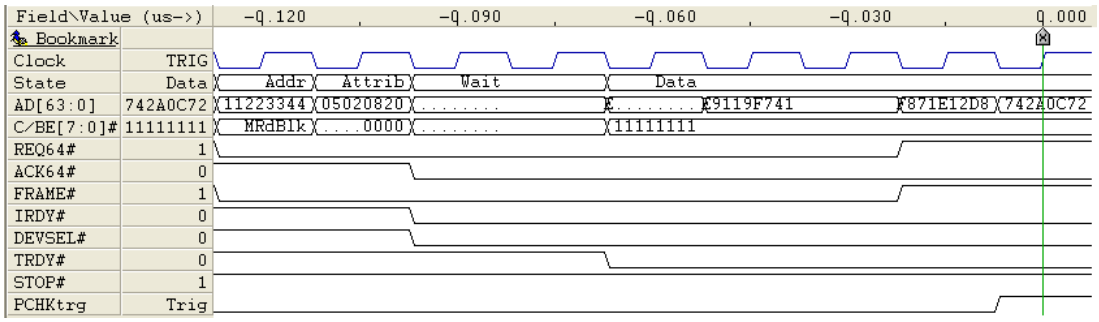
Rule 9

PCI-X Spec. 1.10.7

Rule 7

## Illegal IRDY# Assertion

IRDY# must only be asserted on the second clock after the attribute phase.

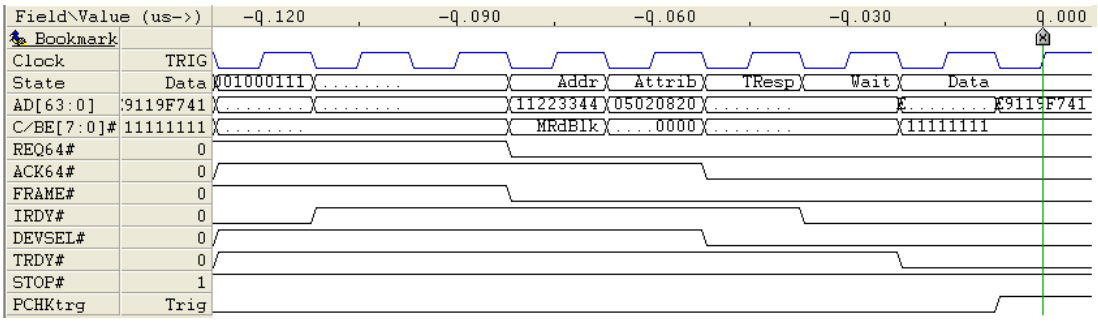



PCI-X Spec. 1.10.2

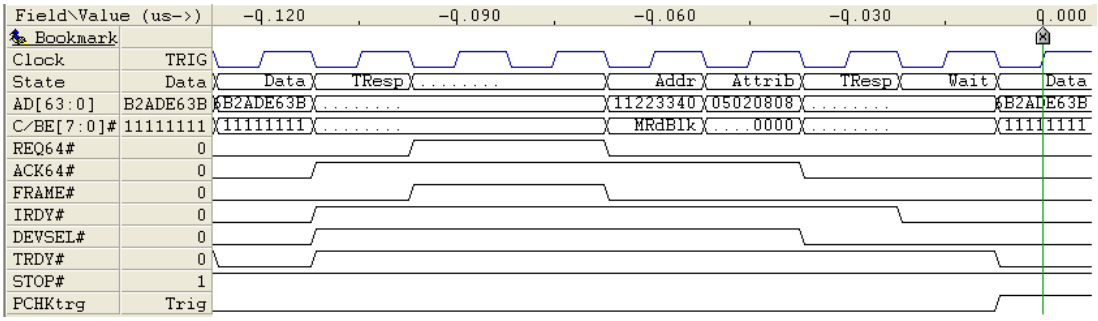
## Illegal IRDY# Deassertion

IRDY# must be deasserted one clock after the last data phase when the transaction has 3 or more data phases. If the transaction has less than 3 data phases, IRDY# is deasserted two clocks after the target asserts TRDY#.

Illegal IRDY# deassertion when the transaction has 3 or more data phases



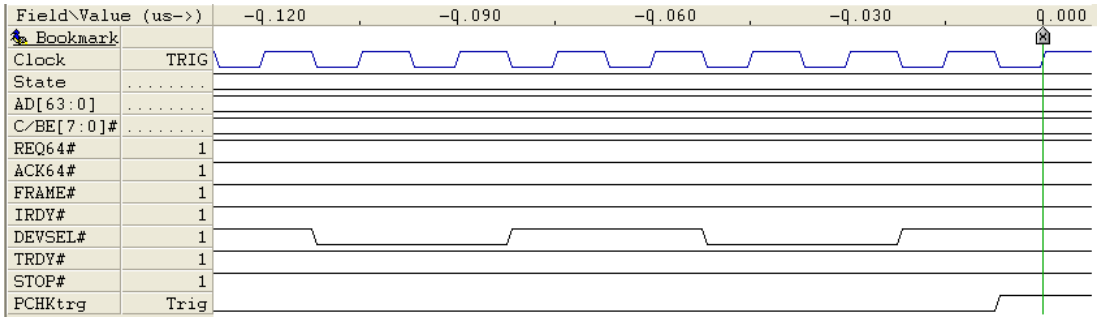

Illegal IRDY# deassertion when the transaction has less than 3 data phases




PCI-X Spec. 1.10.2

## Illegal DEVSEL# Assertion

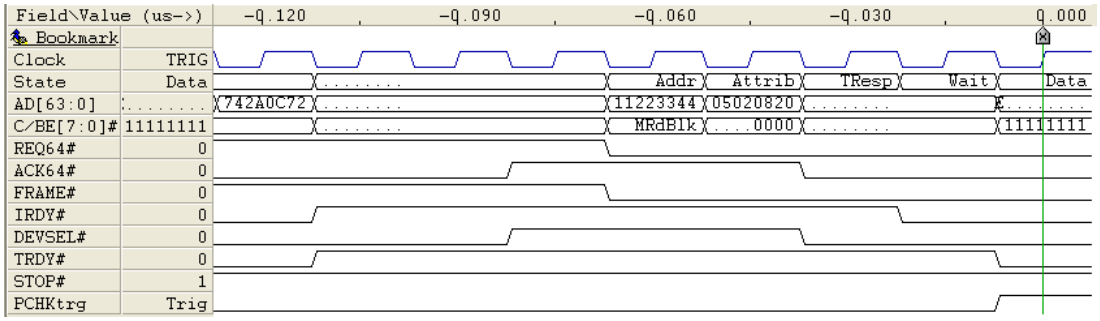
DEVSEL# must not be asserted when the bus is idle or in Address Phase, except by host bridge to signal bus mode during bus initialization.




PCI-X Spec. 1.10.3.2

## Illegal DEVSEL# Deassertion

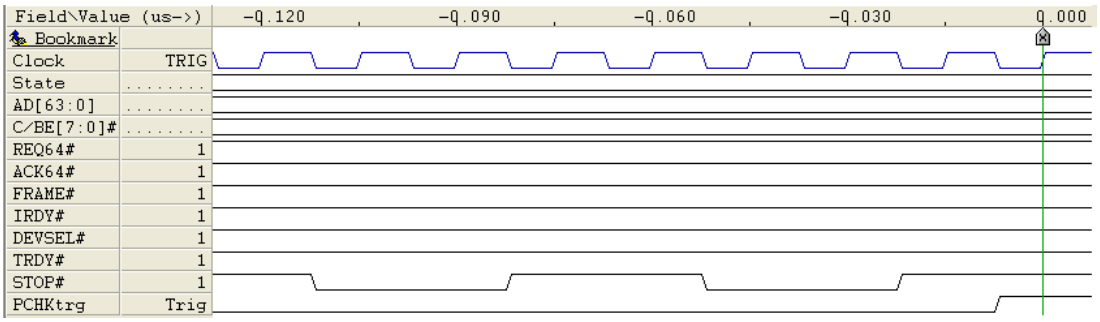
DEVSEL# must be deasserted one clock after the last data phase, except when signaling Split Response, Target Abort or Single Data phase Disconnect.




PCI-X Spec. 1.10.3.8

## Illegal STOP# Assertion

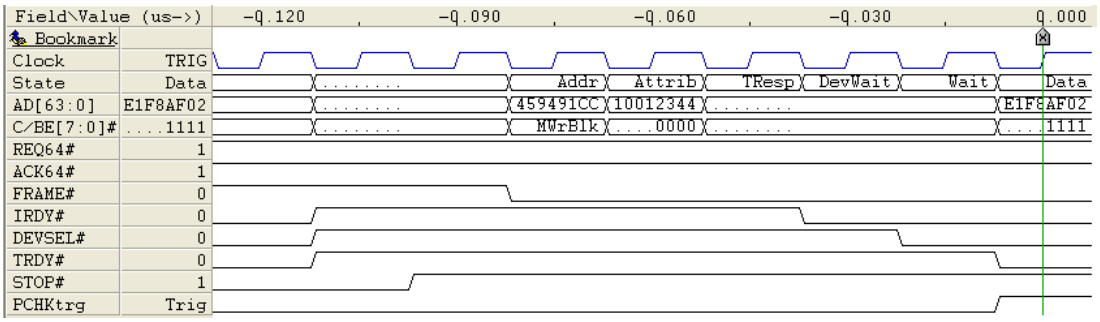
STOP# can only be asserted during Data Phases, except by host bridge to signal bus mode during bus initialization.




PCI-X Spec. 2.11.2, 1.10.3.8

## Illegal STOP# Deassertion

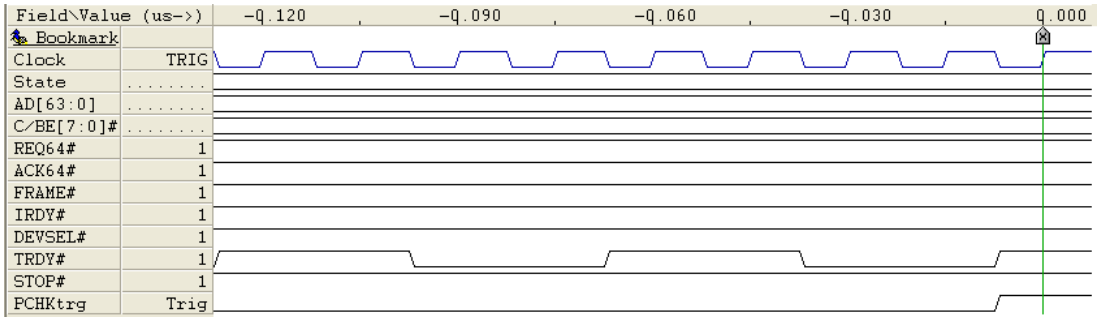
STOP# must be deasserted one clock after the last data phase, except by host bridge to signal bus mode during bus initialization




PCI-X Spec. 2.11.2, 1.10.3.8

## Illegal TRDY# Assertion

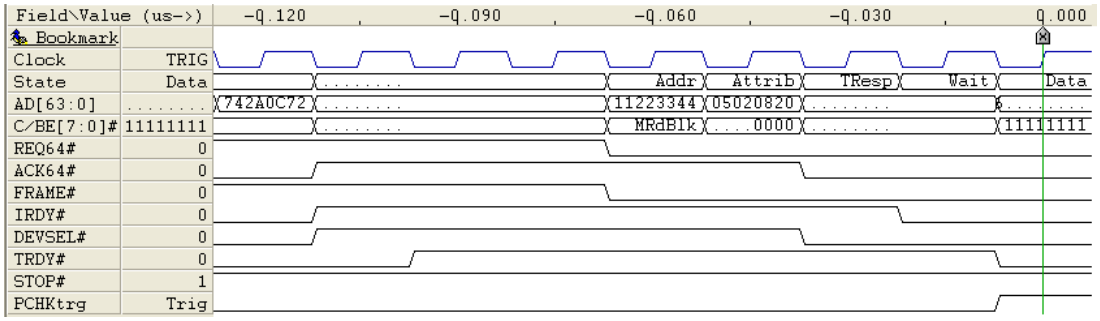
TRDY# can only be asserted in Data Phases, and no earlier than the clock after DEVSEL# is asserted, except by host bridge to signal bus mode during bus initialization.




PCI-X Spec. 2.11.2, 1.10.3.8

## Illegal TRDY# Deassertion

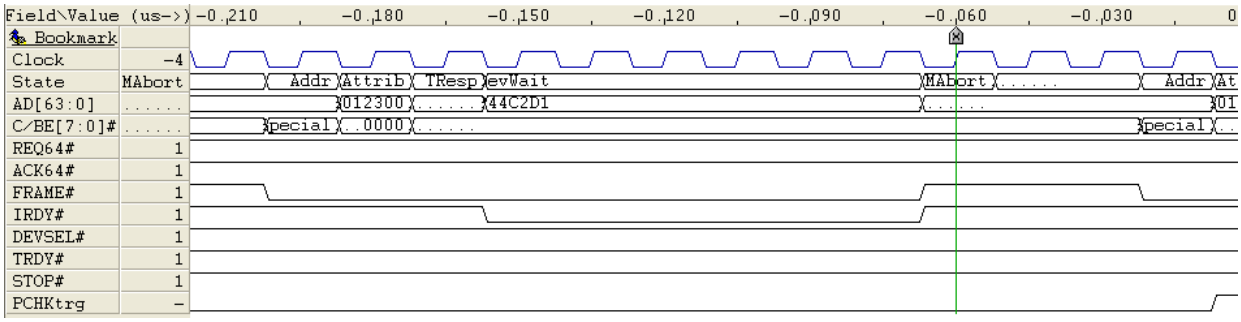
TRDY# must be deasserted one clock after the last data phase, except by host bridge to signal bus mode during bus initialization.




PCI-X Spec. 2.11.2, 1.10.3.8

## Missing Master Abort

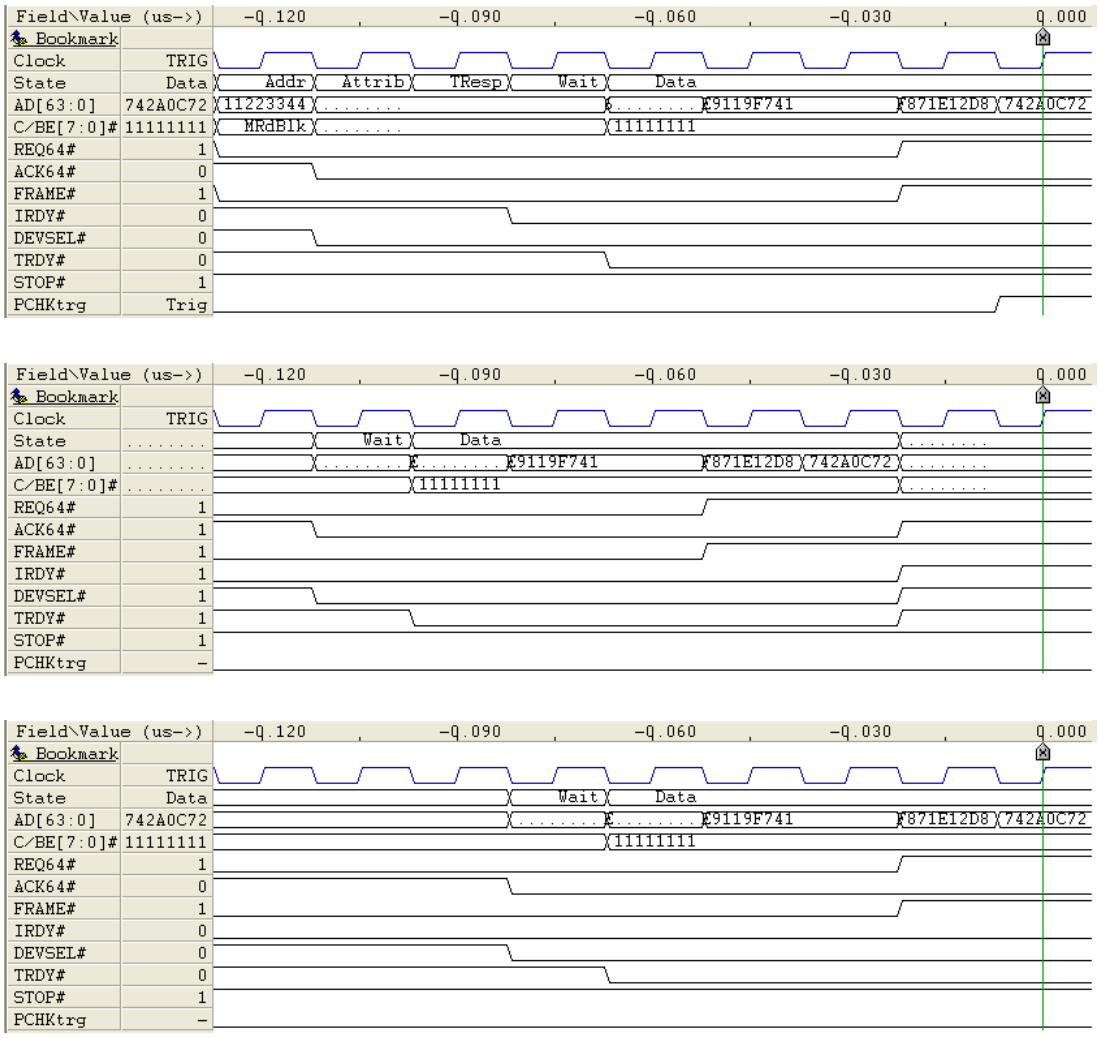
No target asserts DEVSEL# within six clocks after the address phase(s). The Initiator must deassert FRAME# and IRDY# eight clocks after the address phase(s).




PCI-X Spec. 2.8

## Illegal Decode Time

DEVSEL# must not be asserted 1, 5 or more than 6 clocks after the address phase(s).

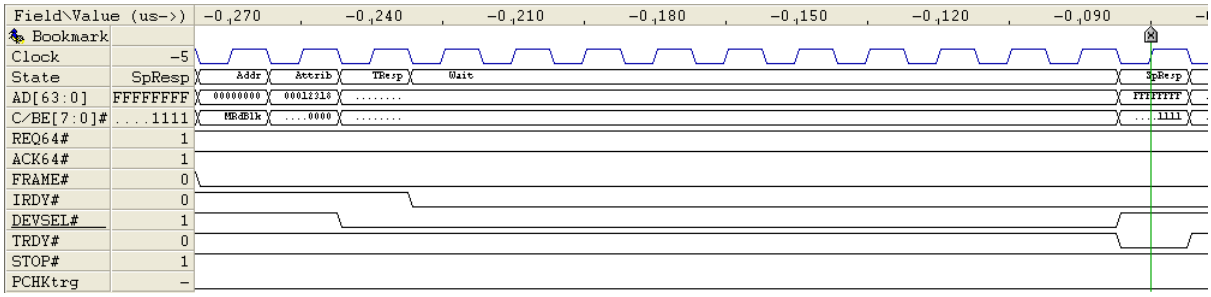



PCI-X Spec. 2.8

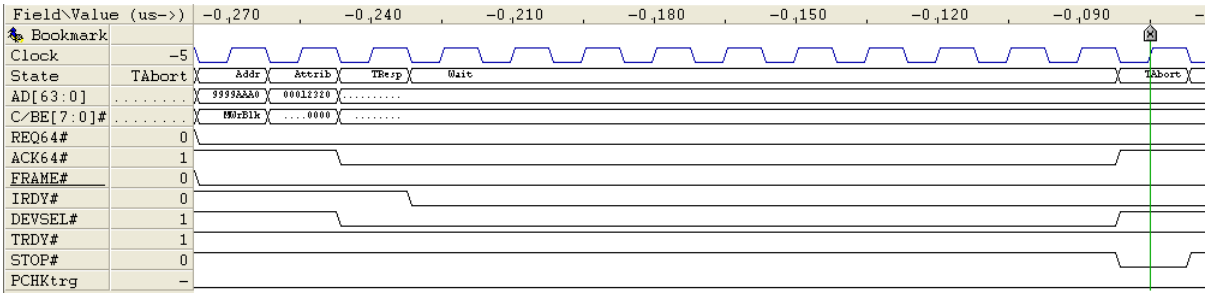
## Split Response, Target Abort, or Retry after 8 Clocks

If the target signals Split Response, Target-Abort, or Retry, the target must do so within eight clocks of the assertion of FRAME#.

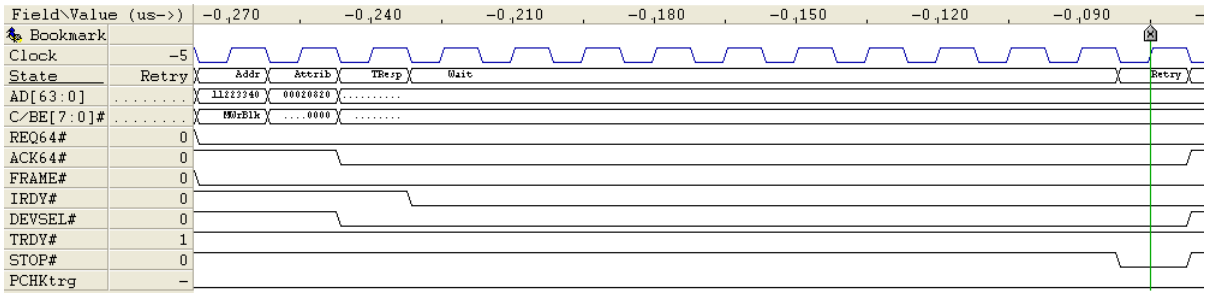
Split Response after 8 clock




Target-Abort after 8 clocks




Retry after 8 clocks

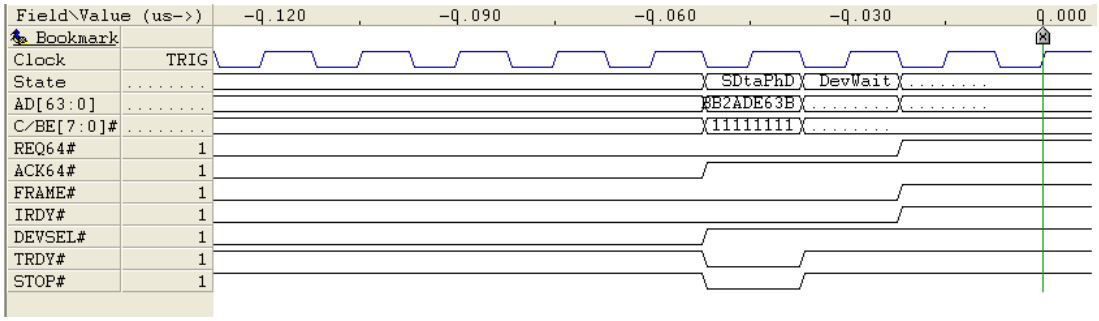


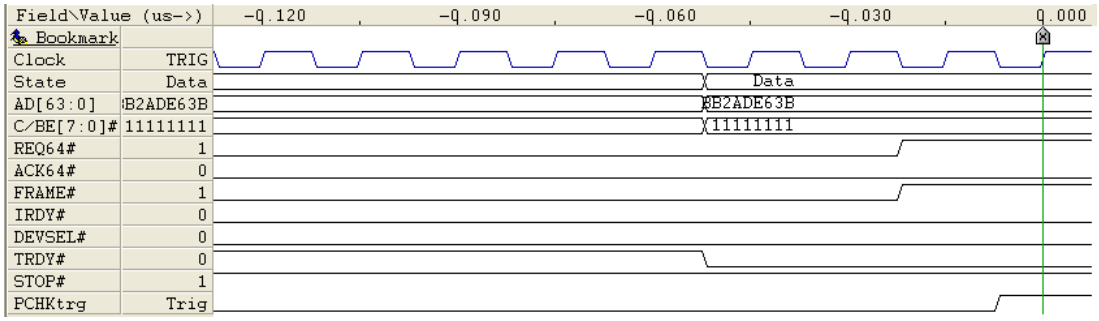

PCI-X Spec. 1.10.3

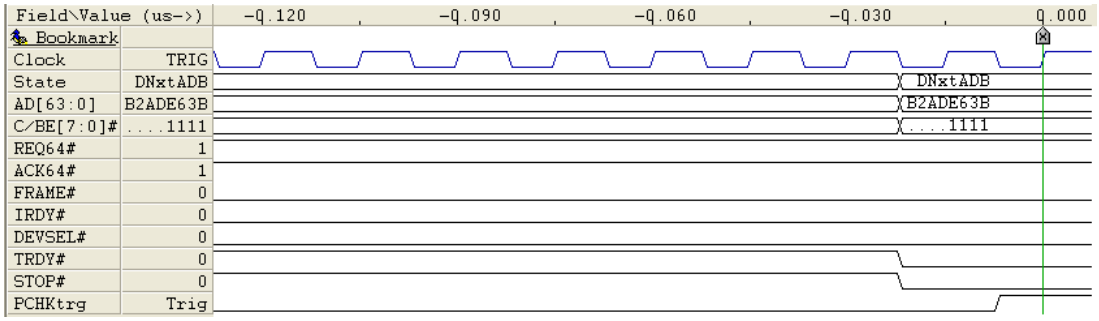
Rule 4

## Single Data Phase Disconnect, Data Transfer, or Disconnect at Next ADB after 16 Clocks

If the target signals Single Data Phase Disconnect, Data Transfer, or Disconnect at Next ADB, the target must do so within 16 clocks of the assertion of FRAME#.



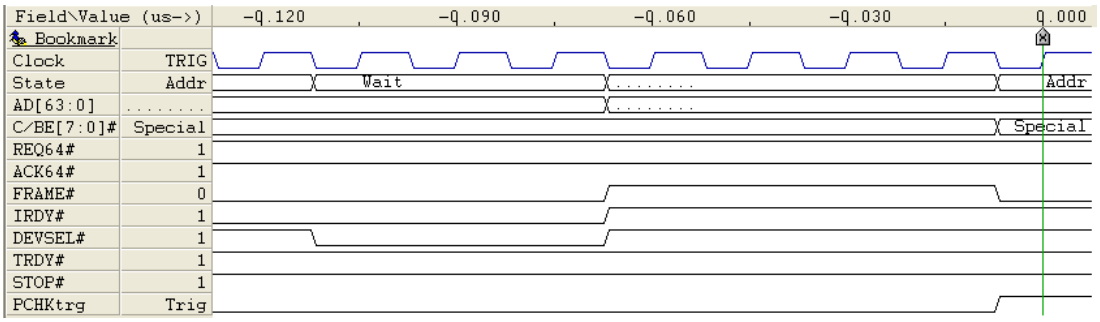




PCI-X Spec. 1.10.3

Rule 4

## DEVSEL# Asserted in Special Cycle

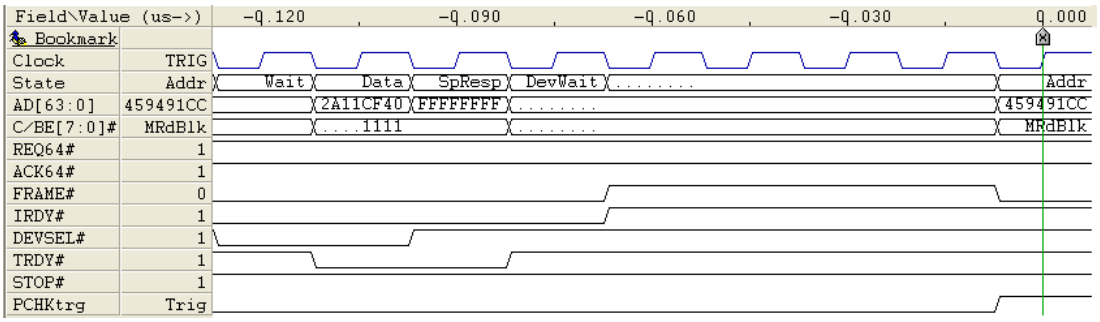
No target is permitted to assert DEVSEL# during a Special Cycle




PCI-X Spec. 2.7.3

## Split Response after Data Transfer

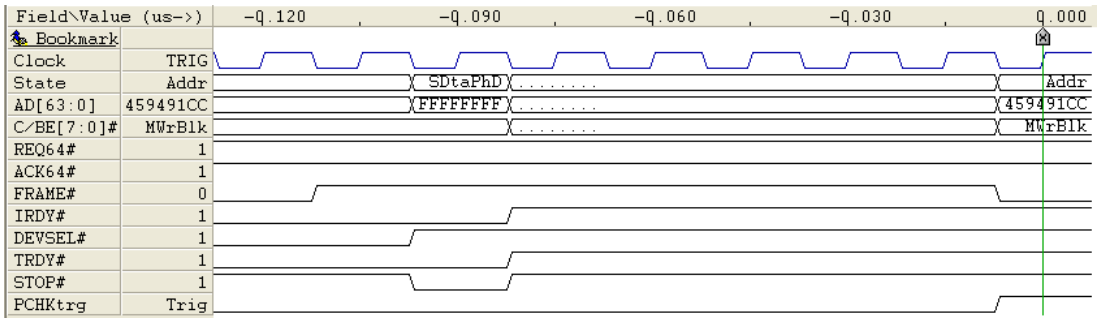
A target is not permitted to signal Split Response after signaling data transfer.




PCI-X Spec. 2.11.2.4

## Single Data Phase Disconnect after Data Transfer

A target is not permitted to signal Single Data Phase Disconnect after signaling Data Transfer.

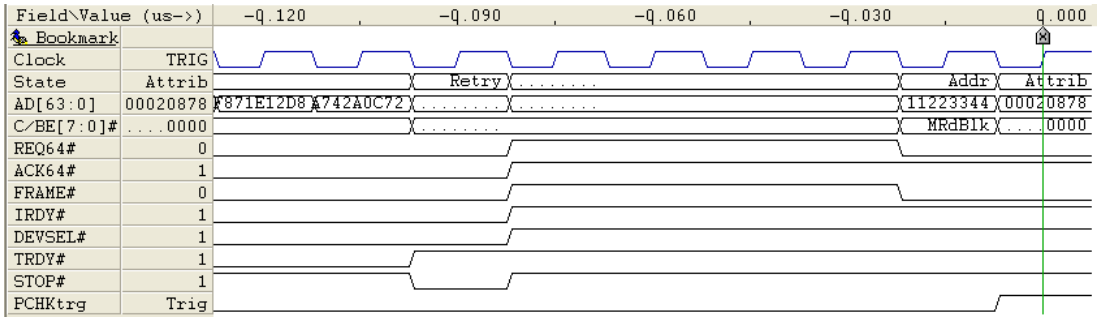



PCI-X Spec. 1.10.3

Rule 6

## Retry after Data Transfer

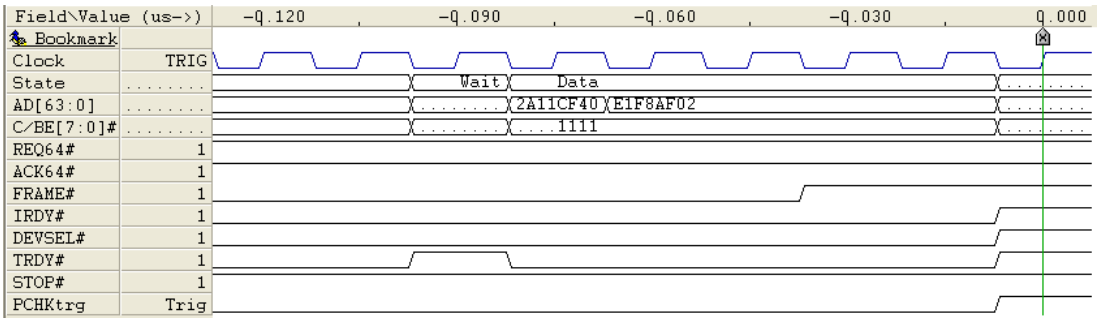
A target is not permitted to signal Retry after signaling Data Transfer.




PCI-X Spec. 2.11.2.3

## Illegal Target Wait States

A target is not permitted to insert wait states after signaling Data Transfer

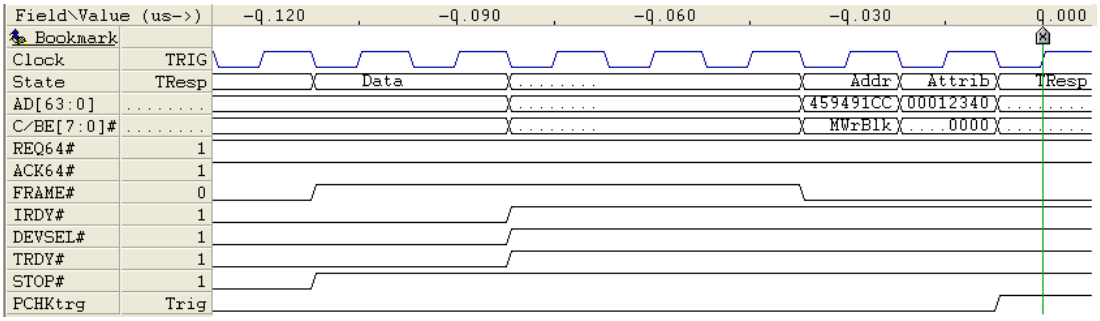



PCI-X Spec. 1.10.3

Rule 4

## Illegal Change of Target Signaling

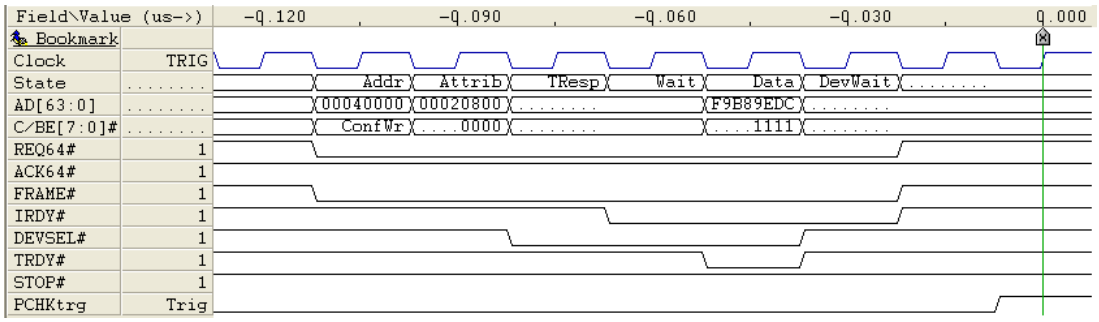
Once the target has signaled Disconnect at Next ADB, it is limited to signaling Disconnect at Next ADB or Target-Abort on all subsequent data phases until the end of the transaction.

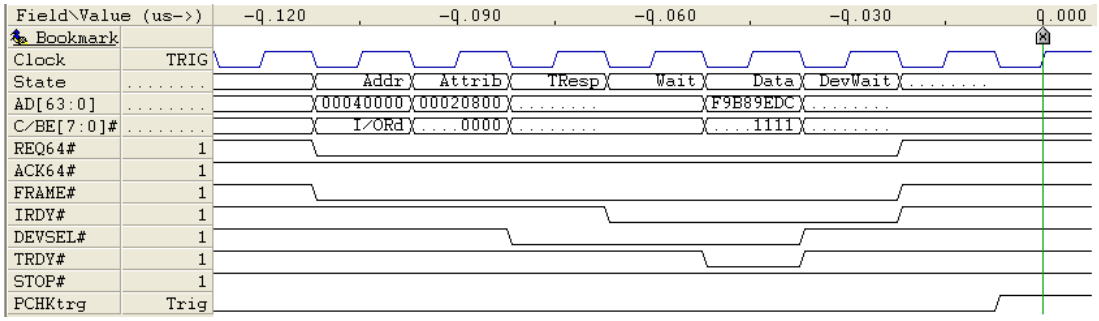



PCI-X Spec. 2.11.2.2

## REQ64# Asserted with DWORD Command

An initiator is not permitted to assert REQ64# when using DWORD commands.



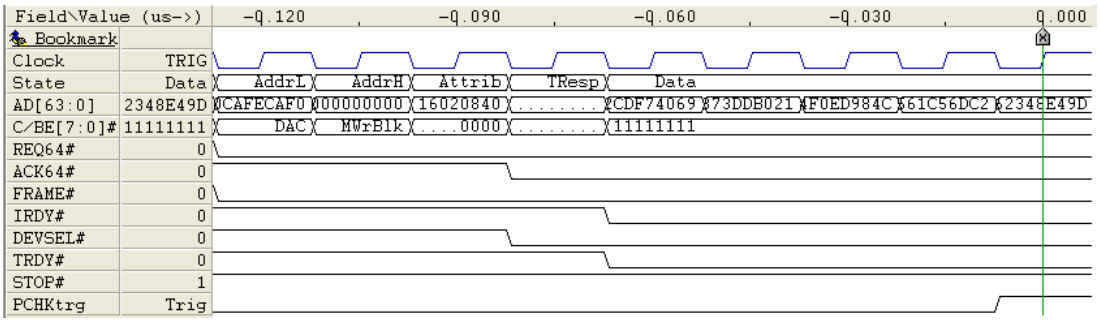




PCI-X Spec. 1.10.1

Rule 7 and 9

## DAC with High Address = 0

An initiator is not permitted to generate a Dual Address Cycle when the high address = 0.

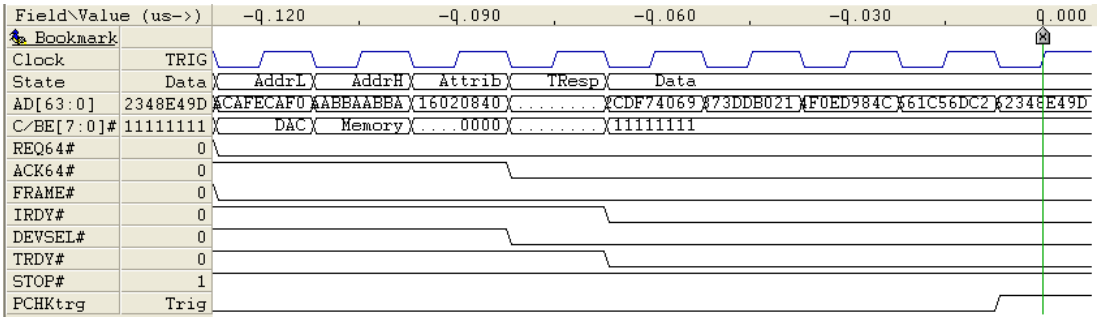



PCI-X Spec. 1.10.7

Rule 5

## DAC Followed by DAC

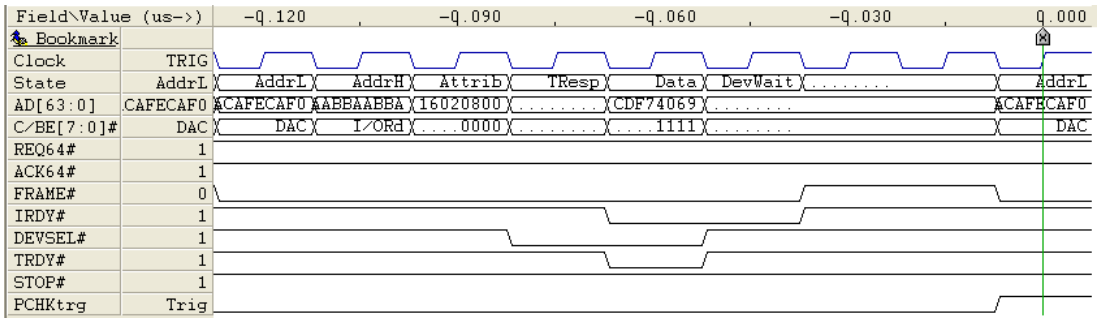
An initiator is not permitted to signal DAC immediately followed by a DAC command




PCI-X Spec. 1.12.1

## DAC Used With No Memory Command

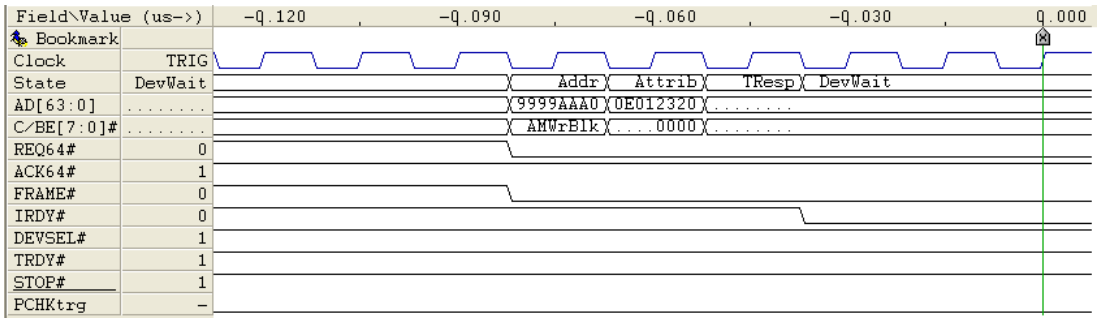
An initiator can only use Dual Address Cycle in combination with a Memory Command, except Split Completion.




PCI-X Spec. 1.12.1

## Use of Reserved Command

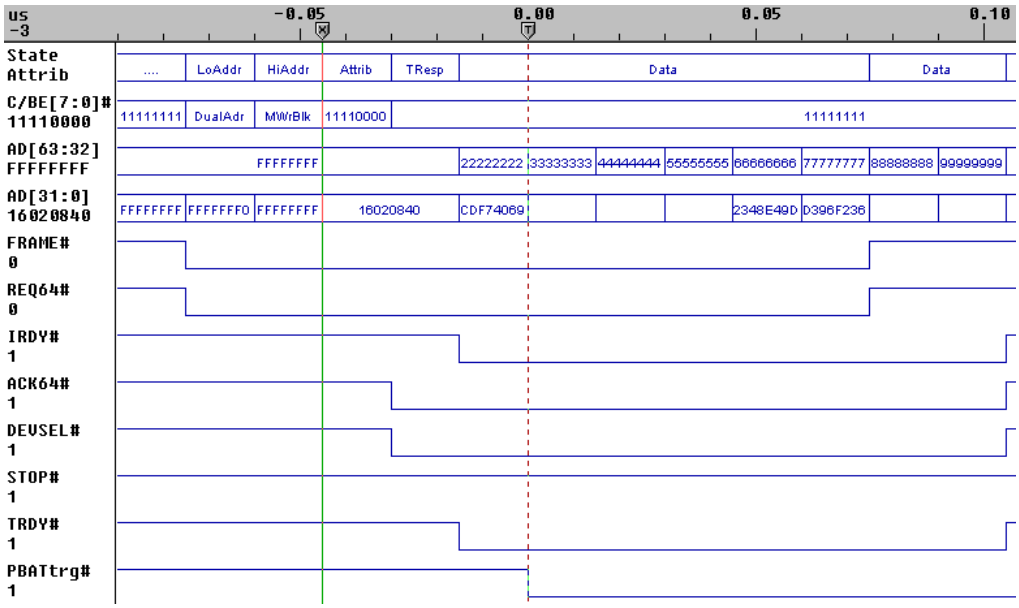
Initiators are not permitted to use any of the reserved commands, including Alias to Memory Read Block and Memory Write Block.




PCI-X Spec. 2.4 and 2.6

## Exceeding 64-bit Address Range

Burst transactions are not permitted to exceed the 64-bit memory address space. E.g. the start address plus byte count minus one must not exceed "FFFF FFFF FFFF FFFF".




PCI-X Spec. 2.6

*This page intentionally left blank*

## Illegal Address and Byte Enables

Valid Byte Enable and address combinations for I/0 and DWORD are shown in the table below.

AD[1:0] Valid BEs

00 xxx0 or 1111

01 xx01 or 1111

10 x011 or 1111

11 0111 or 1111

Valid Byte Enable and address combinations for Memory Write are shown in the table below.

AD[1:0] Valid BEs

00 xxxx

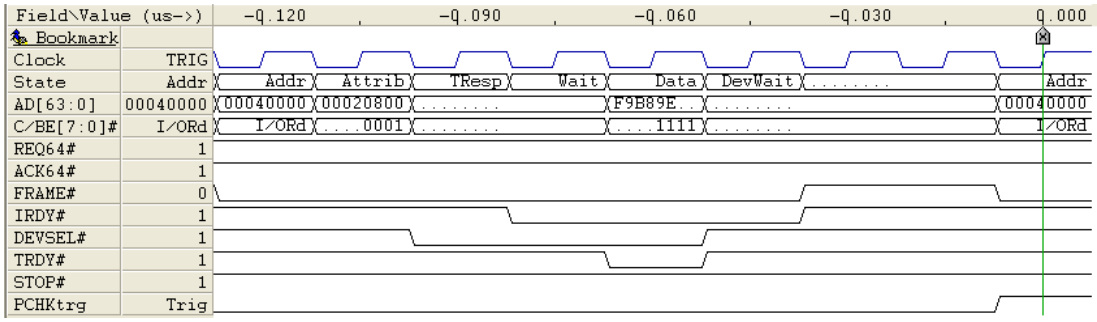
01 xxx1

10 xx11

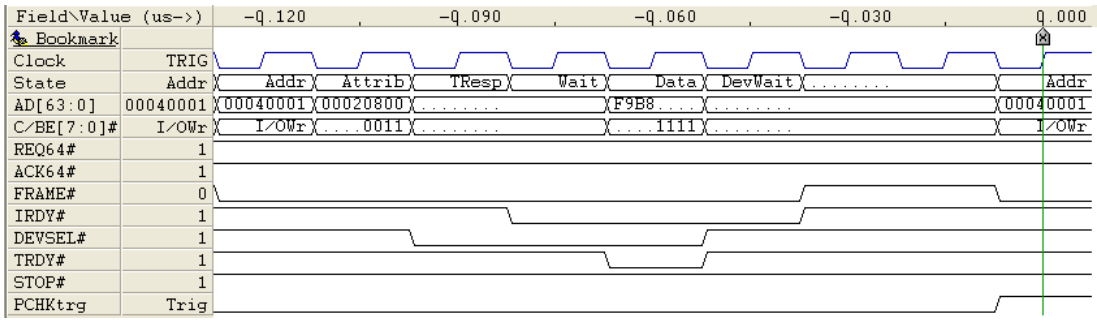
11 x111

## Illegal Address and Byte Enables (Continued)

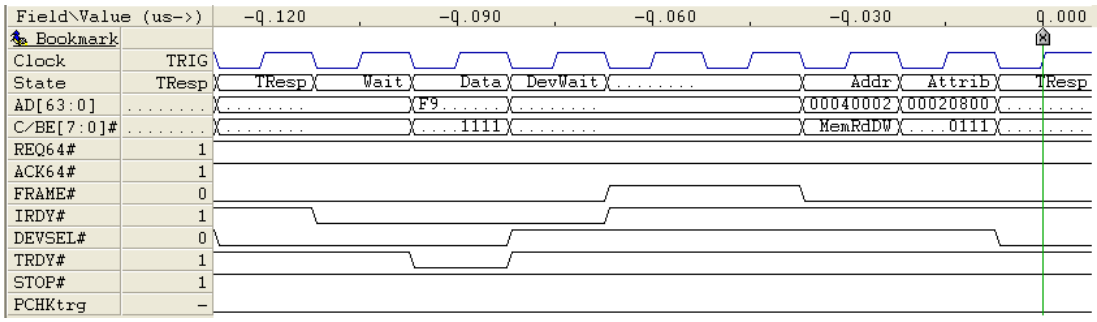
AD[1:0] = 00



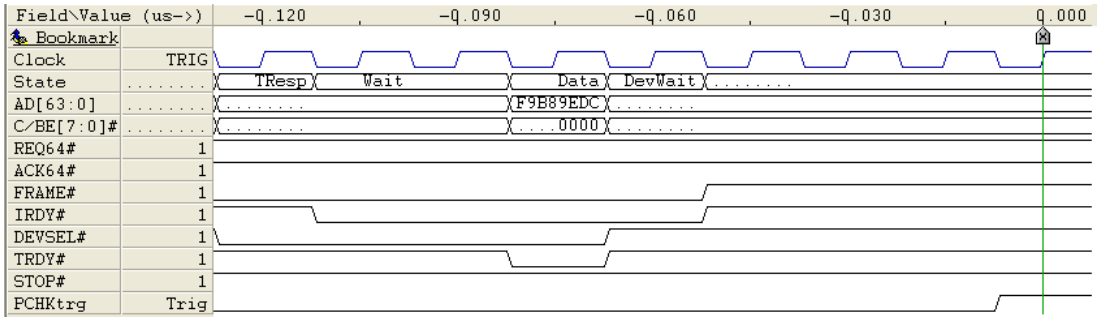

AD[1:0] = 01




AD[1:0] = 10



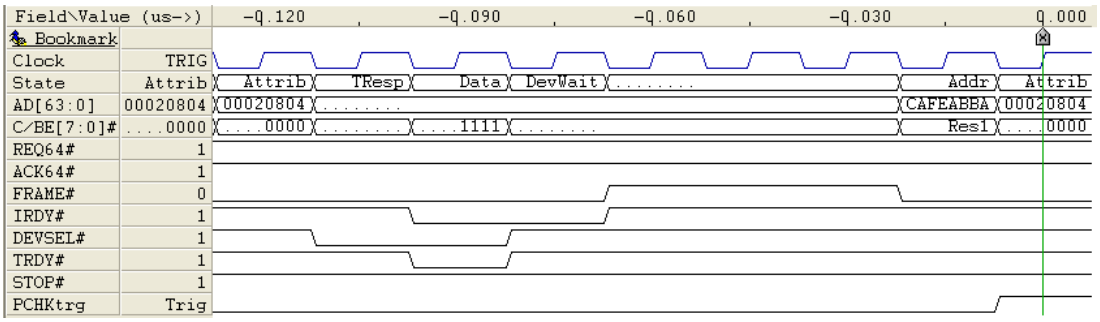

AD[1:0] = 11




PCI-X Spec. 2.3

## Target Respond to Reserved Command

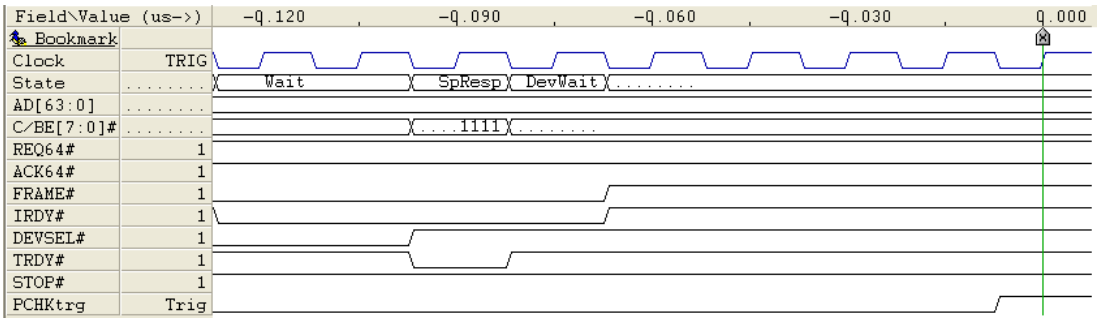
A target is not permitted to respond to any of the reserved commands, except Alias to Memory Read Block and Memory Write Block.




PCI-X Spec. 2.4 and 2.6

## Illegal Split Response

A target is not permitted to respond with Split Response to a Burst Write Transaction.

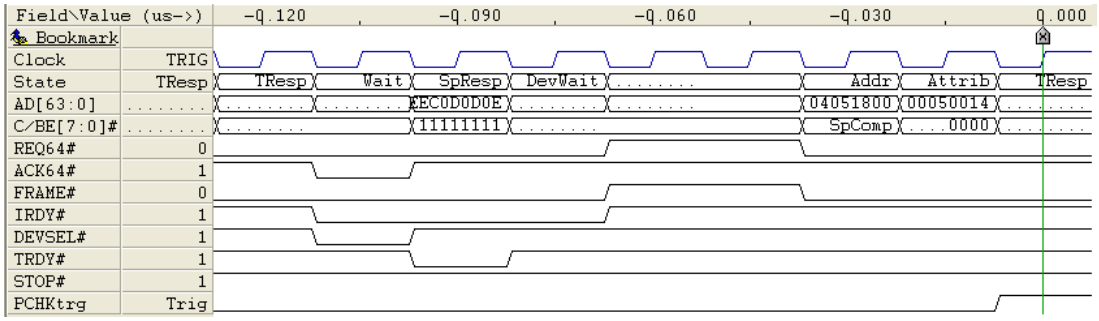



PCI-X Spec. 2.6.1

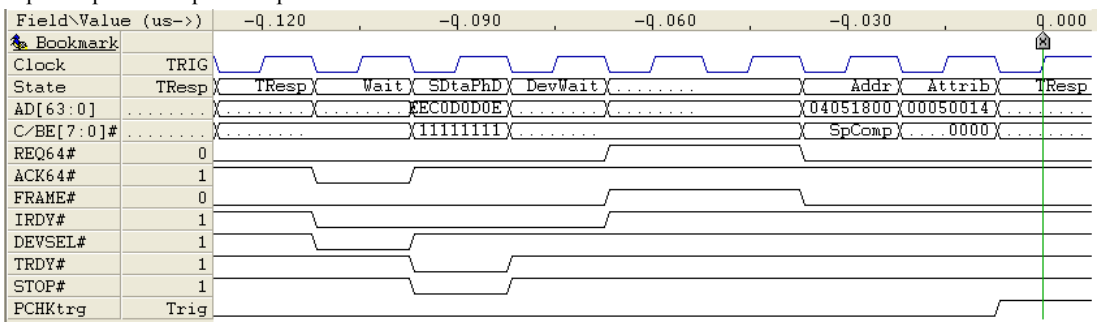
## Illegal Target Response to Split Completion

A target is not permitted to signal Split Response or Single Data Phase Disconnect in response to a Split Completion transaction

Illegal Single Data Phase Disconnect to Split Completion



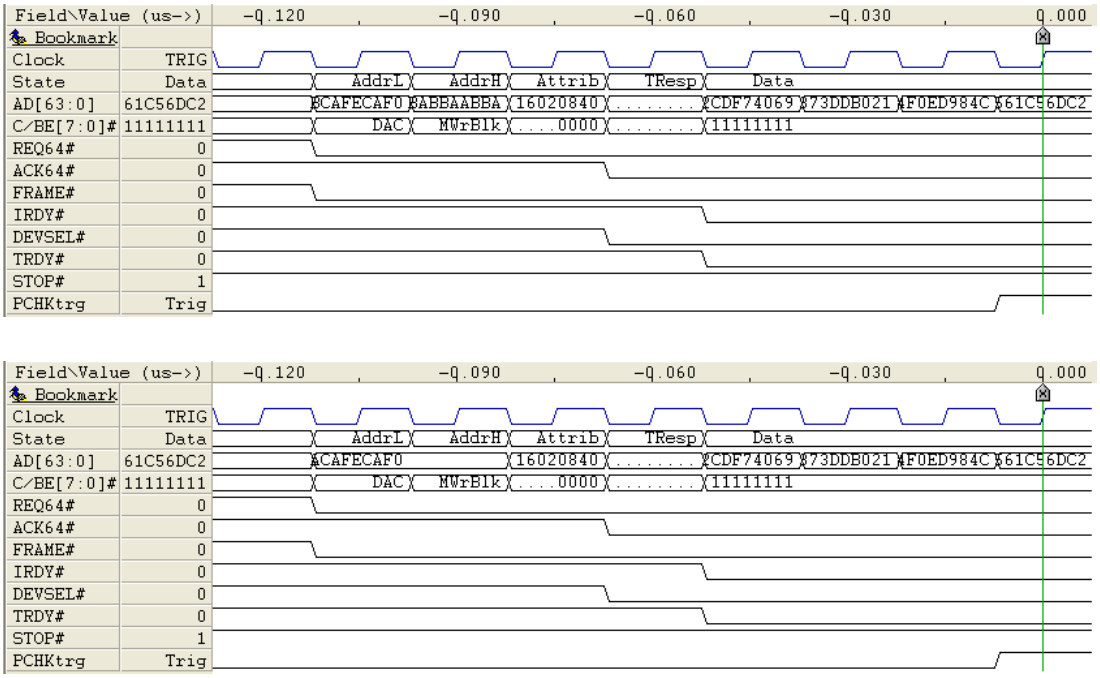

Illegal Split response to Split Completion




PCI-X Spec. 1.10.8.4 and 2.6.1

## High Address Change in DAC

In a 64-bit system, the initiator is required to drive the full 64-bit address on AD[63:0] during the first address phase of a Dual address cycle. During the second address phase, the high 32-bits of the address is copied to AD[31:0].

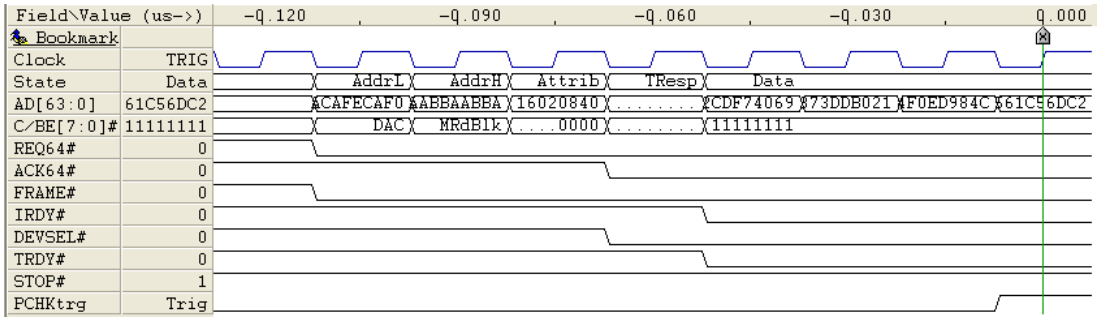



PCI-X Spec. 2.12.1

Rule 3

## Bus Command Change in DAC

In a 64-bit system, the initiator is required to drive the Memory Command on C/BE[7:4] during the first address phase of a Dual Address Cycle. During the second address phase, the Memory Command is copied to C/BE[3:0].

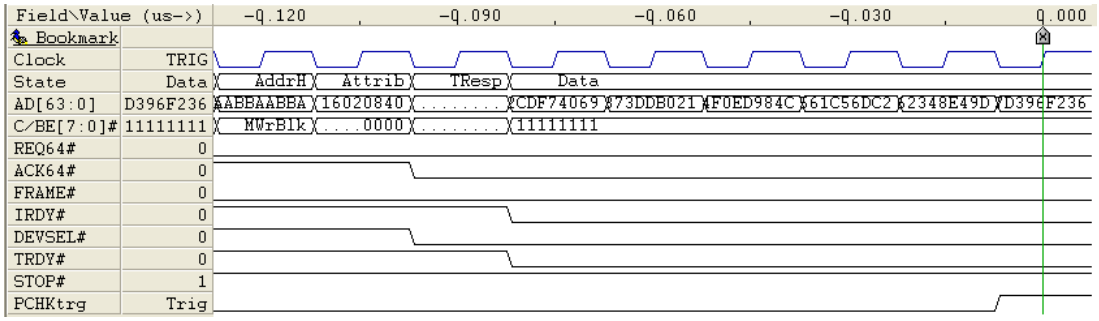



PCI-X Spec. 2.4

Note 1

## AD[63:32] Not High in Attribute Phase

AD[63:32] is reserved and must be driven high by a 64-bit initiator during the Attribute phase.

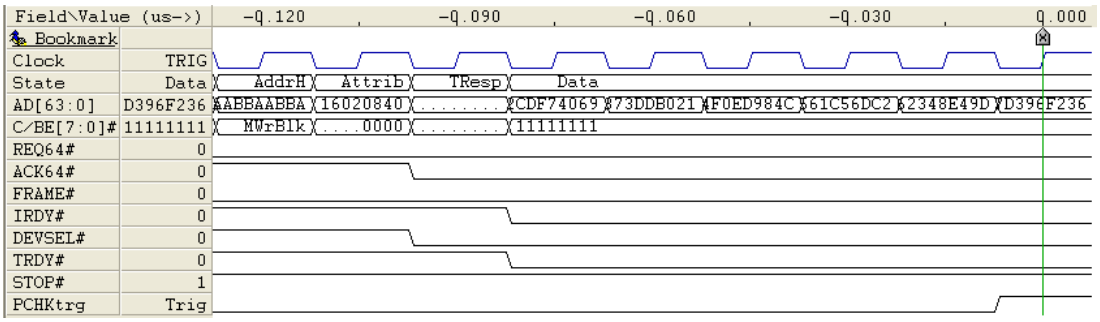



PCI-X Spec. 1.10.1

Rule 3

## C/BE[7:4]# Not High in Attribute Phase

C/BE[7:4]# is reserved and must be driven high by a 64-bit initiator during the Attribute phase.

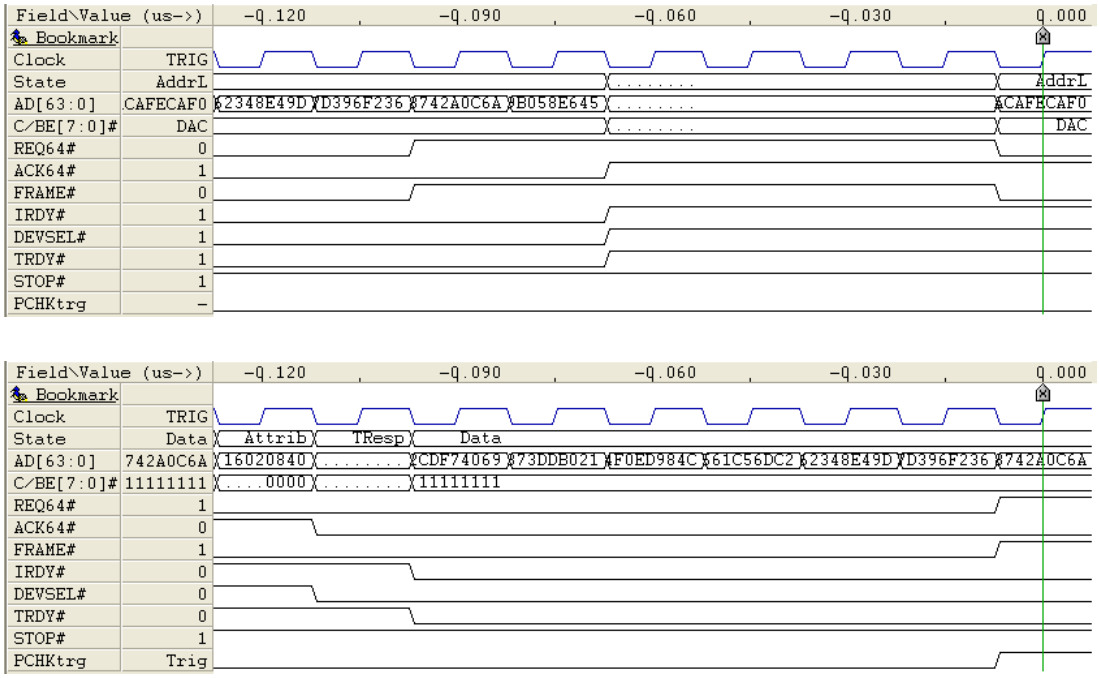



PCI-X Spec. 1.10.1

Rule 3

## C/BE# Not High in Response Phase

C/BE# must be driven high in the first Response Phase clock cycle. For all transactions except Memory Write, C/BE# must also be driven high for all following Response clock cycles.



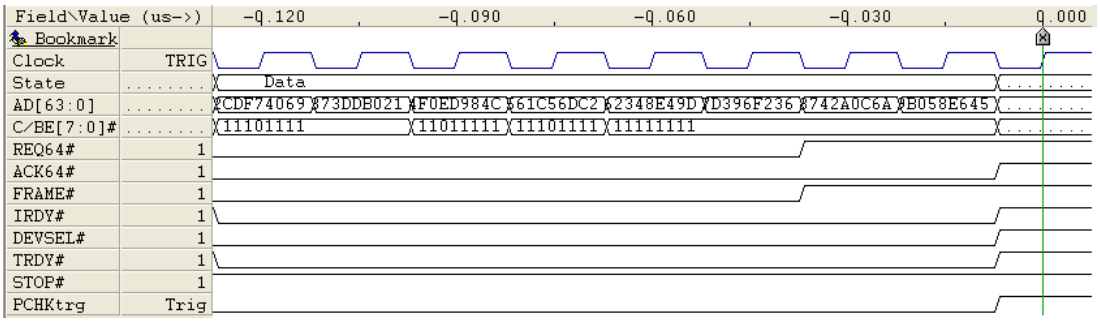

PCI-X Spec. 1.10.1

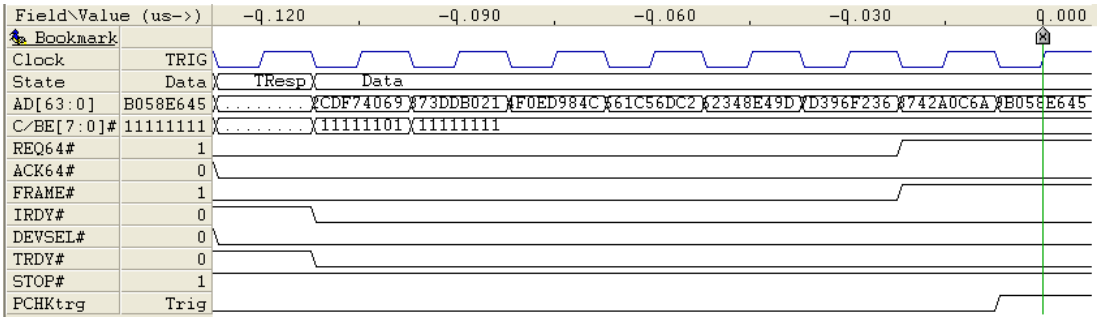
Rule 4 and 11b

PCI-X Spec. 2.6.1

## C/BE# Not High in Data Phase

C/BE# are reserved and must be driven high during Data Phase of all transactions, except Memory Write.



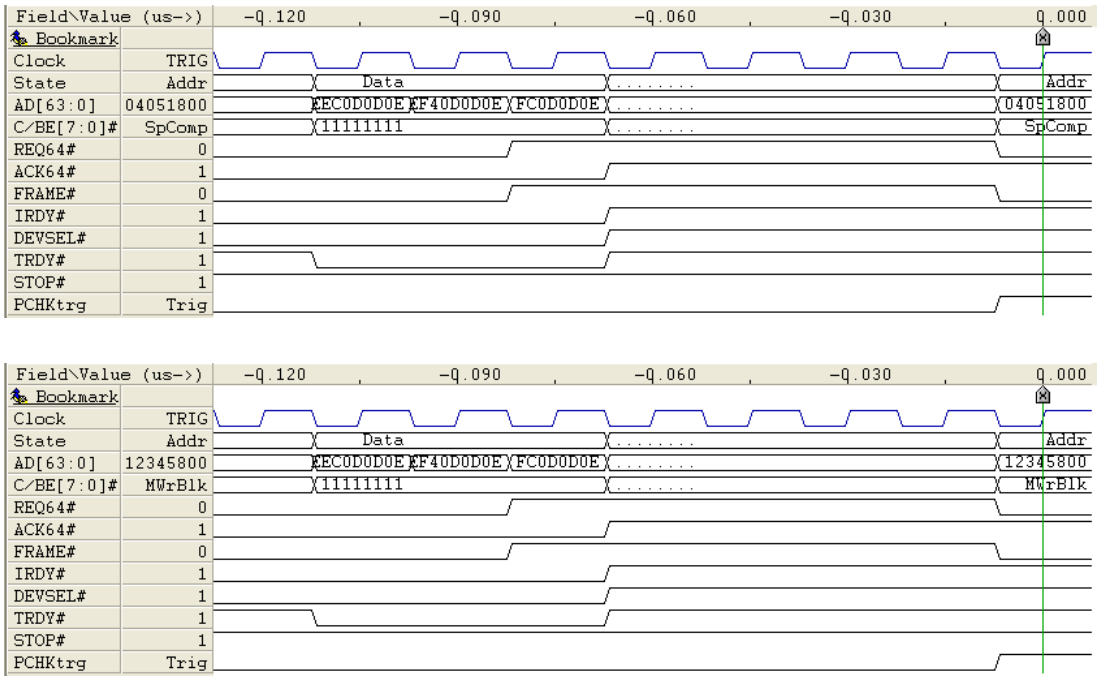




PCI-X Spec. 1.10.1

Rule 11b

## Wait States Not in Pair for Write/Split Completion

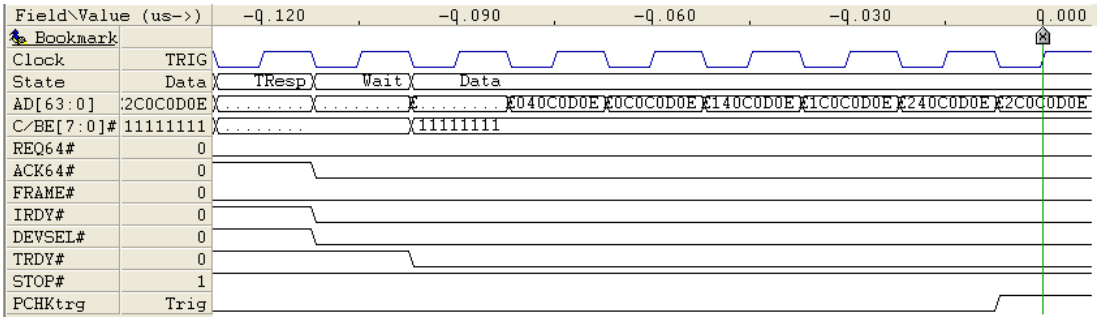
Initial Target wait states are required to be in pairs for all burst write transactions, including Split Completion




PCI-X Spec. 2.9

## Wrong Odd DWORD Data Copy

When starting Memory Write, Memory Write Block, Alias to Memory Write Block or a Split Completion transaction at an odd address (E.g. AD[2]='1'), a 64-bit initiator is required to drive the first addressed DWORD on both AD[31:0] and AD[63:32] the second clock after Attribute Phase.




PCI-X Spec. 1.10.2

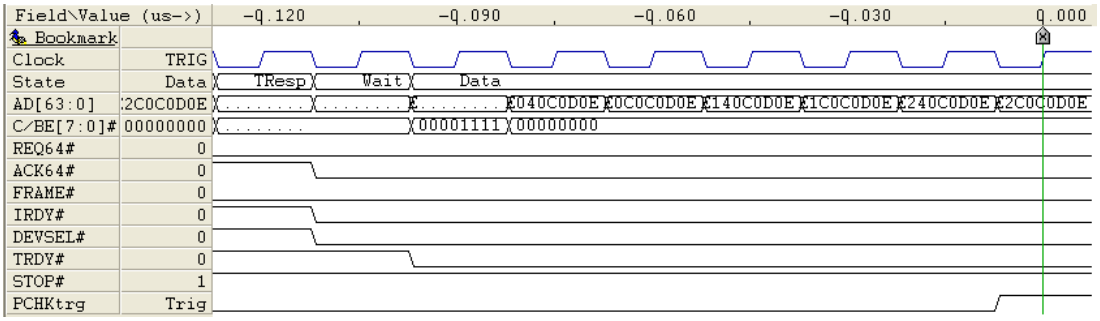
Rule 5

PCI-X Spec. 2.12.3

Rule 5

## Wrong Odd DWORD BE Copy

When starting a Memory Write transaction at an odd address (E.g. AD[2]='1'), a 64-bit initiator is required to drive the first addressed DWORD Byte Enables on both C/BE[3:0]# and C/BE[7:4]# the second clock after Attribute Phase.




PCI-X Spec. 1.10.2

Rule 5

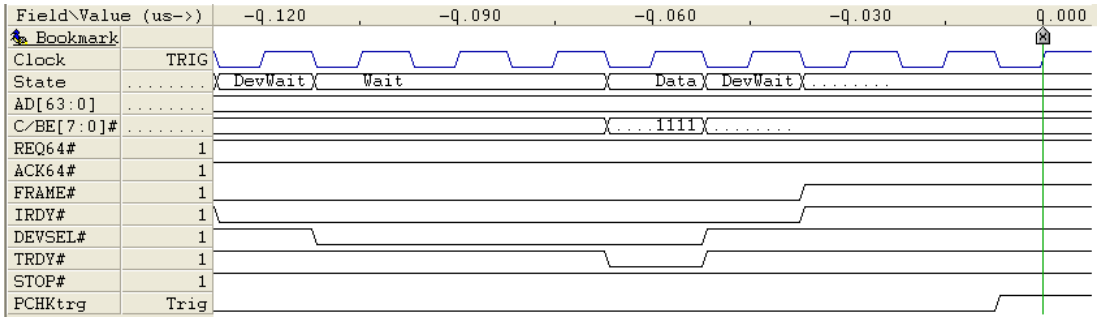
PCI-X Spec. 2.12.3

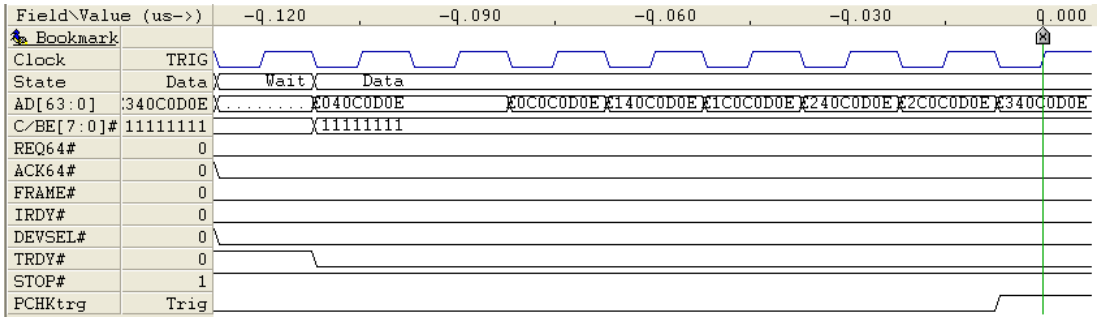
Rule 5

PCI-X Spec. 2.6.1

## Data Change in Target Response Phase

In any write or a Split Completion Transaction, the initiator is required to drive the first set of data from it asserts IRDY# until two clocks after the target asserts DEVSEL#.




PCI-X Spec. 1.10.2

Rule 5

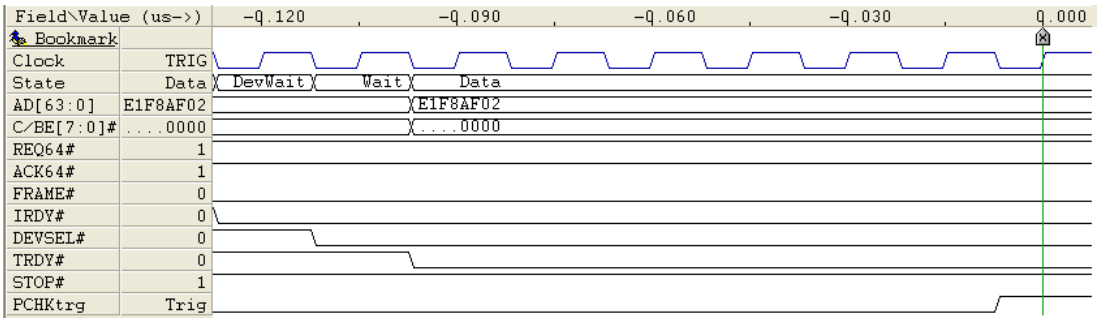
PCI-X Spec. 2.12.3

Rule 5

PCI-X Spec. 2.9.2

## BE Change in Target Response Phases

In a Memory Write transaction, the initiator is required to drive the first set of Byte Enables from it asserts IRDY# until two clocks after the target asserts DEVSEL#.




PCI-X Spec. 1.10.2

Rule 5

PCI-X Spec. 2.12.3

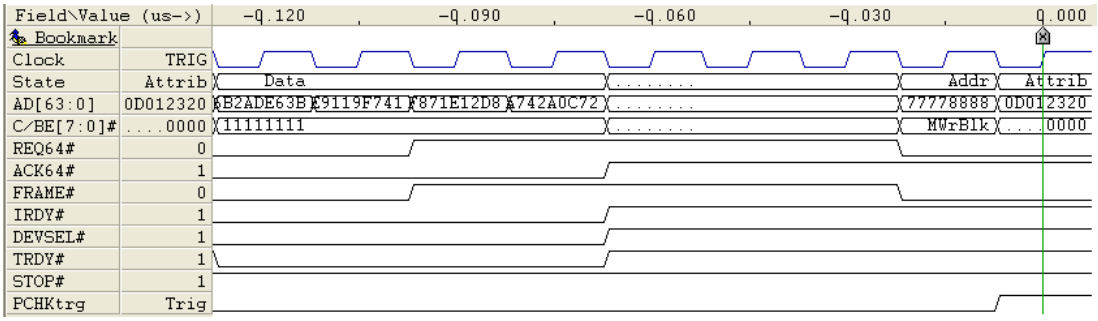
Rule 5

PCI-X Spec. 2.6.1

PCI-X Spec. 2.9.2

## Wait State Data Toggle Error

In Burst Write Cycles and Split Completion transactions, the initiator is required to drive the first set of data on the first, third, fifth, etc clocks if the target inserts wait states. Similarly, the second set of data must be driven on even clocks, as long as the target inserts wait states.



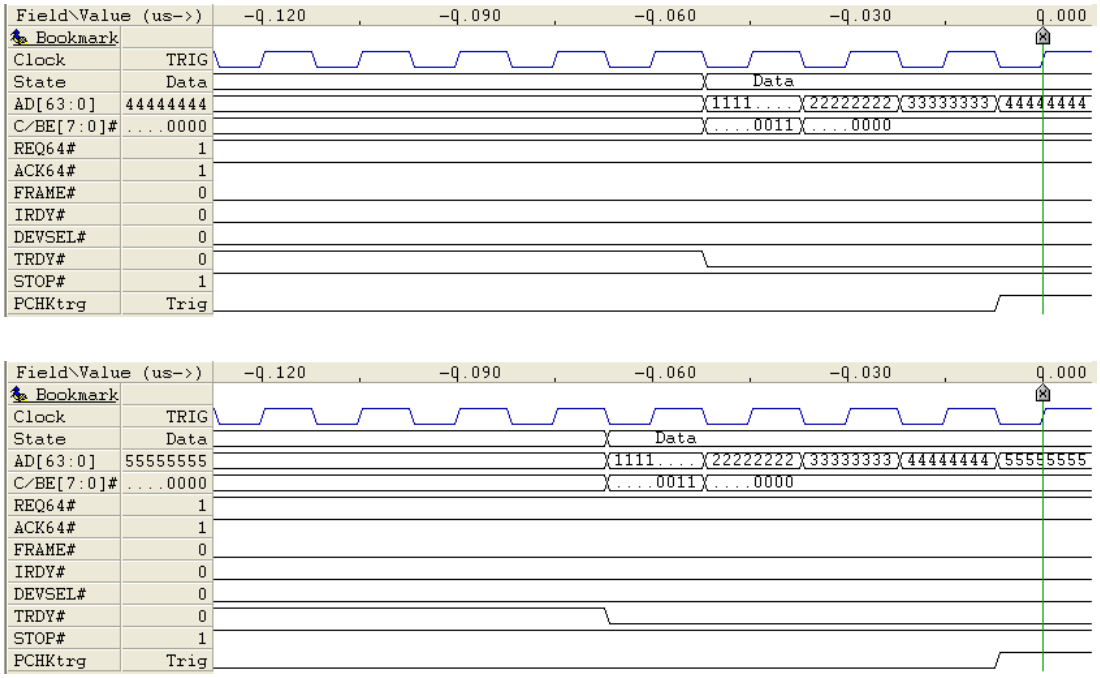

PCI-X Spec. 1.10.2

Rule 5

PCI-X Spec. 2.6.1

## Wait State BE Toggle Error

In Burst Write Cycles, the initiator is required to drive the first set of Byte Enables on the first, third, fifth, etc clocks if the target inserts wait states. Similarly, the second set of Byte Enables must be driven on even clocks, as long as the target inserts wait states.







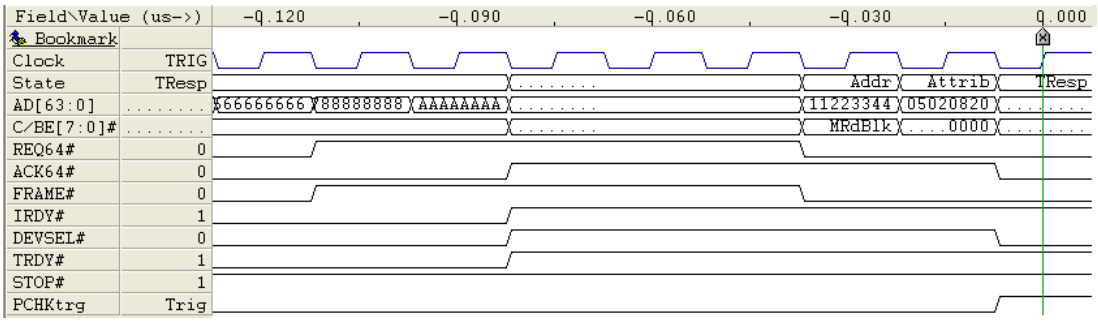
PCI-X Spec. 1.10.2

Rule 5

PCI-X Spec. 2.6.1

## Missing PERR# on High Bus Parity Error

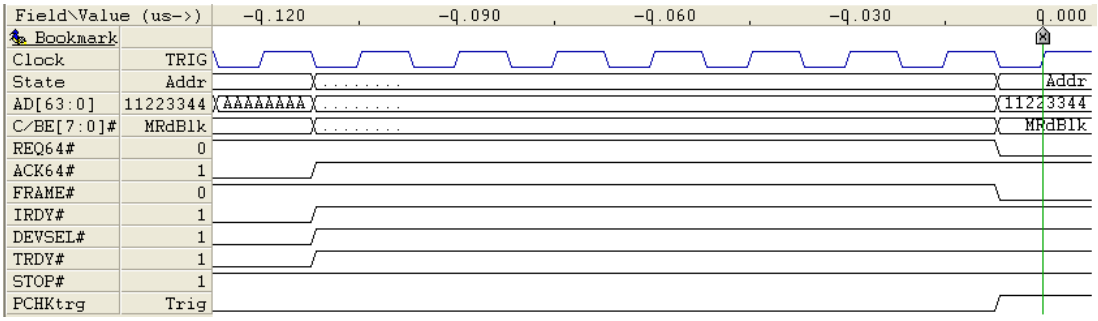
A data parity error is detected on AD[63:32] and C/BE[7:4], but PERR# is not asserted.




PCI-X Spec. 1.10.6

## Missing PERR# on Low Bus Parity Error

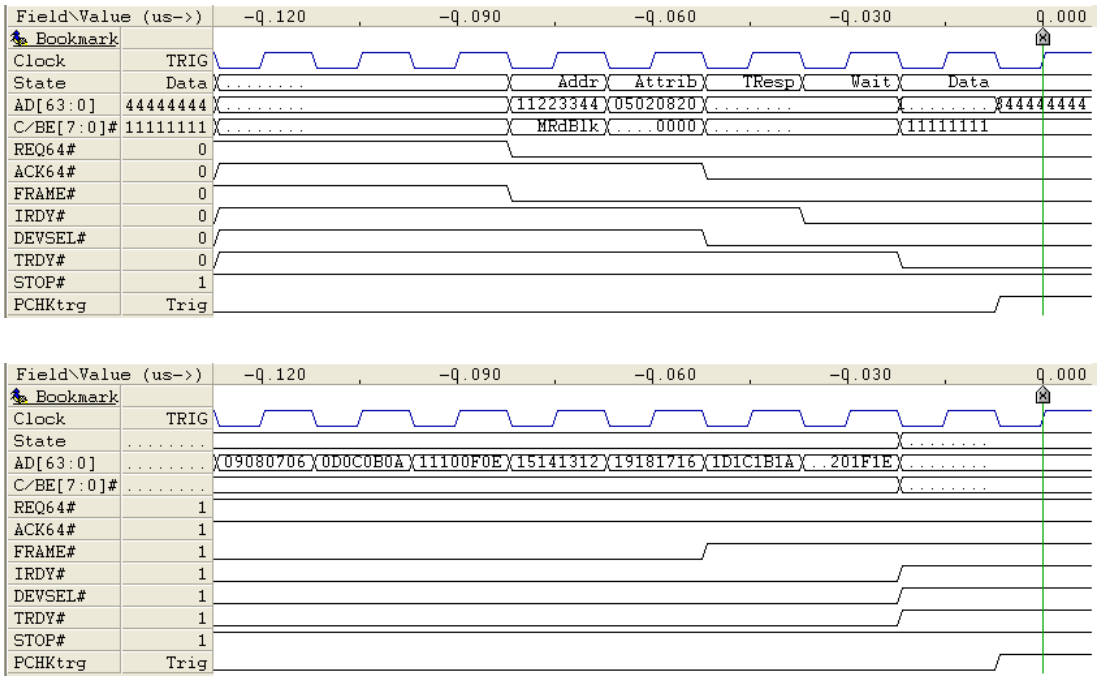
A data parity error is detected on AD[31:0] and C/BE[3:0], but PERR# is not asserted.




PCI-X Spec. 1.10.6

## PERR# Reported When No Parity Error

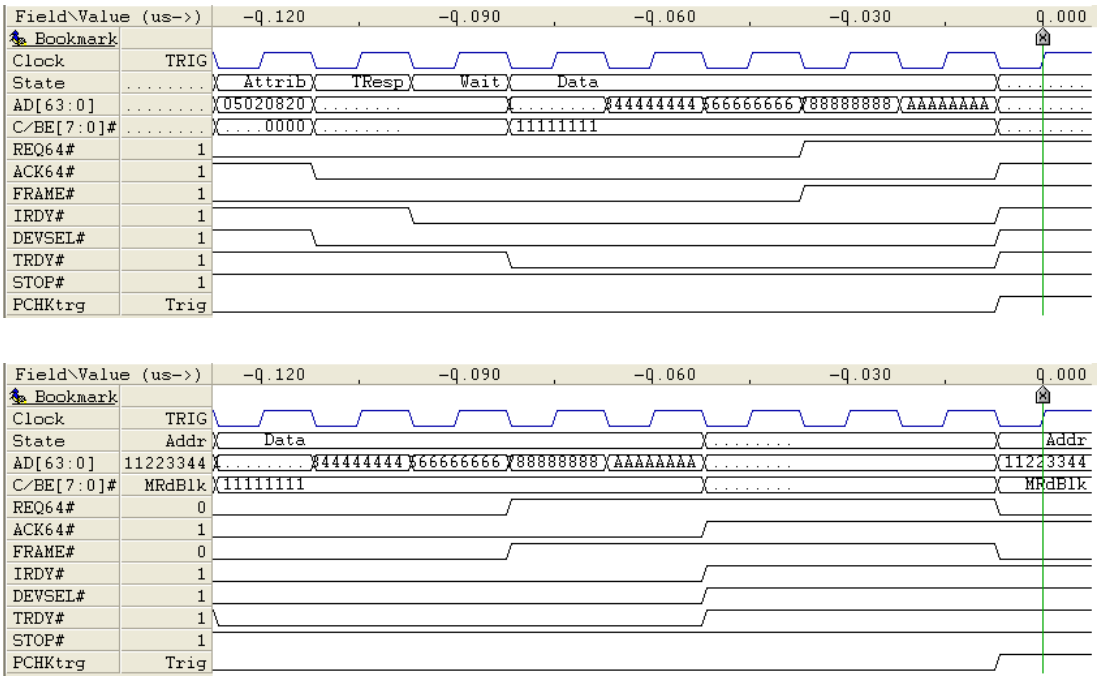
PERR# asserted when no data parity error has occurred.




PCI-X Spec. 1.10.6

## Illegal PERR# Assertion

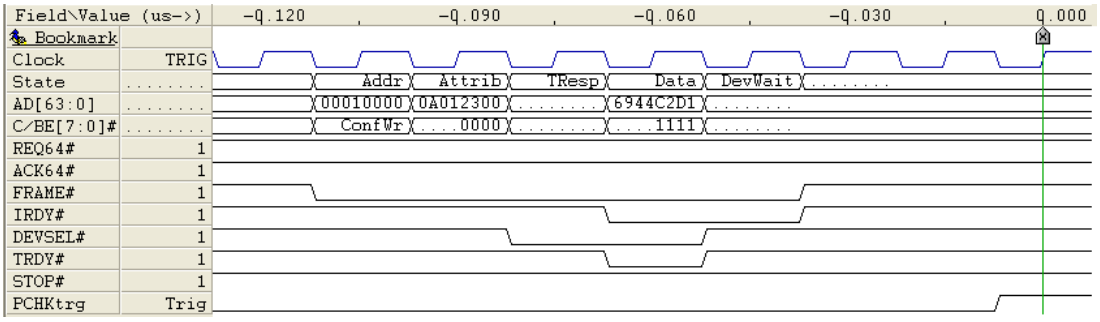
PERR# can only be asserted 3 data phases after the data transfer to which it belongs occurred on the bus.




PCI-X Spec. 5.3

## Address Change in Configuration Cycles

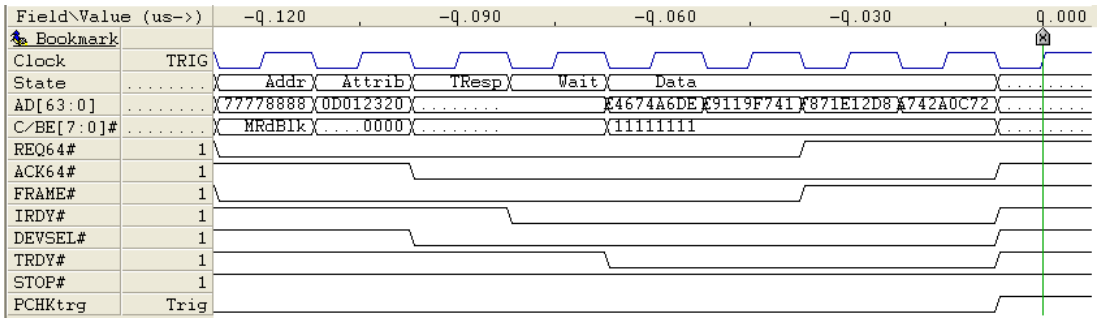
For Configuration Cycles, the initiator is required to start driving the address lines four clocks before asserting FRAME#, and continue driving the same address until the attribute phase.




PCI-X Spec. 1.10.5.1

## Illegal LOCK# Assertion

LOCK# must be asserted the clock after FRAME# is asserted



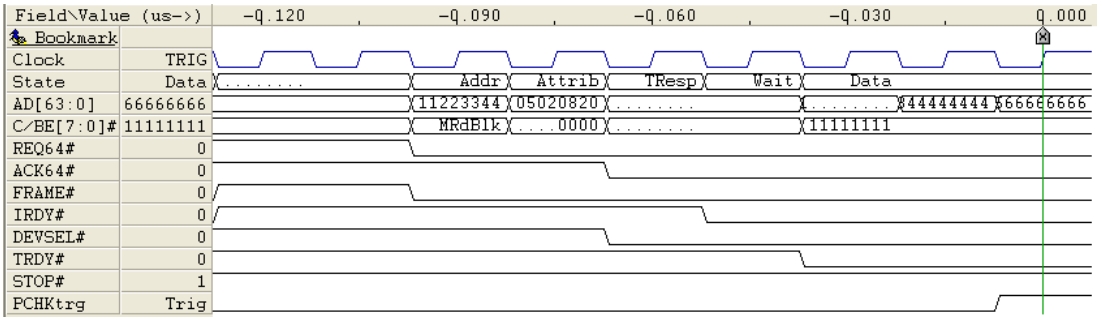

PCI Spec. 2.2: 8.5.1

Rule 3

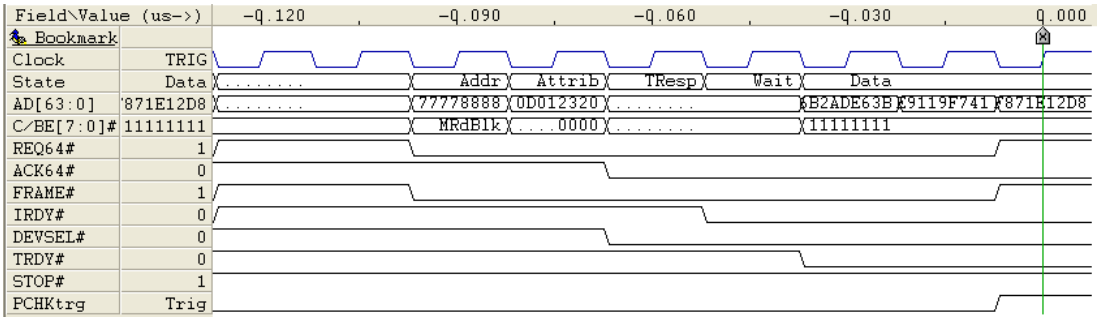
## Illegal LOCK# Deassertion

LOCK# must be deasserted if the initial transaction of the Exclusive transaction is terminated by Retry. LOCK# must also be deasserted if the transaction is terminated with a Master Abort or Target Abort. In cycles where LOCK# is asserted, it must be kept asserted until the transaction completes.

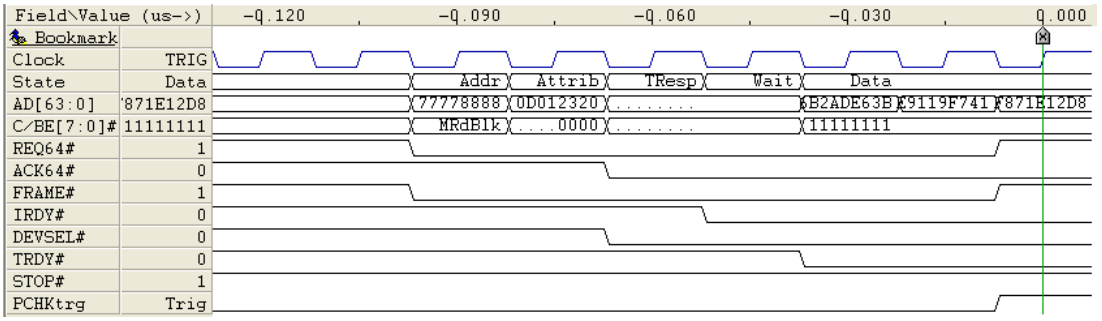
Illegal LOCK# deassertion when transaction terminated with a Retry




Illegal LOCK# deassertion when transaction terminated with a Master Abort




Illegal LOCK# deassertion when transaction terminated with a Target Abort

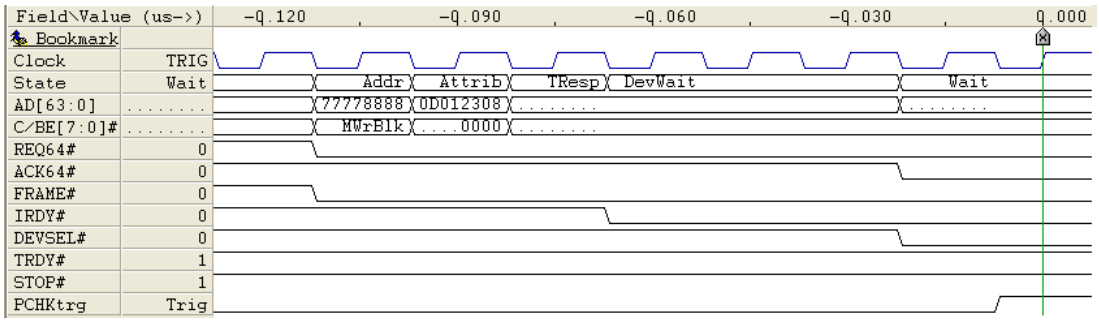



PCI Spec. 2.2: 8.5.1

Rule 5 and 6

## First Transaction on LOCK# Not Read

For exclusive access, the first transaction after LOCK# is asserted must be a Memory Read command.




PCI Spec. 2.2: 8.5.1

Rule 2

*This page intentionally left blank*

## Attribute Phase Reserved Bits Not Zero

Initiators are required to insert zeros for all reserved bits in the attribute phase, as shown in the table below.

Transaction Type Reserved Bits

Configuration Type 0 Bit(31:29)

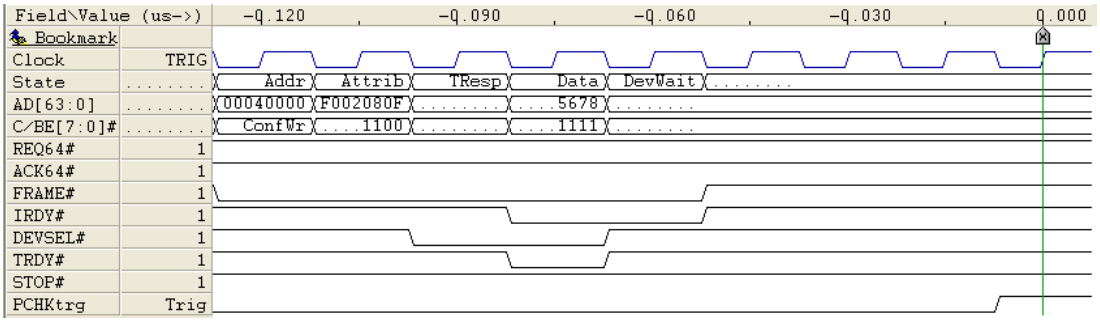
DWORD Bit(31, 7:0)

Split Completion Bit(28:24)

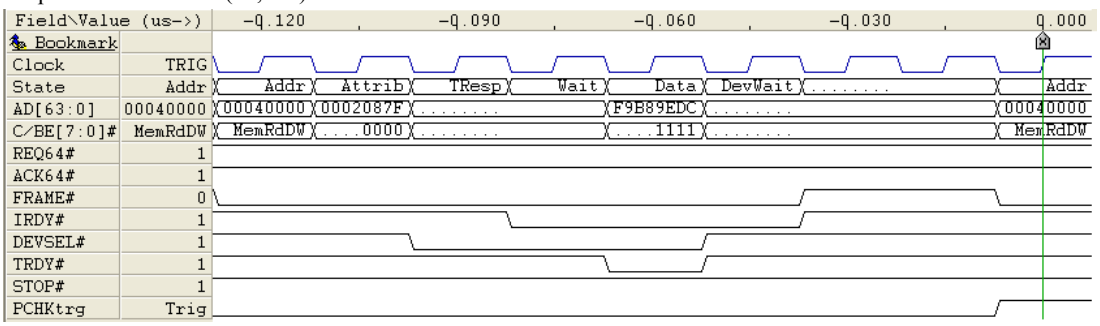
Burst (not Split Completion) Bit(31)

## Attribute Phase Reserved Bits Not Zero (Continued)

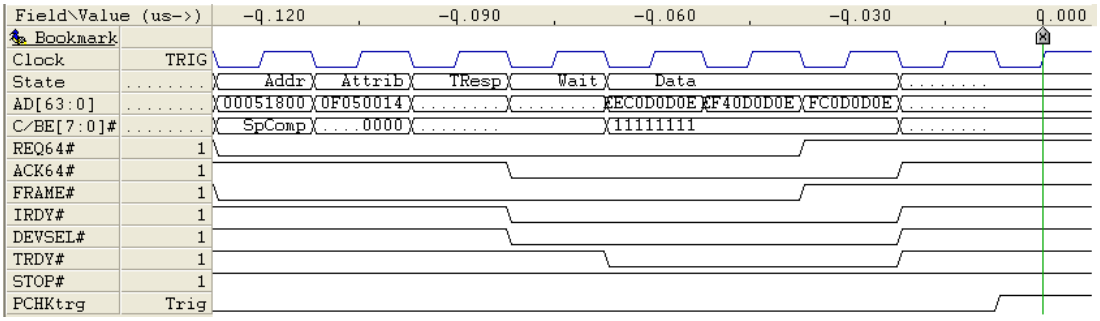
Attribute phase reserved Bits(31:29) not zero in Configuration Type 0 transaction




Attribute phase reserved Bits(31, 7:0) not zero in DWORD transaction

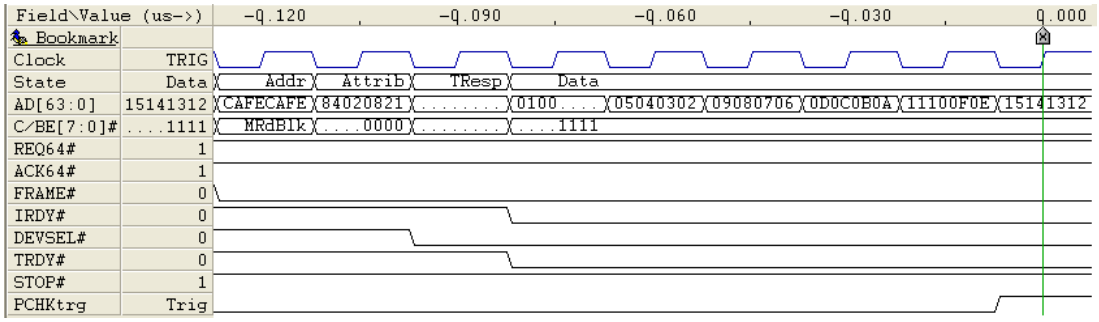



Attribute phase reserved Bits(28:24) not zero in Split Completion



## Attribute Phase Reserved Bits Not Zero (Continued)

Attribute phase reserved Bit(31) not zero in Burst transaction




PCI-X Spec. Figure 2-1, 2-2, 2-12, 2-38, 2-39 and Table 2-4

## Address Phase Reserved Bits Not Zero

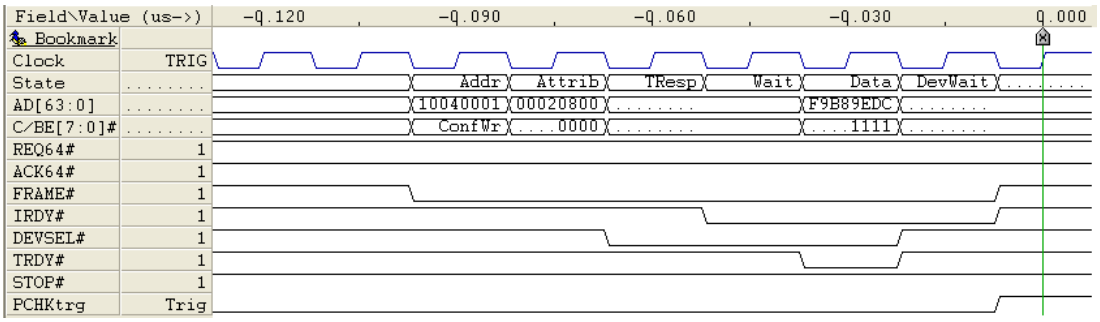
Initiators are required to insert zeros for all reserved bits in the address phase for Configuration type 1 and Split Completion cycles, as shown in the table below:

Transaction Type Reserved Bits

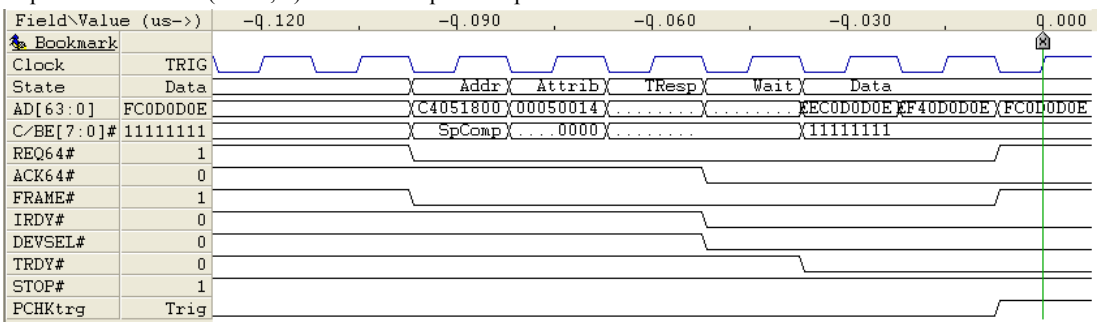
Configuration Type 1 Bit(31:24)

Split Completion Bit(31:30, 7)

Address phase reserved Bit(31:24) not zero in Configuration Type 1 transaction



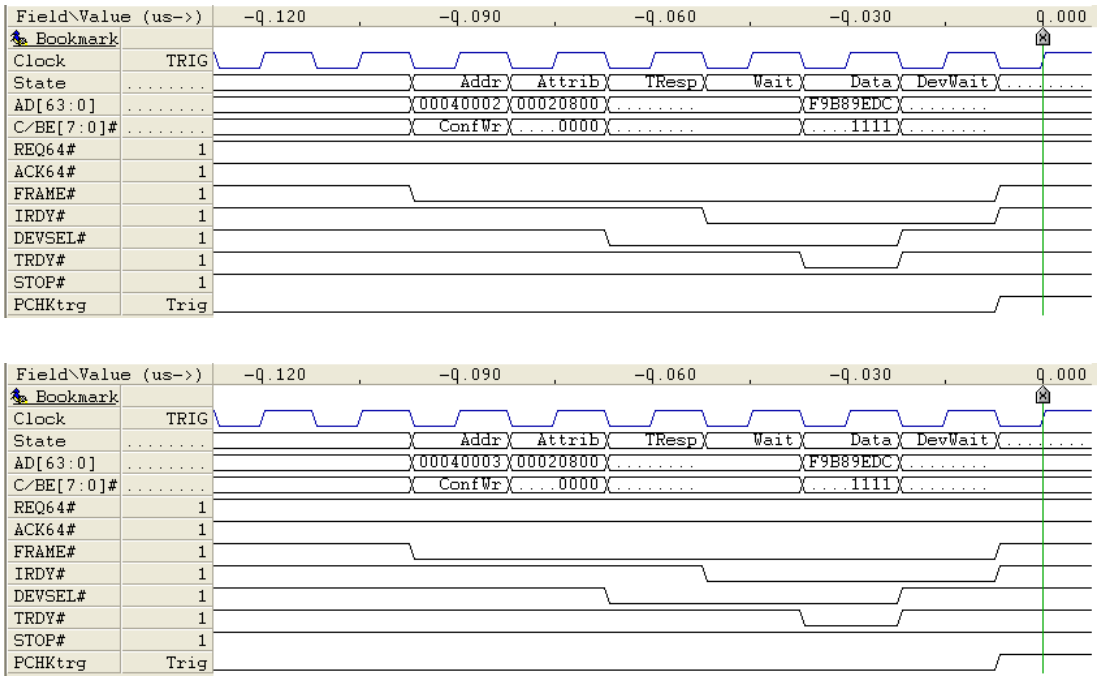

Address phase reserved Bit(31:30, 7) not zero in Split Completion




PCI-X Spec. Figure 2-11, 2-38 and Table 2-4

## Use of Reserved Configuration Type

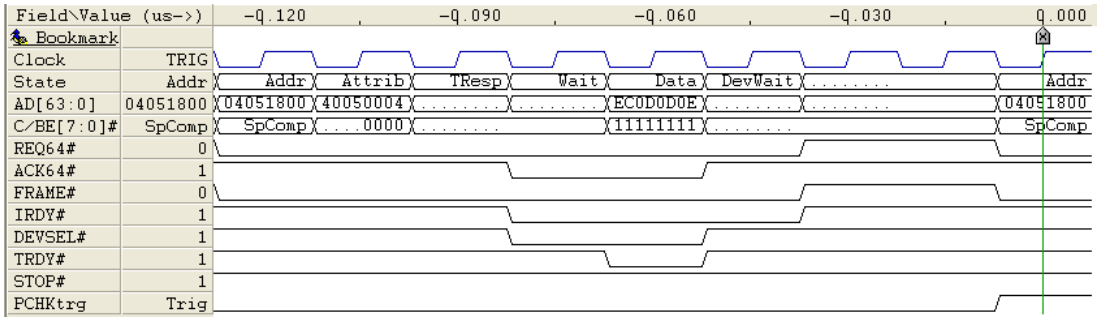
For configuration cycles, only type "00" and "01" are defined. Type "1X" is not defined.




PCI-X Spec. Figure 2-11

## SCE Without SCM

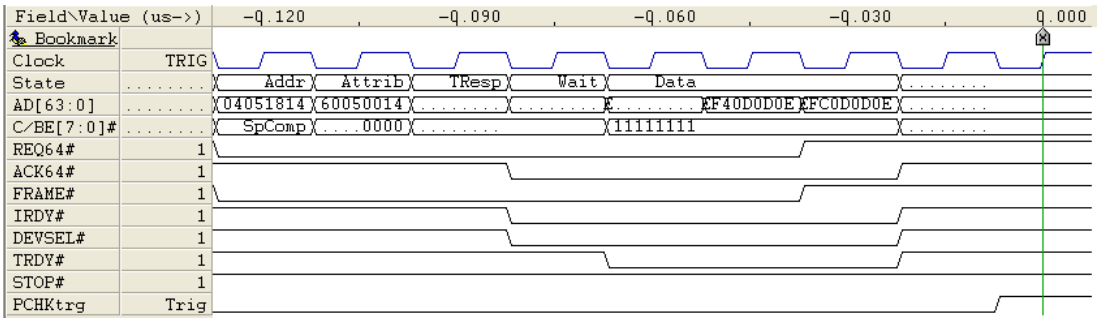
In a Split Completion Error message, the Split Completion Message bit must also be set.




PCI-X Spec. 2.10.3

## SCM Address Not Zero

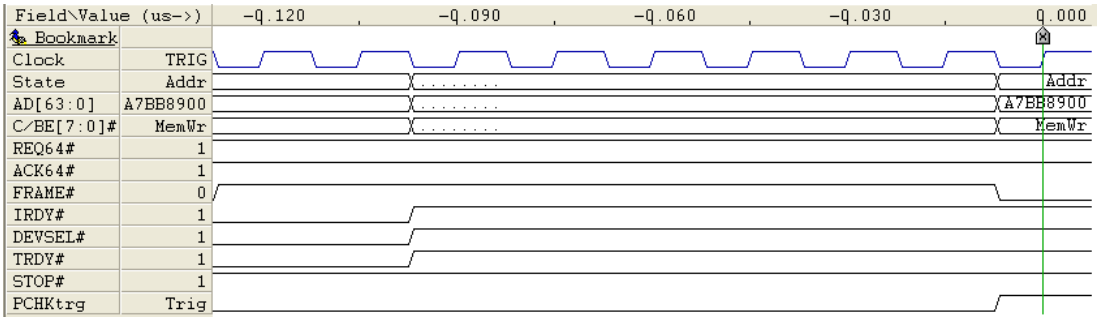
In the address phase of a Split Completion Message, AD[6:0] are required to be zero.




PCI-X Spec. Figure 2-38 and Table 2-8

## Byte Enable Out of Range

Byte Enables asserted for address beyond the end address of last data transfer of Memory Write transactions.



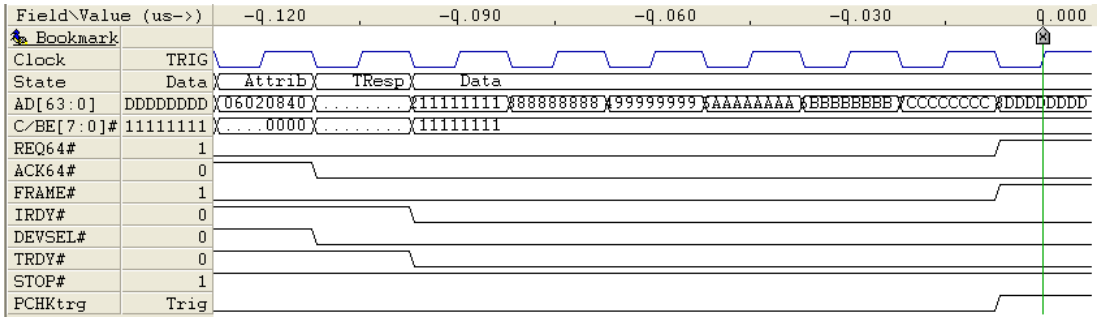

PCI-X Spec. 2.3

## High Address Parity Error

Parity error detected in AD[63:32], C/BE[7:4] and PAR64 in Address cycle.

Note

This is not a violation, but trigger output will be generated to detect such cycles.



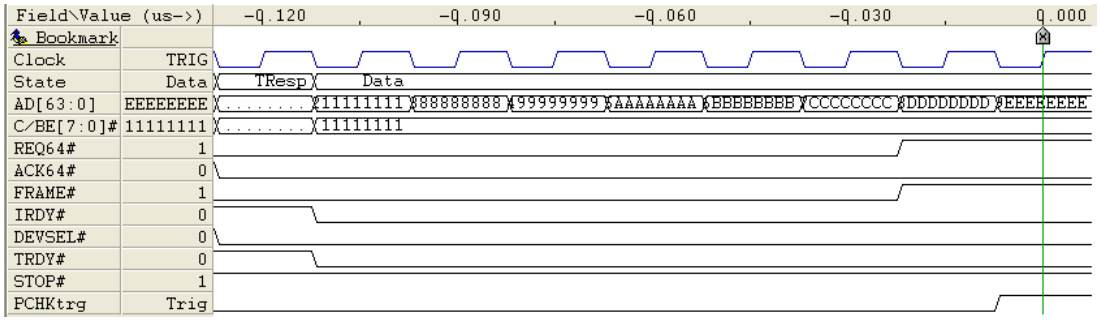

PCI-X Spec. 5

## Low Address Parity Error

Parity error detected in AD[31:0], C/BE[3:0] and PAR in Address cycle.

Note

This is not a violation, but trigger output will be generated to detect such cycles.




PCI-X Spec. 5

## High Data Parity Error

Parity error detected in AD[63:32], C/BE[7:4] and PAR64 in Data cycle.

Note

This is not a violation, but trigger output will be generated to detect such cycles.


PCI-X Spec. 5

## Low Data Parity Error

Parity error detected in AD[31:0], C/BE[3:0] and PAR in Data cycle

Note

This is not a violation, but trigger output will be generated to detect such cycles.

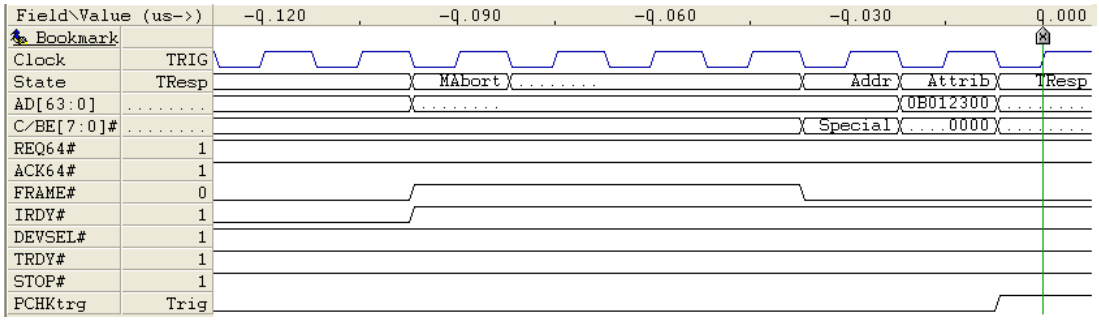

PCI-X Spec. 5

## Master Abort

A cycle was terminated with Master Abort.

Note

This is not a violation, but trigger output will be generated to detect such cycles.




PCI-X Spec. 1.10.2

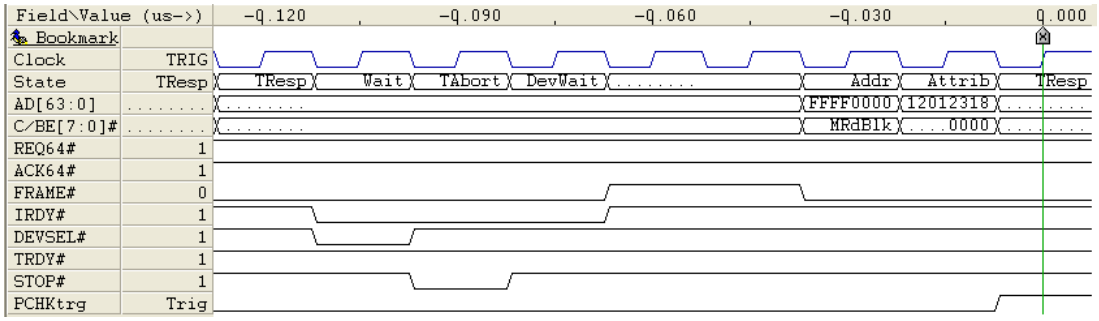
Rule 4

## Target Abort

A cycle was terminated with Target Abort.

Note

This is not a violation, but trigger output will be generated to detect such cycles.



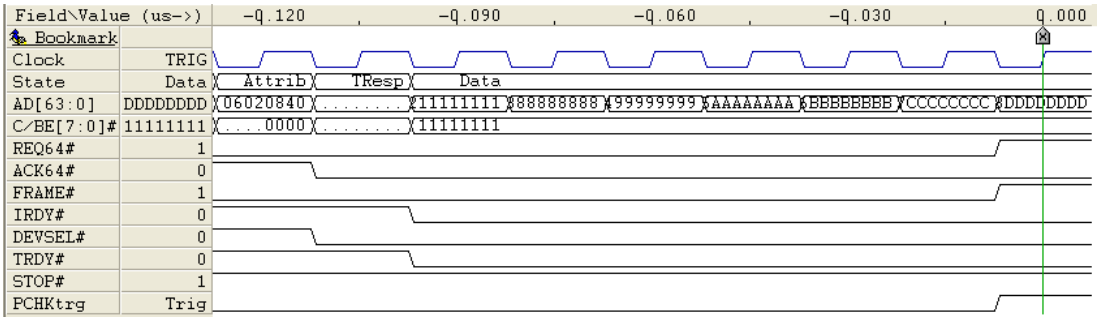

PCI-X Spec. 2.11.2.5

## SERR# Asserted

A device has asserted System Error (SERR#)

Note

This is not a violation, but trigger output will be generated to indicate that SERR# is/was active on bus.




PCI-X Spec. 1.10.6

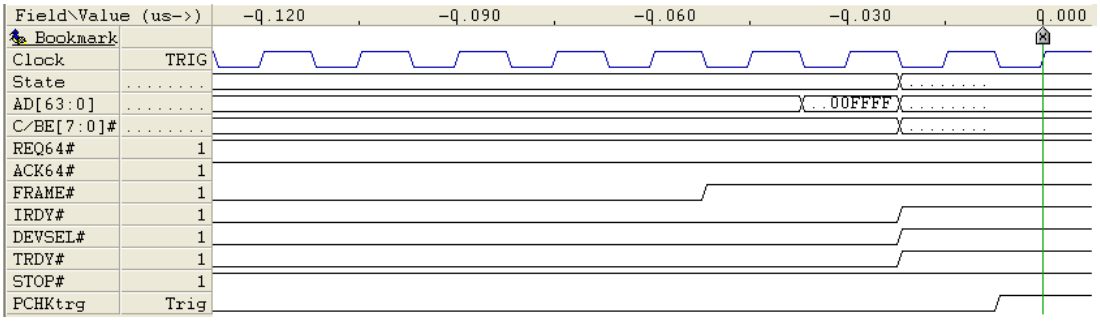
Rule 4

## PERR# Asserted

A device has asserted Parity Error (PERR#).

Note

This is not a violation, but trigger output will be generated to indicate that PERR# is/was active on bus.




PCI-X Spec. 1.10.6

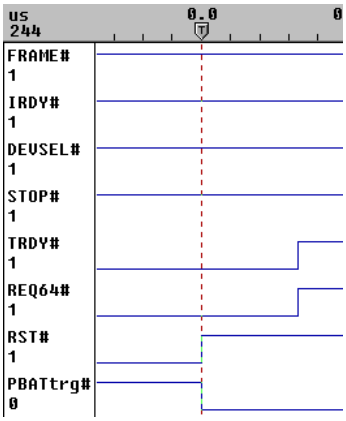
Rule 1

## RST# Asserted

PCI Reset Occurred (RST# Asserted).

Note

This is not a violation, but trigger output will be generated to indicate that RST# is/was active on bus.




PCI Spec. 2.2.1

# 7 Exerciser

The Exerciser is used to generate cycles with known characteristics on the PCI bus.

- Introduction
- Operation
- Executing Commands
- Configuration Scan
- Using Scripts
- Exerciser Script Commands

Use of the Exerciser requires the “VG-E” license to have been purchased. See “Ordering Information” on page 427.

## 7.1 Overview

### Features

Some options require the Enhanced Exerciser "VG-E2" license. See "Ordering Information" on page 427.

> **E2 Only –** Information that pertains to the Enhanced Exerciser will be described in a box like this one.

**Emulation**

The Exerciser can be used in conjunction with, and concurrently with, the Analyzer; providing a test environment in which it is possible to emulate any add-in board and get the same analysis of bus traffic and control issues as you would using the actual add-in board.

The Exerciser contains a target interface that has its own address decoder for a user defined address window which can respond to accesses from another module.

**Scripting**

A built-in script engine allows test scripts to be created with a convenient record and playback function. Scripts are recorded by manually running through the various commands of the Exerciser. Scripts can be stored and edited as text files where each script can consist of sequences of bus cycles of any kind, with varying sizes, cycle types, etc. The script playback function can be set to run single, multiple or infinite playbacks, while the Analyzer may perform bus monitoring in the background.

**DMA Transfers**

The Exerciser has the ability to act as a target or as an initiator with small or large DMA transfers. Since the DMAs run in the background, the DMA command can be used to produce bus traffic, while other Exerciser commands are available for other purposes. It is also possible to set up a DMA that transfers data between two other agents.

**Target Memory (PCI)**

The Exerciser gives access to I/O memory (256 bytes) and target memory (8 MB in the E version, and 8KB in the E2 version). The Exerciser supports both 32 bit addressing and 64 bit addressing (using Dual Address Cycles) and 8, 16, 32, and 64 bit data which can be located anywhere in the address map. Data can be written to and read from this memory as single cycles or as zero-wait-state burst cycles (after initial latency). This way the Exerciser can emulate a device or PCI/PCI-X board.

### Generate Interrupts

The Exerciser can generate any of the four PCI/PCI-X Interrupts.

### Scan PCI/PCI-X Config Space

The Exerciser can do a complete scan of PCI bus configuration space. It systematically probes for all possible devices on the bus, and if it finds a PCI-to-PCI bridge, it probes through the bridge for all devices on any bus behind the bridge.

PCI-to-PCI bridges only allow downstream configuration transactions. Consequently, the Vanguard can only see devices on the same bus segment as itself, and below. It can not see upstream in the bus hierarchy. If the Vanguard is not on the highest level in the hierarchy, usually bus 0, you may have devices in the system which are not visible to the Vanguard.

The devices found are displayed with parameters such as Config Space address, Class Code, Device/Vendor ID, Prefetchable Memory, Memory and I/O target windows, which bus the device is found on, etc.

A scan of PCI bus configuration space is useful to get an overview of the devices in the system, i.e. to find where already enabled targets are in PCI memory, and to get the information needed to manually configure and enable the devices found.

## 7.2 Introduction

The main features of the Exerciser

- Provides real PCI cycles with nominal bus timing.
- Can be used in conjunction with the Vanguard Analyzer, Protocol Checker and Statistics.
- A script engine is included.
- Can emulate any PCI board and has 8 MByte target memory available for use.

**E2 Only –** The E2 Exerciser has 8KBytes of target memory.

- DMA transfers can be set up with the Exerciser acting as a “target” or an “initiator”.
- Can generate any PCI interrupts.
- Can scan the PCI configuration space.

**E2 Only –** The E2 Exerciser can generate Protocol Violations.

The Exerciser is useful for:

- Assist in the debugging of boards by sending signals to a board under inspection, and monitor it’s behavior.
- Inspect and patch data in memory without interfering with the operation of other masters.
- Testing behavior during violations.

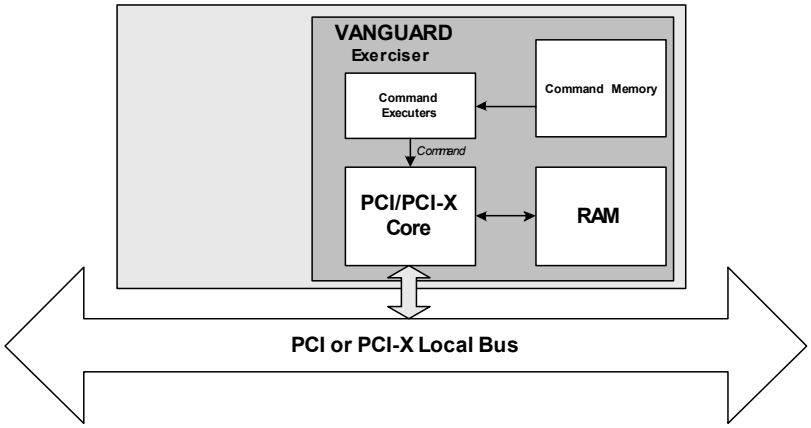



**FIGURE 7-1** *Exerciser Block Diagram*

**Command Memory** This area contains the list of commands that will be executed onto the PCI bus.

**Command Executer** Controls the PCI Core. It reads commands from the Command Memory and sends status reports back to BusView.

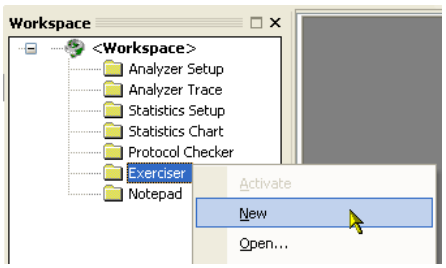
**PCI/PCI-X Core** These are the FPGA images used to act as a PCI or PCI-X Exerciser.

**RAM** This memory area is for use in DMA transfers and in mapping target memory.

**E2 Only –** The E2 Exerciser uses FPGA internal memory only.

## 7.3 Operation

The Exerciser is disabled by default when BusView is installed for the first time. To access the “Exerciser Function/Bus Mode.” options click on “Options” in the Tools, Hardware menu.




Start the Exerciser by right-clicking on the Exerciser folder in the Workspace Window and clicking on New, or click on Activate in the Exerciser menu item.

**E2 Only –** The E2 Exerciser is activated from the Options dialog in the Hardware Menu.

### Exerciser Window

The Exerciser Window provides a command line interface to issue commands, and an area where the results of issued commands are displayed.

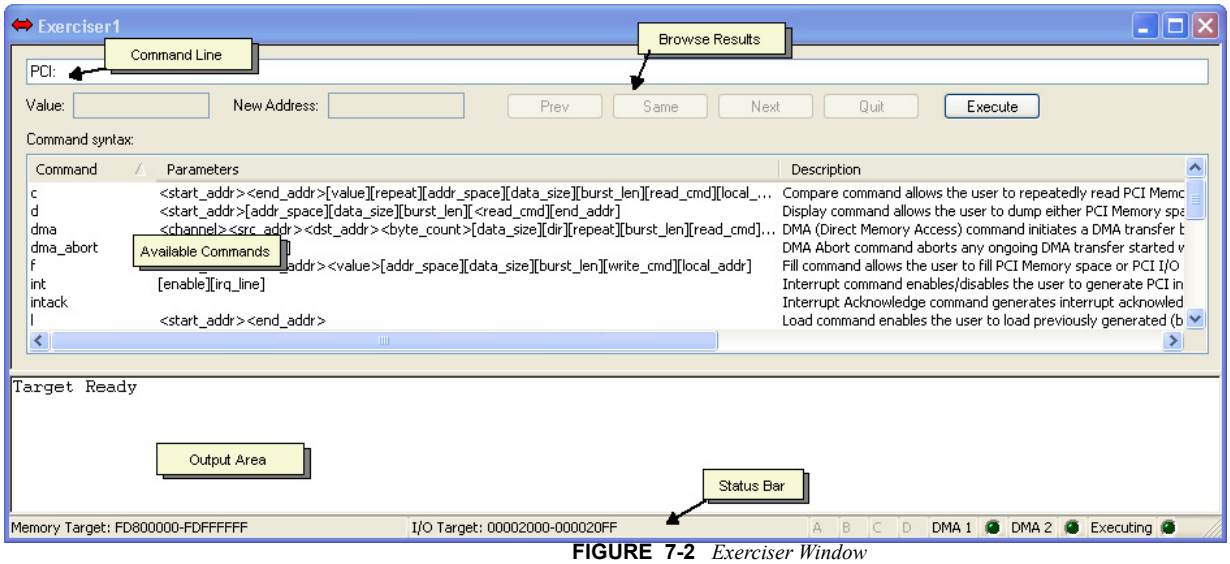
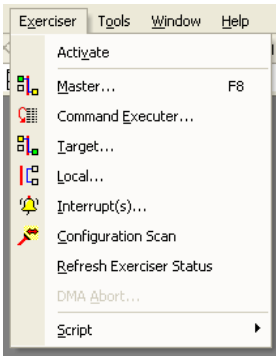



**FIGURE 7-2** *Exerciser Window*

## The Exerciser toolbar and menu




As an alternative to executing Exerciser commands from the Command Line, the Exerciser Menu and toolbar can be used to define and execute commands

**Activate** Enables the Exerciser

**Master** Opens the Exerciser Master Commands dialog.

**Command Executer** Opens the Command Executer dialog.

**Target** Opens the Exerciser Target dialog.

**Local** Opens the Exerciser Local Commands dialog.

**Interrupts** Opens the Interrupt dialog.

**Configuration Scan** Executes the Scan command.
Scan can also be executed from the command line by typing "`scan`".

**Refresh Exerciser Status** Get an updated Exerciser Status.

**DMA Abort** Abort any DMA operation currently in progress.

**Script** Use the Exerciser Scripting features. See "Using Scripts" on page 278.

## Help

The Exerciser provides several ways of getting help.

- When typing a command in the command line, the text color will change. When the text is red, this means that the command does not yet have all necessary parameters to be executed.
- Context sensitive help. Type a command in the Exerciser window, and press Ctrl-F1 keys. The on-line help opens at the page describing the command.
- Type "h" or "?" followed by a command, and a list of arguments appears in the Exerciser window.

Use the on-line help manual available by pressing the help button at the tool bar.

## Exerciser Options

**E2 Only –** E2 options are listed in "E2 Exerciser Options (Enhanced Exerciser)" on page 225

Right-click on the Exerciser Window and click Options to open the Exerciser Options Window.

### Performance Options

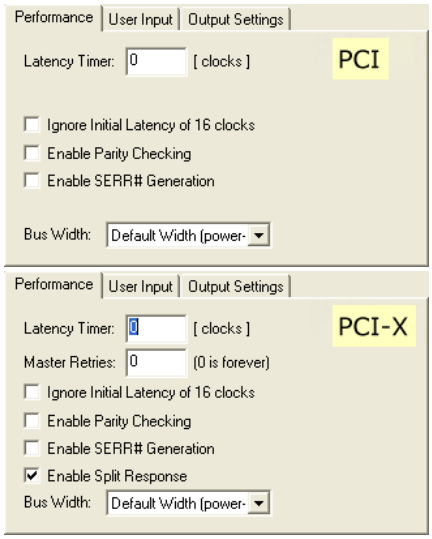



**FIGURE 7-3** *Exerciser Performance Options*

- Latency Timer - Number of clocks (0 to 255) remaining between losing GNT and having to release the bus if another device is requesting the bus.
- Master Retries - PCI-X option.
- Ignore Initial Latency of 16 Clocks - The Exerciser will continue to issue wait states until it has data ready.
- Enable Parity Checking - Checks parity on bus transfers if enabled.
- Enable #SERR Generation - Allows the Exerciser to generate SERR# on the bus
- Enable Split Response - PCI-X option.

- Bus Width -
  - Default Width - Use the bus width as detected on power-up.
  - Force 32 Bit Operation - Only 32 bit address and data cycles are issued.
  - Force 64 Bit Operation - Only 64 bit address and data cycles are issued.

These options can also be set via the command line prompt with the *opt* command:
opt <enable>[latency_timer] [serr][parity][bus_width] [ignore_init_lat]
with [split_resp_en] and [master_retry] also available for PCI-X only.

| **Argument** | | | **Description** | **Default** |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| enable | | BOOL | 1= Enable Exerciser<br>0= Disable Exerciser | 0 |
| | latency_timer | number | 0-255 time in nanoseconds | 0 PCI, 64 PCI-X |
| | serr | BOOL | 1= Enable System Error driver | 0 |
| | parity | BOOL | 1= Enable Parity Error response | 0 |
| | bus_width | number | 0=Deafult width (set on PCI reset)<br>1= Force 32 bit operation<br>2= Force 64 bit operation | 0 |
| | ignore_init_lat | BOOL | 1= Ignore initial latency | 0 |
| | split_resp_en | BOOL | PCI-X ONLY<br>1= Enable Split Response on target reads. | 1 |
| | master_retry | number | PCI-X ONLY<br>Number of retries before Exerciser as a master terminates the cycle (0-255)<br>0= Forever | 255 |

**User Input**

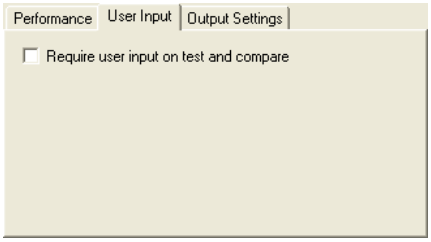



**FIGURE 7-4** *Exerciser User Input Options*

Require user input on test and compare -

- If this is enabled and a Test command is running; if an error occurs then testing will stop and wait until Next is pressed before continuing.
- If disabled and a Test command is running; if an error occurs then testing will continue. Errors are displayed in the Output Log.

### Output Settings

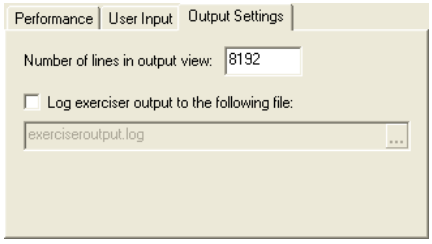
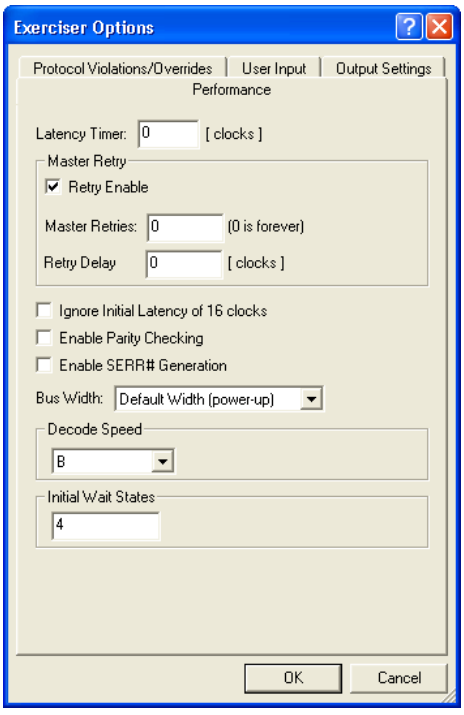



**FIGURE 7-5** *Exerciser Output Settings Options*

- Number of lines in output to view - states the maximum number of lines to be viewed in the Output Log. When the maximum is reached, the oldest line is removed when a new line is created.
- Log exerciser output to the following file: - Selecting this will allow you to specify a file in which all Exerciser output will be saved.

## E2 Exerciser Options (Enhanced Exerciser)

### Performance Extended



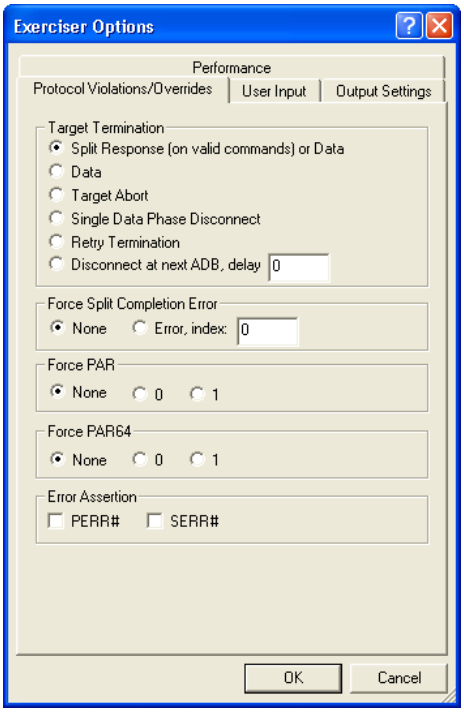

- Latency Timer - Number of clocks (0 to 255) remaining between losing GNT and having to release the bus if another device is requesting the bus.

- Master Retry
  - Enable / Disable Master Retry.
  - Master Retries - The number of Retries to perform before the Retry Counter expires.
  - Retry Delay - The number of clocks between each retry attempt.
- Ignore Initial Latency of 16 Clocks - The Exerciser will continue to issue wait states until it has data ready.
- Enable Parity Checking - Enables/Disables Parity Checking
- Enable #SERR Generation - Allows the Exerciser to generate SERR# on the bus
- Bus Width -
  - Default Width - Use the bus width as detected on power-up.
  - Force 32 Bit Operation - Only 32 bit address and data cycles are issued.
  - Force 64 Bit Operation - Only 64 bit address and data cycles are issued.
- Decode Speed - Specify Target to support decode speed B, C and subtractive.
- Wait States - Set the Target's number of initial wait states. Minimum is 4 and maximum is 15

Some options can be set via the command line prompt with the *opt* command, other are set with the *optx* command:
opt <enable>[latency_timer] [serr][parity][bus_width] [ignore_init_lat] [master_retry] .

| **Argument** | | | **Description** | **Default** |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| enable | | BOOL | 1= Enable Exerciser<br>0= Disable Exerciser | 0 |
| | latency_timer | number | 0-255 time in nanoseconds | 64 |
| | serr | BOOL | 1= Enable System Error driver | 0 |
| | parity | BOOL | 1= Enable Parity Error response | 0 |
| | bus_width | number | 0=Deafult width (set on PCI reset)<br>1= Force 32 bit operation<br>2= Force 64 bit operation | 0 |
| | ignore_init_lat | BOOL | 1= Ignore initial latency | 0 |
| | master_retry | number | Number of retries before Exerciser as a master terminates the cycle (0-255)<br>0= Forever | 255 |

**Protocol Violations/Overrides**




- Target Termination - Select the type of termination that the Target should terminate with.
  - SpResp = Split response on legal commands, else Data termination.
  - Data = Always data termination.
  - TAbort = Target Abort.
  - SDtaPhD = Single Data Phase Disconnect.
  - Retry = Retry.
  - DNxtADB = Disconnect on Next ADB.
- Force Split Completion Error - When enabled, completer will always respond with a split completion error message. Split completion error message can be specified.
- Force PAR - Forcing the PAR signal high or low will effectively generate random parity errors on the lower AD and C/BE bus.
- Force PAR64 - Forcing the PAR64 signal high or low will effectively generate random parity errors on the upper AD and C/BE bus.
- Error Assertion - Manually asserts PERR# and SERR# signals.

These options can also be set via the command line prompt with the *optx* command:
optx <enable> [retry_enable] [retry_delay] [decode_speed] [wait_states] [target_term] [SpComp_err] [DNxtADB_delay] [PAR_gen] [PAR64_gen] [PERR_asset] [SERR_asset]

| **Argument** | | | **Description** | **Default** |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| enable | | BOOL | 1= Enable Exerciser, 0= Disable Exerciser | 0 |
| | retry_enable | BOOL | 1 = Enable retry, 0 = Disable retry | |
| | retry_delay | integer | Number of clocks to delay before master retry, 0 - 65535. | 0 |
| | decode_speed | hex number | Number of clocks from last address phase to target asserts DEVSEL#.<br>3 or B = decode speed B.<br>4 or C = decode speed C.<br>5 = Invalid decode speed according to the PCI-X spec.<br>6 or SUB = decode speed Subtractive.<br>7 = Invalid decode speed according to the PCI-X spec. | B |
| | wait_states | char | Force number of initial wait states.<br>None = None forced wait states.<br>4-15 = Number of wait states. | 4 |
| | target_term | char | Force target termination<br>SpResp = Split response on legal commands, else Data termination.<br>Data = Always data termination.<br>TAbort = Target Abort.<br>SDtaPhD = Single Data Phase Disconnect.<br>Retry = Retry.<br>DNxtADB = Disconnect on Next ADB. | SpResp |
| | SpComp_err | char | None = No force of split completion error.<br><number> = Split completion error message index in hex (range 0-FF) | None |
| | DNxtADB_delay | integer | Number of clocks from DEVSEL# before Disconnect on Next ADB is signaled 1- 4095 | 1 |
| | PAR_gen | char | Force PAR.<br>None = Normal parity generation on PAR.<br>0 = Force PAR to 0<br>1 = Force PAR to 1 | None |
| | PAR64_gen | char | Force PAR64.<br>None = Normal parity generation on PAR64.<br>0 = Force PAR to 0<br>1 = Force PAR to 1 | None |
| | PERR_assert | BOOL | Assert PERR on 2. clock after each data phase.<br>0 = Normal PERR generation<br>1 = PERR always asserted | 0 |
| | SERR_assert | BOOL | Assert SERR on 2. clock after each address phase.<br>0 = Normal SERR generation<br>1 = SERR always asserted | 0 |

## 7.4 Executing Commands

### Executing Single commands

There are 2 ways of executing commands on the Exerciser:

- By typing commands at the command line prompt in the Exerciser Window.
- By using the dialog boxes available from the Script, Master, Target, Interrupt, Local, and the Options menus.

Some of the commands have several arguments. Arguments can be required or optional:

- <argument> = required.
- [<argument>] = optional.

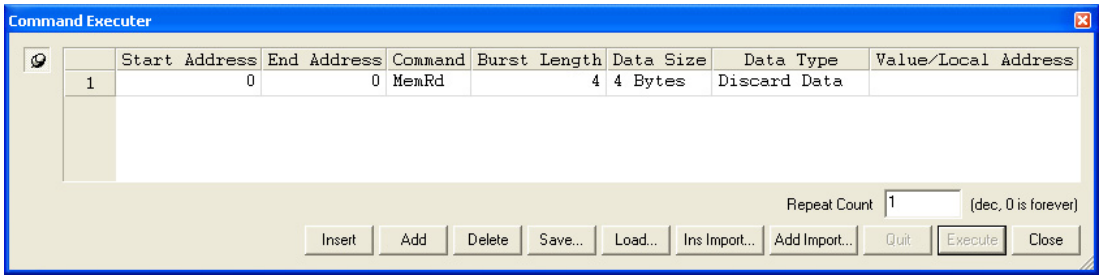
When using the command line interface, all required arguments have to be specified. When using dialog boxes, the arguments have default values.

**"target only"**If the Vanguard is put in a slot designated "target only", the Exerciser can only act as a target on the bus, i.e. it cannot perform any commands from the Master and the Interrupt menus.

### Executing Multiple Commands

#### Command Executer

The Command Executer is used to issue a list of Cycle Sequences. The Fields available in this dialog will alter depending on the Command used.




#### Scripts

Scripts can be made to allow lists of commands to be executed without user intervention. Scripts are text files which can be edited in any text editor and simulate commands typed in by the keyboard. each command is executed before the next one is read. See “Using Scripts” on page 278.

## *7.5 Summary of Commands*

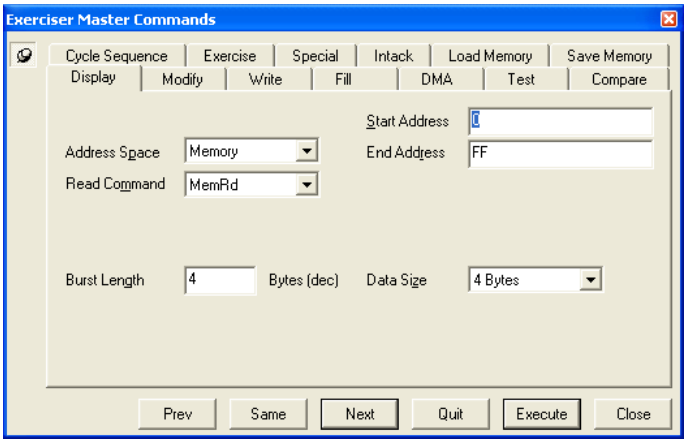
## Display

Displays Memory, I/O or Configuration space in 256 Byte blocks.

**Command Line format:**

d <start_addr> [<addr_space>] [<data_size>][<burst_len>][<read_cmd>] [<end_addr>]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| start_addr | | Hex number | Start address of area to display. | |
| | addr_space | unsigned int | Address space<br>m = Memory Space<br>i = I/O Space<br>c = Configuration Space | m |
| | data_size | unsigned int | Size of each data object 1, 2, 4 or 8 bytes | 4 |
| | burst_len | unsigned int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| | read_cmd | char[2] | **PCI Command**<br>r = Memory read<br>rm = Memory read multiple<br>rl = Memory read line<br>**PCI-X Command**<br>r = Memory read DWORD<br>rb = Memory read block<br>ra = Alias to Memory read block (Not Legal according to PCI-X Specification) | r |
| | end_addr | hex number | End address of the area to display.<br>Maximum is start_addr + FFFF | FF |



**Example:**`d 80001000 m 4 0`

`d` = Display command
`1000` = start address
`m` = Memory space
`4` = 4 byte accesses (DWORD)
`0` = number of bytes which is system dependent in PCI when `0` is chosen.

**Note –** The `end_addr` argument is used to set the block size of the `Display` command, i.e. if the `end_addr` is set to s`tart_addr+0x7f`, only `0x80` bytes of data are displayed in each block.

## Config Display

cd <bus> <device> [function][start_register] [end_register]

---

**Note –** In Configuration space all accesses are DWORD aligned, since A1 and A0 are used to indicate Configuration cycle type 0 or 1.

---

When displaying PCI/PCI-X Configuration space using Configuration cycles type 0, and when more than one of the address bits AD[31::11] are "1", a warning will be displayed in the Exerciser window.

A PCI/PCI-X device is a target of a configuration cycle only if its IDSEL is asserted, and AD[1::0] is "00".

Most PCI/PCI-X systems implements generation of IDSEL by connecting the IDSEL associated with device number 0 to AD16, the IDSEL associated with device number 2 to AD17, etc.

When more than one of the bits AD[31::11] are "1", several PCI/PCI-X devices might try to respond to the same cycle, leading to system crash and, in worst case, a permanent hardware failure.

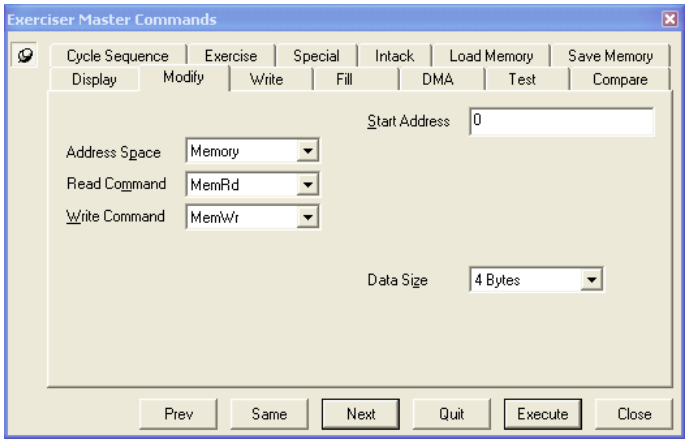
## Modify

Read and Modify data in memory, I/O or Configuration space.

**Command Line format:**

m <start_addr>[<addr_space>] [<data_size>] [<read_cmd>][<write_cmd>]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| start_addr | | Hex number | Start address of area to display. | |
| | addr_space | char | Address space<br>m = Memory Space<br>i = I/O Space<br>c = Configuration Space | m |
| | data_size | unsigned int | Size of each data object 1, 2, 4 or 8 bytes | 4 |
| | read_cmd | char[2] | **PCI Command**<br>r = Memory read<br>rm = Memory read multiple<br>rl = Memory read line<br>**PCI-X Command**<br>r = Memory read DWORD<br>rb = Memory read block<br>ra = Alias to Memory read block (Not Legal according to PCI-X Specification) | r |
| | write_cmd | char[2] | **PCI Command**<br>w = Memory write<br>wi = Memory write and invalidate<br>**PCI-X Command**<br>w = Memory write<br>wb = Memory write block<br>wa = Alias to Memory write block (Not Legal according to PCI-X Specification) | w |



**Example:**`m 1000 m 4 r w`

`m` = Modify command
`1000` = PCI start address
`m` = Memory space
`4` = 4 byte accesses
`r` = read command is Memory Read
`w` = write command is Memory Write.

## Config Modify

cm <bus> <device> [function][register]

---

**Note –** In Configuration space all accesses are 4 bytes aligned, since A1 and A0 are used to indicate Configuration cycle type 0 or 1.

---

When modifying PCI/PCI-X Configuration space using Configuration cycles type 0, and more than one of the address bits AD[31::11] are 1, a warning will be displayed in the Exerciser window.

A PCI/PCI-X device is a target of a configuration cycle only if its IDSEL is asserted, and AD[1::0] is "00".

Most PCI/PCI-X systems implements generation of IDSEL by connecting the IDSEL associated with device number 0 to AD16, the IDSEL associated with device number 2 to AD17, etc.

When more than one of the bits AD[31::11] are "1", several PCI/PCI-X devices might try to respond to the same cycle, leading to system crash and, in worst case, a permanent hardware failure.

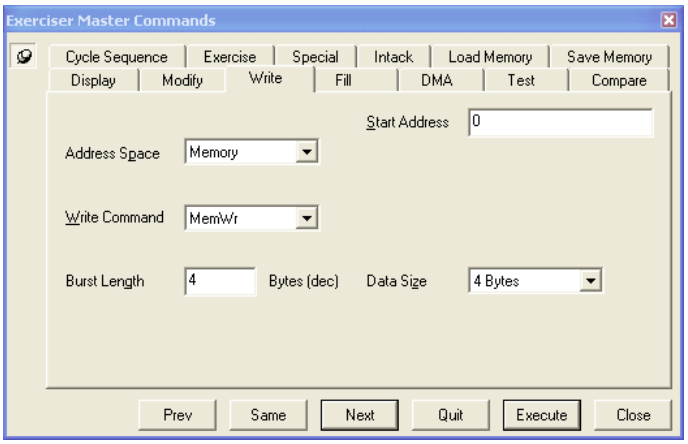
## Write

Used to write data into Memory, I/O or Configuration space.

**Command Line format:**

w <start_addr>[<addr_space>] [<data_size>] [<burst_len>] [<write_cmd>]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| start_addr | | Hex number | Start address of area to display. | |
| | addr_space | char | Address space<br>m = Memory Space<br>i = I/O Space<br>c = Configuration Space | m |
| | data_size | unsigned int | Size of each data object 1, 2, 4 or 8 bytes | 4 |
| | burst_len | unsigned int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| | write_cmd | char[2] | **PCI Command**<br>w = Memory write<br>wi = Memory write and invalidate<br>**PCI-X Command**<br>w = Memory write<br>wb = Memory write block<br>wa = Alias to Memory write block (Not Legal according to PCI-X Specification) | w |




**Example:** `w 1000`

`w` = Write command
`1000` = PCI start address.

When the burst option is used, the data is entered at the desired address in the same way as for single cycle. However, the data is not written until the command is executed, or until the number of bytes entered is equal to the burst length.

## Config Write

cm <bus> <device> [function][register]

---

**Note –** In Configuration space all accesses are 4 bytes aligned, since A1 and A0 are used to indicate Configuration cycle type 0 or 1.

---

When modifying PCI/PCI-X Configuration space using Configuration cycles type 0, and more than one of the address bits AD[31::11] are "1", a warning will be displayed.

A PCI/PCI-X device is a target of a configuration cycle if: its IDSEL is asserted and AD[1::0] is "00", and if C/BE#3:0 is ConfigRd or ConfigWr.

Most PCI/PCI-X systems implements generation of IDSEL by connecting the IDSEL associated with device number 0 to AD16, the IDSEL associated with device number 2 to AD17, etc.

When more than one of the bits AD[31::11] are "1", several PCI/PCI-X devices might try to respond to the same cycle, leading to system crash and, in worst case, a permanent hardware failure.

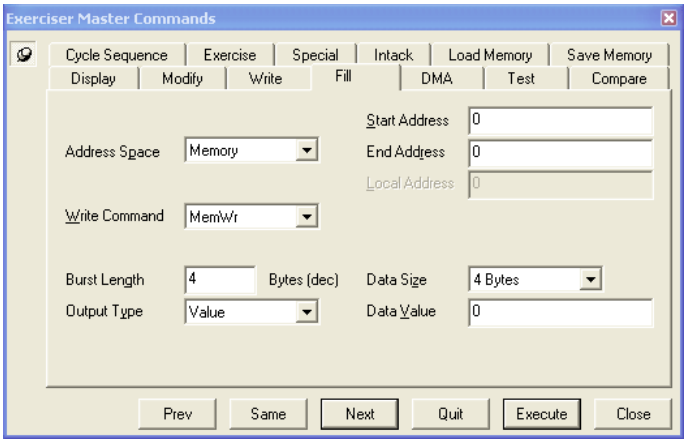
## Fill

Fills Memory, I/O or Configuration space with a given pattern or value.

**Command Line format:**

f <start_addr><end_addr><value>
[<addr_space>][<data_size>][<burst_len>][<write_cmd>][<local_addr>]]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| start_addr | | Hex number | Start address of the fill area | |
| end_addr | | Hex number | End address of the fill area | |
| value | | Hex number or char | The value to fill into the area<br>char values:<br>o = Walking one pattern<br>z = Walking zero pattern<br>s = address as data<br>r = random data<br>l = use data in "local addr" | o |
| | addr_space | char | Address space<br>m = Memory Space<br>i = I/O Space<br>c = Configuration Space | m |
| | data_size | unsigned int | Size of each data object 1, 2, 4 or 8 bytes | 4 |
| | burst_len | unsigned int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| | write_cmd | char[2] | **PCI Command**<br>w = Memory write<br>wi = Memory write and invalidate<br>**PCI-X Command**<br>w = Memory write<br>wb = Memory write block<br>wa = Alias to Memory write block (Not Legal according to PCI-X Specification) | w |
| | local_addr | hex number | Local memory address of data pattern | |




**Example:** `f 1000 10ff o m 4 8`

`f` = Fill command
`1000` = PCI/PCI-X start address
`10ff` = PCI/PCI-X end address
`o` = Walking One fill pattern
`m` = Memory space
`4` = 4 bytes data size and 8 = burst length in bytes

**Fill a user-defined pattern:**

The value argument specifies which kind of fill pattern to be used. Using value=l in conjunction with the local_addr argument, causes the pattern at local_addr to be used as fill pattern.

A user-defined pattern can be filled into the local user memory area starting with local_addr, before the Fill command is started. This can be done using the Local Fill and the Local Modify commands.

See also the Save and Load commands for saving patterns to disk.

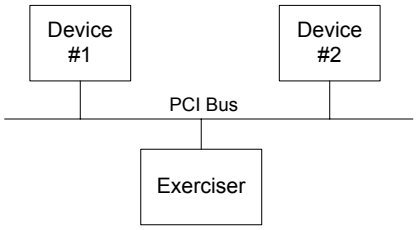
## DMA

The DMA (Direct Memory Access) command initiates a DMA transfer between a target and the Exerciser Local User Memory. The DMA command may take an argument specifying that the DMA transfer should loop forever, transferring the same block in an endless loop. This option is effective for creating heavy traffic on the bus.

The Vanguard has 2 DMA channels available so two different DMA transfers can be running at the same time.

DMA Transfers can be made in three different ways:

- Local to PCI - Transfers data from Local (Target) memory to a PCI location. i.e from Exerciser to Device #1 in the diagram shown.
- PCI to Local - Transfers data from a PCI location to local memory. i.e. from Device #1 to the Exerciser in the diagram shown.
- PCI to PCI - Transfers data from a PCI location to local memory, and then from local memory to another PCIlocation. i.e. from Device #1 to Device #2 via the Exerciser in the diagram shown.



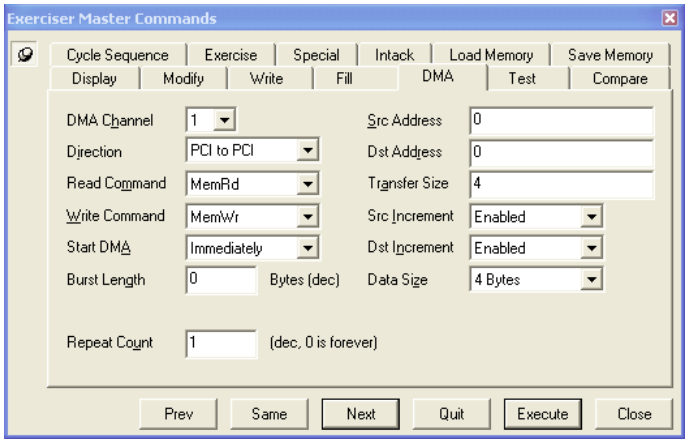

**Note –** The DMA command runs in the background and the command prompt will return after starting the DMA transfer. This means that other commands can be entered without having to wait for the DMA transfer to finish.

**Note –** A DMA transfer can be aborted by running the DMA abort command.

**Command Line format:**

dma <channel> <src_addr> <dst_addr> <transfer_size> <dir> [<repeat>] [<data_size>][<burst_len>] [<read_cmd>] [<write_cmd>] [<src_incr>] [<dst_incr>][start_on_trg] ]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| channel | | Unsigned int | DMA channel number 1 or 2 | |
| src_addr | | Hex number | Source start address | |
| dst_addr | | Hex number | Destination start address | |
| transfer_size | | Hex number | Number of bytes to transfer | |
| dir | | char[2] | pp = PCI to PCI memory<br>pl = PCI to local memory<br>lp = local to PCI memory | pp |
| | repeat | unsigned_int | Number of times to repeat the DMA transfer<br>0 = forever | 1 |
| | data_size | unsigned int | Size of each data object 1, 2, 4 or 8 bytes. | 4 |
| | burst_len | unsigned int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| | read_cmd | char[2] | **PCI Command**<br>r = Memory read<br>rm = Memory read multiple<br>rl = Memory read line<br>**PCI-X Command**<br>r = Memory read DWORD<br>rb = Memory read block<br>ra = Alias to Memory read block (Not Legal according to PCI-X Specification) | r |
| | write_cmd | char[2] | **PCI Command**<br>w = Memory write<br>wi = Memory write and invalidate<br>**PCI-X Command**<br>w = Memory write<br>wb = Memory write block<br>wa = Alias to Memory write block (Not Legal according to PCI-X Specification) | w |
| | src_incr | BOOL | Source address is incremented for every successful data transfer.<br>1 = Enabled<br>0 = Disabled | 1 |
| | dst_incr | BOOL | Destination address is incremented for every successful data transfer.<br>1 = Enabled<br>0 = Disabled | 1 |
| | start_on_trg | BOOL | Start DMA transfer on trigger from the Analyzer<br>1= Enable<br>0= Disabled | 0 |




**Example:** `dma 2 10000 20000 1000 4 pp 4`

`dma` = DMA command
`2` = DMA channel 2
`10000` = source start address
`20000` = destination start address
`1000` = bytes to transfer
`4`= 4 bytes data size
`pp` = PCI/PCI-X to PCI/PCI-X transfer
`4` = repeat 4 times.

A DMA transfer of 0x1000 bytes from address 0x10000 to address 0x20000 is executed 4 times, using DMA channel 2.

A user-defined pattern can be filled into the local user memory area starting with `local_addr`, before the DMA transfer is started. This can be done using the Local Fill and the Local Modify commands.

See also the Save and Load commands for saving patterns to disk.

**Status Line Information**

The status line at the bottom of the BusView window shows a green indicator in the DMA fields if a DMA (started with the DMA command) is active.

If no DMA is active, and the DMA status line field is grayed, a double mouse click on the DMA field, will open the DMA dialog box.

If a DMA is active, and the green indicator is on, a double mouse click on the field will terminate the DMA. The indicator will then change color to dark green.

**Arbitration:**The arbitration between the DMA channels is set in the Exerciser Options, including the hidden DMA channel 0, which is used internally by all exerciser user commands.

## Test

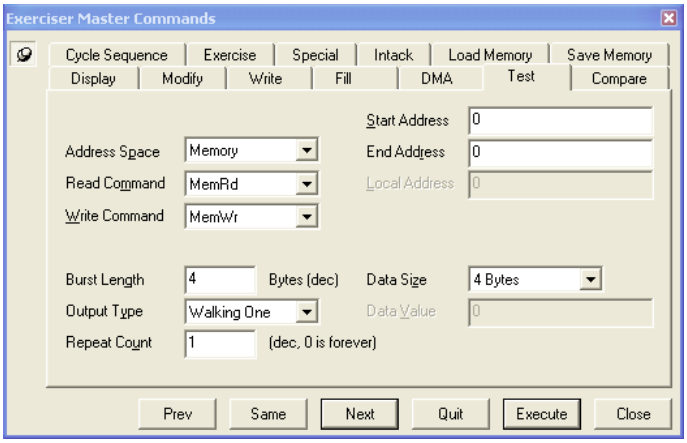
Repeatedly fills Memory , I/O space or Configuration space with a given pattern, reads it back, and checks for errors.

**Command Line format:**

t <start_addr> <end_addr> [<value>] [<repeat>] [<addr_space >] [<data_size>] [<burst_len>] [<read_cmd >] [<write_cmd>] [<local_addr>]]

| Argument | | | Description | Default |
|---|---|---|---|---|
| Required | Optional | Type | | |

| | | | | |
|---|---|---|---|---|
| start_addr | | Hex number | Start address of the fill area | |
| end_addr | | Hex number | End address of the fill area | |
| | value | Hex number or char | The value to fill into the area<br>char values:<br>o = Walking one pattern<br>z = Walking zero pattern<br>s = address as data<br>r = random data<br>l = use data in “local addr” | o |
| | repeat | unsigned_int | Number of times to repeat the DMA transfer<br>0 = forever | 1 |
| | addr_space | char | Address space<br>m = Memory Space<br>i = I/O Space<br>c = Configuration Space | m |
| | data_size | unsigned int | Size of each data object 1, 2, 4 or 8 bytes | 4 |
| | burst_len | unsigned int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| | read_cmd | char[2] | **PCI Command**<br>r = Memory read<br>rm = Memory read multiple<br>rl = Memory read line<br>**PCI-X Command**<br>r = Memory read DWORD<br>rb = Memory read block<br>ra = Alias to Memory read block (Not Legal according to PCI-X Specification) | r |
| | write_cmd | char[2] | **PCI Command**<br>w = Memory write<br>wi = Memory write and invalidate<br>**PCI-X Command**<br>w = Memory write<br>wb = Memory write block<br>wa = Alias to Memory write block (Not Legal according to PCI-X Specification) | w |
| | local_addr | hex number | Local memory address of data pattern | 0 |




**Example:** `t 10000 1ffff o 32 m 4 1`

`t` = Test command
`10000` = start address
`1ffff` = end address
`o` = Walking One pattern
`32` = repeat 32 times
`m` = Memory space
`4` = 4 byte accesses
`1` = burst length.

The result is a test that writes `0xffff` bytes of walking a one pattern to PCI/PCI-X address `0x10000`, reads them back, and checks whether any errors have occurred in the operation. This procedure is performed 32 times, or until the user terminates the test by pressing the Quit button, or one of the keyboard keys ‘q’, ‘Q’, ‘Esc’, or ‘.’.

**Test with a user-defined pattern:**

The `value` argument specifies which kind of test pattern to be used. Using `value=1` in conjunction with the `local_addr` argument, causes the pattern at `local_addr` to be used as test pattern.

A user-defined pattern can be filled into the local user memory area starting with `local_addr`, before the test is started. This can be done using the Local Fill and the Local Modify commands. See also the Save and Load commands for saving patterns to disk.

**EXERtrg# on verify error:**

The Test command asserts a trigger signal (EXERtrg#) to the PCI/PCI-X Bus Analyzer when a verify error occurs. For each verify error, the test stops and waits for user input, either to terminate the test, or to continue. Use the buttons at the bottom of the Exerciser window. Keyboard keys are ‘q’, ‘Q’, ‘Esc’, or ‘.’ to quit, or any other key to continue.

It is required to run the test with single cycles when the EXERtrg# trigger signal is taken into use in the analyzer, because the trigger signal is asserted during the verify process. When burst cycles are used, an error can occur in the first transfer of the burst, but the trigger signal will not be asserted until the burst is finished and the verify process has started.

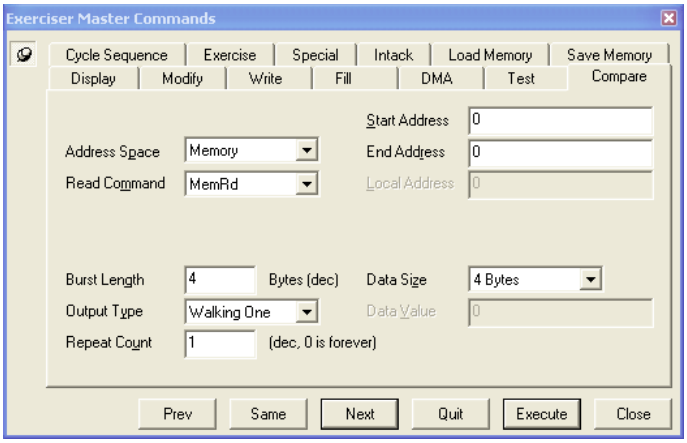
## Compare

Repeatedly reads Memory, I/O space or Configuration space, and compares the data with a given pattern.

**Command Line format:**

c <start_addr> <end_addr> [<value>] [<repeat>] [<addr_space >] [<data_size>] [<burst_len>] [<read_cmd>] [<local_addr>]]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| start_addr | | Hex number | Start address of the fill area | |
| end_addr | | Hex number | End address of the fill area | |
| | value | Hex number or char | The value to fill into the area<br>char values:<br>o = Walking one pattern<br>z = Walking zero pattern<br>s = address as data<br>r = random data<br>l = use data in "local addr" | o |
| | repeat | unsigned_int | Number of times to repeat the DMA transfer<br>0 = forever | 1 |
| | addr_space | char | Address space<br>m = Memory Space<br>i = I/O Space<br>c = Configuration Space | m |
| | data_size | unsigned int | Size of each data object 1, 2, 4 or 8 bytes | 4 |
| | burst_len | unsigned int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| | read_cmd | char[2] | **PCI Command**<br>r = Memory read<br>rm = Memory read multiple<br>rl = Memory read line<br>**PCI-X Command**<br>r = Memory read DWORD<br>rb = Memory read block<br>ra = Alias to Memory read block (Not Legal according to PCI-X Specification) | r |
| | local_addr | hex number | Local memory address of data pattern | 0 |




**Example:** `c 10000 1ffff o 1 m 4 4 rm`

`c` = Compare command
`10000` = start address
`1ffff` = end address
`o` = Walking one
`1` = do not repeat
`m` = Memory space
`4` = 4 bytes accesses
`4` = 4 bytes burst length
`rm` = Memory read multiple

The result is a test that reads `0xffff` bytes from address `0x10000` with four bytes accesses, and compares the data with a walking one pattern.

This procedure is performed once, or until the user terminates the test by pressing the Quit button at the bottom of the Exerciser window, or one of the keyboard keys ‘q’, ‘Q’, ‘Esc’, or ‘.’.

**Test with a user-defined pattern:**

The value argument specifies which kind of test pattern to be used. Using `value=l` in conjunction with the `local_addr` argument, causes the pattern at `local_addr` to be used as test pattern.

A user-defined pattern must be filled into the local user memory area starting with `local_addr`, before the test is started. This can be done using the Local Fill and the Local Modify commands.

See also the Save and Load commands for saving patterns to disk.

**EXERtrg# on verify error:**

The Compare command asserts a trigger signal (EXERtrg#) to the PCI/PCI-X Bus Analyzer when a verify error occurs. For each verify error, the test stops and waits for user input, either to terminate the test, or to continue. Use the buttons at the bottom of the Exerciser window. Keyboard keys are ‘q’, ‘Q’, ‘Esc’, or ‘.’ to quit, or any other key to continue.

It is required to run the test with single cycles when the EXERtrg# trigger signal is taken into use in the analyzer, because the trigger signal is asserted during the verify process. When burst cycles are used, an error can occur in the first transfer of the burst, but the trigger signal will not be asserted until the burst is finished and the verify process has started.

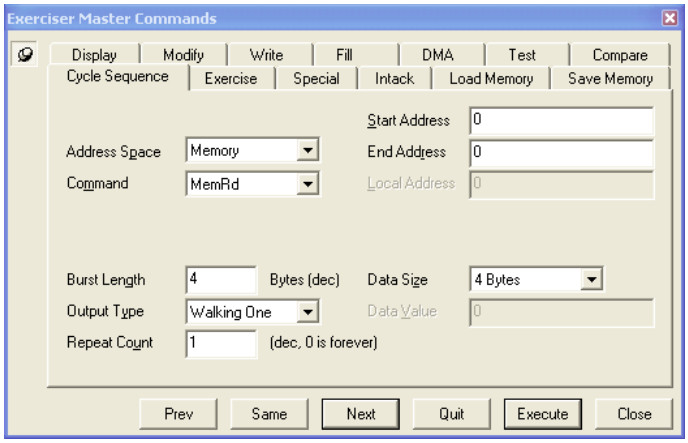
## Cycle Sequence

Generates a sequence of PCI/PCI-X cycles. Only error messages are output. This command is useful to put traffic on the bus.

**Command Line format:**

q <start_addr> <end_addr> [<value>] [<repeat>] [<addr_space >] [<data_size>] [<burst_len>] [<cmd>] [<local_addr>]

| **Argument** | | | **Description** | **Default** |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| start_addr | | Hex number | Start address of the fill area | |
| end_addr | | Hex number | End address of the fill area | |
| | value | Hex number or char | The value to fill into the area<br>char values:<br>o = Walking one pattern<br>z = Walking zero pattern<br>s = address as data<br>r = random data<br>l = use data in "local addr" | o |
| | repeat | unsigned_int | Number of times to repeat the sequence<br>0 = forever | 1 |
| | addr_space | char | Address space<br>m = Memory Space<br>i = I/O Space<br>c = Configuration Space | m |
| | data_size | unsigned int | Size of each data object 1, 2, 4 or 8 bytes | 4 |
| | burst_len | unsigned int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| | cmd | char[2] | **PCI Command**<br>r = Memory, I/O or Config read<br>rm = Memory read multiple<br>rl = Memory read line<br>w = Memory, I/O or Config write<br>wi = Memory write and invalidate<br>**PCI-X Command**<br>r = Memory DWORD, I/O or Config read<br>rb = Memory read block<br>ra = Alias to Memory read block (Not Legal according to PCI-X Specification)<br>w = Memory, I/O or Config write<br>wb = Memory write block<br>wa = Alias to Memory write block (Not Legal according to PCI-X Specification) | r |
| | local_addr | hex number | Local memory address of data pattern | 0 |




**Example:** `q 20000 200ff 1 c`

`q` = Cycle Sequence command
`20000` = PCI/PCI-X start address
`200ff` = PCI/PCI-X end address
`1` = never repeat (repeat count 1)
`c` = PCI Configuration space.

**Place a user-defined pattern on the bus:**

The value argument specifies which kind of pattern to be used. Using `value=l` in conjunction with the `local_addr` argument, causes the pattern at `local_addr` to be used.

A user-defined pattern can be filled into the local user memory area starting with `local_addr`, before the Cycle Sequence command is started. This can be done using the Local Fill and the Local Modify commands.

See also the Save and Load commands for saving patterns to disk.

## Exercise

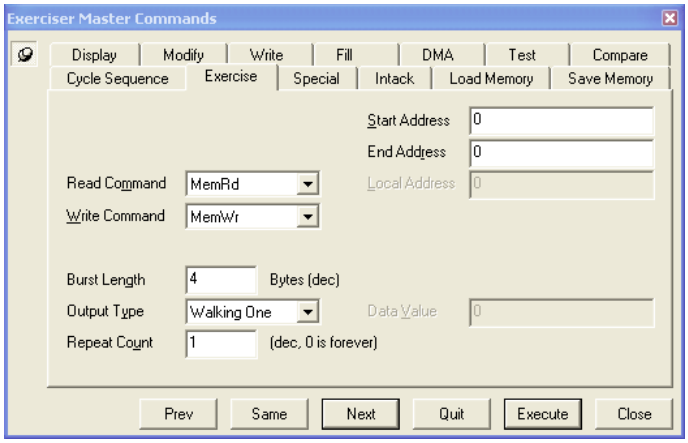
Reads and writes cycles with varying data sizes.

For example: if a memory area, a pattern, and a Write command is selected, then the pattern is applied to the bus four times (presuming repeat count is 1). First with a data size of 1 byte, second with a data size of 2 bytes, third with a data size of 4 bytes, and fourth with a data size of 8 bytes.

**Command Line format:**

x <start_addr> <end_addr> [<value>] [<repeat>] [<burst_len>] [<read_cmd>] [<write_cmd>] [<local_addr>]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| start_addr | | Hex number | Start address of the fill area | |
| end_addr | | Hex number | End address of the fill area | |
| | value | Hex number or char | The value to fill into the area<br>char values:<br>z= Walking zero pattern is filled into the area<br>o = Walking one pattern is filled into the area<br>s = address as data is filled into the area<br>r = random data is filled into the area<br>l = use data in "local addr" | o |
| | repeat | unsigned int | Number of times to repeat the DMA transfer<br>0 = forever | 1 |
| | burst_len | unsigned int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| | read_cmd | char[2] | **PCI Command**<br>r = Memory read<br>rm = Memory read multiple<br>rl = Memory read line<br>**PCI-X Command**<br>r = Memory read DWORD<br>rb = Memory read block<br>ra = Alias to Memory read block (Not Legal according to PCI-X Specification) | r |
| | write_cmd | char[2] | **PCI Command**<br>w = Memory write<br>wi = Memory write and invalidate<br>**PCI-X Command**<br>w = Memory write<br>wb = Memory write block<br>wa = Alias to Memory write block (Not Legal according to PCI-X Specification) | w |
| | local_addr | hex number | Local memory address of data pattern | 0 |




**Example:** `x 0 8000000 ABBACAFE 0`

`x` = Exercise command
`0` = PCI/PCI-X start address
`8000000` = PCI/PCI-X end address
`ABBACAFE` = data fill pattern
`0` = infinite repeat.

**Place a user-defined pattern on the bus:**

The value argument specifies which kind of pattern to be used. Using `value=l` in conjunction with the `local_addr` argument, causes the pattern at `local_addr` to be used.

A user-defined pattern can be filled into the local user memory area starting with `local_addr`, before the Exercise command is started. This can be done using the Local Fill and the Local Modify commands.

See also the Save and Load commands for saving patterns to disk.

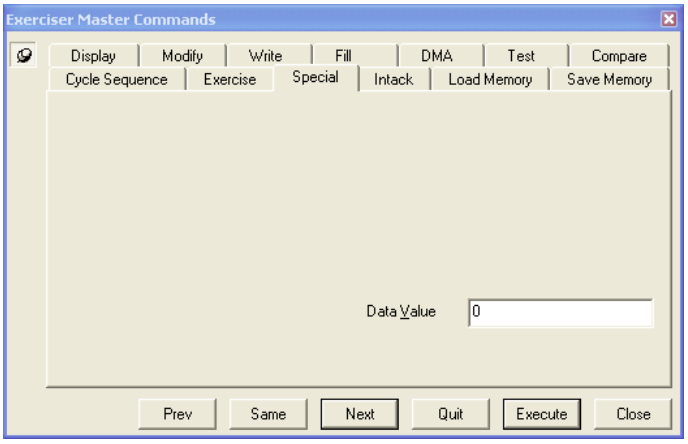
## Special

The Special Cycle command generates special cycles on the PCI/PCI-X bus.

**Command Line format:**

special [<data>]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| | data | Hex number | Data value | 0 |




**Example:**special 12345678

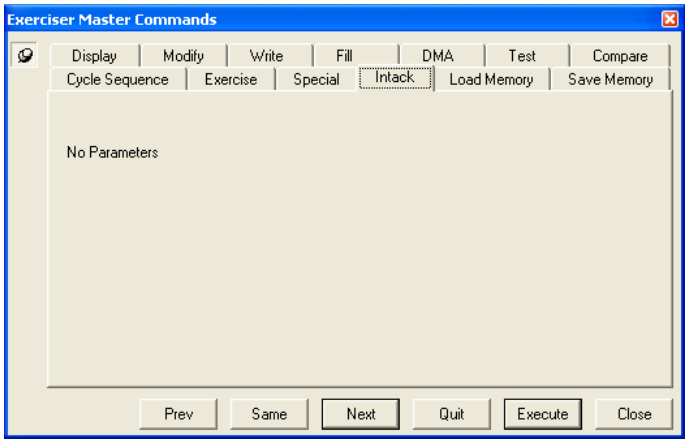
special = Special Cycle command
12345678 = data.

## Intack

The Interrupt Acknowledge command generates interrupt acknowledge cycles on the PCI/PCI-X bus.

**Command Line format:**

intack



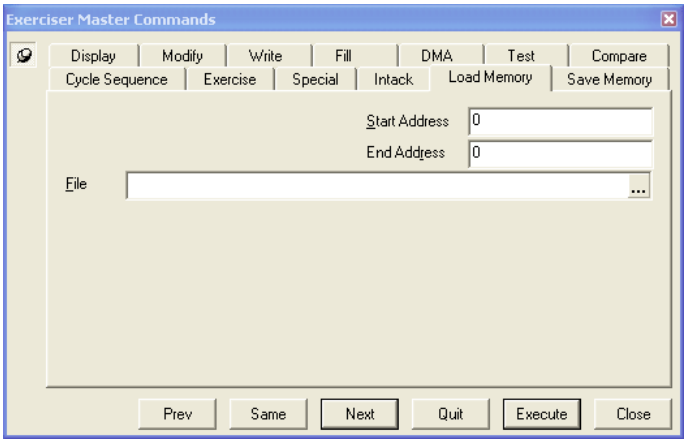
## Load Memory

Data files previously generated with the Save or Local Save commands, can be loaded into PCI/PCI-X memory using the Load command.

**Command Line format:**

l <start_addr><end_addr> [file_name]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| start_addr | | Hex number | Start address of the fill area | |
| end_addr | | Hex number | End address of the fill area | |
| | [<filename>] | string | Name of the file to load. | |




**Example:** l 0 100

`0x100` bytes of data will be loaded from the file to PCI/PCI-X memory, starting from PCI/PCI-X address `0x0`.

**Note –** Address must be DWORD aligned.

If no filename is entered, a dialog box will open, asking for the location and name of the file to load.

**Abort operation**

Press the Quit button at the bottom of the PCI/PCI-X Exerciser window, or any of the ‘Q’, ‘q’, ‘Esc’, or ‘.’ keys, to abort the Load command.

**Note –** The memory file is always big endian and the Load Memory command understands this.

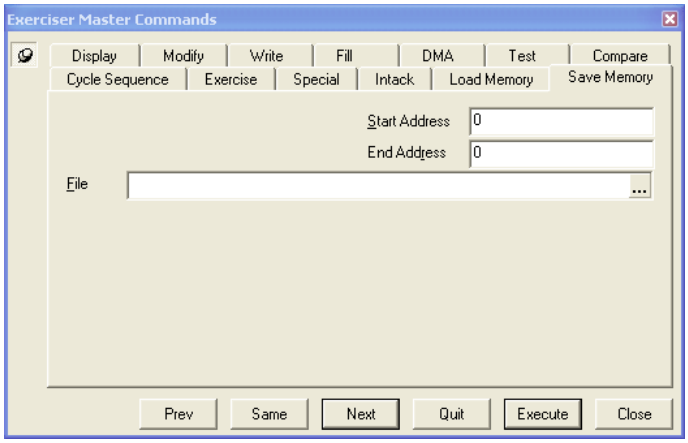
## Save Memory

Save PCI/PCI-X memory to file.

**Command Line format:**

s<start_addr><end_addr> [file_name]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| start_addr | | Hex number | Start address of the fill area | |
| end_addr | | Hex number | End address of the fill area | |
| | [<filename>] | string | Name of the file to save the data in. | |




**Example:** s 0 ffc

The PCI/PCI-X Exerciser will perform Memory Read cycles on the PCI/PCI-X bus, from start address `0x0` to end address `0xFFC` (inclusive), and save the data to a file.If no filename is entered, a dialog box will open, asking for the location and name of the file to save.

**Note –** Address must be DWORD aligned.

The file is in standard ASCII format, and can be read in any text editor. It is also possible to edit the file manually, but be careful not to change the length or format of the file, as this may cause unusual behavior when trying to load it at a later time.

**Abort operation**

Press the Quit button at the bottom of the PCI/PCI-X Exerciser window, or any of the ‘Q’, ‘q’, ‘Esc’, or ‘.’ keys, to abort the Save command.

## DMA Abort

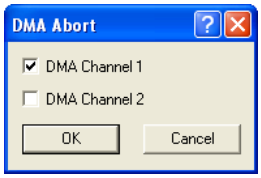
The DMA Abort command aborts any ongoing DMA transfer started with the DMA command.

**Command Line format:**

dma_abort [<ch1>] [<ch2>]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| | ch1 | unsigned int | Channel 1, 2 | |
| | ch2 | unsigned int | Channel 1, 2 | |

**Note –** If no channel is specified, all running DMAs are aborted.




**Example:** dma_abort 1 2

dma_abort = DMA Abort command
1= abort DMA channel 1
2 = abort DMA channel 2.

The DMA channels can be listed in an arbitrary order. If no DMA channel is specified, all running DMAs are aborted.

**Status Line Information**

If a DMA is active, and the green indicator is on, a double mouse click on the field will terminate the DMA.

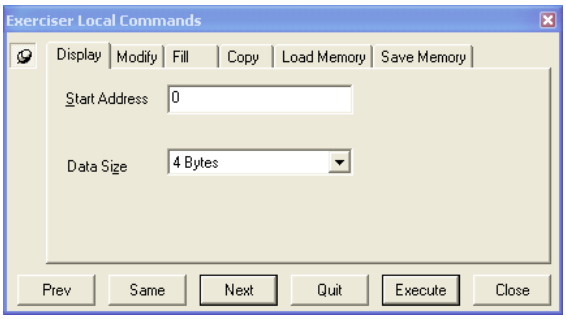
## Local Display

The Vanguard Exerciser has 8MB of Local User Memory (E2 Exerciser has just 8kB, 0x0 to 0x1FFF). The Local Display command allows the user to dump Local User Memory in 256Byte blocks for display. The Local User memory ranges from address 0x0 to 0x7FFFFF and can be mapped as target memory using the Target command.

**Command Line format:**

ld <addr>[<data_size>]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| start_addr | | Hex number | Start address of the area | |
| | data_size | unsigned int | Size of each data object 1, 2, 4 or 8 bytes | 4 |




**Example:** `ld 200000 2`

`ld` = Local Display command
`200000` = Local User Memory start address
`2` = 2 bytes display format.

## Local Modify

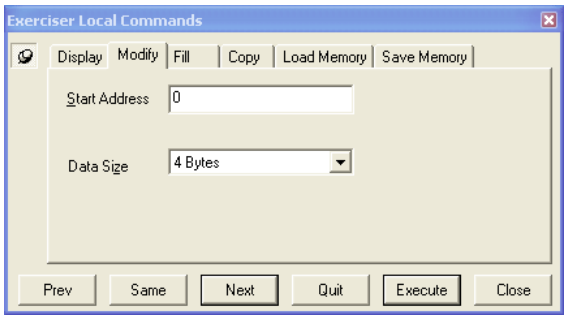
Displays data with data size 1, 2, 4, or 8 bytes, at a given local user address, and optionally allows Local User Memory modification.

The local memory addresses range from 0x0 to 0x7FFFFF. (E2 has 0x0 to 0x1FFF)
Use hexadecimal values to modify the local memory.

**Command Line format:**

lm <start_addr> [<data_size>]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| start_addr | | Hex number | Start address of the area | |
| | data_size | unsigned int | Size of each data object 1, 2, 4 or 8 bytes | 4 |




**Example:** `lm a000 1`

`lm` = Local Modify command
`a000` = Local User Memory start address
`1` = 1 byte display format.

## Local Fill

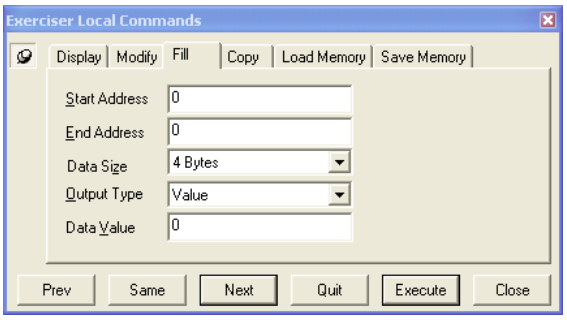
Fills local user memory on the Exerciser, with a given data pattern or value

(E 0x0 to 0x7FFFFF, E2 0x0 to 0x1FFF)

**Command Line format:**

lf <start_addr> <end_addr> <value> [<data_size>]

| **Argument** | | | **Description** | **Default** |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| start_addr | | Hex number | Start address of the fill area | |
| end_addr | | Hex number | End address of the fill area | |
| value | | Hex number or char | The value to fill into the area<br>char values:<br>z = Walking zero pattern is filled into the area<br>o = Walking one pattern is filled into the area<br>s = address as data is filled into the area<br>r = random data is filled into the area | o |
| | data_size | unsigned int | Size of each data object 1, 2, 4 or 8 bytes | 4 |




**Example:**`lf 100000 1fffff 1234 2`

`lf` = Local Fill command
`1000` = Local User Memory start address
`1fff` = Local User Memory end address
`1234` = data to fill into area
`2` = data size.

## Local Copy

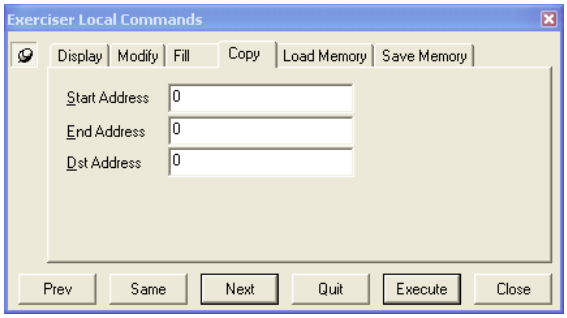
Copies local user Exerciser memory from one location to another.

(E 0x0 to 0x7FFFFF, E2 0x0 to 0x1FFF)

**Command Line format:**

lc <start_addr> <end_addr> <dst_addr>

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| start_addr | | Hex number | Start address of the fill area | |
| end_addr | | Hex number | End address of the fill area | |
| dst_addr | | Hex number | Start address of destination area | |




**Example:** `lc 100000 10ffff 400000`

`lc` = Local Copy command
`1000` = Local User Memory start address
`10ff` = Local User Memory end address
`4000` = Local User Memory destination address.

## Local Load

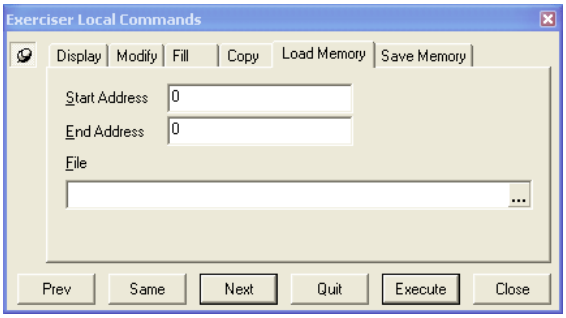
Data files previously generated with the Save or Local Save commands, can be loaded into Local User Memory using the Local Load command.

(E 0x0 to 0x7FFFFF, E2 0x0 to 0x1FFF)

**Command Line format:**

ll <start_addr> <end_addr> [<filename>]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| start_addr | | Hex number | Start address of the fill area | |
| end_addr | | Hex number | End address of the fill area | |
| | [<filename>] | string | Name of the file to load. | |




**Example:** ll 0 1000

In the above example, `0x100` bytes of data will be loaded from a file to Local User Memory, starting from local address `0x0`.

**Note –** Address must be DWORD aligned.

If no filename is entered, a dialog box will open, asking for the location and name of the file to load.

**Abort operation**

Press the Quit button at the bottom of the PCI/PCI-X Exerciser window, or any of the ‘Q’, ‘q’, ‘Esc’, or ‘.’ keys, to abort the Local Load command.

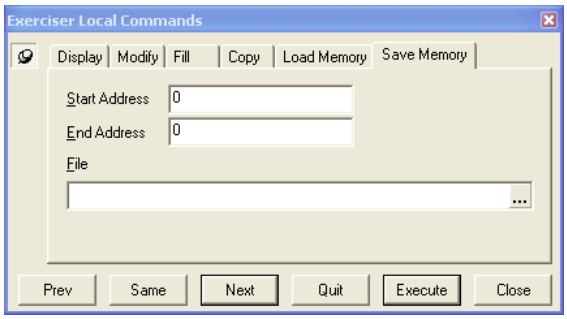
## Local Save

Saves Local User Memory to file.

**Command Line format:**

ls <start_addr> <end_addr>]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| start_addr | | Hex number | Start address of the fill area | |
| end_addr | | Hex number | End address of the fill area | |
| | [<filename>] | string | Name of the file to save the data in. | |




**Example:**ls 0 1000

The Exerciser will read the Local User Memory from start address `0x0` to end address `0x1000` (inclusive), and save the data to a file.

**Note –** Address must be DWORD aligned.

If no filename is entered, a dialog box will open, asking for the location and name of the file to save.

The file is in standard ASCII format, and can be read in any text editor. It is also possible to edit the file manually, but be careful not to change the length or the format of the file, as this may cause unusual behavior when trying to load it at a later time.

**Abort operation**

Press the Quit button at the bottom of the PCI/PCI-X Exerciser window, or any of the ‘Q’, ‘q’, ‘Esc’, or ‘.’ keys, to abort the Local Save command.

## Refresh

Refresh the Exerciser DMA, interrupt and target status.

## Target

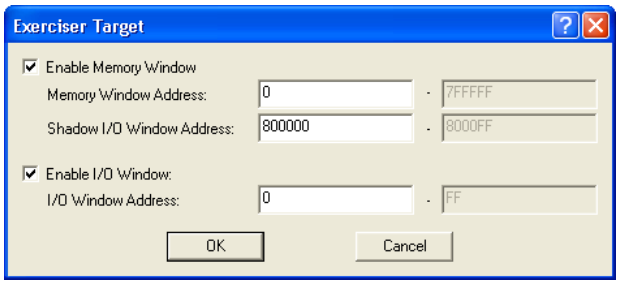
**Note –** See “Target (E2 command)” on page 269 for details on using the Target command for the Enhanced Exerciser (E2).

Maps PCI target window into Local User Memory. Windows can be located in both I/O and Memory space. When activated, parts of the Local User Memory can be accessed by other PCI/PCI-X bus masters. This is how the Exerciser can emulate a PCI/PCI-X target memory device.

**Command Line format:**

target <enable>[<pci_base>][<addr_space>][io_shadow_base]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| | enable | BOOL | Enable/Disable:<br>1 = Enable<br>0 = Disable | 0 |
| | pci_base | hex number | PCI/PCI-X start address of the window<br>Value must be specified if enabled. | |
| | addr_space | char | PCI/PCI-X address space<br>m = PCI/PCI-X Memory space<br>i = PCI/PCI-X I/O space | |
| | io_shadow_base | hex number | PCI-X base address of memory mapped IO Window. | |




**Example 1:**target 1 80000000 m

target = Target command
1 = enable a target window
80000000 = PCI/PCI-X start address of window
m = PCI/PCI-X Memory space.

**Example 2:**target 0 0 i

target = Target command
0 = disable the target window
0 = PCI/PCI-X start address of window
i = PCI/PCI-X I/O space.

**Status Line Information**

The status line at the bottom of the BusView window displays the current target window status.

## Target (E2 command)

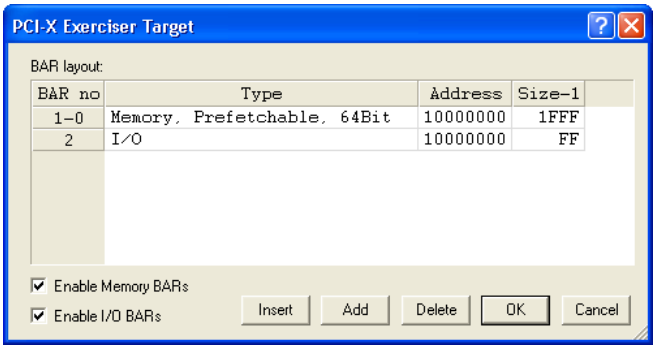
Enables and Disables the use of Memory and IO BARs.

When enabled, parts of the Local User Memory can be accessed by other PCI-X bus masters. This is how the Exerciser can emulate a PCI-X target memory device.

**Command Line format:**

target <enable>[<addr_space>]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| enable | | BOOL | Enable/Disable:<br>1 = Enable<br>0 = Disable | |
| | addr_space | char | PCI-X address space<br>m = PCI-X Memory space<br>i = PCI-X I/O space | m |




In the Target dialog, the checkboxes are used to enable and disable Memory and IO BARs. See “BAR(E2 command)” on page 270 for details about the BAR command

**Example:**target 1 m

target = Target command
1 = enable a target window
m = Memory space

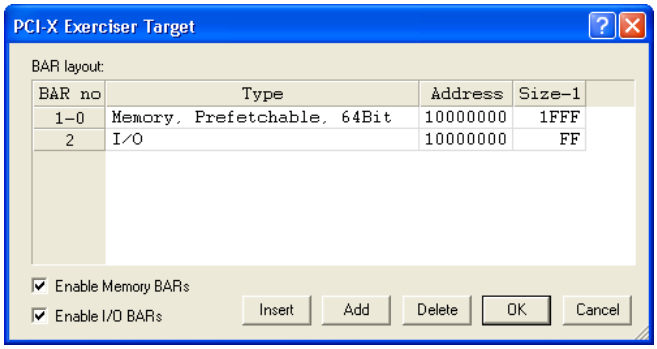
## BAR(E2 command)

Configures the BAR layout in the PCI-X E2 Exerciser. When used, parts of the Local User Memory can be accessed by other PCI-X bus masters. This is how the Exerciser can emulate a PCI-X target memory device. First set of parameters are for BAR0, the next for BAR1, and so on. If a BAR is set as a 64-bit value, two BARS are used.

**Command Line format:**

bar <type> <addr> <size> [<type> <addr> <size>] [<type> <addr> <size>] [<type> <addr> <size>] [<type> <addr> <size>] [<type> <addr> <size>]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| type | type | char | Memory type<br>m = prefetchable 32bit<br>mn = memory, non-prefetchable 32bit<br>m64 = prefetchable 64bit<br>mn64 = non-prefetchable 64bit<br>i = I/O | |
| addr | addr | hex | Address range<br>0-FFFFFFFFFFFFFFFF<br>Least significant bits ignored according to size | |
| size | size | hex | Size minus one in bytes<br>All significant bits must be 1.<br>Range is F - FFFFFFFFFFFFFFFF<br>16 bytes - whole PCI address space | |




The dialog box can be used to compile a list of BARs. Use the ‘Insert’ and ‘Add’ buttons to create new BARs and the ‘Delete’ button to remove them.

BARs with no parameter set are not defined.

Each BAR can be of the type:

- 32-bit Prefetchable Memory
- 32-bit Non-prefetchable Memory
- 64-bit Prefetchable Memory
- 64-bit Non-prefetchable Memory
- I/O

**Example:**bar m64 10000000 1FFF i 10000000 FF

`bar` - BAR Command
`m64` - set BAR to be prefetchable 64-bit memory
`10000000` - Start address of BAR
`1FFF` - Size of BAR
`i` - set another BAR as IO memory
`10000000` - Start address of BAR
`FF` - Size of BAR

## Interrupt

The Interrupt command generates PCI/PCI-X interrupts.

**Command Line format:**

int <enable> <irq_line>

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| enable | | BOOL | 1 = Enable Interrupt<br>0 = Disable Interrupt | |
| irq_line | | char | PCI interrupt request line. Valid values are any of the letters; a, b, c, d. | |

This command can also be executed by pressing the Interrupt button  on the toolbar, or clicking Interrupt(s) from the Exerciser menu, and then choosing an interrupt from the dialog box that appears:

**Example 1:** int 1 b

int = int command
1 = enable interrupt
b = interrupt line.

**Example 2:** int 0 b

int = int command
0 = disable interrupt
b = interrupt line.

**Status Line Information**

The status line at the bottom of the BusView window identifies the active interrupt line. LEDs marked A, B, C, or D are active if the corresponding interrupt line is enabled.

The same information can be found by simply typing “refresh” at the PCI/PCI-X Exerciser prompt.

## Intexe

Interrupts exercise. Activate and deactivate the interrupt lines after specified time parameters.

**Command Line format:**

intexe <irq_line>[time_on][time_off][repeat]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| irq_line | | char | Interrupt line.<br>Valid value is 'a', 'b', 'c', 'd' or '0'. Set this to '0' to stop exercising.<br>Note, to exercise a new interrupt, you must first stop exercising the current one. | |
| | time_on | int | Time in ms the interrupt line is activated. | 20ms |
| | time_off | int | Time in ms the interrupt line is deactivated. | 30ms |
| | repeat | int | Number of times to repeat.<br>0 is forever | 1 |

## IO

I/O port control.

**Command Line format:**

io <port> [value]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| port | | int | port number to access (0-3). | |
| | value | int | 0 - drive the port low.<br>1 - drive the port high.<br>If no value parameter specified, read and return current value from the port. | 20ms |

## 7.6 Configuration Scan

**Command Line format:**

scan [read_res_cfg][scan_uninit_bridges][scan_bridges]

| Argument | | | Description | Default |
|---|---|---|---|---|
| **Required** | **Optional** | **Type** | | |
| | read_res_cfg | | 1 = all configuration registers on a found device are read (default)<br>0 = reserved configuration registers are not read | 1 |
| | scan_uninit_bridges | | 1 = Uninitialized bridges are being programmed and scanned (default)<br>0 = Ignore uninitialized bridges | 1 |
| | scan_bridges | | 1 = Initialized bridges are being programmed and scanned (default)<br>0 = Ignore initialized bridges | 1 |

The Exerciser can perform a complete scan of PCI/PCI-X configuration space. It systematically probes for all possible devices on the bus that the Vanguard is located on. If it finds a PCI-to-PCI bridge, it probes through the bridge for all devices on the busses behind the bridge.

All properties of all found devices are displayed, such as:

- PCI-to-PCI Bridges:
  - Device/Vendor ID.
  - Class Code. If the class code is unknown, "Unknown device" is displayed.
  - Vendor. If the vendor is not on the PCISIG (PCI Special Interest Group) list of vendors, "Unknown" is displayed.
  - Primary, Secondary, and Subordinate bus numbers.
  - Master and Memory/IO space enable.
  - Target windows in both prefetchable memory, memory and IO space.
  - etc.
- Other devices:
  - Device/Vendor ID.
  - Class Code. If the class code is unknown, "Unknown device" is displayed.
  - Device/Vendor ID.
  - Vendor. If the vendor is not on the PCISIG (PCI Special Interest Group) list of vendors, "Unknown" is displayed.
  - Subsystem ID/Subsystem Vendor ID is displayed if it is different from the Device/ Vendor ID.
  - Master and Memory/IO space enable.
  - Target windows in both prefetchable memory, memory and IO space.

A scan of PCI/PCI-X configuration space gives an overview of the devices in the PCI/PCI-X system. This is useful in finding where already enabled targets are in PCI memory, and to get the information needed for manual configuration and enabling of the devices found.

**Note –** PCI/PCI-X configuration space can only be scanned down stream, i.e. it is not possible to do configuration cycles from the secondary side to the primary side of a PCI-to-PCI bridge.

In order for the Config Scan command to find all the PCI/PCI-X devices located in the PCI/PCI-X system, the Primary, Secondary, and Subordinate bus numbers in all the PCI-to-PCI bridges have to be configured correctly.

The Primary, Secondary, and Subordinate bus number registers are located at offset 0x18, 0x19, and 0x1A, respectively, in the PCI-to-PCI bridge configuration space header. The primary bus number is the bus number on the primary side of the bridge, the secondary bus number is the bus number on the secondary side of the bridge, and the subordinate bus number is the highest numbered bus behind the bridge.

Performing a Configuration Scan

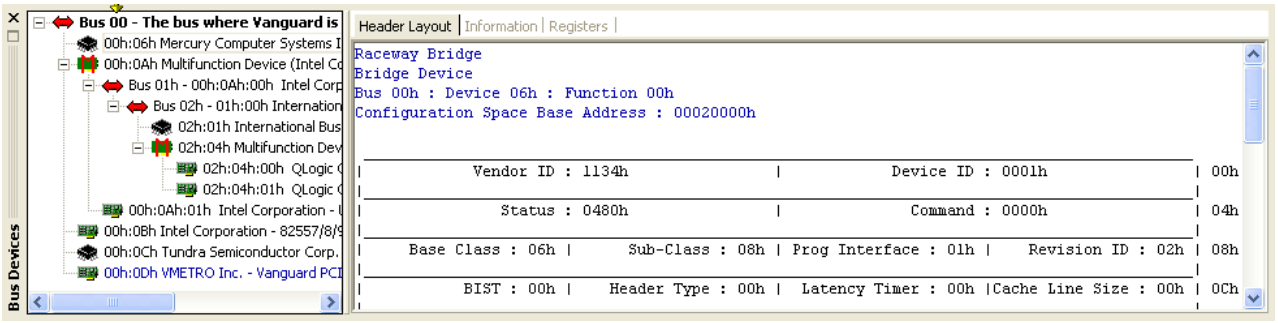
The Configuration Scan can be started using one of the following methods:

- Click the Config Scan button on the toolbar.
- Click Configuration Scan on the Exerciser Menu.
- Type scan into the Exerciser Command Prompt.

When the scan is complete, the Bus Devices window will open, listing the results of the scan.

## Bus Devices

The results of a Configuration Scan are displayed in the Bus Devices window.




*Configuration Scan Results*

Figure  shows an example results of a Config Scan command. The tabs Header Layout, Information and Register display the same information, but in different ways.

- Header Layout - Information is formatted the same as that issued in the PCI specification.
- Information - Relevant information is organised together. for example, IDs are shown together.
- Registers - Information is organised by address.

It is also possible to perform a scan from the Host PC. See “Host PC” on page 51 for details.

## *7.7 Using Scripts*

An Exerciser script is a list of commands that can be executed without user intervention. These commands are listed in a text file which can be edited in any text editor.

In addition to the usual Exerciser commands, there are also Script Commands for use in scripts. These commands allow scripts to incorporate loops and delays.

### Record a Script

1. Click Start Recording from the Exerciser, Script menu or type `rec_start` in the command line.
2. Type a filename for the script file and choose a location to store it.
3. Every Exerciser command executed now will be recorded into this script file. The Script Commands are also now available.
4. Click Stop Recording from the Exerciser, Script menu or type `rec_stop` in the command line to finish.

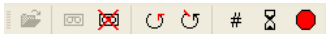
### Execute a Script

1. Click Load Script from the Exerciser, Script menu or type `dls` in the command line.
2. The script is now loaded into memory and awaiting execution.
   To run the script once; click Run from the Exerciser, Script menu, type `run` in the command line, or press F10.
   To run the script a number of times; click Run Loop from the Exerciser, Script menu, type `run [number_of_times]` in the command line, or press Ctrl + F10.
3. The script will now run and any results displayed in the Output Area.

## *7.8 Exerciser Script Commands*

Script commands can be entered via the command line found in the Exerciser window, or by using


the menus “Exerciser, Script” or by right-clicking the mouse in the Exerciser Window. It is also possible to customize the BusView toolbar to include Script Commands. This is done by right-clicking on the busview toolbar and selecting “Scripts”

 A Script toolbar can be added to BusView by right clicking on the toolbar and selecting “Scrip”.

### Start Recording

The Start Recording command starts recording of an Exerciser script. It does this by recording all executed commands to a file.

**Command Line format:**
`rec_start [file_name]`

**Optional Command argument:**
`[file_name]` - used to specify the path and filename of the script file to be recorded. If this parameter is not used, a dialog will prompt for the path and filename details.

The header of the Exerciser window displays the name of the script file being recorded.

The script file is in standard ASCII format and can be edited manually with any text editor. Any errors in the file will simply make the Exerciser display a help text when the script is run, in the same way as when an erroneous command is typed at the Exerciser prompt.

To enter a comment into the script file, the line has to start with a “#” character.

Blank lines are interpreted as CRs.

### Stop Recording

The Stop Recording command displays a dialog box to confirm that you wish to stop recording. If the OK button is pressed recording is stopped.

**Command Line format:**
`rec_stop`

## Load Script

The Load command opens a previously recorded script file.

**Command Line format:**

```
dls [file_name]
```

**Optional Command argument:**
`[file_name]` - used to specify the path and filename of the script file to be loaded. If this parameter is not used, a dialog will prompt for the path and filename details.

## Run Script

Runs the script previously loaded with the Load command. The name of the script file appears in the header of the PCI Exerciser window.

Run Loop can be selected from the Script Menu.

**Command Line format:**
`run [number_of_times] [silent_mode]`

**Optional Command argument:**
`[number_of_times]` - Number of times to run the script. 1 is default, 0 is run forever.

`[silent_mode]` - 0= Running with output to view (default), 1= Without output view (errors are displayed). NOTE: Silent Mode is only accessible from the command line.

## Wait

Enter a number of milliseconds to wait.

**Command Line format:**
`wait <number_of_ms>`

**Command argument:**
`number_of_ms` - Number of milliseconds to wait.

## Pause

Inserts a pause statement in the script file. When a pause is present, press a key to continue.

**Command Line format:**
`pause`

## Loop

Inserts a loop into the script.

**Command Line format:**
`loop <number_of_loops>`

**Command argument:**
`number_of_loops` - Number of times to iterate the loop segment.

---

**Note –** A loop segment must consist of: a “loop n” command, and an “end” command to indicate the end of the looped segment.

```
loop n
..
 "loop segment"
..
end
```

---

## End

Inserts an "end" statement into the script file which marks where the end of the loop is.

**Command Line format:**
`end`

## Show

Shows the contents of the script currently held in memory.

**Command Line format:**
`show`

## Comment

Inserts a comment to the script file indicated with a "#" at the beginning of the line.

# 8 The Simulator

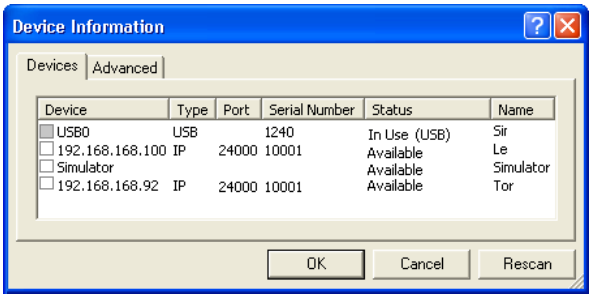
The Simulator can be used where the Vanguard Hardware is not available. It allows most of the features in BusView to be used without having a connection to the Vanguard Hardware.

- Starting the Simulator
- Using the Analyzer with the Simulator
- Using the Statistics Functions with the Simulator
- Using the Protocol Checker with the Simulator
- Using the Exerciser with the Simulator (PCI/PCI-X, VME)

**Note –** All features are available in Simulator mode regardless of which options have been purchased.

## 8.1 Starting the Simulator

BusView connects to the Simulator in the same way it connects to the Vanguard hardware. The Simulator is presented as an available device in the Device Information dialog box.




### Step-by-step Instructions to start the Simulator

These steps assume that you have already installed BusView onto your PC.

1. Start BusView by double-clicking on the BusView icon on your desktop.
2. BusView will begin by performing a scan to find available connections.
3. Select the Simulator by clicking on the check box and click OK.
4. The next dialog box will ask which mode you wish to run the Simulator, PCI, PCI-X, or VME. Choose one by clicking on it's radio button and then click OK.

BusView is now connected to the Simulator.

---

**Note –** To switch between modes it is necessary disconnect from the Simulator, and re-connect in the required mode.

---

## *8.2 Using the Analyzer with the Simulator*

The Simulator makes use of several pre-saved Trace Files. These are used to simulate Sampling of the bus and the acquisition of Trace/Trigger Control data.

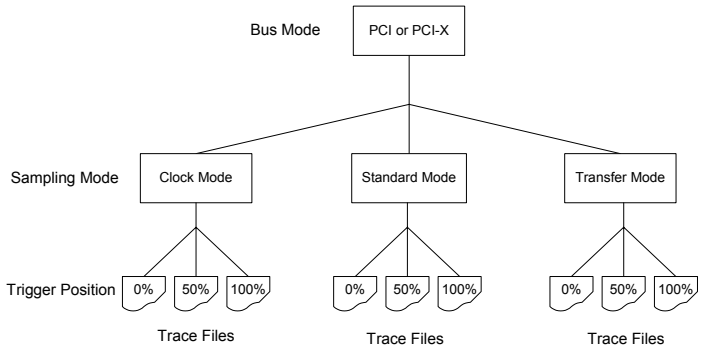
It is important to realize that the trace shown after the Analyzer has triggered in Simulator mode, bears no relation to the Event Patterns defined in the Analyzer Setup Window. However, there are some conditions used to determine which trace file is displayed.

There are a number of trace files available, depending on the bus or link type:

- PCI/PCIX Clock Mode (Multiplexed)
- PCI/PCIX Standard Mode (Demultiplexed)
- PCI/PCIX Transfer Mode (Demultiplexed)
- VME State Mode
- VME Timing Mode

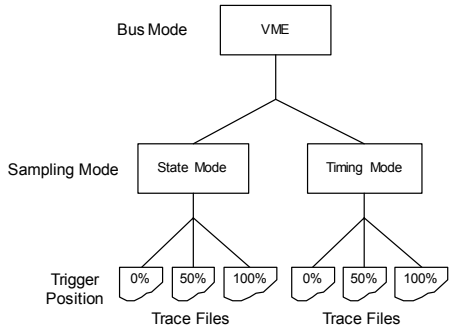
Each of the trace files have a different Trigger Position, 0%, 50% and 100%. When the Analyzer is run, the trace file chosen for display is determined as that which is closest to the chosen trigger position defined in the Analyzer Setup Options. To summarize:

For PCI and PCIX:




A total of eighteen trace files exist for PCI simulation, nine each for PCI and PCI-X bus modes.

For VME:



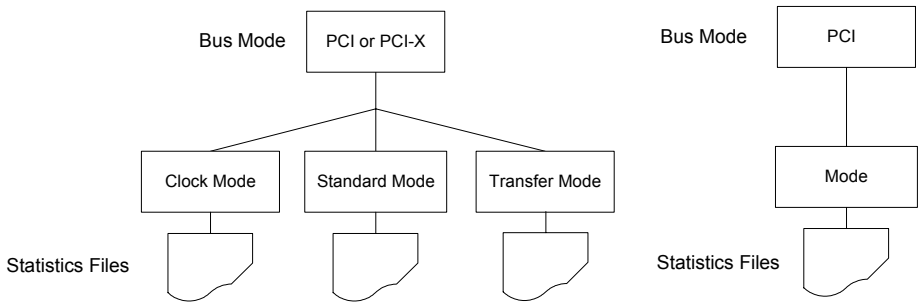

A total of six trace files exist for VME simulation, three each for State and Timing sampling modes.

*This page intentionally left blank*

## *8.3 Using the Statistics Functions with the Simulator*

There are pre-recorded statistics files which are used by the Simulator to display the play back of statistics charts.

The Simulator selects the pre-recorded statistics file depending on which bus mode or link type (PCI, PCI-X or, VME) and Sampling Modes has been chosen from the Statistics Setup Window




The Statistics Charts displayed after the Statistics Function has been run bears no correlation to the settings implemented in the Statistics Setup Window.

## *8.4 Using the Protocol Checker with the Simulator*

The Simulator will generate random errors when the Protocol Checker is run. The error will be displayed on the Protocol Checker window according to the setting chosen in the Protocol Checker Options dialog.

## *8.5 Using the Exerciser with the Simulator (PCI/PCI-X, VME)*

The Simulator reacts depending on the type of Exerciser command

Master Commands issued will result in OK status being returned. No real bus traffic is generated.

Local Commands operate on allocated PC memory and operate normally.

Exerciser Script Commands operate normally and it is possible to build and execute scripts using the Simulator. This is useful for debugging scripts offline.

# 9 Application Programming Interface

This document explains use of the Vanguard Application Programming Interface (Vanguard API), which is used to access Vanguard features from applications different from BusView. The first section gives an introduction to the Vanguard API, while subsequent sections explain the API commands in more detail.

- Introduction
- API client interfaces
- Status codes
- Command overview
- Command description

Use of the Vanguard API requires the VG-API license to have been purchased.

**E2 Only –** Information that pertains to the Enhanced Exerciser will be described in a box like this one.

**Note –** This documentation is valid for version 1.10 of the API.

## 9.1 Introduction

The Vanguard API gives users of Vanguard Analyzers the option to access Vanguard device functionality without using the BusView GUI. This makes it possible to incorporate Vanguard functionality in dedicated applications running on a variety of platforms and using a variety of programming languages.

The main features of the Vanguard API are:

- Provides a programming language- and platform independent interface to Vanguard features
- Provides two alternative API client interfaces, XML-RPC and “Raw ASCII”, both using a Remote Procedure Call (RPC) paradigm.
- Supports most of the Exerciser functionality found in BusView.
- Supports basic trace capturing with predefined setups.
- Supports Protocol Checker functionality.

The Vanguard API engine is implemented as a TCP/IP based server on the Vanguard device, using simple ASCII-formatted data for both the requests and the responses. The API is therefore available to every client platform that has basic TCP/IP support. The API engine integrates with the main Vanguard device firmware as indicated in Figure 9-1.

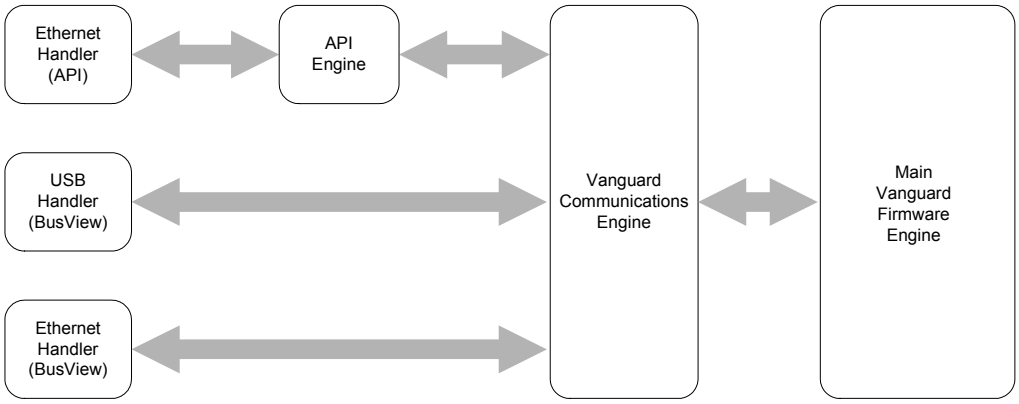



**FIGURE 9-1** *Vanguard API firmware integration*

The API engine integrates with the device firmware as another communications handler, in addition to the BusView Ethernet and USB communication handlers.

As seen by the other Vanguard firmware, the API engine is acting as a BusView emulator. This has the following implications:

- Simultaneous API- and BusView connections are not possible
- A certain initialization overhead is needed each time an API client connects to the Vanguard API engine. When using the API, the client should therefore keep the same TCP/IP connection throughout the API session.

## *9.2 API client interfaces*

Two different client interfaces are available for using the Vanguard API[1]. The proprietary Raw ASCII interface transmits RPC requests and responses as simple ASCII-coded strings, while the XML-RPC interface transmits the requests and responses as XML-formatted messages. The Raw ASCII interface uses basic TCP/IP socket client/server communication for the transport, while the XML-RPC interface uses HTTP.

Both client interfaces are available at TCP/IP port number 25000. The Vanguard API server selects the interface mode based on the received data.

### Raw ASCII interface

To use the Raw ASCII interface, a standard TCP/IP socket connection to the Vanguard API server must be established. Due to a certain API initialization overhead when establishing a server connection, the client should always keep the same connection open during the lifetime of the API session.

The Raw ASCII interface accepts requests as a set of ASCII strings, and returns data in a similar format. Each string must be terminated with a pair of carriage-return and line-feed ASCII characters (CR/LF pair), i.e. ASCII characters $13_{10}$ and $10_{10}$. The general message format is given in Table 9-1

**TABLE 9-1. Raw ASCII message format**

| Line | Data | Description |
|---|---|---|
| 1 | "<<<<" | Start Tag (terminated with CR/LF) |
| 2..n | Request- or response message data | **Request**: Line 2 contains the API function name while subsequent lines, if present, contain any function parameters. Lines must be terminated with CR/LF<br>**Response**: Line 2 contains a status code while subsequent lines, if present, contain any returned data. Lines are terminated with CR/LF. |
| n+1 | ">>>>" | End Tag (terminated with CR/LF) |

Each message is enclosed by a start/end tag pair that explicitly mark the start and the end of a message.

A sample `vanguard_api_option` request could look as follows:

```
<<<<
vanguard_api_option
0
3
>>>>
```

1. A third, optional interface is available for event-based data (See "Event notification interface" on page 297). This additional interface can be used together with both the Raw ASCII- and the XML-RPC interface.

The second line holds the API function name, while the three following lines contain function parameters.

A sample response for a `vanguard_device_info` request that succeeds with no errors is as given below:

```
<<<<
0
1000105
0
15000
0
0
>>>>
```

The second line gives the status code, which is always zero for a no-error response. The other 5 response data lines give the returned data.

A sample response for a request that returns with an error is given below:

```
<<<<
4
wrong argument
>>>>
```

The second line now contains the non-zero status code, while the third line contains extended information about the error. Not all error codes return additional extended information.

## XML-RPC interface

XML-RPC is a Remote Procedure Calling (RPC) protocol that uses HTTP for the transport and XML for the encoding. XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned. XML-RPC client libraries are freely available for a number of programming languages.

See www.xmlrpc.org for further information on the XML-RPC specification.

The Vanguard API XML-RPC server accepts API request parameters passed both as a single XML-RPC array parameter and as individual XML-RPC parameters (strings, integers and boolean parameters).

Using individual XML-RPC parameters for each API parameter, a sample `vanguard_api_option` XML-RPC request could look as follows:

```
POST / HTTP/1.1
User-Agent: dummy_xml_rpc_client
Host: 192.168.168.58:25000
Content-Type: text/xml
Content-Length: 236
Connection: keep-alive

<?xml version="1.0"?>
<methodCall>
   <methodName>vanguard_api_option</methodName>
   <params>
      <param><value><int>0</int></value></param>
      <param><value><int>3</int></value></param>
   </params>
</methodCall>
```

Using a single XML-RPC array parameter for the API parameters, a sample `vanguard_api_option` XML-RPC request could look as follows:

```
POST / HTTP/1.1
User-Agent: dummy_xml_rpc_client
Host: 192.168.168.58:25000
Content-Type: text/xml
Content-Length: 219
Connection: keep-alive

<?xml version="1.0"?>
<methodCall>
   <methodName>vanguard_api_option</methodName>
   <params>
      <param><value><array><data>
         <value><int>0</int></value>
         <value><int>3</int></value>
      </data></array></value></param>
   </params>
</methodCall>
```

Unless an error occurs, in which case a standard XML-RPC fault message is used, the individual data values of the XML-RPC responses are always transferred using a single XML-RPC array. A sample XML-RPC response will look as follows, showing the `vanguard_device_info` command:

```
HTTP/1.1 200 OK
Content-length: 282
Content-Type: text/xml
Server: Vanguard Analyzer

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param><value><array><data>
         <value><int>1000105</int></value>
         <value><int>0</int></value>
         <value><int>15000</int></value>
         <value><int>0</int></value>
         <value><boolean>0</boolean></value>
      </data></array></value></param>
   </params>
</methodResponse>
```

When an error occurs when running a procedure, the standard XML-RPC fault message is used. The contents of this message is the numeric non-zero status code and possibly a string with extended error information. A sample error response will look as follows:

```
HTTP/1.1 200 OK
Content-length: 182
Content-Type: text/xml
Server: Vanguard Analyzer

<?xml version="1.0"?>
<methodResponse>
   <fault>
      <value>
         <struct>
            <member>
               <name>faultCode</name>
               <value><int>4</int></value>
            </member>
            <member>
               <name>faultString</name>
               <value><string>wrong argument</string></value>
            </member>
         </struct>
      </value>
   </fault>
</methodResponse>
```

## *9.3 Event notification interface*

By default, the Vanguard API is pull-based, i.e. the user has to use various status commands to check for certain interrupt-type of events, e.g. when the analyzer has triggered. This is the normal behaviour in a RPC system, i.e. the user only get replies from the RPC-server when data is explicitly requested. In addition to the normal pull-based engine, the Vanguard API also provides a separate event notification server that enables users to get notified of certain type of Vanguard events without having to use extensive polling.

### Configuration

To use event notifications, a standard TCP/IP socket connection to the Vanguard API notification server must be established. The Vanguard API notification server is available at TCP/IP port number 25001.

The notification server is not available until a connection to the main API engine has been established. Likewise, any connection to the notification server will be automatically shut down when a normal API engine session ends.

The notification server is read-only. Configuration of the notification interface is done through the main API engine, using the `vanguard_api_option` command.

### Message formatting

Notifications are received as an event type identifier. The available values are given in Table 9-2. Note that the same event types are used by the `vanguard_api_option` command to set up the notification event mask.

**TABLE 9-2. Event notifications**

| | Type | Description |
|---|---|---|
| event_type | int | Notification event identifier. The following events are defined:<br>0x01: PCI Reset Assert event<br>0x02: PCI Reset De-assert event<br>0x04: Bus Frequency Changed event<br>0x08: Analyzer Triggered event<br>0x10: Analyzer Trace Full event<br>0x20: Protocol Checker Triggered event |

By default, the notification event messages are formatted using the Raw ASCII format used with the main API engine. To comply with parsers for the Raw ASCII format, the message data starts with a status code field. The status code is however always set to 0 (i.e. success) for the notification messages. Following the status code comes the event type identifier.

A sample notification message for an Analyzer Triggered event is as given below:

```
<<<<
0
8
>>>>
```

The second line gives the status code (always 0), while the third line gives the event type.

Users who already use the XML-RPC interface with the main API engine, and who wants to utilize their XML-RPC parser also for the notification messages, can optionally turn on XML-RPC type of formatting in the event notification server. This is done using the `vanguard_api_option` command.

A sample XML-RPC formatted notification message will look as follows, showing the message for an Analyzer Triggered event:

```
Content-Length: 156
<?xml version="1.0"?>
<methodResponse>
   <params>
      <param><value><array><data>
         <value><int>8</int></value>
      </data></array></value></param>
   </params>
</methodResponse>
```

The first line gives the length of the XML-RPC data that follows, in the same manner as for a normal XML-RPC response (using HTTP). This line is ended with a CR/LF pair.

## 9.4 Status codes

The Vanguard API status codes are listed in Table 9-3 below. Note that some codes are only valid for certain bus types.

**TABLE 9-3. Vanguard API status codes**

| Code | Name | Description |
|---|---|---|
| *0* | VG_OK | *no error, i.e. success* |
| 1 | VG_GENERAL_ERROR | A general, non-specified error occurred |
| 2 | VG_UNDEF_BUS | The current bus is undefined |
| 3 | VG_UNDEF_CMD | Illegal API command |
| 4 | VG_PARSER | An error occurred when parsing the command |
| 5 | VG_CMD_ORDER | The API commands are used in an invalid command order. |
| 6 | VG_PARAM_COUNT | The API function was called with an invalid number of parameters |
| 7 | VG_AUTHORIZATION | Authorization failed for the command |
| 8 | VG_PARAM | A parameter has an invalid value |
| 9 | VG_BUS_RESET | A bus reset occurred during the execution of the command |
| 10 | VG_DEV_INFO_REQUIRED | A `vanguard_device_info` command is needed prior to running this command |
| 11 | VG_SELFTEST | Vanguard selftest failed |
| 12 | VG_API_INIT | API initialization failed |
| *13-29* | *Reserved* | |
| 30 | VG_ACCESS | A bus access failed |
| 31 | VG_SPLIT_COMPLETION | A split completion error occurred |
| 32 | VG_EXERCISER | General exerciser error |
| 33 | VG_NO_EXERCISER | No exerciser was found |
| 34 | VG_VERIFY | Data verification failed |
| 35 | VG_DMA_DESC | The Vanguard is out of DMA descriptors |
| 36 | VG_MASTER_ABORT | A master abort condition occurred |
| 37 | VG_TARGET_ABORT | A target abort condition occurred |
| 38 | VG_DATA_PERR | A data parity error occurred |
| 39 | VG_ADDR_PERR | An address parity error occurred |
| 40 | VG_SYSTEM | A system error occurred |

**TABLE 9-3. Vanguard API status codes (Continued)**

| Code | Name | Description |
|---|---|---|
| 41 | VG_RETRY_EXPIRED | A retry expired error occurred |
| 42 | VG_TARGET_DISCONNECT | A target disconnect condition occurred |
| 43 | VG_OUT_OF_TAGS | Out of tags |
| 44 | VG_BUS_ERROR | A bus error occurred |
| *45-59* | *Reserved* | |
| 60 | VG_ALREADY_RUNNING | The module, e.g. the analyzer, is already running. The module might have to be halted to make the command succeed. |
| *61* | *Reserved* | |
| 62 | VG_NO_SETUP | No valid setup has been loaded |
| 63 | VG_INVALID_SETUP | The setup is invalid, e.g. an analyzer setup does not match the current bus type or the API version. |
| 64 | VG_SETUP | An error occurred during handling of setup data |

Some of the status (error) codes may have a corresponding string with extended information, as described in "API client interfaces" on page 293. These strings are described in Table 9-4. Note that theses error codes may also be received without any extended error information.

**TABLE 9-4. Vanguard API extended error information**

| Code | Name | Extended Error Description |
|---|---|---|
| 1 | VG_GENERAL_ERROR | Basic string that further describes the error. |
| 4 | VG_PARSER | Basic string that further describes the error. |
| 30 | VG_ACCESS | String with the following format:<br>"<address>: access failed",<br>where <address> is the address for which the access failure occurred. |
| 31 | VG_SPLIT_COMPLETION | String containing a single hexadecimal encoded 32-bit unsigned value. This value contains the Split Completion Message and should be parsed according to the 32-bit Split Completion Message Format from the PCI-X specification. |
| 34 | VG_VERIFY | String containing three hexadecimal encoded integers separated with white-space characters. The integers represent the failure address, the read value and the correct value respectively. |
| 63 | VG_INVALID_SETUP | Basic string that further describes the error. |

## 9.5 Command overview

### Miscellaneous Commands

The miscellaneous API commands are listed below. These commands require only a VG-API license



### Analyzer Commands

The analyzer commands are listed below. In order to access the analyzer commands, a default analyzer license is required in addition to the API license.



### Protocol Checker Commands

The protocol checker commands are listed below. In order to access the protocol checker commands, a default protocol checker license is required in addition to the API license.

## Exerciser Commands

The exerciser commands are listed below. In order to access the exerciser commands, a default exerciser license is required in addition to the API license.

**E2 Only –** In addition to the default exerciser license, an E2 license is required to access the `exerciser_enhanced_target` and `exerciser_enhanced_bar` commands

## 9.6 Command description

Detailed descriptions of all Vanguard API commands are given in the following sections. Note that in the descriptions, the common parts like e.g. tags and status code for the Raw ASCII interface, are skipped.

In the API descriptions, 5 different data types are used for parameters and returned values. The mapping of these types are different for each API client interface, as described in Table 9-5.

**TABLE 9-5. Data type mapping**

| API Type | Raw ASCII mapping | XML-RPC mapping |
|---|---|---|
| `int` | 32-bit signed integer formatted as a string | XML-RPC `<int>` value |
| `boolean` | Formatted as a "1" string for `true` and a "0" string for `false` | XML-RPC `<boolean>` value |
| `string` | Normal ASCII string | XML-RPC `<string>` value |
| `int64` | Unsigned integer (up to 64 bit) formatted as a hexadecimal string.<br>The number $200000_{10}$ (0x30D40) would e.g. be formatted as the string "30D40". | Unsigned integer (up to 64 bit) formatted as a hexadecimal string, as for Raw ASCII mode.<br>The encoded string is transferred as an XML-RPC `<string>` value |
| `binary` | Hexadecimal encoded binary data, i.e. each 8-bit byte is encoded as two ASCII characters in the range [0..9,A..F].<br>The binary sequence $0x1234ABBA_{16}$ would e.g. be formatted as the string "1234ABBA".<br>See the individual API specifications for information on e.g. byte ordering. | Hexadecimal encoded binary data. The encoding is as for the Raw ASCII mode, and the encoded string is transferred as an XML-RPC `<string>` value. |

The hexadecimal encoding of the `int64` type is partly due to the fact that the XML-RPC specification does not support 64-bit integers. Additionally, the hexadecimal form makes e.g. addresses more readable, which could ease debugging.

## vanguard_version_info

Returns various version information for the Vanguard device.

**Arguments:**

None

**Returned data:**

| | Type | Description |
|---|---|---|
| api_major | int | Vanguard API's major version number |
| api_minor | int | Vanguard API's minor version number |
| fw_major | int | Vanguard device firmware's major version number |
| fw_minor | int | Vanguard device firmware's minor version number |
| setup_major | int | Supported analyzer setup file's major version number |
| setup_minor | int | Supported analyzer setup file's minor version number |

The `setup_major` and `setup_minor` fields give the highest API analyzer setup file version supported by this API engine. The API engine supports all previous versions of the analyzer setup file format.

## vanguard_device_info

Returns various device information for the Vanguard.

**Arguments:**

None

**Returned data:**

| | Type | Description |
|---|---|---|
| serial_no | int | Vanguard serial number |
| bus_type | int | Current bus type:<br>-1 = Unknown<br>0 = PCI<br>1 = PCI-X<br>2 = VME<br>*3 = Reserved* |
| bus_freq | int | Bus frequency, given as the period measured in picoseconds. |
| vg_type | int | Vanguard type:<br>0 = Vanguard PCI<br>1 = Vanguard CompactPCI<br>2 = Vanguard PMC<br>3 = Vanguard VME<br>*4 = Reserved*<br>5 = Vanguard PCI0SL<br>6 = Vanguard XMC<br>7 = Vanguard SAE |
| sys_slot | boolean | `true` if the Vanguard is inserted in a system slot. Valid for CompactPCI only. |
| | | |

The API forces the user to call this function initially after a connection has been established to the API server, or after a PCI reset has occurred. Apart from the `vanguard_version_info` function, all API functions will return the VG_DEV_INFO_REQUIRED status code in such conditions, until this function has been called.

## vanguard_api_option

Set/get general API options.

**Note –** These settings are not persistent, i.e. they are reset each time a new API session starts.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | option | | int | The selected option. See Table 9-6 below for available alternatives. A value of -1 resets all options to their default values | |
| 1 | | option_value | int | Value to set for the selected `option`. If this argument is skipped or invalid, the function returns the current value of the selected `option`. | |

**Returned data:**

| | Type | Description |
|---|---|---|
| option_value | int | Current value of the selected option. This value is returned only if a valid non-zero `option` is given, and if the `option_value` parameter isn't used or is invalid. |

**TABLE 9-6. Vanguard API options**

| Option | Values | Default |
|---|---|---|
| 0 = event_mask | Select which event classes to get notification messages for when using the event notification server. Could be a logical OR combination of the following flags:<br>0x01: Enable PCI Reset Assert event<br>0x02: Enable PCI Reset De-assert event<br>0x04: Enable Bus Frequency Changed event<br>0x08: Enable Analyzer Triggered event<br>0x10: Enable Analyzer Trace Full event<br>0x20: Enable Protocol Checker Triggered event | 0 |
| 1 = event_format | 1 = Format event notification messages as XML<br>0 = Format event notification messages as Raw ASCII | 0 |

## vanguard_device_option

Set/get various options on the Vanguard device.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | option | | int | The selected option. See Table 9-7 on the next page for available options. | |
| 1 | | option_value | int | Value to set for the selected `option`. If this argument is skipped or invalid, the function returns the current value of the selected `option`. | |

**Returned data:**

| | Type | Description |
|---|---|---|
| option_value | int | Current value of the selected option. This value is returned only if a valid `option` is given, and if the `option_value` parameter isn't used or is invalid. |

For the exerciser function option (2), the following notes apply:

- If the user has no valid exerciser license and the image selection is different from No Exerciser (3), the function fails with a VG_AUTHORIZATION error code.
- If the image section is E2 (2) and the user have no valid E2 license, the function fails with a VG_AUTHORIZATION error code.
- If the image selection is E2 and the bus type isn't PCI-X, the function fails with a VG_PARAM error code.
- If the image selection is E2 and the bus mode (option 0) is set to PCI 33MHz or PCI 66MHz, the function fails with a VG_PARAM error code.
- If the image selection is PCI Only (1) and the bus mode (option 0) is set to a different value than PCI 33MHz or PCI 66MHz, the function fails with a VG_PARAM error code. PCI Only (1) will also fail with a VG_PARAM error code when the bus type is PCI-X.
- If the bus frequency is above 100MHz, only the E2 option (if licensed) and the No Exerciser option is valid. Other values will make the function fail with a VG_PARAM error code

**TABLE 9-7. Vanguard device options**

| Option | Description |
|---|---|
| 0 = Bus mode | Controls the behavior of PCIXCAP and M66EN during PCI reset. For 0-slot, this option controls the test bus. Valid values are:<br>0 = PCI 33MHz<br>1 = PCI 66MHz<br>2 = PCI-X 66MHz<br>3 = PCI-X 100MHz (CompactPCI only)<br>4 = PCI-X 133MHz |
| 1 = Host bus mode | 0-slot only. Controls the behavior of PCIXCAP and M66EN during PCI reset for the 0-slot host bus. Valid values are:<br>0 = PCI 33MHz<br>1 = PCI 66MHz<br>2 = PCI-X 66MHz<br>*3 = Reserved*<br>4 = PCI-X 133MHz |
| 2 = Exerciser function | Controls the Vanguard exerciser functionality. Valid values are:<br>0 = PCI/PCI-X exerciser with automatic mode detection<br>1 = PCI-only exerciser (Disable PCI-X capabilities)<br>2 = 133MHz PCI-X exerciser (E2)<br>3 = No exerciser |
| 3 = PCI Clock Frequency | CompactPCI and 0-slot only. Controls the PCI clock Frequency. To change this setting on CompactPCI, the Vanguard must be in a system slot. For 0-slot, this setting affects the test bus. Valid values are:<br>0 = Automatic selection based on Bus Mode Negotiation<br>1 = Manual 25MHz<br>2 = Manual 33MHz<br>3 = Manual 50MHz<br>4 = Manual 66MHz<br>5 = Manual 100MHz<br>6 = Manual 133MHz<br>The manual settings will take effect after the first PCI reset. |
| 4 = Bridge configuration | 0-slot only. Controls 0-slot Bridge configuration:<br>0 = Allow Bridge configuration.<br>1 = Prevent Bridge from being configured.<br>Setting this option to '1' prevents the Bridge from being seen by the Host system. This is done by disconnecting IDSEL from the bridge. |
| 5 = Trigger Output | Trigger output option. This setting is lost when the Vanguard is powered off.<br>0 = State Analyzer Source, Active High Polarity<br>1 = State Analyzer Source, Active Low Polarity<br>2 = Protocol Checker Source, Active High Polarity<br>3 = Protocol Checker Source, Active Low Polarity<br>4 = Exerciser Source, Active High Polarity<br>5 = Exerciser Source, Active Low Polarity |

## vanguard_reset

Miscellaneous Vanguard Reset options.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | option | | int | The selected reset option. See Table 9-8 below for available alternatives. | |

**Returned data:**

None

**TABLE 9-8. Vanguard reset options**

| Option | Values | Default |
|---|---|---|
| 0 = Device reset | Trigger a reboot of the Vanguard device. This command will return a status code of VG_OK (0) to the user, then wait a little while before the Vanguard is rebooted. Note that since the device is rebooted, the API connection will go down and the API user will have to manually reconnect to the Vanguard afterwards. | |
| 1 = PCI Reset | Trigger a PCI Reset. CompactPCI and 0-slot only. Additionally, the exerciser must be enabled.<br>On CompactPCI, the Vanguard must be in a system slot for this command to be valid.<br>On 0-slot, the Bridge Configuration bit must be set to "1" for this command to be valid. See option 4 for the `vanguard_device_option` command for information on how to set this bit.<br>The `vanguard_device_info` function must be called following this command. Otherwise all API functions will return the VG_DEV_INFO_REQUIRED status code. | |

## vanguard_selftest

Runs a set of self tests on the Vanguard device.

**Arguments:**

None

**Returned data:**

None

---

**Note –** This command takes quite some time to run, approximately 1 minute.

---

If the self test fails, a status code of VG_SELFTEST is returned.

## analyzer_setup

Set up the analyzer with a BusView generated setup.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | setup_file | | string | BusView-generated API-compliant setup (contents of a *.ast file) | |

**Returned data:**

None

Creating a Vanguard API compliant setup is done using BusView GUI in a normal manner. When the setup is finished, an API specific setup file is created using BusView's "Save as..." menu option (from the File menu), and selecting "Analyzer Setup API Files (*.ast)" as the file type in the dialog box.

The contents of the BusView generated API setup file is already in a pure ASCII string format, and the contents of such files can be copied directly into the `setup_file` argument. For the Raw ASCII mode, the file contents must be terminated with a CR/LF pair.

---

**Note –** From BusView's point of view, ast-files are write-only, i.e. BusView can't open ast-files for e.g. modification. If the API user wants the option to modify or view the setup later, the setup should also be saved as a default BusView setup (stp-file).

---

## analyzer_run

Runs the analyzer.

**Arguments:**

None

**Returned data:**

None

## analyzer_halt

Halts the analyzer.

**Arguments:**

None

**Returned data:**

None

## analyzer_status

Gets analyzer- and trace status information.

**Arguments:**

None

**Returned data:**

| | Type | Description |
|---|---|---|
| trace_status | int | The state of the analyzer:<br>0: Trace is Empty<br>1: Analyzer is running<br>2: Analyzer has triggered<br>3: Trace is full<br>4: Analyzer has halted<br>5: Error |
| | *int* | *Reserved* |
| | *int* | *Reserved* |
| trace_length | int | Number of valid samples in trace. This field is only valid if the analyzer is halted or trace is full, e.g. `trace_status` field equals 3 or 4. |
| trigger_pos | int | Sample (sequence) number of trigger position (zero-based). This field is only valid if the analyzer is halted or trace is full, e.g. `trace_status` field equals 3 or 4. |
| samp_speed | int | Sampling speed, given as the period measured in picoseconds |
| samp_mode | int | Sampling mode:<br>0 = Transfer Mode<br>1 = Clock Mode<br>2 = Standard Mode<br>3 = Undefined Mode |
| triggered | boolean | `true` if analyzer has triggered. This field is only valid if the analyzer is halted or trace is full, e.g. `trace_status` field equals 3 or 4. |

## analyzer_trace_data

Gets analyzer trace data.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | start_line | | int | Sequence number (zero-based) of first trace line (sample) to download. If `start_line` is out of range, as given by the `trace_length` returned from the `analyzer_status` command, this function will return no trace data. | |
| 1 | end_line | | int | Sequence number (zero-based) of last trace line (sample) to download. If `end_line` is out of range, as given by the `trace_length` returned from the `analyzer_status` command, `end_line` will be adjusted to the last valid sample. | |

**Returned data:**

| | Type | Description |
|---|---|---|
| start_line | int | Sequence number of the first trace line downloaded (zero-based). If the returned `data_length` is 0, `start_line` is always 0. |
| end_line | int | Sequence number of the last trace line downloaded (zero-based). If the returned `data_length` is 0, `end_line` is always 0. |
| data_length | int | length of `trace_data` |
| trace_data | binary | downloaded trace data. Each line of trace data is 320 bits (40 bytes) wide. The format of each trace line is given in Table 9-9 below. |

**TABLE 9-9. PCI/PCI-X trace line format**

| Bits | Name | Description |
|---|---|---|
| 63-0 (64) | TimeTag | Absolute time tag for this trace line. This is a 64-bit signed integer stored in a little endian format, i.e. byte 0 is stored in bits 0-7. The absolute time tag gives the PCI clock count of the trace line, relative to the trigger line. See also bit 311 (TT_overflow). The corresponding clock frequency is given by the `samp_speed` value from the `analyzer_status` command. |
| 64-127 (64) | A[63:0] | This is the PCI address A[63:0] from the PCI lines AD[31:0], stored in a little endian format. A[63:0] is latched in Address Phase(s), and A[62:32] is only used in DAC commands. A[63:0] is incremented in hardware for each data phase |
| 191-128 (64) | D[63:0] | This is the PCI data D[63:0] from the PCI lines AD[63:0], stored in a little endian format. D[63:32] is only used when transferring 64-bit data. These bits contain PCI AD[63:0] for all clocks when sampling in Clock- and Standard mode. |
| 227-192 (36) | Attrib[35:0] | These are the signals from the PCI-X Attribute phase transferred on PCI lines C/BE[3:0]# and AD[31:0]. Attrib[35:0] is only used in PCI-X mode and is latched in the PCI-X Attribute phase and kept throughout the transaction. The byte count is decremented in hardware for each data phase.<br>Bits 199-192 (8): Attribute Phase, ByteCount Lower bits / Secondary Bus Number in Configuration Transaction.<br>Bits 203-200 (4): Attribute Phase, ByteCount Upper / BE[3:0]# PCI-X cycles using the DWORD command.<br>Bits 219-204 (16): Attribute Phase, Requester ID /Split Completion Message Bit.<br>Bits 224-220 (5): Attribute Phase, Tag<br>Bit 225: Attribute Phase, Relaxed Ordering bit / Split Completion Message bit<br>Bit 226: Attribute Phase, No Snoop bit / Split Completion Error bit<br>Bit 227: Attribute Phase, Split Completion Byte Count modified bit |
| 231-228 (4) | C[3:0] | This is the PCI command from PCI lines C/BE[3:0]#. C[3:0] is latched in the Address Phase and kept throughout the transaction. |
| 232 | A64 | A64 indicates Dual Address Cycle, and is set in the Address Phase when 64-bit address, and kept throughout the transaction |
| 233 | D64 | This signal indicates that the transfer is 64-bit |
| 241-234 (8) | BE[7:0] | These are the Byte Enable bits 7:0 from PCI lines C/BE{7:0]#. These bits contain C/BE[7:0]# for all clocks in Clock- and Standard mode. |
| 242 | FRAME# | PCI FRAME# |
| 243 | REQ64# | PCI REQ64# |
| 244 | IRDY# | PCI IRDY# |
| 245 | DEVSEL# | PCI DEVSEL# |
| 246 | ACK64# | PCI ACK64# |

**TABLE 9-9. PCI/PCI-X trace line format (Continued)**

| Bits | Name | Description |
|---|---|---|
| 247 | TRDY | PCI TRDY# |
| 248 | STOP# | PCI STOP# |
| 249 | LOCK# | PCI LOCK# |
| 250 | IDSELA | PCI IDSEL |
| 251 | IDSELB | IDSELB is only used on PMC and is for the second PMC device on the same connector |
| 252 | PAR | PCI PAR. In Transfer mode with data, PAR is advanced one clock cycle to be in sync with the corresponding data on D[31:0] |
| 253 | PAR64 | PCI PAR64. In Transfer mode with data, PAR is advanced one clock cycle to be in sync with the corresponding data on D[63:32] |
| 254 | PERR# | PCI PERR#. Advanced two clock cycles to be in sync with the corresponding data in Transfer mode with data. |
| 255 | SERR# | PCI SERR# |
| 256 | RST# | PCI RST# |
| 258-257 (2) | GNT[1:0]# (GNTB#, GNTA#) | These signals reflects the PCI GNT# signals in the slot where the Vanguard is installed GNTA# is the GNT# signal for the Vanguard Exerciser, and GNTB# will be the GNT# signal to the second PMC device on the same connector.<br>The GNTA# and GNTB# signals are latched in the arbitration phase if GNT latching is turned on for these signals |
| 259 | DWORD_CMD | Active when a DWORD command occurs on PCI-X |
| 260 | DAC | DAC is active in both the first and second address phase of the Dual Address Cycle |
| 262-261 (2) | REQ[1:0]# (REQB#, REQA#) | REQA# is the PCI REQ# signal where the Vanguard is installed. REQB# is for the second PMC device on the same connector |
| 263 | INTA# | PCI INTA# |
| 264 | INTB# | PCI INTB# |
| 265 | INTC# | PCI INTC# |
| 266 | INTD# | PCI INTD# |
| 267 | INTS | Compact PCI signal INTS |
| 268 | INTP | Compact PCI signal INTP |
| 269 | ENUM# | Compact PCI signal ENUM# |
| 270 | PME# | PCI PME# |
| 271 | Start | This signal will have different meaning depending on the sampling mode. In Transfer mode, Start will be active at the first sample of a burst transfer. In Clock- and Standard mode, Start will be active in the address phase(s). |
| 272 | Burst | Burst indicates burst transfers, and thus will be active in the data phase in transactions containing more than one data phase |

**TABLE 9-9. PCI/PCI-X trace line format (Continued)**

| Bits | Name | Description |
|---|---|---|
| 273 | Attrib/Retry | This signal will have different meaning depending on PCI or PCI-X bus, and on the sampling mode.<br>Attrib is used on PCI-X and in Clock- and Standard mode only to indicate Attrib phase. Retry is used on PCI only to indicate Target Retry. |
| 274 | Master Abort | This signal will be active when Master Abort is detected on PCI or PCI-X |
| 280-275 (6) | Latency[5:0] | Latency[5:0] shows the number of clock cycles from FRAME# to TRDY# (Target Initial Latency) |
| 283-281 (3) | DecSpeed[2:0] | Decode speed. Shows the number of clocks from address phase (FRAME# high to low) to DEVSEL# on SAC, and from the 2nd address phase when DAC |
| 287-284 (4) | ExtInput[3:0] / GNT[3:0]# | External inputs [3:0] or GNT# signals [3:0] (from pin header on front panel).<br>The GNT[3:0]# signals are latched in the arbitration phase if GNT latching is turned on for these signals (individually for each signal) |
| 291-288 (4) | ExtInput[7:4] / GNT[7:4]# | External inputs [7:4] or GNT# signals [7:4] (from pin header on front panel).<br>The GNT[7:4]# signals are latched in the arbitration phase if GNT latching is turned on for these signals (individually for each signal) |
| *307-292 (16)* | *Spare* | *Reserved* |
| 308 | PChkTrg | PChkTrg is the trigger from the protocol checker. |
| 309 | EXERtrg | This is the trigger output from the Exerciser |
| *310* | *Spare* | *Reserved* |
| 311 | TT_overflow | Indicates timetag calculation overflow. When set, the absolute time tag (bits 0-63) is greater than the given value for samples after the trigger trace-line and less than the given value for samples before the trigger trace-line. |
| 319-312 (8) | ExtInput[15:8] / GNT[15:8]# | External inputs [15:8], or GNT# signals [15:8] (from pin header on analyzer card).<br>The GNT[15:8]# signals are latched in the arbitration phase if GNT latching is turned on for these signals (individually for each signal). |

## pcheck_setup

Sets up the Protocol Checker with the proper violation mask.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | mask_bits | | binary | An array of 8-bit bytes, each holding a set of violation mask bits. To enable a violation, set the corresponding mask bit. The number of bytes must match the number of violation mask bytes. PCI uses 6 bytes, while PCI-X uses 9. The bytes must be ordered in ascending order, i.e. the first byte in the array represents mask byte 0.<br>The possible violations are given in Table 9-10 and Table 9-11. | |

**Returned data:**

None

**TABLE 9-10. PCI-X violations**

| Index | Byte:Bit | Description |
|---|---|---|
| *0* | *0:0* | *Reserved* |
| 1 | 0:1 | Illegal FRAME# assertion |
| 2 | 0:2 | Illegal FRAME# de-assertion |
| 3 | 0:3 | No termination when byte count satisfied |
| 4 | 0:4 | Burst termination not on ADB |
| 5 | 0:5 | REQ64# not synced to FRAME# |
| 6 | 0:6 | ACK64# not synced to DEVSEL# |
| 7 | 0:7 | ACK64# without REQ64# |
| 8 | 1:0 | Illegal IRDY# assertion |
| 9 | 1:1 | Illegal IRDY# de-assertion |
| 10 | 1:2 | Illegal DEVSEL# assertion |
| 11 | 1:3 | Illegal DEVSEL# de-assertion |
| 12 | 1:4 | Illegal STOP# assertion |
| 13 | 1:5 | Illegal STOP# de-assertion |
| 14 | 1:6 | Illegal TRDY# assertion |
| 15 | 1:7 | Illegal TRDY# de-assertion |
| 16 | 2:0 | Missing Master Abort |
| 17 | 2:1 | Illegal decode time |

**TABLE 9-10. PCI-X violations (Continued)**

| Index | Byte:Bit | Description |
|---|---|---|
| 18 | 2:2 | SpResp, TAbort or Retry after 8 clocks |
| 19 | 2:3 | SDtaPhD, Data or DNxtADB after 16 clocks |
| 20 | 2:4 | DEVSEL# asserted in Special Cycle |
| 21 | 2:5 | Split response after data transfer |
| 22 | 2:6 | SDtaPhD after data transfer |
| 23 | 2:7 | Retry after data transfer |
| 24 | 3:0 | Illegal target wait states |
| 25 | 3:1 | Illegal change of target signaling |
| 26 | 3:2 | REQ64# asserted with DWORD command |
| 27 | 3:3 | DAC with high address = 0 |
| 28 | 3:4 | DAC followed by DAC |
| 29 | 3:5 | DAC used with no memory command |
| 30 | 3:6 | Use of reserved command |
| 31 | 3:7 | Exceeding 64-bit address range |
| 32 | 4:0 | Illegal address and byte enables |
| 33 | 4:1 | Target respond to reserved command |
| 34 | 4:2 | Illegal split response |
| 35 | 4:3 | Illegal target response to split completion |
| 36 | 4:4 | High address change in DAC |
| 37 | 4:5 | Bus command change in DAC |
| 38 | 4:6 | AD[63:32] not high in attribute phase |
| 39 | 4:7 | C/BE[7:4]# not high in attribute phase |
| 40 | 5:0 | C/BE# not high in response phase |
| 41 | 5:1 | C/BE# not high in data phase |
| 42 | 5:2 | Wait states not in pair for write/split completion |
| 43 | 5:3 | Wrong odd DWORD data copy |
| 44 | 5:4 | Wrong odd DWORD BE copy |
| 45 | 5:5 | Data change in target response phase |
| 46 | 5:6 | BE change in target response phase |
| 47 | 5:7 | Wait state data toggle error |
| 48 | 6:0 | Wait state BE toggle error |
| 49 | 6:1 | Missing PERR# on high bus parity error |
| 50 | 6:2 | Missing PERR# on low bus parity error |
| 51 | 6:3 | PERR# reported when no parity error |
| 52 | 6:4 | Illegal PERR# assertion |
| 53 | 6:5 | Address change in configuration cycles |
| 54 | 6:6 | Illegal LOCK# assertion |
| 55 | 6:7 | Illegal LOCK# de assertion |
| 56 | 7:0 | First transaction on LOCK# not read |

**TABLE 9-10. PCI-X violations (Continued)**

| Index | Byte:Bit | Description |
|---|---|---|
| 57 | 7:1 | Attribute phase reserved bits not zero |
| 58 | 7:2 | Address phase reserved bits not zero |
| 59 | 7:3 | Use of reserved configuration type |
| 60 | 7:4 | SCE without SCM |
| 61 | 7:5 | SCM address not zero |
| 62 | 7:6 | Byte enable out of range |
| 63 | 7:7 | High address parity error |
| 64 | 8:0 | Low address parity error |
| 65 | 8:1 | High data parity error |
| 66 | 8:2 | Low data parity error |
| 67 | 8:3 | Master Abort |
| 68 | 8:4 | Target Abort |
| 69 | 8:5 | SERR# asserted |
| 70 | 8:6 | PERR# asserted |
| 71 | 8:7 | RST# asserted |

**TABLE 9-11. PCI violations**

| Index | Byte:Bit | Description |
|---|---|---|
| *0-7* | *0:0-0:7* | *Reserved* |
| 8 | 1:0 | PERR only on DataXfer |
| 9 | 1:1 | Illegal IRDY assertion |
| 10 | 1:2 | Illegal A1, A0 memory commands |
| 11 | 1:3 | Missing Master Abort |
| 12 | 1:4 | Illegal linear addr inc |
| *13* | *1:5* | *Reserved* |
| 14 | 1:6 | BE[3:0] change in data |
| 15 | 1:7 | Illegal LOCK assertion |
| 16 | 2:0 | Master Abort |
| *17* | *2:1* | *Reserved* |
| *18* | *2:2* | *Reserved* |
| 19 | 2:3 | Abnormal STOP |
| 20 | 2:4 | Illegal STOP to FRAME offset |
| 21 | 2:5 | Target protocol error |
| 22 | 2:6 | Illegal TRDY assertion |
| 23 | 2:7 | Illegal Config 0 cycle |
| 24 | 3:0 | FRAME off before IRDY |
| 25 | 3:1 | Illegal A1, A0 & BE[3:0] |
| 26 | 3:2 | Illegal command response |

**TABLE 9-11. PCI violations (Continued)**

| Index | Byte:Bit | Description |
|---|---|---|
| 27 | 3:3 | Illegal back-to-back |
| 28 | 3:4 | Illegal FRAME re-assertion |
| 29 | 3:5 | Illegal FRAME or IRDY change |
| 30 | 3:6 | Illegal Target change |
| 31 | 3:7 | Illegal FRAME to IRDY |
| 32 | 4:0 | Too many clocks in cycle |
| 33 | 4:1 | Target Abort |
| 34 | 4:2 | Illegal Master Abort |
| 35 | 4:3 | Illegal DEVSEL off |
| 36 | 4:4 | FRAME off in DAC |
| 37 | 4:5 | Illegal DEVSEL on/off |
| 38 | 4:6 | First LOCK not read |
| 39 | 4:7 | RST# asserted |
| 40 | 5:0 | Data parity error |
| 41 | 5:1 | Address parity error |
| 42 | 5:2 | PERR# asserted |
| *43-47* | *5:3-5:7* | *Reserved* |

## pcheck_option

Set/get Protocol Checker options.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | option | | int | The selected option. See Table 9-12 below for available alternatives. A value of -1 resets all options to their default values | |
| 1 | | option_value | int | Value to set for the selected `option`. If this argument is skipped, the function returns the current value of the selected `option`. | |

**Returned data:**

| | Type | Description |
|---|---|---|
| option_value | int | Current value of the selected option. This value is returned only if a valid non-zero `option` is given, and if the `option_value` parameter isn't used. |

**TABLE 9-12. Protocol checker options**

| Option | Values | Default |
|---|---|---|
| 0 = lock_on_first | 1 = Set the lock on first option<br>0 = Clear the lock on first option | 0 |
| 1 = auto_clear | 1 = Set the auto clear option<br>0 = Clear the auto clear option | 0 |
| 2 = clear_on_run | 1 = Set the clear on run option<br>0 = Clear the clear on run option | 0 |

## pcheck_run

Runs the Protocol Checker.

**Arguments:**

None

**Returned data:**

None

## pcheck_halt

Halts the Protocol Checker.

**Arguments:**

None

**Returned data:**

None

## pcheck_status

Gets the Protocol Checker status.

**Arguments:**

None

**Returned data:**

| | Type | Description |
|---|---|---|
| status | int | Protocol Checker status:<br>0 = Protocol Checker is running<br>1 = Protocol Checker has triggered<br>2 = Protocol Checker is halted<br>3 = Protocol Checker has been cleared |
| violation_bits | binary | An array of 8-bit bytes, each holding a set of violation status bits. The number of bytes must match the number of violation status registers. PCI uses 6 registers, while PCI-X uses 9. The bytes must be ordered in a ascending order, i.e. the first byte in the array represents violation byte 0. See the API description for the `pcheck_setup` command for the individual bit definitions. If a bit is set, the corresponding violation has occurred. |

## pcheck_clear

Clears Protocol Checker violations.

**Arguments:**

None

**Returned data:**

None

## exerciser_status

Get general exerciser status information.

**Arguments:**

None

**Returned data:**

| | Type | Description |
|---|---|---|
| exer_type | int | Exerciser type:<br>-2 = Exerciser failed to load<br>-1 = No exerciser available<br>0 = Normal exerciser<br>1 = Enhanced exerciser |
| exer_init_status | int | Exerciser initialization status:<br>0 = No Master Init: The exerciser could not do config cycles to set the master enable bit. Use of the exerciser commands requires that the test system set the master enable bit.<br>1 = Normal<br>2 = No Grant: The exerciser does not respond, possibly due to no grant. |

## exerciser_read

Reads data from Memory, I/O or Configuration space.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | start_addr | | int64 | Start address of area to read. | |
| 1 | | addr_space | int | Address space<br>0= Memory Space<br>1= I/O Space<br>2= Configuration Space | 0 |
| 2 | | data_size | int | Size of each data object 1, 2, 4 or 8 bytes | 4 |
| 3 | | burst_len | int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| 4 | | read_cmd | int | **PCI Command**<br>0= Memory read<br>1= Memory read multiple<br>2= Memory read line<br>**PCI-X Command**<br>0= Memory read DWORD<br>1= Memory read block<br>2= Alias to Memory read block (not allowed, according to PCI-X spec. ) | 0 |
| 5 | | end_addr | int64 | End address of the area to read.<br>Maximum is `start_addr` + FFFF | `start_addr` + FF |

**Returned data:**

| | Type | Description |
|---|---|---|
| base_addr | int64 | Start address of returned data. As the returned data is aligned on a `data_size` boundary, this address could be different from the `start_addr` input parameter. |
| data_lengt | int | Length in bytes of the following data |
| data | binary | Returned data ordered in byte order. |

**Note –** The `end_addr` argument is used to set the block size of the `exerciser_read` command, i.e. if the `end_addr` is set to s`tart_addr+0x7f`, only `0x80` bytes of data are read in each block.

**Note –** In Configuration space all accesses are DWORD aligned, since A1 and A0 are used to indicate Configuration cycle type 0 or 1.

## exerciser_write

Writes data into Memory, I/O or Configuration space.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | data | | binary | Data to write ordered in byte order | |
| 1 | start_addr | | int64 | Start address of area to write to | |
| 2 | | addr_space | int | Address space<br>0= Memory Space<br>1= I/O Space<br>2= Configuration Space | 0 |
| 3 | | data_size | int | Size of each data object 1, 2, 4 or 8 bytes | 4 |
| 4 | | burst_len | int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| 5 | | write_cmd | int | **PCI Command**<br>0= Memory write<br>1= Memory write and invalidate<br>**PCI-X Command**<br>0= Memory write<br>1= Memory write block<br>2= Alias to Memory write block (not allowed, according to PCI-X spec. ) | 0 |

**Returned data:**

None

When the burst option is used, the data is entered at the desired address in the same way as for single cycle. However, the data is not written until the command is executed, or until the number of bytes entered is equal to the burst length.

**Note –** In Configuration space all accesses are DWORD aligned, since A1 and A0 are used to indicate Configuration cycle type 0 or 1.

## exerciser_fill

Fills Memory, I/O or Configuration space with a given pattern or value.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | start_addr | | int64 | Start address of the fill area | |
| 1 | end_addr | | int64 | End address of the fill area | |
| 2 | | pattern_id | int | Pattern identifier defined as follows:<br>0= Walking One pattern<br>1= Walking Zero pattern<br>2= address as data<br>3= random data<br>4= use pattern in `pattern_data`<br>5= use local data from `pattern_data` | 0 |
| 3 | | addr_space | int | Address space<br>0= Memory Space<br>1= I/O Space<br>2= Configuration Space | 0 |
| 4 | | data_size | int | Size of each data object 1, 2, 4 or 8 bytes | 4 |
| 5 | | burst_len | int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| 6 | | write_cmd | int | **PCI Command**<br>0= Memory, I/O or Config write<br>1= Memory write and invalidate<br>**PCI-X Command**<br>0= Memory, I/O or Config write<br>1= Memory write block<br>2= Alias to Memory write block (not allowed, according to PCI-X spec. ) | 0 |
| 7 | | pattern_data | int64 | If `pattern_id` = 4, this value represents the pattern to use.<br>If `pattern_id` = 5, this value is the Local memory address of data pattern. | 0 |

**Returned data:**

None

## exerciser_load

Load Memory space with data. The purpose of this command is to provide an alternative to `exerciser_write` for filling large amounts of memory space, as the (more versatile) `exerciser_write` command has a certain amount of run-time overhead for large data buffers.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | data | | binary | Data to load ordered in byte order | |
| 1 | dst_addr | | int64 | Destination address | |

**Returned data:**

None

## exerciser_exercise

Reads and writes cycles with varying data sizes.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | start_addr | | int64 | Start address of the fill area | |
| 1 | end_addr | | int64 | End address of the fill area | |
| 2 | | pattern_id | int | Pattern identifier defined as follows:<br>0= Walking One pattern<br>1= Walking Zero pattern<br>2= address as data<br>3= random data<br>4= use pattern in `pattern_data`<br>5= use local data from `pattern_data` | 0 |
| 3 | | repeat | int | Number of times to repeat the DMA transfer<br>Must be >= 1 | 1 |
| 4 | | burst_len | int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| 5 | | read_cmd | int | **PCI Command**<br>0 = Memory, I/O or Config read<br>1 = Memory read multiple<br>2 = Memory read line<br>**PCI-X Command**<br>0 = Memory DWORD, I/O or Config read<br>1 = Memory read block<br>2 = Alias to Memory read block (not allowed, according to PCI-X spec. ) | 0 |
| 6 | | write_cmd | int | **PCI Command**<br>0= Memory, I/O or Config write<br>1 = Memory write and invalidate<br>**PCI-X Command**<br>0 = Memory, I/O or Config write<br>1 = Memory write block<br>2 = Alias to Memory write block (not allowed, according to PCI-X spec. ) | 0 |
| 7 | | pattern_data | int64 | If `pattern_id` = 4, this value represents the pattern to use.<br>If `pattern_id` = 5, this value is the Local memory address of the data pattern. | 0 |

**Returned data:**

None

## exerciser_test

Repeatedly fills Memory, I/O or Configuration space with a given pattern, reads it back, and checks for errors.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | start_addr | | int64 | Start address of the fill area | |
| 1 | end_addr | | int64 | End address of the fill area | |
| 2 | | pattern_id | int | Pattern identifier defined as follows:<br>0= Walking One pattern<br>1= Walking Zero pattern<br>2= address as data<br>3= random data<br>4= use pattern in `pattern_data`<br>5= use local data from `pattern_data` | 0 |
| 3 | | repeat | int | Number of times to repeat the DMA transfer<br>Must be >= 1 | 1 |
| 4 | | addr_space | int | Address space<br>0 = Memory Space<br>1= I/O Space<br>2 = Configuration Space | 0 |
| 5 | | data_size | int | Size of each data object 1, 2, 4 or 8 bytes | 4 |
| 6 | | burst_len | int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| 7 | | read_cmd | int | **PCI Command**<br>0 = Memory, I/O or Config read<br>1 = Memory read multiple<br>2 = Memory read line<br>**PCI-X Command**<br>0 = Memory DWORD, I/O or Config read<br>1 = Memory read block<br>2 = Alias to Memory read block (not allowed, according to PCI-X spec. ) | 0 |
| 8 | | write_cmd | int | **PCI Command**<br>0 = Memory, I/O or Config write<br>1 = Memory write and invalidate<br>**PCI-X Command**<br>0 = Memory, I/O or Config write<br>1 = Memory write block<br>2 = Alias to Memory write block (not allowed, according to PCI-X spec. ) | 0 |
| 9 | | pattern_data | int64 | If `pattern_id` = 4, this value represents the pattern to use.<br>If `pattern_id` = 5, this value is the Local memory address of the data pattern. | 0 |

**Returned data:**

None

**EXERtrg# on verify error:**

The `exerciser_test` command asserts a trigger signal (EXERtrg#) to the PCI/PCI-X Bus Analyzer when a verify error occurs. The test stops at the first verify error and returns the corresponding error.

It is required to run the test with single cycles when the EXERtrg# trigger signal is taken into use in the analyzer, because the trigger signal is asserted during the verify process. When burst cycles are used, an error can occur in the first transfer of the burst, but the trigger signal will not be asserted until the burst is finished and the verify process has started.

## exerciser_compare

Repeatedly reads Memory, I/O or Configuration space, and compares the data with a given pattern.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | start_addr | | int64 | Start address of the fill area | |
| 1 | end_addr | | int64 | End address of the fill area | |
| 2 | | pattern_id | int | Pattern identifier defined as follows:<br>0= Walking One pattern<br>1= Walking Zero pattern<br>2= address as data<br>3= random data<br>4= use pattern in `pattern_data`<br>5= use local data from `pattern_data` | 0 |
| 3 | | repeat | int | Number of times to repeat the DMA transfer<br>Must be >= 1 | 1 |
| 4 | | addr_space | int | Address space<br>0 = Memory Space<br>1 = I/O Space<br>2 = Configuration Space | 0 |
| 5 | | data_size | int | Size of each data object 1, 2, 4 or 8 bytes | 4 |
| 6 | | burst_len | int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| 7 | | read_cmd | int | **PCI Command**<br>0 = Memory, I/O or Config read<br>1 = Memory read multiple<br>2 = Memory read line<br>**PCI-X Command**<br>0 = Memory DWORD, I/O or Config read<br>1 = Memory read block<br>2 = Alias to Memory read block (not allowed, according to PCI-X spec. ) | 0 |
| 8 | | pattern_data | int64 | If `pattern_id` = 4, this value represents the pattern to use.<br>If `pattern_id` = 5, this value is the Local memory address of the data pattern. | 0 |

**Returned data:**

None

**EXERtrg# on verify error:**

The Compare command asserts a trigger signal (EXERtrg#) to the PCI/PCI-X Bus Analyzer when a verify error occurs. The test stops at the first verify error and returns the corresponding error.

It is required to run the test with single cycles when the EXERtrg# trigger signal is taken into use in the analyzer, because the trigger signal is asserted during the verify process. When burst cycles are used, an error can occur in the first transfer of the burst, but the trigger signal will not be asserted until the burst is finished and the verify process has started.

## exerciser_cycle_sequence

Generates a sequence of PCI/PCI-X cycles. This command is useful to put traffic on the bus.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | start_addr | | int64 | Start address of the fill area | |
| 1 | end_addr | | int64 | End address of the fill area | |
| 2 | | pattern_id | int | Pattern identifier defined as follows:<br>0= Walking One pattern<br>1= Walking Zero pattern<br>2= address as data<br>3= random data<br>4= use pattern in `pattern_data`<br>5= use local data from `pattern_data` | 0 |
| 3 | | repeat | int | Number of times to repeat the DMA transfer<br>Must be >= 1 | 1 |
| 4 | | addr_space | int | Address space<br>0 = Memory Space<br>1 = I/O Space<br>2 = Configuration Space | 0 |
| 5 | | data_size | int | Size of each data object 1, 2, 4 or 8 bytes | 4 |
| 6 | | burst_len | int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| 7 | | cmd | int | **PCI Command**<br>0 = Memory, I/O or Config read<br>1 = Memory read multiple<br>2 = Memory read line<br>3 = Memory, I/O or Config write<br>4 = Memory write and invalidate<br>**PCI-X Command**<br>0 = Memory DWORD, I/O or Config read<br>1 = Memory read block<br>2 = Alias to Memory read block (not allowed, according to PCI-X spec. )<br>3 = Memory, I/O or Config write<br>4 = Memory write block<br>5 = Alias to Memory write block (not allowed, according to PCI-X spec. ) | 0 |
| 8 | | pattern_data | int64 | If `pattern_id` = 4, this value represents the pattern to use.<br>If `pattern_id` = 5, this value is the Local memory address of the data pattern. | 0 |

**Returned data:**

None

## exerciser_dma

Runs DMA transfers on the bus.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | channel | | int | DMA channel number 1 or 2 | |
| 1 | src_addr | | int64 | Source start address | |
| 2 | dst_addr | | int64 | Destination start address | |
| 3 | transfer_size | | int | Number of bytes to transfer | |
| 4 | | data_size | int | Size of each data object, 4 or 8 bytes | 4 |
| 5 | | dir | int | 0 = PCI to PCI memory<br>1 = PCI to local memory<br>2 = local to PCI memory | 0 |
| 6 | | repeat | int | Repeat count for DMA transfer (0 = forever) | 1 |
| 7 | | burst_len | int | burst length in bytes<br>PCI : 0 = system dependant<br>PCI-X: 0 = 4096 bytes | 4 |
| 8 | | read_cmd | int | **PCI Command**<br>0= Memory read<br>1= Memory read multiple<br>2= Memory read line<br>**PCI-X Command**<br>0= Memory read DWORD<br>1= Memory read block<br>2= Alias to Memory read block (not allowed, according to PCI-X spec. ) | 0 |
| 9 | | write_cmd | int | **PCI Command**<br>0 = Memory write<br>1 = Memory write and invalidate<br>**PCI-X Command**<br>0 = Memory write<br>1 = Memory write block<br>2 = Alias to Memory write block (not allowed, according to PCI-X spec. ) | 0 |
| 10 | | src_incr | boolean | Increment source address for every successful data transfer.<br>`true` = Enabled<br>`false` = Disabled | `true` |
| 11 | | dst_incr | boolean | Increment source address for every successful data transfer.<br>`true` = Enabled<br>`false` = Disabled | `true` |
| 12 | | start_on_trg | boolean | Start DMA transfer on trigger from the Analyzer<br>`true` = Enabled<br>`false` = Disabled | `false` |

**Note –** If no parameters are specified, DMA status is returned. If any parameters are given, the first four parameters are required.

**Returned data:**

| | Type | Description |
|---|---|---|
| dma1_run_status | int | Status of DMA controller 1:<br>0 = Running<br>1 = Waiting for trigger<br>2 = Done<br>3 = Aborted<br>4 = Error |
| dma1_exer_status | int | Exerciser status for DMA controller 1:<br>0 = Status OK<br>1 = Master Abort<br>2 = Target Abort<br>*3 = Reserved*<br>4 = Data Parity error<br>5 = Address Parity error<br>*6 = Reserved*<br>7 = Retry expired<br>8 = Target Disconnect<br>9 = Split Completion error<br>10 = General error<br>*11 = Reserved*<br>12 = Exerciser running<br>13 = Out of tags<br>14 = VME bus error<br>*15 = Reserved*<br>16 = Out of DMA descriptors |
| dma2_run_status | int | Status of DMA controller 2 |
| dma2_exer_status | int | Exerciser status for DMA controller 2 |

## exerciser_dma_abort

Aborts ongoing DMA transfers started with the `exerciser_dma` command.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | | dma_ch1 | int | Channel 1 or 2 | |
| 1 | | dma_ch2 | int | Channel 1 or 2 | |

**Note –** If no channel is specified, all running DMAs are aborted.

**Return format:**

None

## exerciser_special_cycle

Generates special cycles on the PCI/PCI-X bus.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | | data | int64 | Data value. Although having a 64-bit resolution, only the corresponding 32-bit value is used. | 0 |

**Returned data:**

None

## exerciser_local_read

Dumps Local User Memory. The Vanguard Exerciser has 8MB of Local User Memory. The Local Display command allows the user to dump Local User Memory in 256Byte blocks for display. The Local User memory ranges from address 0x0 to 0x7FFFFF and can be mapped as target memory using the Target command.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | start_addr | | int64 | Start address of the area to read. | |
| 1 | | data_size | int | Size of each data object 1, 2, 4 or 8 bytes | 4 |

**Returned data:**

| | Type | Description |
|---|---|---|
| base_addr | int64 | Start address of returned data. As the returned data is aligned on a data_size boundary, this address could be different from the address given in the input command |
| data_length | int | Length in bytes of the following data |
| data | binary | Returned data ordered in byte order |

## exerciser_local_fill

Fills local user memory on the Exerciser, with a given data pattern or value

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | start_addr | | int64 | Start address of the fill area | |
| 1 | end_addr | | int64 | End address of the fill area | |
| 2 | | pattern_id | int | Pattern identifier defined as follows:<br>0= Walking One pattern<br>1= Walking Zero pattern<br>2= address as data<br>3= random data<br>4= use pattern in `pattern_data` | 0 |
| 3 | | data_size | int | Size of each data object 1, 2, 4 or 8 bytes | 4 |
| 4 | | pattern_data | int64 | If `pattern_id` = 4, this value represents the pattern to use. | 0 |

**Returned data:**

None

## exerciser_local_copy

Copies local user Exerciser memory from one location to another.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | start_addr | | int64 | Start address of the area to copy | |
| 1 | end_addr | | int64 | End address of the area to copy | |
| 2 | dst_addr | | int64 | Destination address | |

**Returned data:**

None

## exerciser_local_load

Load local user Exerciser memory with data.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | data | | binary | Data to write ordered in byte order | |
| 1 | dst_addr | | int64 | Local destination address | |

**Returned data:**

None

## exerciser_io

I/O port control.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | port | | int | Port number to access (0-3) | |
| 1 | | value | boolean | If no `value` argument is specified, the selected `port` is read and the current value from the port is returned.<br>If the `value` argument is specified, the selected `port` is set as follows:<br>`true` = Drive the port high<br>`false` = Drive the port low | |

**Returned data:**

If the port is written (the `value` argument is set), no data is returned. If the port is read, the following table is valid,

| | Type | Description |
|---|---|---|
| port | int | The read port |
| value | boolean | The read value of the port |

## exerciser_scan

Scan for bus devices.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | | uninit_bridges | boolean | `true` = Uninitialized bridges are being programmed and scanned<br>`false` = Ignore uninitialized bridges | `true` |
| 1 | | init_bridges | boolean | `true` = Initialized bridges are being programmed and scanned<br>`false` = Ignore initialized bridges | `true` |

**Returned data:**

| | Type | Description |
|---|---|---|
| device_count | int | Number of devices found during the scan. Depending on this count, a set of device specific values will follow as stated below. Each device found will return the following 5 values. |
| vendor_id | int64 | Vendor ID for this device |
| device_id | int64 | Device ID for this device |
| bus | int | Bus number for this device |
| device | int | Device number for this device |
| function | int | Function number for this device |

## exerciser_config_data

Get configuration space data for a given device on the bus.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | bus | | int | Bus number of device to get data from | |
| 1 | device | | int | Device number of device to get data from | |
| 2 | | function | int | Function number of device to get data from | 0 |
| 3 | | reg_start | int | First 32-bit register to read. Start of configuration space is register 0 | 0 |
| 4 | | reg_end | int | Last 32-bit register to read. Start of configuration space is register 0 | 63 |

**Returned data:**

| | Type | Description |
|---|---|---|
| valid_bars | int | Number of valid BARs in BAR data given below. |
| bar0 | int64 | BAR0 contents. Valid if `valid_bars` >= 0 |
| bar0_size | int | Size of BAR0. Valid if `valid_bars` >= 0 |
| bar1 | int64 | BAR1 contents. Valid if `valid_bars` >= 1 |
| bar1_size | int | Size of BAR1. Valid if `valid_bars` >= 1 |
| bar2 | int64 | BAR2 contents. Valid if `valid_bars` >= 2 |
| bar2_size | int | Size of BAR2. Valid if `valid_bars` >= 2 |
| bar3 | int64 | BAR3 contents. Valid if `valid_bars` >= 3 |
| bar3_size | int | Size of BAR3. Valid if `valid_bars` >= 3 |
| bar4 | int64 | BAR4 contents. Valid if `valid_bars` >= 4 |
| bar4_size | int | Size of BAR4. Valid if `valid_bars` >= 4 |
| bar5 | int64 | BAR5 contents. Valid if `valid_bars` >= 5 |
| bar5_size | int | Size of BAR5. Valid if `valid_bars` >= 5 |
| data_length | int | Length of config data dump below |
| data | binary | Configuration data dump |

## exerciser_config_read

Read configuration space data for a given device on the bus.

**Arguments:**

| | Argument | | | Description | Default |
|---|---|---|---|---|---|
| | **Required** | **Optional** | **Type** | | |
| 0 | bus | | int | Bus number of device to get data from | |
| 1 | device | | int | Device number of device to get data from | |
| 2 | | function | int | Function number of device to get data from | 0 |
| 3 | | reg_start | int | First 32-bit register to read. Start of configuration space is register 0 | 0 |
| 4 | | reg_end | int | Last 32-bit register to read. Start of configuration space is register 0 | 63 |

**Returned data:**

| | Type | Description |
|---|---|---|
| reg_base | int | First 32-bit register in returned data below |
| data_length | int | Length of register data dump below |
| data | binary | Register data dump |

## exerciser_config_write

Write configuration space data for a given device on the bus.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | data | | binary | Data to write ordered in byte order. The length of the data array must correspond to an integer number of 32-bit DWORDs. | |
| 1 | bus | | int | Bus number of device to write data to | |
| 2 | device | | int | Device number of device to write data to | |
| 3 | | function | int | Function number of device to write data to | 0 |
| 4 | | reg_start | int | First 32-bit register to write. Start of configuration space is register 0 | 0 |

**Returned data:**

None

## exerciser_target

**Note –** See “exerciser_enhanced_target” on page 353 and “exerciser_enhanced_bar” on page 354 for details on mapping target windows for the Enhanced Exerciser (E2).

Maps PCI target window into Local User Memory. Windows can be located in both I/O and Memory space. When activated, parts of the Local User Memory can be accessed by other PCI/PCI-X bus masters. This is how the Exerciser can emulate a PCI/PCI-X target memory device.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | | enable | boolean | `true` = Enable<br>`false` = Disable | |
| 1 | | pci_base | int64 | PCI/PCI-X start address of the window<br>Value must be specified if `enable` is set to `true`. | 0 |
| 2 | | addr_space | int | PCI/PCI-X address space<br>0= PCI/PCI-X Memory space<br>1 = PCI/PCI-X I/O space | 0 |
| 3 | | io_shadow_base | int64 | PCI-X base address of memory mapped IO Window. Only used for PCI-X. | 0 |

**Note –** If no parameters are specified, target status is returned.

**Returned data:**

| | Type | Description |
|---|---|---|
| mem_bar_en | boolean | `true` if Memory target is enabled, otherwise `false`. |
| mem_start_addr | int64 | If `mem_bar_en` is `true`, holds the start address of the Memory bar. |
| mem_end_addr | int64 | If `mem_bar_en` is `true`, holds the end address of the Memory bar |
| io_bar_en | boolean | `true` if I/O target is enabled, otherwise `false`. |
| io_start_addr | int64 | If `io_bar_en` is `true`, holds the start address of the I/O bar |
| io_end_addr | int64 | If `io_bar_en` is `true`, holds the end address of the I/O bar |

## exerciser_enhanced_target

Enables and Disables the use of Memory and IO BARs on the PCI-X E2 exerciser.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | | enable | boolean | `true`= Enable<br>`false` = Disable | |
| 1 | | addr_space | int | PCI-X address space<br>0 = PCI-X Memory space<br>1 = PCI-X I/O space | 0 |

**Note –** If no parameters are specified, target status is returned.

**Returned data:**

| | Type | Description |
|---|---|---|
| bar0_type | int | Type of BAR:<br>-1 = Undefined BAR<br>0 = prefetchable 32bit<br>1 = memory, non-prefetchable 32bit<br>2 = prefetchable 64bit<br>3 = non-prefetchable 64bit<br>4 = I/O |
| bar0_start | int64 | Start address for BAR0. Valid if `bar0_type` >= 0 |
| bar0_end | int64 | End address for BAR0. Valid if `bar0_type` >= 0 |
| bar1_type | int | Type of BAR |
| bar1_start | int64 | Start address for BAR1. Valid if `bar1_type` >= 0 |
| bar1_end | int64 | End address for BAR1. Valid if `bar1_type` >= 0 |
| bar2_type | int | Type of BAR |
| bar2_start | int64 | Start address for BAR2. Valid if `bar2_type` >= 0 |
| bar2_end | int64 | End address for BAR2. Valid if `bar2_type` >= 0 |
| bar3_type | int | Type of BAR |
| bar3_start | int64 | Start address for BAR3. Valid if `bar3_type` >= 0 |
| bar3_end | int64 | End address for BAR3. Valid if `bar3_type` >= 0 |
| bar4_type | int | Type of BAR |
| bar4_start | int64 | Start address for BAR4. Valid if `bar4_type` >= 0 |
| bar4_end | int64 | End address for BAR4. Valid if `bar4_type` >= 0 |
| bar5_type | int | Type of BAR |
| bar5_start | int64 | Start address for BAR5. Valid if `bar5_type` >= 0 |
| bar5_end | int64 | End address for BAR5. Valid if `bar5_type` >= 0 |

**E2 Only –** This function requires a valid license for the Enhanced Exerciser..

## exerciser_enhanced_bar

Configures the BAR layout in the PCI-X E2 Exerciser. When used, parts of the Local User Memory can be accessed by other PCI-X bus masters. This is how the Exerciser can emulate a PCI-X target memory device. First set of parameters are for BAR0, the next for BAR1, and so on. If a BAR is set as a 64-bit value, two BARS are used.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | type | | int | Memory type<br>0 = prefetchable 32bit<br>1 = memory, non-prefetchable 32bit<br>2 = prefetchable 64bit<br>3 = non-prefetchable 64bit<br>4 = I/O | |
| 1 | addr | | int64 | Address range<br>0-FFFFFFFFFFFFFFFF<br>Least significant bits ignored according to `size` | |
| 2 | size | | int64 | Size minus one in bytes<br>All significant bits must be 1.<br>Range is F - FFFFFFFFFFFFFFFF<br>16 bytes - whole PCI address space | |

**Returned data:**

None. See the `exerciser_enhanced_target` command for BAR status

BARs with no parameter set are not defined.

**E2 Only –** This function requires a valid license for the Enhanced Exerciser..

## exerciser_interrupt

Generates PCI/PCI-X interrupts.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | enable | | boolean | `true` = Enable Interrupt<br>`false` = Disable Interrupt | |
| 1 | irq_lines | | int | PCI interrupt request line. Valid values are a combination of:<br>0x1= int A<br>0x2= int B<br>0x4= int C<br>0x8= int D | |

**Note –** If no parameters are specified, interrupt status is returned. If any parameters are given, all parameters are required.

**Returned data:**

| | Type | Description |
|---|---|---|
| intA | boolean | int A status:<br>`true` = Interrupt is enabled<br>`false` = Interrupt is disabled |
| intB | boolean | int B status |
| intC | boolean | int C status |
| intD | boolean | int D status |

## exerciser_interrupt_acknowledge

Generates interrupt acknowledge cycles on the PCI/PCI-X bus.

**Arguments:**

None

**Returned data:**

None

---

**Note –** As the normal response for an Interrupt Acknowledge cycle is Master Abort, this command will return a VG_OK status to the user if a Master Abort condition occurs on the bus.

---

## exerciser_option

Set/get PCI/PCI-X exerciser options.

**Arguments:**

| Argument | | | | Description | Default |
|---|---|---|---|---|---|
| | Required | Optional | Type | | |
| 0 | option | | int | The selected option. See Table 9-13 and Table 9-14 below for available alternatives. A value of -1 resets all options to their default values | |
| 1 | | option_value | int | Value to set for the selected option. If this argument is skipped, the function returns the current value of the selected option. | |

**Returned data:**

| | Type | Description |
|---|---|---|
| option_value | int | Current value of the selected option. This value is returned only if a valid non-zero `option` is given, and if the `option_value` parameter isn't used or is invalid. |

**TABLE 9-13. PCI/PCI-X options**

| Option | Values | Default |
|---|---|---|
| 0 = latency_timer | Number of clocks (0-255) remaining between losing GNT and having to release the bus if another device is requesting the bus. | PCI: 0<br>PCI-X: 64 |
| 1 = serr_en | 1 = Enable System Error driver<br>0 = Disable System Error driver | 0 |
| 2 = perr_en | 1 = Enable Parity Error response<br>0 = Disable Parity Error response | 0 |
| 3 = bus_width | 0 =Default width (set on PCI reset)<br>1 = Force 32 bit operation<br>2 = Force 64 bit operation | 0 |
| 4 = ignore_initial_latency | 1 = Ignore initial latency of 16 clocks - the exerciser will continue to issue wait states until it has data ready. | 0 |
| 5 = split_response_en | PCI-X, non-E2, ONLY<br>1 = Enable Split Response on target reads. | 1 |
| 6 = retry_counter | PCI-X ONLY<br>Number of retries before Exerciser as a master terminates the cycle (0-255)<br>0 = Forever | 255 |

**TABLE 9-14. Additional PCI-X E2 options**

| Option | Values | Default |
|---|---|---|
| 10 = retry_enable | 1 = Enable master retry<br>0 = Disable master retry | 1 |
| 11 = retry_delay | Number of clocks to delay before master retry, 0 - 65535 | 0 |
| 12 = decode_speed | Number of clocks from last address phase to target asserts DEVSEL#.<br>3 = decode speed B<br>4 = decode speed C<br>5 = Invalid decode speed according to the PCI-X spec.<br>6 = decode speed Subtractive<br>7 = Invalid decode speed according to the PCI-X spec. | 3 |
| 13 = wait_states | Force number of initial wait states.<br>4-15 = Number of wait states. | 4 |
| 14 = target_term | Force target termination<br>0 = Split response on legal commands, else Data termination.<br>1 = Always data termination.<br>2 = Target Abort.<br>3 = Single Data Phase Disconnect.<br>4 = Retry.<br>5 = Disconnect on Next ADB. | 0 |
| 15 = SpComp_err | 256 = No force of split completion error.<br><number> = Split completion error message index (0-255) | 256 |
| 16 = DNxtADB_delay | Number of clocks from DEVSEL# before Disconnect on Next ADB is signaled, 0 - 4095 | 1 |
| 17 = PAR_gen | Force PAR. Forcing the PAR signal high or low will effectively generate random parity errors on the lower AD and C/BE bus<br>0 = Normal parity generation on PAR.<br>1 = Force PAR to 0<br>2 = Force PAR to 1 | 0 |
| 18 = PAR64_gen | Force PAR64. Forcing the PAR64 signal high or low will effectively generate random parity errors on the upper AD and C/BE bus.<br>0 = Normal parity generation on PAR64.<br>1 = Force PAR to 0<br>2 = Force PAR to 1 | 0 |
| 19 = PERR_assert | Assert PERR on 2. clock after each data phase.<br>0 = Normal PERR generation<br>1 = PERR always asserted | 0 |
| 20 = SERR_assert | Assert SERR on 2. clock after each address phase.<br>0 = Normal SERR generation<br>1 = SERR always asserted | 0 |
| 21 = local_mem_select | Local memory select for local commands and local address parameters<br>0 = 8M external memory<br>1 = 4K low latency memory | 0 |

*This page intentionally left blank*

*This page intentionally left blank*

# 10 Compliance Checker

The compliance checker is a suite of tests designed to aid in the verification of ASICS, Components, Motherboards, Expansion Cards and System compliance with PCI/PCI-X Local Bus Specification. The tests performed follow the PCI Compliance Checklist version 2.2, 2.3, 3.0 and PCI-X Compliance Checklist version 1.0b - as released by the PCI SIG.

The PCI/PCI-X Compliance Checklist is required in order to use the Vanguard Compliance Checker. This is not supplied with the standard Vanguard distribution but can be obtained from CWCDS or the PCI SIG.

It is important to note that not all items on the PCI SIG - PCI Compliance Checklist are tested in this test suite. The fact that all tests in this suite pass should not be interpreted to mean that the device tested is 100% PCI Compliant.

This chapter describes the use of the compliance checker and you should refer to the relevant PCI SIG documents for:

- PCI Compliance Checklist Revision 2.2, 2.3, 3.0
- PCI-X Compliance Checklist Revision 1.0b

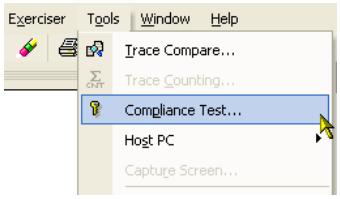
## *10.1 Operation*

Please note the following when using the Compliance Checker:

1. We recommend performing the tests in a system where there are no other active PCI devices or memory regions enabled. This is because some tests may fail if other active devices respond to tests aimed at the IUT (Implementation Under Test). This should preferably be a passive backplane.
2. There must be NO other traffic on the bus while performing these tests. This will cause the tests to fail.
3. Some tests will set the IUT BARs to various IO and memory addresses. If any of these cause a conflict in your system, for example with system memory, there are adjustment possibilities discussed later in this chapter.
4. The Compliance Checker will test only one PCI/PCI-X function at a time. The other functions will be disabled during the test.
5. The Compliance Checker will scan the bus and display all PCI/PCI-X functions on all devices found. The test will attempt to disable all devices and functions other than IUT.

### Getting Started

Install the first PCI/PCI-X device for testing into the system. The device must be on the same bus segment as the Vanguard.




The Exerciser MUST be enabled under Tools/Hardware/Options for the Compliance Checker to run.

The Compliance Checker will not open if there are other tasks running, such as the analyzer sampling or the exerciser is generating transactions.

The Compliance Checker can be started by clicking “Compliance Test” under “Tools”, or by clicking on the button in the tool bar. Once Compliance Test has been selected the Exerciser will perform a scan of the PCI/PCI-X bus for all devices.

This will open the Compliance Test dialog from which the device to be tested and type of tests are selected.

The menu contains three different sections:

- **Setup** contains all of the Compliance Tests.
- **Settings** contains the Path Settings, IUT settings, and Memory Ranges .
- **Transcript** contains the test results after a test has been generated.

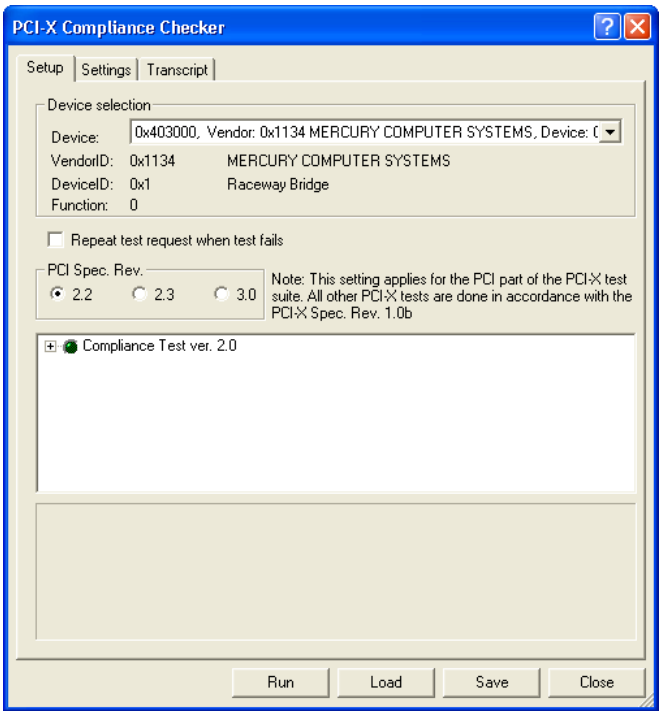



**FIGURE 1.** *Compliance Checker setup*

## Setup

### Device Selection

Select the device to be tested from the list of devices found in the “Device Selection” drop down box.

### Repeat test request when test fails

Checking this box will cause any failed tests to bring up a dialog to request a repeat.

### PCI Spec. Rev.

Select the PCI Specification Revision you wish to test against. When in PCI-X mode, this selects the PCI revision for the PCI parts of the PCI-X Specification.

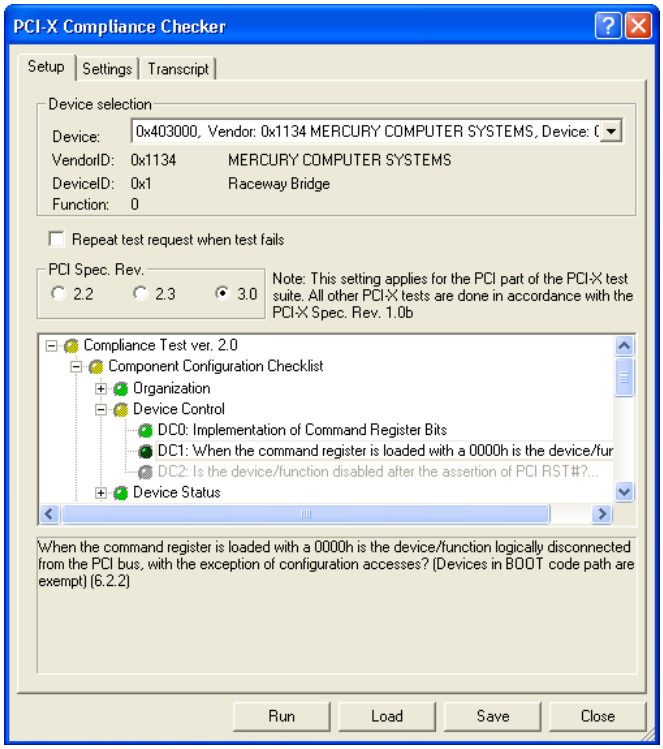
The setup for different Spec Rev is saved in the registry on every Run, Close command or when changing PCI Specification Revision.

### Compliance Check settings

When “Compliance Test” is expanded, a series of checklists will appear. Each of these can be expanded to view each individual tests listed under the specified checklists.

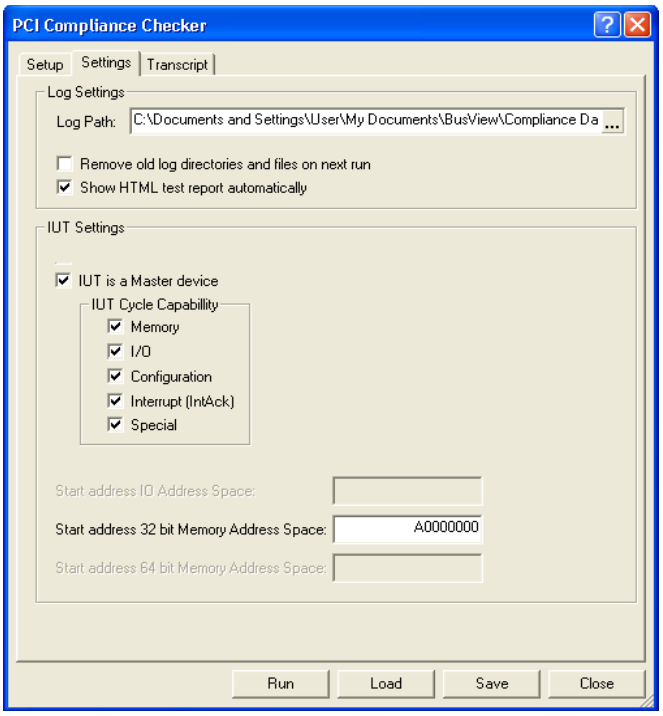
Individual tests or checklists can be enabled or disabled by clicking on the lamp next to the test or checklist. The color of the lamp has the following meaning:

- Yellow -Partial selection/All tests are not active
- Light Green -Test is active and the device will complete the selected test
- Dark Green -Test is not active or masked out
- Gray- Test is not implemented in the Compliance Checker at the present time




All of the testing selections are made from the Setup tab. Once the desired compliance test has been enabled, click “Run” and the test will start.

## Settings



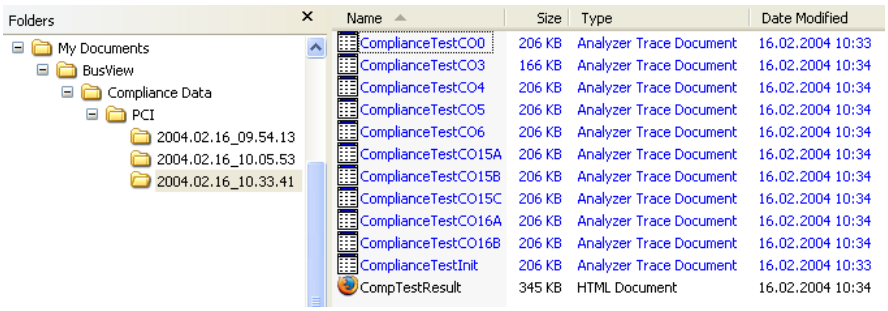

### Log Settings

The results of the Compliance Check are written to the directory defined in the Log Path; by default this is: *C:\Documents and Settings\<User>\My Documents\BusView\Compliance Data/<BusType>/<Timestamp>/*. Where:

- <User> is a folder labeled with the Windows Username used on the Host machine.
- <BusType> is the Bus mode in which the Vanguard is operating (PCI, PCI-X or VME)

- <Timestamp> is a folder labelled with the date and time at which the Compliance Check was executed.

Each time a test is run, the results are written to files in a directory named with a timestamp of when the test was performed. This directory also contains all trace files and test reports generated during the selected tests.




The results are presented in a HTML Report File. The default name for this file is: **CompTestResult.htm**

***Remove old log directories and file on next run***

Selecting the check box called “Remove old log directories and file on next run” will remove **all** previous test results each time the Compliance Checker is run.

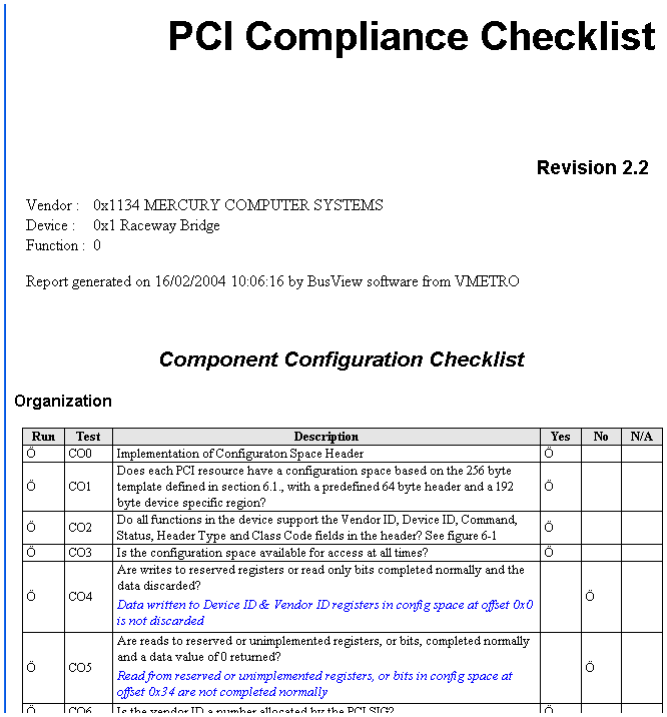
**Show HTML test report automatically**

This option must be selected in order to show the HTML report automatically after a test. If this is not selected then the HTML report must be opened manually.

---

**Note –** All settings are saved in the Registry when a test is Run. There are different Registry settings for PCI and PCI-X

---



# PCI Compliance Checklist

**Revision 2.2**

Vendor : 0x1134 MERCURY COMPUTER SYSTEMS
Device : 0x1 Raceway Bridge
Function : 0

Report generated on 16/02/2004 10:06:16 by BusView software from VMETRO

***Component Configuration Checklist***

Organization

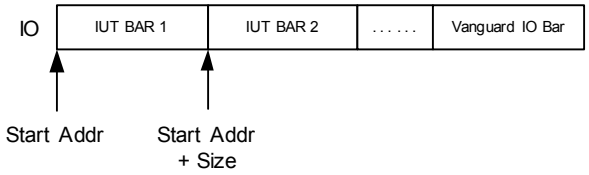
| Run | Test | Description | Yes | No | N/A |
|---|---|---|---|---|---|
| Ö | CO0 | Implementation of Configuraton Space Header | Ö | | |
| Ö | CO1 | Does each PCI resource have a configuration space based on the 256 byte template defined in section 6.1., with a predefined 64 byte header and a 192 byte device specific region? | Ö | | |
| Ö | CO2 | Do all functions in the device support the Vendor ID, Device ID, Command, Status, Header Type and Class Code fields in the header? See figure 6-1 | Ö | | |
| Ö | CO3 | Is the configuration space available for access at all times? | Ö | | |
| Ö | CO4 | Are writes to reserved registers or read only bits completed normally and the data discarded?<br>*Data written to Device ID & Vendor ID registers in config space at offset 0x0 is not discarded* | | Ö | |
| Ö | CO5 | Are reads to reserved or unimplemented registers, or bits, completed normally and a data value of 0 returned?<br>*Read from reserved or unimplemented registers, or bits in config space at offset 0x34 are not completed normally* | | Ö | |
| Ö | CO6 | Is the vendor ID a number allocated by the PCI SIG? | Ö | | |

**FIGURE 2.** *Example Compliance Checker html results page*

**IUT Settings**

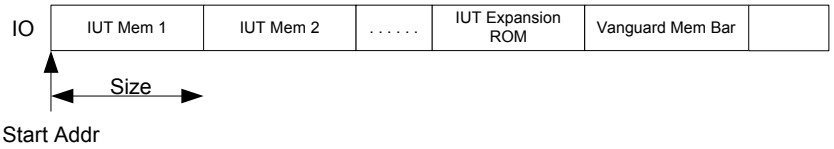
- **IUT is a 64 bit device** - Select this if the IUT can perform 64-bit addressing. PCI Only.
- **IUT is a Master device** - Select this if the IUT can act as a Master device.
- IUT Cycle Capability - Selects which cycles the IUT is capable of doing when acting as Master.

- **Start address IO Address Space**- Defines the start address of the first IUT IO address space.
  If there are several IUT IO BARs then the remaining IO BARs and the Vanguard IO BAR will be added in the following manner:




  The Default IO Start Address is: 0x8000000

- **Start address 32 bit Memory Address Space** - Defines the start address of the IUT 32-bit memory address space. Enter the start address of the first 32-bit IUT BAR. If there are several, then the others will be appended. After the last 32-bit IUT BAR, the Expansion ROM BAR (if defined) is added. Finally, the Vanguard 32-bit memory BAR is added to the end:

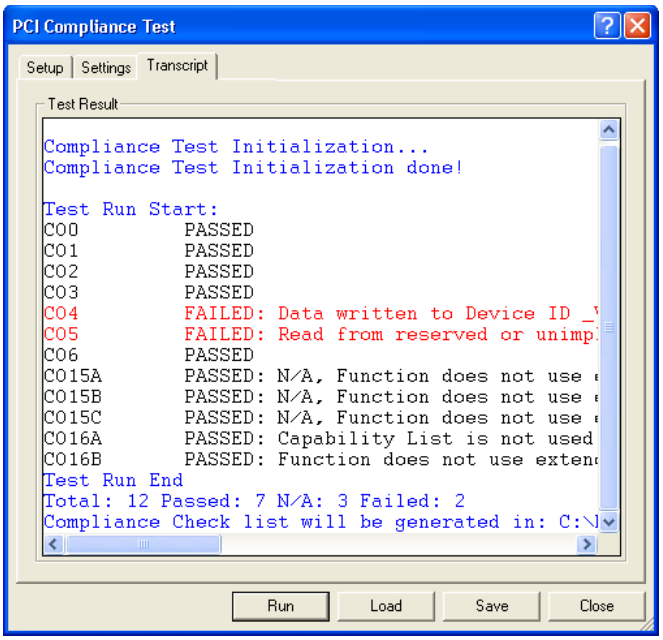



  The Default 32 bit memory Start Address is: 0xA0000000

- **Start address 64 bit Memory Address Space** - Defines the start address of the IUT 64-bit memory address space. Enter the start address of the first 64-bit IUT memory BAR. If there are several, then the others will be appended in the same manner as the 32-bit BARs.

  The Default 64 bit Start Address is: 0x100000000

## Transcript

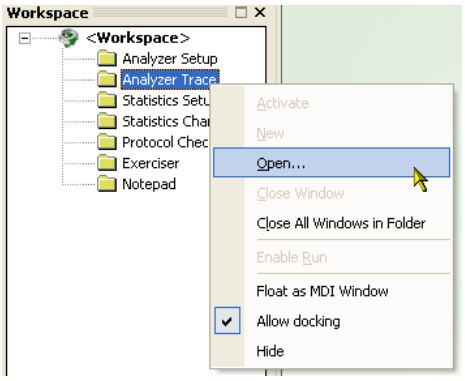



This page gives feedback on the progress of selected tests.

## *10.2 Debugging a failed test*

Each test saves one or more analyzer trace files that can be viewed using BusView. These trace files are saved in the directory location specified in the report tab as described in “Log Settings” on page 365.

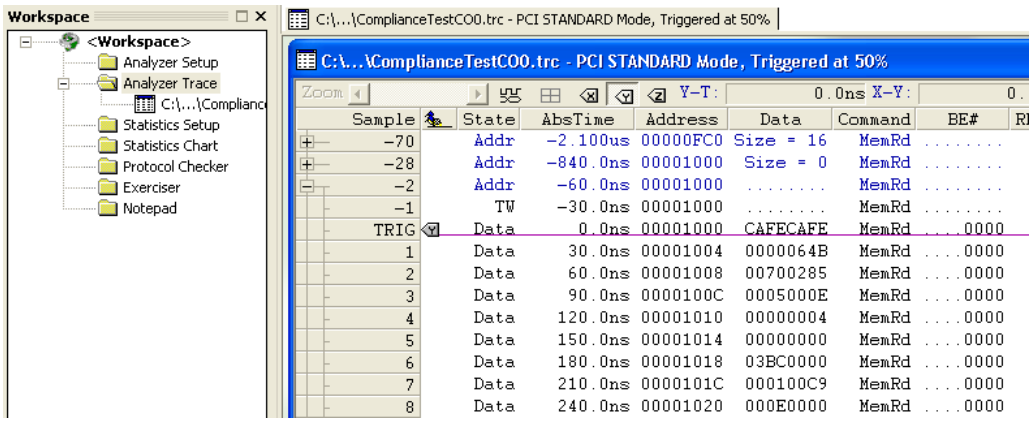
To view a Compliance Checker test result Trace file:




**Loading an Analyzer Trace File**

1. Right click on the Analyzer Trace folder in the Workspace window.
2. Browse to the directory in which the Compliance Checker test traces are located and choose the file you wish to open. The file names correspond to the test name.

Once a file is selected, the trace screen will appear showing the transactions generated during the particular test that was selected.




### Saving and Loading a File

The save and load buttons save a workspace file with the settings from the compliance checker dialog, such as which test you have selected, etc.

## *10.3 Test Requirements*

This section defines the requirements for the Compliance Checker tests and describes how the tests are performed. This will help in understanding the test results and error messages. For some of the more general requirements see previous sections.

Note that CWCDS has added some initial tests such as CO0, DS0 and DC0 to fill in the implementation tables in the Compliance Checklists. These are not in the Compliance Checklist from the PCI SIG.

The Exerciser will perform reads and writes to the device under test and the Analyzer and Protocol Checker will verify the results. The Exerciser must not to be interrupted by a driver or other devices accessing the IUT when the test is running. Some tests employ the Protocol Checker, which can be run as a separate function, but not in addition to the Compliance Checker.

During execution, the BusView log window will occasionally display that a Protocol Error is found. Unless the test displays a failure, this means that the IUT behaved as it should.

This test suite does not specify corner cases and special situation where error conditions might appear. This is because all devices are different in functionality, performance and implementation. There is no way for CWCDS to know what these corner cases are. Rather than suggest something that may be leading the majority of the users down the wrong path, we choose not to do this.

Due to the difference in capabilities of the Vanguard PCI and the PCI-X Exercisers, some tests can only be performed in PCI-X mode. Also, some tests are skipped in PCI-X because they do not apply or can not be tested.

Most tests are only run once. A device which is unstable or has marginal timing may pass a test in one scenario, and fail the same test in a different setting. We recommend running the full test suite in as many different test systems as possible to detect this type of marginal behavior.

The addressing used is controlled from the settings page. These can be changed if there are conflicts with other devices.

### Configuration Tests

If CO6 fails and the Vendor ID is just recently assigned by the PCI SIG, then please contact CWCDS so that we can a
used is from http://www.pcidatabase.com. You can register you PCI products on the database also.

### Master Tests

All Master tests rely on IUT being programmed to perform certain cycles. These tests can not be performed otherwise. Most of the tests do not care about the data value, those where the data value is important are noted in the dialog.

When the dialog box is displayed asking you to perform cycles, it is important that those cycles are done exactly as specified. Tests will only fail if the particular condition tested against exists when that test is being performed. This means that generating the requested cycles inaccurately may cause the test to pass or fail incorrectly.

Where the dialog box requests data parameters for Write cycles, the data values requested must be used in order for the test to be made correctly. Some data values may be presented as XXXXs, this means any values can be used.

Where Read cycles are concerned, the data value presented in the dialog box must be compared to that read in the IUT.

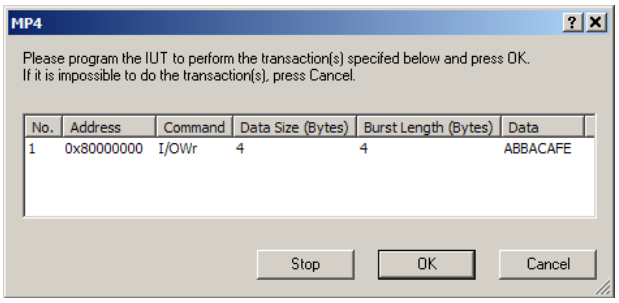



**FIGURE 3.** *Example cycles request*

The Stop button will terminate the test, this has the same functionality as the Run/Stop button in the main Compliance Checker setup dialog shown in Figure 1.

## Target Tests

Some devices request larger memory space through the BAR than they actually use. Some devices issue TARGET ABORT above a certain offset. This will cause problems for some tests that rely on a larger minimum memory space size than the device allows. This can not be adjusted.

If the IUT enables both Memory and I/O space, most tests perform the test using memory space.

## Component Protocol Checklist for a Master Device

All of these test require that the IUT can be programmed to perform certain cycles. Make sure to read the description carefully before executing each of these tests. Some test may fail if the transaction executed differ from what is being asked for. Tests that require master capabilities that have not been checked in the settings page will be skipped and marked passed.

Some tests ask you to verify that the data read back by the IUT is correct. If you cannot do this without stopping the test click OK and rerun the test separately later.

When the Dialog box is displayed asking you to perform cycles, the cycles asked for are the absolute minimum. CWCDS recommends running as much traffic as possible and to exercise the device well using the command specified. The test will only fail if the particular condition when the error appears is found.

## Component Protocol Checklist for a Target Device

Test 2.1.1 and 2.1.2 require that the analyzer and the IUT are in a system where the analyzer can see the same interrupt signals as in the IUT slot. It is OK if they are staggered. If the system does not rout interrupts between the slots, skip the test.

Test 2.9.6 will be set as N/A if the IUT responds with a Split Completion. There is no way for the Compliance Test to force a device to issue immediate data.

## Tests specific to PCI-X

### General PCI-X Protocol Checklist (XGP)

Some of these tests will ask for specific BE# configurations. This is typically hard to initiate for most devices. Read the request thoroughly and skip these tests if this can not be done.

### General PCI-X Initiator Checklist (XIP)

All of these test require the IUT to perform certain cycles

### PCI-X Target Checklist (XTP)

There are no special considerations or instructions for these target tests.

### PCI-X Simple Device Checklist (XSP)

There are no special considerations or instructions for these target tests.

### PCI-X Bus Arbitration Checklist (XAP)

These test require that the IUT GNT# signal is connected to the external input EXT0 on the analyzer. The IUT REQ# signal must be connected to Ext1. PCI Arbitration signals are always point to point.

### PCI-X Configuration Checklist (XCP)

There are no special considerations or instructions for these target tests.

### PCI-X Error Checklist (XEP)

This set of test include parity tests so it is important that the IUT perform the exact actions with data and address as prescribed. There are also tests on Split Completion Errors etc that the user must be aware of and the IUT may require certain recovery mechanisms from the driver etc.

### PCI-X Bus Width Checklist (XWP)

XWP22 must receive the exact data value requested. Otherwise the test will fail.

**PCI-X Split Transaction Checklist (XST)**

There are no special considerations or instructions for these target tests.

**Inter operability and Initialization Checks (XBP)**

No tests implemented

**PCI-X Bridge Checklist (XIN)**

No tests implemented

## *10.4 Troubleshooting Compliance Checker*

- The Compliance Checker can take a few seconds to start due to various housekeeping tasks necessary when initializing the Compliance Checker. If you find the Compliance Checker fails to start, click “Reset All” in the Tools, Hardware, Reset sub menu or perform a full PCI reset.
- If the test hangs while running a compliance test, reset the Vanguard by clicking “Reset All” in the Tools, Hardware, Reset sub menu.
- If an infinite retry condition or similar occurs, then a PCI reset is required.
- Make sure that there are no other tasks running, such as the analyzer sampling or the exerciser is generating transactions.
- The transcript window only displays the result. The BusView status log window displays more details of the progression of the tests. The output of this window can also be logged to a file.

# 11 PCI Zero Slot Adapter (VG-PCI0SL)

This section explains the use and operation of the VG-PCI0SL.

- Introduction
- Operating Modes

## *11.1 Introduction*

The Vanguard PCI Zero Slot Adapter (VG-PCI0SL) includes a PCI to PCI bridge and an edge connector to allow for the addition of another PCI card on top of the Vanguard.

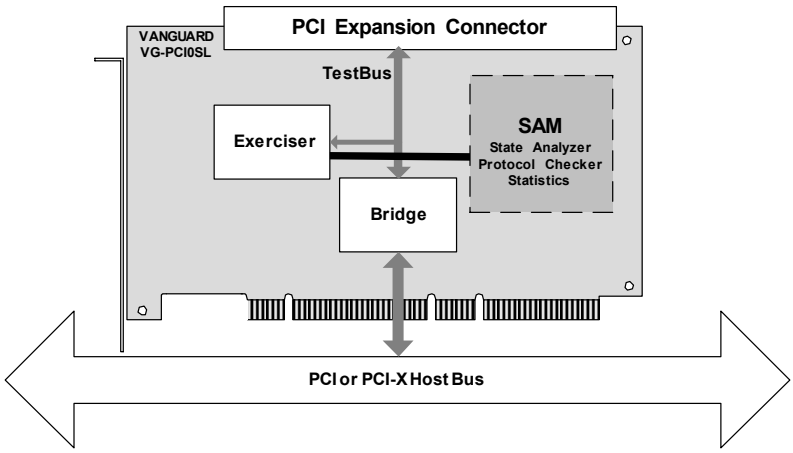



**FIGURE 11-1** *Block diagram of the VG-PCI0SL*

- **Exerciser** - The Exerciser is loaded with different FPGA images depending on which operating mode is required. PCI 0-66MHz and PCI-X 0-133MHz is supported.
- **Bridge** - The bridge transparently connects two electrically separate PCI/PCI-X domains.
- **SAM** - The SAM is a daughter card that contains the Analyzer, Statistics and Protocol Checker functions.
- **Host Bus** - This is the bus on host machine in which the VG-PCI0SL is inserted.
- **Test Bus** - This is the bus in which the card under test is operating. The Bus Mode is controlled by the device under test or the VG-PCI0SL from BusView.
- **PCI Expansion Connector** - The device under test is inserted into the PCI Expansion Connector. This connector is keyed for 3.3V devices only.

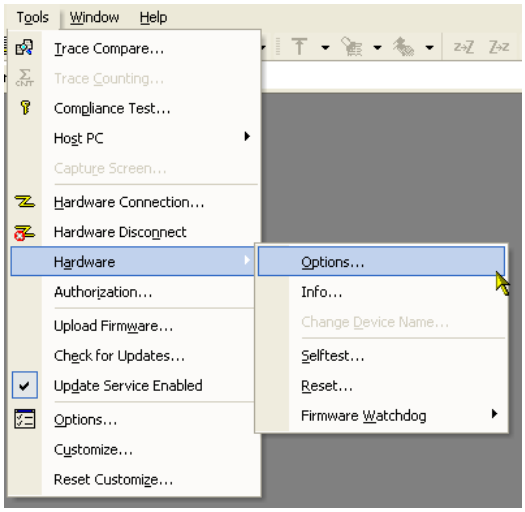
### Test Bus Reset button

The Test Bus Reset button on the front panel will only function if the “Prevent Bridge from being configured.” option is set. This is because the bridge can not accept resets on the secondary side while the primary side remains operational. Therefore, the primary side reset propagates through the bridge in normal operation, but if the bridge is disabled, then the secondary side can be reset independently.

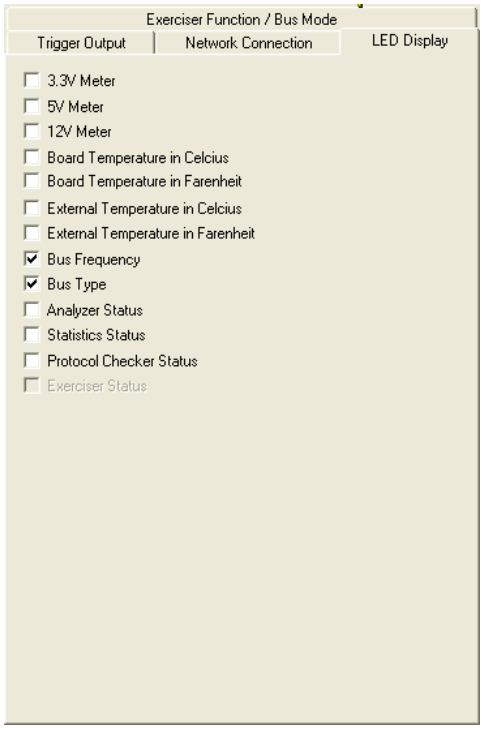
The option to prevent the bridge from being configured is described in “Host Bus” on page 382.

### Hardware Options

The Options sub menu is found under Hardware in the Tools menu. Click Options to open a dialog box in which the operating modes of the VG-PCI0SL can be set.



**Exerciser Functions / Test Bus Mode**




- PCI/PCI-X Exerciser with automatic mode detection - The Vanguard will auto-detect PCI and PCI-X bus modes.
- PCI Only Exerciser (Disable PCI-X capabilities) - Use this option to force the Vanguard to simulate a PCI Only device. The Exerciser will not allow any PCI-X capabilities. This option will ground the PCIXCAP.
  See section "*7.1 part 2 Status Register*" and section "*7.2 PCI-X Capability List Item*" in the "*PCI-X 1.0a Addendum to the PCI Local Bus Specification*" published by the PCISIG.
- 133MHz PCI-X Exerciser (E2) - The Vanguard will set the test bus to PCI-X mode at 133 MHz. The Enhanced Exercise (part number VG-E2) is required to use this mode.
- No Exerciser. - Disable all Exerciser functionality in the Vanguard.

***Test Bus Mode***

- Test Bus Mode - Mode detection. See section "*6.2 Initialization Requirements*" and "*Table 6.1: M66EN PCIXCAP Encoding*" in the "*PCI-X 1.0a Addendum to the PCI Local Bus Specification*" published by the PCISIG. Table 11-1 summarizes the options listed in the dialog box:

**TABLE 11-1. M66EN and PCIXCAP Encoding**

| M66EN | PCIXCAP | Bus Mode |
|---|---|---|
| Ground | Ground | PCI 33 MHz |
| Not Connected | Ground | PCI 66MHz |
| Ground | Pulled Down | PCI-X 66 MHz |
| Not Connected | Not Connected | PCI-X 133 MHz |

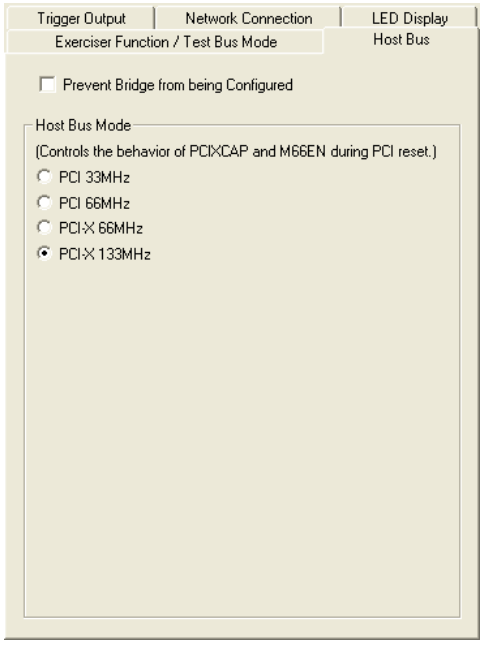
***Test Bus PCI Clock Frequency***

Controls PCI clock frequency of the test bus segment.

- Automatic Bus Negotiation - The Vanguard will use the frequency according to Table 11-1 .
- Manual - Force the PCI clock frequency. The resulting frequency will never be higher than the one specified by PCIXCAP and M66EN (Table 11-1 ).
    - In PCI mode, the Vanguard can generate: 25, 33, 50 or 66MHz frequencies.
    - In PCI-X mode, the Vanguard can generate 25, 33, 50, 60, 100or 133MHz. The frequencies 25 and 33 are lower than that stated in the PCI-X specification and may not be supported by the device under test.

### Host Bus

These settings control the behaviour of the bridge on the host bus segment.




***Prevent Bridge from being configured.***

Checking this box will prevent the Bridge from being seen by the Host system. This is done by disconnecting IDSEL from the bridge.

**Note –** The Test Bus segment will not be configured. Configuration and enumeration will have to be done manually.

***Host Bus Mode***

Determines the Bus Mode of the Host bus in the same manner as the Test Bus Mode selection shown in Table 11-1 .

# *APPENDIXES*

# A Vanguard vs. PBT Series

**TABLE A-1. Vanguard Comparisons with PBT series Analyzers**

| Analyzer Features | Vanguard | PBT Series |
|---|---|---|
| Interface | Ethernet or USB | RS232 or USB |
| Trigger Events | 8 | 4 |
| Trace Buffer | 2M x 256bits<br>64bit HW demultiplex | 32K - 256K x 128 bits<br>32bit HW demultiplex |
| PCI/PCI-X Clock Frequency | 0-133MHz | 12-100MHz |
| Field Upgradeable | Yes | No |
| Software | BusView 4.0 | BusView 3.x |
| Real-time Statistics | 6 pre defined plus user defined statistics and Event Counting | 1-2 |
| Asynchronous Timing | No | Yes with PTIMBAT |
| Graphical Sequencer | Yes | No |
| Compliance Checker | Yes | No |
| Exerciser | E: 0-100MHz<br>E-2: 0-133MHz | 12-100MHz |
| Error Injection | Yes | No |
| Protocol Violations PCI-X | 71 | 71 |
| Protocol Violations PCI | 45 | 45 |
| Temperature and Voltage monitoring. | Yes | No |
| Power Consumption | 6W (idle) | 10.5W (idle) |
| Fan for cooling | No | Yes |

# B Specifications

- Power Consumption
- General
- Analyzer
- Exerciser
- Protocol Checker

## Power Consumption

**TABLE B-1. Vanguard PCI (Carrier + SAM) 3.3 V**

| PCI CLK (MHZ) | All Running A | Note |
|---|---|---|
| **33** | **2.93** | **PCI / PCI-X** |
| **50** | **3.27** | **PCI / PCI-X** |
| **66** | **3.6** | **PCI / PCI-X** |
| **75** | **3.77** | **PCI-X** |
| **100** | **4.3** | **PCI-X** |
| **133.33** | **5.0** | **PCI-X** |

## General

| | |
|---|---|
| PCI-X/PCI bus: | 32/64-bit, up to 133MHz |
| Interfaces: | USB port, 12 Mb/s.<br>Ethernet port 10/100 Mb/s |
| Power supply requirements: | +3.3VDC +/-5% from PCI backplane or from ext. power supply via front panel inlet.<br>5A Max |
| Dimensions: | 174.6 x 106.7mm (Short card) |
| | One PCI slot. |
| Compliant to: | PCI Rev. 2.3<br>PCI-X rev. 1.0a |
| Measurements: | Temperature: 0-120 °C/32-248°F<br>Voltage: 3.3V, 5V, 12V |

## Analyzer

| | |
|---|---|
| Trace Memory: | 2M x 256 bits (64 MBytes total) |
| Input channels: | 93 bus signals, plus 16 ext. inputs on pin headers.<br>One trigger output in front panel |
| PCI-X/PCI clock requirements: | Max. 133MHz, min. 1 kHz |
| Signal levels: | Base module 3.3V |
| Monitored signals: | AD[31::0], AD[63::32], C/BE[3::0]#, C/BE[7::4]#, FRAME#, TRDY#, IRDY#, STOP#, DEVSEL#, PAR, PAR64, PERR#, SERR#, RST#, INTA:D#, LOCK#, ACK64#, REQ64#, REQ#, GNT#, IDSEL.<br>cPCI: INTS#, INTP#, ENUM#<br>(Plus GNT3:0#, REQ3:0#, IDSEL via pin headers). |
| Trigger: | 8 word recognizers covering all 92 (95) PCI signals and 16 Ext. inputs.<br>True Range & NOT operator on Address/ Data. Edge Triggering. |
| Range: | 8 A64 address ranges, 8 D64 data ranges. Inside/Outside. |
| Sequencer: | 16 levels with If, Else, Elsif, Goto, Count, Delay, Trigger, Store, Halt. |
| Trigger position: | 0-100%, 1% resolution |
| Occurrence/ delay counters: | 3 x 32-bits |
| Event counters: | 8 x 30-bits for Statistics |
| Real-Time Statistics Counters: | 47 x 30-bits counters |
| Decode Speed Counter: | 1 dedicated counter |
| Time Tag: | Range:<br>30ns-4688min@33MHz,<br>15ns-2344min30sec@66MHz.<br>10ns-1562 min@100MHz<br>7.5ns-1172min@133MHz |
| Latency Tag: | Counts latency (wait states) from FRAME# to TRDY# asserted. Max count: 64 clocks. |
| Trigger Output: | LVTTL level trigger output with programmable polarity, level or pulse. May pulse on each stored sample. Available on pin header in front panel. |
| External Inputs: | 15 TTL level inputs on pin header on back panel.1 TTL level input in front panel |

## Exerciser

| | |
|---|---|
| Master: | Zero-wait-states, 1 GB/s @133MHz peak burst rate, 64-bits Address, 2 DMA Controllers. |
| Target: | 8MBytes SDRAM memory, 1 GB/s peak burst rate @133MHz. 32/64-bit address. 256 bits I/O space memory. |

## Protocol Checker

| | |
|---|---|
| PCI-X Violations: | 71 Protocol Violations |
| PCI Violations: | 45 Protocol Violations |

# C Signals

This section is a reference to all the signals used by the Vanguard. All the signals, except the JTAG signals, are monitored by the analyzer.

The following sections are covered here:

- Pin Function Groups of PCI and PCI-X signals
- PCI Signals
    - PCI Transfer operation
    - System Pins
    - Address and Data Pins
    - Interface Control Pins
    - Arbitration Pins
    - Error Reporting Pins
    - Interrupt Pins
    - 64-bit Bus Extension Pins
    - Vanguard Signals
- PCI-X Signals
    - PCI-X Transfer operation
    - System Pins
    - Address and Data Pins
    - Interface Control Pins
    - Arbitration Pins
    - Error Reporting Pins
    - Interrupt Pins
    - 64-bits Bus Extension Pins
    - Vanguard Signal Groups

# C-1 Pin Function Groups of PCI and PCI-X signals

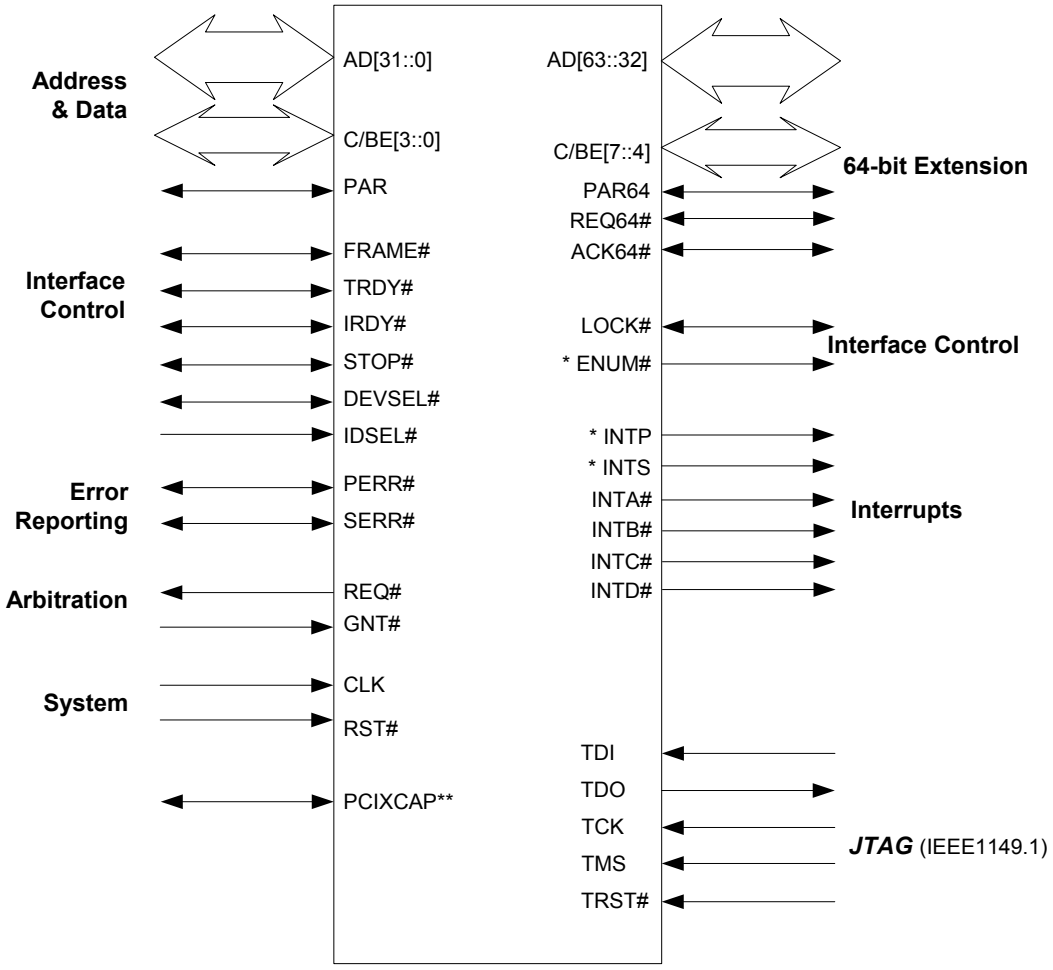



**FIGURE C-1** *PCI and PCI-X Signals*

PCI bus pin list, with required signals to the left, and optional signals to the right.

* CompactPCI signals only.
** PCI-X signal only.

Most PCI data transfers are accomplished using Burst Transfers. A burst transfer consists of one single address phase followed by two or more data phases. The start address and type of transaction are transferred during the address phase, and then the target has to increment the address for each data phase.

## PCI Transfer operation

**Address Phase:** The transfer Initiator applies an address to the Address/Data bus. The Target device recognizes this as an address within its range. At the same time the type of transaction is applied to the Command/Byte Enable (C/BE [3::0]#) bus, and FRAME# is asserted.

The FRAME# signal indicates that there is a valid address and transaction type on the bus, and is asserted from the moment the Initiator starts the address phase, until the Initiator is ready to complete the last data phase.

The Address Phase is one PCI clock cycle, except for 64-bits addresses which use DAC (Dual Address Command). These take two clock cycles.

When a PCI target has determined that it is the target of the transaction, it asserts DEVSEL#.

**Data Phase(s):** There can be one or more data phases in a transfer. Each data phase one PCI clock cycle. The number of bytes transferred in each data phase is determined in the C/BE [3:0]# field. If bit 0 is asserted, then the least significant byte is transferred, etc.

**Completing the Transaction:** The Initiator de-asserts FRAME# and keeps IRDY# asserted after the n-1 data phase, and de-asserts IRDY# after n data phases. The next bus master that has been granted ownership of the bus, must detect when both FRAME# and IRDY# are deasserted before it can start its transaction.

## System Pins

**CLK.** Clock provides timing for all transactions and is input to every device. All signals in PCI are asserted relative to CLK except INT#.

**RST#.** The Reset signal forces all PCI configuration registers, state machines, and output drivers to an initialized state. RST# may be asserted both synchronously and asynchronously to the PCI CLK edge.

**M66EN.** This signal is a static signal that is grounded on 33MHz devices and pulled high on 66 MHz devices.

For information on CompactPCI bus mode and clock frequencies, See “Bus Mode” on page 27 and “PCI Clock Frequency” on page 27.

## Address and Data Pins

**AD[31::0].** The PCI bus uses a multiplexed architecture, meaning that the address and data busses are multiplexed on the same PCI pins. A transaction consists of one address phase followed by two or more data phases.

**C/BE[3::0].** The Bus Command and the Byte Enables, are multiplexed on the same pins. During the address phase the type of transaction is defined with a Bus Command, and during the data phase the number of bytes transferred are set with the Byte Enables. Bit 0 enables the least significant byte, and bit 3 enables the most significant byte.

**PAR.** PCI uses even parity across AD[31::0] and C/BE[3::0] for error detection. PAR is valid one clock cycle after the address phase. For data phases the PAR is valid one clock after IRDY# or TRDY# is asserted, for write and read cycles respectively.

**TABLE C-1. PCI Bus Commands**

| Predefined Symbol | C3# | C2# | C1# | C0# |
|---|---|---|---|---|
| **Mem** | X | 1 | x | x |
| **Conf** | 1 | 0 | 1 | x |
| **I/O** | 0 | 0 | 1 | x |
| **IntAck** | 0 | 0 | 0 | 0 |
| **Special** | 0 | 0 | 0 | 1 |
| **I/ORd** | 0 | 0 | 1 | 0 |
| **I/OWr** | 0 | 0 | 1 | 1 |
| **Res1** | 0 | 1 | 0 | 0 |
| **Res2** | 0 | 1 | 0 | 1 |
| **MemRd** | 0 | 1 | 1 | 0 |
| **MemWr** | 0 | 1 | 1 | 1 |
| **Res3** | 1 | 0 | 0 | 0 |
| **Res4** | 1 | 0 | 0 | 1 |
| **ConfRd** | 1 | 0 | 1 | 0 |
| **ConfWr** | 1 | 0 | 1 | 1 |
| **MRdMul** | 1 | 1 | 0 | 0 |
| **MRdLn** | 1 | 1 | 1 | 0 |
| **MWrInv** | 1 | 1 | 1 | 1 |

## Interface Control Pins

**FRAME#.** FRAME# is asserted by the current master indicating that a valid address and command are available.

**IRDY#.** Initiator Ready, is driven by the bus master, and indicates that the bus master is ready to complete the current data phase.

**TRDY#.** Target Ready, is driven by the target, and indicates that the target is ready to complete the current data phase.

Data is only transferred when TRDY# and IRDY# are asserted at the same time.

**STOP#.** STOP# is asserted by the target if it wants the master to abort the transaction.

**LOCK#.** LOCK# is used by a master to lock the currently addressed memory target during an atomic transaction series.

**IDSEL.** Initialization Device Select is used as chip select during configuration transactions.

**DEVSEL#.** This is used by the Target to "claim" the transaction. If a transaction results in no DEVSEL# assertion within 4 clocks, it is considered a Master Abort. The Initiator terminates the transaction and continues.

**ENUM#.** ENUM# is a CompactPCI signal only, and shall be driven by hot swap compatible boards after insertion, and prior to removal. The system master uses this interrupt signal to force software to interrogate all boards within the system for resource allocation regarding I/O, memory, and interrupt usage.

## Arbitration Pins

**REQ#.** Request is a message to the arbiter that the requesting agent wants access to the bus.

**GNT#.** Grant is a message back to the requesting agent that access is granted.

## Error Reporting Pins

**PERR#.** Parity Error reports parity errors in all transactions except Special Cycle.

**SERR#.** System Error reports parity errors in a Special Cycle, and all other critical errors.

## Interrupt Pins

**INT(A-D)# .** Interrupts are requested on these lines. Interrupts are level sensitive. PCI defines one interrupt line (INTA#) for a single function device, and up to four lines for a multifunction device. In the event patterns and in the trace display the left most digit is INTD# and the right most is INTA#.

**INTP and INTS.** The INTP and INTS signals are CompactPCI signals only, and are defined for IDE boards. Both primary and secondary ISA interrupts, designated INTP and INTS respectively, are available and may be used by the masters.

## 64-bit Bus Extension Pins

**AD[63::32].** Extends the address/data bus to 64 bits.

**C/BE[7::4]#.** Additional Bus Commands and Byte Enables.

**REQ64#.** Request 64-bits transfer is generated by the master indicating the desire to use 64-bits transfer.

**ACK64#.** Acknowledge 64-bits transfer is generated by the target signaling that 64-bits transfer is OK.

**PAR64#.** Parity of the upper double word. Parity is calculated for the AD[63::32] and the C/BE[7::4]# busses too.

## Vanguard Signals

The selection of PCI signals presented in each sampling mode (default configuration), has been carefully selected to make it convenient to form triggers and storage filters. In addition, there are several signals and signal groups that need further discussion.

**MAbort**

Active when a transaction has been terminated with a Master Abort.

**DAC**

Active in the address phase, when Dual Address Cycle (DAC) is used.

**Start**

Active in the first address phase.

**Retry**

Active when the target signals “Retry”.

Note – Inactive when the target signals disconnect without data.

**Status - in Transfer mode only**

The Status signal is a combination of five signals, Retry, MAbort, DEVSEL#, STOP#, and TRDY#. This is listed only in Transfer mode.

**TABLE C-2. PCI Status Field**

| Predef. Symbol | Retry | MAbort | DEVSEL# | STOP# | TRDY# |
|---|---|---|---|---|---|
| **TAbort** | 0 | 0 | 1 | 0 | 1 |
| **MAbort** | 0 | 1 | 1 | 1 | 1 |
| **Retry** | 1 | 0 | 0 | 0 | 1 |
| **Tdwod** | 0 | 0 | 0 | 0 | 1 |
| **Tdwd** | 0 | 0 | 0 | 0 | 0 |
| **Data** | 0 | 0 | x | 1 | 0 |

**State - Standard mode and Clock mode.**

The State field indicates the state of the bus, i.e. if it is an address/data phase, whether the master (IW), the target (TW), or both (W), have inserted wait cycles, etc. The State field is available in Standard and Clock mode.

(Tdwd = Target disconnect with data.)

(Tdwod = Target disconnect without data.)

**TABLE C-3. PCI State Field**

| State | Start | A64 | FRAME# | IRDY# | Retry | MAbort | DEVSEL# | STOP# | TRDY# |
|---|---|---|---|---|---|---|---|---|---|
| **Addr** | 1 | x | 0 | 1 | x | x | 1 | 1 | 1 |
| **AddrL** | 1 | 1 | 0 | 1 | x | x | 1 | 1 | 1 |
| **AddrH** | 0 | 1 | 0 | 1 | x | x | 1 | 1 | 1 |
| **W** | 0 | 0 | 0 | 1 | 0 | 0 | x | x | 1 |
| **IW** | x | x | x | 1 | 0 | 0 | x | x | 0 |
| **TW** | x | x | x | 0 | 0 | 0 | x | 1 | 1 |
| **Data** | x | x | x | 0 | 0 | 0 | x | 1 | 0 |
| **Tdwd** | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 |
| **Tdwod** | x | x | x | 0 | 0 | 0 | 0 | 0 | 1 |
| **Retry** | x | x | x | 0 | 1 | 0 | 0 | 0 | 1 |
| **MAbort** | x | x | x | x | x | 1 | x | x | x |
| **TAbort** | x | x | x | 0 | 0 | 0 | 1 | 0 | 1 |
| **---** | x | x | 1 | 1 | x | 0 | x | x | x |

Note – When the C/BE# field is signaling a 64-bit address by asserting the Dual Address Command, the State field will be decoded as HiAddr during the first address cycle, and LowAddr during the second address cycle.

**Burst - Clock mode and Standard mode**

The Burst bit is activated when both FRAME# and IRDY# go active, and is deactivated when FRAME# and IRDY# both go inactive. As the name implies, the Burst bit indicates a burst cycle on the bus. The Burst bit can be inserted in the Event Patterns window and used as a trigger.

**Transfer - Transfer mode**

The Transfer field is a combination of the Burst bit and the Start bit, as shown in Table C-4 . The Start bit indicates the start of a transaction, i.e. it is active in the address phase. The Start bit is only visible as a part of the Transfer field, i.e. it cannot alone be used as a trigger condition, and it is not visible in the trace. The Transfer field can be inserted in the Event Patterns window and used as a trigger in Transfer mode. The Transfer field is very useful to identify burst cycles and the start of burst cycles, both as triggers and store qualifiers, and to visualize bursts in the trace.

**TABLE C-4. PCI Transfer Field**

| Burst | Start | Burst |
|---|---|---|
| - | x | 0 |
| Start | 1 | 1 |
| Burst | 0 | 1 |

**Wait**

The Wait field indicates the number of wait states from the address phase to the first data phase, and between the data phases in a burst transaction, i.e. the latency in the transaction.

Without decoding (default): Number of wait states.
With decoding: Time, in number of wait cycles multiplied with the PCI clock period.

Note – When a Store qualifier is used, the Wait field is not valid in the trace, and will be displayed as a ".".

**IT Latency - Initial Target Latency.**

This is the latency measured between the initiator asserting FRAME# and the target initiating something other than a Wait State.

**PCHKTrg**

Protocol Checker cross trigger. If the Protocol Checker is enabled, then this signal will be active when the Protocol Checker has detected a violation. Note that this is an active high signal as opposed to previous generation tools.

**TABLE C-5. PCHKTrg Field**

| Predef. Symbol | Start |
|---|---|
| Trig | 1 |
| - | 0 |

### EXERtrg

Exerciser cross trigger. The Exerciser activates this signal if the Compare and Test functions fail.

**TABLE C-6. EXERtrg Field**

| Predef. Symbol | Start |
|---|---|
| **Trig** | 1 |
| **-** | 0 |

### INTx#

Combines the interrupt signals INTA# to INTD# into one group.

**TABLE C-7. INTx# Field**

| Predef. Symbol | Start |
|---|---|
| **---A** | 1110 |
| **--B-** | 1101 |
| **-C---** | 1011 |
| **D---** | 0111 |
| **--BA** | 1100 |
| **-C-A** | 1010 |
| **D--A** | 0110 |
| **--CB-** | 1001 |
| **D-B-** | 0101 |
| **DC--** | 0011 |
| **-CBA** | 1000 |
| **D-BA** | 0100 |
| **DC-A** | 0010 |
| **DCB-** | 0001 |
| **DCBA** | 0000 |
| **----** | 1111 |

**Decode**

**TABLE C-8. PCI Decode field**

| Predef. Symbol | Bit 2 | Bit 1 | Bit 0 | Speed in Clocks |
|---|---|---|---|---|
| **FAST** | 0 | 0 | 1 | **1** |
| **MED** | 0 | 1 | 0 | **2** |
| **SLOW** | 0 | 1 | 1 | **3** |
| **SUB** | 1 | 0 | 0 | **4** |
| **ILL0** | 0 | 0 | 0 | **0** |
| **ILL5** | 1 | 0 | 1 | **5** |
| **ILL6** | 1 | 1 | 0 | **6** |
| **ILL7** | 1 | 1 | 1 | **7** |

## PCI-X Transfer operation

**Address Phase:** The transfer Initiator applies an address to the Address/Data bus. The Target device recognizes this as an address within its range. At the same time the type of transaction is applied to the Command/Byte Enable (C/BE [3::0]#) bus, and FRAME# is asserted.

The FRAME# signal indicates that there is a valid address and transaction type on the bus, and is asserted from the moment the Initiator starts the address phase, until the Initiator is ready to complete the last transaction.

The Address Phase is one PCI-X clock cycle, except for 64-bits addresses which use DAC (Dual Address Command). These take two clock cycles.

**Attribute Phase:** The transfer Initiator applies an attribute to the Address/ Data and Command/ Byte Enable bus. Attributes are additional information included with each transaction that further defines the transaction. The attribute phase is always a single clock regardless of the width of the data transfer or the width of the address (single or dual address cycle). The upper buses (AD[63::32] and C/BE[7::4]#) of 64-bit devices are reserved and driven high during the attribute phase. There are three different attribute formats for requesters and one format for completers. The Requester Attribute formats are burst and DWORD transactions and Type 0 configuration transactions. The Completer attribute is Split Completion. See Figure C-2 on page 413 for example.

When a PCI-X target has determined that it is the target of the transaction, it asserts DEVSEL#.

**Data Phase(s):** There can be one or more data phases in a transfer. Each takes one PCI-X clock cycle. The number of bytes transferred in each data phase is determined in the C/BE [3:0]# field. If bit 0 is asserted, the least significant byte is transferred, etc.

**Completing the Transaction:** The Initiator de-asserts FRAME# and keeps IRDY# asserted when the last data phase is about to be transmitted, and de-asserts IRDY# again on completion. The next bus master that has been granted ownership of the bus, has to detect when both FRAME# and IRDY# are deasserted before it can start its transaction.

In PCI-X mode, disconnects are only allowed at 128 byte naturally aligned boundaries (ADB), De-asserting FRAME# indicates that the master intends to disconnect at the next ADB.

**TABLE C-9. PCI-X Initialization Pattern.**

| DEVSEL# | STOP# | TRDY# | Mode | Max.Clock Period (ns) | Min.Clock Period (ns) | Min.Clock Freq (MHz) (ref) | Max.Clock Freq (MHz) (ref) |
|---|---|---|---|---|---|---|---|
| Deasserted | Deasserted | Deasserted | | | 30 | 0 | 33 |
| | | | | 30 | 15 | 33 | 66 |
| Deasserted | Deasserted | Asserted | PCI-X | 20 | 15 | 50 | 66 |
| Deasserted | Asserted | Deasserted | PCI-X | 15 | 10 | 66 | 100 |
| Deasserted | Asserted | Asserted | PCI-X | 10 | 7,5 | 100 | 133 |
| Asserted | Deasserted | Deasserted | PCI-X | Reserved | Reserved | Reserved | Reserved |
| Asserted | Deasserted | Asserted | | Reserved | Reserved | Reserved | Reserved |
| Asserted | Asserted | Deasserted | | Reserved | Reserved | Reserved | Reserved |
| Asserted | Asserted | Asserted | | Reserved | Reserved | Reserved | Reserved |

## System Pins

**CLK.** Clock provides timing for all transactions on PCI-X, and is an input to every device. All signals, except the interrupt signals, are sampled on the rising edge of CLK.

**RST#.** The Reset signal forces all PCI-X configuration registers, state machines, and output drivers to an initialized state. RST# may be asserted both synchronously and asynchronously to the PCI-X CLK edge.

**M66EN.** The 66MHZ_ENABLE pin indicates to a device whether the bus segment is operating at 66 or 33 MHz.

**PCIXCAP.** Add-in cards indicate that they are PCI-X capable and at what speed using this pin. (See Table C-10 )

**CompactPCI Only –** For information on CompactPCI bus mode and clock frequencies, See “Bus Mode” on page 27 and “PCI Clock Frequency” on page 27.

Table C-10 shows how the combinations of M66EN and PCIXCAP decide the speed and capabilities of the Add-in card.

**TABLE C-10. M66EN and PCIXCAP Encoding for PCI and PMC versions**

| M66EN | PCIXCAP | Conventional Device Frequency Capability | PCI-X Device Frequency Capability |
|---|---|---|---|
| Ground | Ground | 33 MHz | Not capable |
| Not connected | Ground | 66 MHz | Not capable |
| Ground | Pull-down | 33 MHz | PCI-X 66 MHz |
| Not connected | Pull-down | 66 MHz | PCI-X 66 MHz |
| Ground | Not connected | 33 MHz | PCI-X 133 MHz |
| Not connected | Not connected | 66 MHz | PCI-X 133 MHz |

**PME#.** The Power Management Event signal is an optional signal that can be used by a device to request a change in the device or system power state. The assertion and de-assertion of PME# is asynchronous to CLK. The use of this pin is specified in the PCI-X Bus Power Management Interface Specification.

## Address and Data Pins

**AD[31::0].** The PCI-X bus uses a multiplexed architecture, meaning that the address- and data-busses are multiplexed on the same pins. A transaction consists of one address phase and one attribute phase followed by two or more data phases.

**C/BE[3::0].** The Bus Command and the Byte Enables, are multiplexed on the same pins. During the address phase the type of transaction is defined with a Bus Command, and during the data phase the number of bytes transferred are set with the Byte Enables. Bit 0 enables the least significant byte, and bit 3 enables the most significant byte.

Byte Enables are only valid for DWORD transactions in PCI-X. For burst transactions they are invalid, and should be driven high.

During the Attribute phase these pins, in combination with the AD lines carry the transaction Attribute.

**PAR.** PCI-X uses even parity across AD[31::0] and C/BE[3::0] for error detection. PAR is valid one clock cycle after the address phase. For data phases the PAR is valid one clock after IRDY# or TRDY# is asserted, for write and read cycles respectively.

**TABLE C-11. PCI-X Bus Commands**

| Predefined Symbol | C3# | C2# | C1# | C0# |
|---|---|---|---|---|
| **Mem** | x | 1 | x | x |
| **Conf** | 1 | 0 | 1 | x |
| **I/O** | 0 | 0 | 1 | x |
| **IntAck** | 0 | 0 | 0 | 0 |
| **Special** | 0 | 0 | 0 | 1 |
| **I/ORd** | 0 | 0 | 1 | 0 |
| **I/OWr** | 0 | 0 | 1 | 1 |
| **Res1** | 0 | 1 | 0 | 0 |
| **Res2** | 0 | 1 | 0 | 1 |
| **MRdDW** | 0 | 1 | 1 | 0 |
| **MWr** | 0 | 1 | 1 | 1 |
| **AMRdBlk** | 1 | 0 | 0 | 0 |
| **AMWrBlk** | 1 | 0 | 0 | 1 |
| **ConfRd** | 1 | 0 | 1 | 0 |
| **ConfWr** | 1 | 0 | 1 | 1 |
| **SpComp** | 1 | 1 | 0 | 0 |
| **MRdBlk** | 1 | 1 | 1 | 0 |
| **MWrBlk** | 1 | 1 | 1 | 1 |

## Interface Control Pins

**FRAME#.** FRAME# is asserted by the current master indicating that a valid address and command are available.

**IRDY#.** Initiator Ready, is driven by the bus master, and indicates that the bus master is ready to complete the current data phase.

**TRDY#.** Target Ready, is driven by the target, and indicates that the target is ready to complete the current data phase.

**STOP#.** STOP# is asserted by the target if it wants the master to terminate the transaction.

**LOCK#.** LOCK# is used by a master to lock the currently addressed memory target during an atomic transaction series.

**IDSEL.** Initialization Device Select is used as chip select during configuration transactions.

**DEVSEL#.** DEVSEL# is asserted by a target when it has decoded the current address as its own address.

**ENUM#.** ENUM# is a CompactPCI-X signal only, and shall be driven by hot swap compatible boards after insertion, and prior to removal. The system master uses this interrupt signal to force software to interrogate all boards within the system for resource allocation regarding I/O, memory, and interrupt usage.

During initialization the DEVSEL#, STOP# and TRDY# signals are used to specify the frequency range as described in Table C-9 on page 404.

## Arbitration Pins

**REQ#.** Request is a message to the arbiter that the requesting agent wants access to the bus.

**GNT#.** Grant is a message back to the requesting agent that access is granted.

## Error Reporting Pins

**PERR#.** Parity Error reports parity errors in all transactions except Special Cycle.

**SERR#.** System Error reports parity errors in a Special Cycle, and all other critical errors.

## Interrupt Pins

**INT(A-D)# .** Interrupts are requested on these lines. Interrupts are level sensitive. PCI-X defines one interrupt line (INTA#) for a single function device, and up to four lines for a multifunction device. In the event patterns and in the trace display the left most digit is INTD# and the right most is INTA#.

## 64-bits Bus Extension Pins

**AD[63::32].** Extends the address/data bus to 64 bits.

**C/BE[7::4]#.** Additional Byte Enables.

**REQ64#.** Request 64-bits transfer is generated by the master indicating the desire to use 64-bits transfer.

**ACK64#.** Acknowledge 64-bits transfer is generated by the target signaling that 64-bits transfer is OK.

**PAR64#.** Parity of the upper double word. Parity is calculated for the AD[63::32] and the C/BE[7::4]# busses too.

## Vanguard Signal Groups

The selection of PCI-X signals presented in each sampling mode (default configuration), has been carefully selected to make it convenient to form triggers and storage filters. In addition, there are several signals and signal groups that need further discussion.

**Attrib**

Active in the Attribute phase.

**MAbort**

Active when a transaction has been terminated with a Master Abort.

**DAC**

Active in the address phase, when Dual Address Cycle (DAC) is used.

**Start**

Active in the first address phase.

**Status - Transfer mode.**

The Status signal is a combination of five signals, MAbort (Internal generated sampling channel), DEVSEL#, STOP# and TRDY#. Status is only available in Transfer mode.

**TABLE C-12. PCI-X Status Field**

| Predef. Symbol | MAbort | DEVSEL# | STOP# | TRDY# |
|---|---|---|---|---|
| - - - | 0 | x | x | x |
| **MAbort** | 1 | 1 | 1 | 1 |
| **SpResp** | 0 | 1 | 1 | 0 |
| **TAbort** | 0 | 1 | 0 | 1 |
| **SDtaPhD** | 0 | 1 | 0 | 0 |
| **Data** | 0 | 0 | 1 | 0 |
| **Retry** | 0 | 0 | 0 | 1 |
| **DNxtADB** | 0 | 0 | 0 | 0 |

MAbort Master Abort
SpResp Split response
TAbort Target Abort
SDtaPhD Single Data Phase Disconnect
DNxtADB Disconnect on next address disconnect boundary

**State**

The State field indicates the state of the bus. The State field is only available in Clock mode)

**TABLE C-13. PCI-X State Field**

| State | Start | A64 | FRAME# | IRDY# | Attrib | MAbort | DEVSEL# | STOP# | TRDY# |
|---|---|---|---|---|---|---|---|---|---|
| **Addr** | 1 | x | 0 | 1 | x | x | 1 | 1 | 1 |
| **AddrL** | 1 | 1 | 0 | 1 | x | x | 1 | 1 | 1 |
| **AddrH** | 0 | 1 | 0 | 1 | x | x | 1 | 1 | 1 |
| **Attrib** | x | x | 0 | x | 1 | x | x | x | x |
| **TResp** | 0 | 0 | 0 | 1 | 0 | x | x | 1 | 1 |
| **DevWait** | x | x | 0 | 0 | x | x | 1 | 1 | 1 |
| **Wait** | x | x | x | 0 | x | x | 0 | 1 | 1 |
| **SpResp** | x | x | x | 0 | x | x | 1 | 1 | 0 |
| **Data** | x | x | x | 0 | x | x | 0 | 1 | 0 |
| **SDtaPhD** | x | x | x | 0 | x | x | 1 | 0 | 0 |
| **DNxtADB** | x | x | x | 0 | x | x | 1 | 0 | 0 |
| **Retry** | x | x | x | 0 | x | x | 0 | 0 | 1 |
| **TAbort** | x | x | x | 0 | x | x | 1 | 0 | 1 |
| **MAbort** | x | x | x | x | x | 1 | x | x | x |
| **---** | x | x | 1 | 1 | x | 0 | x | x | x |

| | |
|---|---|
| TResp | Target response |
| SpResp | Split response |
| TAbort | Target Abort |
| SDtaPhD | Single Data Phase Disconnect |
| DNxtADB | Disconnect on next address disconnect boundary |
| MAbort | Master Abort |

Note – When the C/BE# field is signaling a 64-bit transfer by asserting the Dual Address Command, the State field will be decoded as AddrL during the first address cycle, and AddrH during the second address cycle.

**Burst - Clock mode and Standard mode**

The Burst bit is activated when both FRAME# and IRDY# go active, and is deactivated when FRAME# and IRDY# both go inactive. As the name implies, the Burst bit indicates a burst cycle on the bus. The Burst bit can be inserted in the Event Patterns window and used as a trigger in Clock mode.

### Transfer - Transfer mode

The Transfer field is a combination of the Burst bit and the Start bit, as shown below. The Start bit indicates the start of a transaction, i.e. it is active in the address phase. The Start bit is only visible as a part of the Transfer field, i.e. it cannot alone be used as a trigger condition, and it is not visible in the trace. The Transfer field can be inserted in the Event Patterns window and used as a trigger in Transfer mode. The Transfer field is very useful to identify burst cycles and the start of burst cycles, both as triggers and store qualifiers, and to visualize bursts in the trace.

**TABLE C-14. PCI-X Transfer Field**

| Predef. Symbol | Start | BURST |
|---|---|---|
| - | x | 0 |
| Start | 1 | 1 |
| Burst | 0 | 1 |

### Wait

The Wait field indicates the number of wait states from the address phase to the first data phase, and between the data phases in a burst transaction, i.e. the latency in the transaction.

Without decoding (default): Number of wait states.
With decoding: Time, in number of wait cycles multiplied with the PCI-X clock period.

Note – When a Store qualifier is used, the Wait field is not valid in the trace, and will be displayed as a ".".

### Decode

The Decode Speed is only available in the trace display. The Decode Speed signifies the number of clock cycles the data appears on the bus after the address phase(s).

**TABLE C-15. PCI-X Decode field**

| Predef. Symbol | Bit 2 | Bit 1 | Bit 0 | Speed in Clocks | Comment |
|---|---|---|---|---|---|
| **A** | 0 | 1 | 0 | 2 | Decode A |
| **B** | 0 | 1 | 1 | 3 | Decode B |
| **C** | 1 | 0 | 0 | 4 | Decode C |
| **SUB** | 1 | 1 | 0 | 6 | Subtractive |
| **ILL0** | 0 | 0 | 0 | 0 | Illegal Decode Speed |
| **ILL1** | 0 | 0 | 1 | 1 | Illegal Decode Speed |
| **ILL5** | 1 | 0 | 1 | 5 | Illegal Decode Speed |
| **ILL7** | 1 | 1 | 1 | 7 | Illegal Decode Speed |

### IT Latency - Initial Target Latency.

This is the latency measured between the initiator asserting FRAME# and the target initiating something other than a Wait State.

### PCHKTrg

Protocol Checker cross trigger. If the Protocol Checker is enabled, then this signal will be active when the Protocol Checker has detected a violation.

**TABLE C-16. PCHKTrg Field**

| Predef. Symbol | Start |
|---|---|
| **Trig** | 1 |
| **-** | 0 |

### EXERtrg

Exerciser cross trigger. This signal can be activated by the Exerciser.

**TABLE C-17. EXERtrg Field**

| Predef. Symbol | Start |
|---|---|
| **Trig** | 1 |
| **-** | 0 |

### INTx#

Combines the interrupt signals INTA# to INTD# into one group.

**TABLE C-18. INTx# Field**

| Predef. Symbol | Start |
|---|---|
| **---A** | 1110 |
| **--B-** | 1101 |
| **-C---** | 1011 |
| **D---** | 0111 |
| **--BA** | 1100 |
| **-C-A** | 1010 |
| **D--A** | 0110 |
| **--CB-** | 1001 |
| **D-B-** | 0101 |
| **DC--** | 0011 |
| **-CBA** | 1000 |
| **D-BA** | 0100 |
| **DC-A** | 0010 |
| **DCB-** | 0001 |
| **DCBA** | 0000 |
| **----** | 1111 |

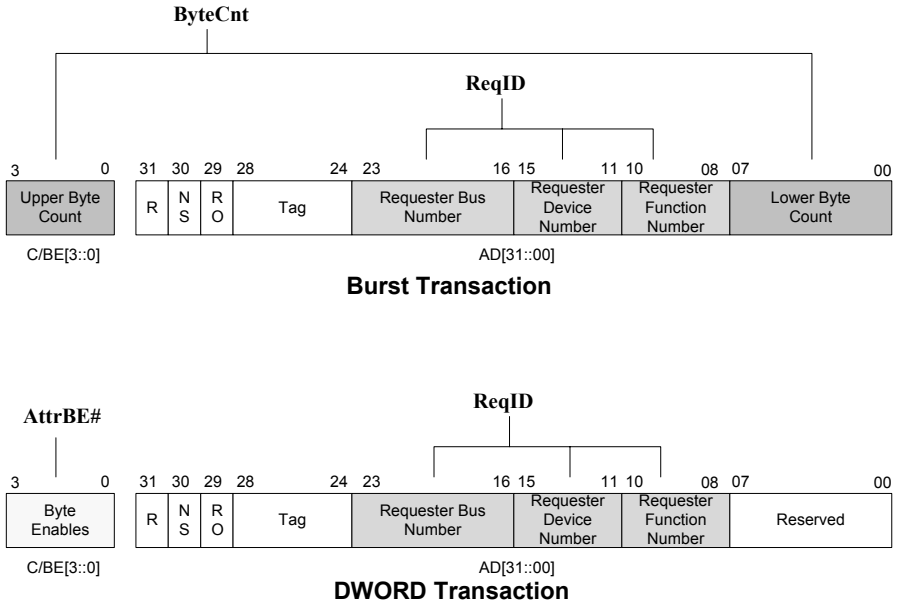



**FIGURE C-2** *Burst and DWORD transaction Attribute Bit Assignments*

### ReqID

Requester ID combines Requester Bus Number, Requester Device Number, and Requester Function Number attributes of the AD[31::00] bus during the Attribute Phase.

### ByteCnt

Combines the Upper and Lower byte counts of the AD[31::00] bus during the attribute phase of a burst transaction.

### AttrBE#

The Byte Enables in the C/BE# bus during the attribute phase of a DWORD transaction.

### CmdDWORD

True when command is a DWORD command

**Tag, NS, RO, R**

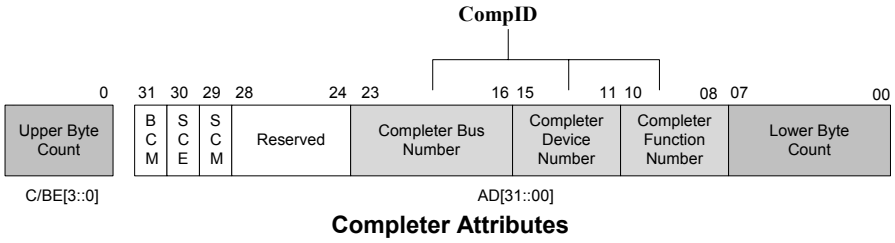



**FIGURE C-3** *Completer Attribute Bit Assignments*

**CompID**

Completer ID combines Completer Bus Number, Completer Device Number, and Completer Function Number attributes of the AD[31::00] bus during a Split Completion command.

**SCM, SCE, BCM :** Split completion control bits.

**SCMsg, SCMsg Class, SCRemByteCnt :** Decodes the split completion message data field when SCE, SCM is set.

# D Embedded Environments

This section discusses using the VanguardPCI in embedded and custom environments.

# *D-1 Using the Vanguard in Embedded and custom environments*

## Introduction

Our tools are frequently used in applications where connectivity to the PCI bus for the analyzer is only added for analysis purposes, or the application is of such an embedded nature that shortcuts with respect to the PCI specification are allowable. In these cases, the minimum requirements for getting the Vanguard Analyzer to operate must be considered. These requirements are discussed here.

**General Requirements** CWCDS uses standard PCI, PMC and cPCI connectors and pinout. If the respective sp is followed when laying out the connector, the analyzer will work and not cause damage. If any deviation from this is present, please consult CWCDS to determine the impact.

None of the Vanguard analyzers connect the Reserved pins or use them for any purpose. The same is true for the VIO signals. These are de coupled with a capacitor. However the 5V adapter that ships with the PCI analyzer does connect VIO to the 5V rail, so if the system or add-on card does not conform to the specification, this may damage the analyzer and the motherboard.

Note – The requirements described below are the minimum for using the analyzer only. If the exerciser is to be used, the requirements listed in the exerciser section also apply.

## PCI Clock

The Vanguard PCI and PCI-X analyzer relies on the PCI clock and only samples if a rising edge on the clock is detected, therefore a good PCI clock is extremely important in order to get reliable results.

If the PCI clock frequency is below 25MHz, make sure to set the "Low Frequency" mode in the Exerciser Functions options found in the Tools, Hardware, Options menu. A clock frequency of 1MHz or higher is recommended, but frequencies down to 1KHz are supported. Low frequencies will result in a slower response from the analyzer.

The correct clock frequency is measured by the analyzer at the rising edge of reset.

## PCI Reset

PCI Reset (RST#) must be asserted at power up and then released with correct timing with respect to Power. This is so that the analyzer can determine the correct Bus Mode (PCI or PCI-X). The analyzer determines the mode by monitoring REQ64#, DEVSEL#, TRDY# and STOP#, as per the PCI-X specification. At the rising edge of RST# and if REQ64 is asserted, then this indicates a 64 bit PCI bus. When RST# is rescinded and there is a certain pattern present on the bus on the signals DEVSEL#, TRDY# and STOP#, this indicates PCI-X mode. Please refer to the PCI-X specification for further details.

The analyzer controls PCIXCAP and M66EN to signal requested mode of operation, as per the PCI-X specification. Please refer to the VANGUARD manual for further details.

## Signal pull-ups and Termination

The Vanguard PCI and PMC analyzers do not provide any pull-ups or termination in addition to what a standard add in card is required to have. The mother board must comply to the specification with respect to this.

The Vanguard Compact PCI analyzer detects whether it is in a System or Peripheral slot and enables the appropriate termination as required by the specification. It does not require or provide any additional pull-ups or termination.

## 32 bit vs. 64 bit systems

The Vanguard Analyzer detects whether the bus is 64 bit when RST is rescinded on the bus. If REQ64# is not asserted, the analyzer will assume 32 bit and behave accordingly. This will not

impact the bus itself. It will operate as normal, but certain features may not be available in the bus analyzer user interface.

## Power

The Vanguard PCI and Compact PCI analyzers require 3.3V power only. The Hex display on the board is the only device that uses 5V, but this display is for information only and does not affect the operation of the Vanguard Analyzer. If 5V is not supplied, the display will be black, but the board will operate normally.

The Vanguard PMC analyzer uses 3.3V power only. However, if the top spacer is being used to stack the Device Under Test (DUT) on top, then this will require 5V from the system. Otherwise the IDSEL, REQ# and GNT# signals will not be passed to the DUT and render it unusable.

The Vanguard PCI and PMC analyzers can be powered externally. (An external power supply may be purchased separately from CWCDS. See “Ordering Information” on page 427) This is especially helpful if the embedded system is unable to provide sufficient power for the Analyzer due to power budgets etc. The external power supply does not provide 5V, and the display may not work if external power is used. Vanguard PMC analyzers still require 5V to be supplied if the top spacer is used to host a DUT on top.

## *D-3 Exerciser Requirements*

The requirements described in this section are the minimum in order to use the Exerciser and in addition to the requirements for the Analyze.

### Arbitration

Due to the self configuration that takes place when the Exerciser is started, arbitration is required. This self configuration involves the Exerciser issuing Config Read and Write cycles to itself. The Exerciser will assert REQ# and expect the arbiter to return a GNT#. If a GNT# is not returned the exerciser will hang and no prompt or messages will be displayed.

### IDSEL routing

In order for the exerciser to complete the self configuration, the IDSEL signal must be connected to one of the AD signals on the PCI bus. How this is done is up to the system designer, but the IDSEL signal to the slot where the exerciser is installed must be able to be asserted.

The PCI-X specification recommends IDSEL to be tied to one of the AD lines through a 2K Ohm series resistor. If the system operates at 66MHz PCI, and no address stepping is used, a 2K Ohm resistor will cause the system to fail. If the system is to operate in PCI only, or both PCI and PCI-X mode, a 100-200 Ohms resistor is recommended. Although this value violates the PCI-X specification, it is widely adopted in the PCI community.

### Workarounds

If RST# is not asserted at power up, the user can wire this in with an external switch or similar. Note that if RST# is toggled by itself (REQ64#, DEVSEL#, TRDY# and STOP# does not contain a valid initialization pattern), the bus mode will be 32 bit PCI.

If the exerciser is the ONLY initiator on the PCI bus, the GNT# signal can be tied to GND through a resistor. This will look like to the Exerciser that it was issued a GNT#. If there are multiple masters in the system this is not recommended since two devices can in this case clobber one another. In PCI-X mode, this method can not be used, since Split Transactions may be issued by any completer.

If IDSEL is not connected correctly, one first has to make sure it is not shared with another device. IDSEL can be wired to one of the AD[31:16] signals through a 100-200 ohm series resistor. Make sure that no other PCI devices on the particular PCI segment is using this AD signal as IDSEL.

# *Index*

# *Ordering Information*

**Vanguard VME**

**VG-VME**

VME State Analyzer, 133 MHz Timing Analyzer and Statistics Module.
This board has NO P0 connector mounted. Includes BusView GUI.

**VG-VMEP0**

VME State Analyzer, 133 MHz Timing Analyzer and Statistics Module.
Includes P0 connector to allow Vanguard PMC module to access
PCI/PCI-X on the P0 connector. Includes BusView GUI.

**VG-VP**

VMEbus Protocol Checker license key for Vanguard product line.

**VG-VE**

Exerciser Module for VG-VME product line.

**VG-PMC**

PMC State Analyzer and Statistics Module. (Single PMC for all functions)
Includes BusView GUI. Can operate as P0 PCI/PCI-X Bus Analyzer for VG-VMEP0.

**Vanguard PCI**

| | |
|---|---|
| **VG-PCI** | 133MHz PCI-X & PCI State Analyzer and Statistics Module for 5V and 3.3V systems.<br>Includes PCI-X carrier, SAM and BusView GUI. |
| **VG-P** | PCI-X & PCI Protocol Checker license key for Vanguard product line. |
| **VG-E** | 100MHz PCI-X & PCI Exerciser and Compliance Checker license key for Vanguard product line. |
| **VG-E2** | 133MHz PCI-X & PCI Enhanced Exerciser, Error Injector and Compliance Checker license key for Vanguard product line. |

**Related Products**

| | |
|---|---|
| **VG-cPCI** | CompactPCI-X & CompactPCI State Analyzer and Statistics Module for 5V and 3.3V systems.<br>Includes cPCI carrier, SAM and BusView GUI.<br>(See separate data sheet) |
| **VG-PMC** | PMC State Analyzer and Statistics Module.<br>(Single PMC for all functions)<br>Includes PCI-X carrier, SAM and BusView GUI. |

Individual boards and SAM modules may be purchased separately. Packages are also available. Please consult CWCDS.

**North and South America**

| | |
|---|---|
| Phone | (281) 584-0728 |
| Fax | (281) 584-9034 |
| Website | www.cwcdefense.com |

**Europe and the rest of the world**

| | |
|---|---|
| Phone | +47 22 10 60 90 |
| Fax | +47 22 10 62 02 |
| Website | www.cwcdefense.com |

# *Technical Support*

In order for us to provide fast technical support, please provide the following information:

- The version of Busview you are using.

  The full version number can be found by clicking About on the Help menu.
- Target Bus / Link Type
- Hardware information (Serial Number, ECO level, etc.).

  The hardware information dialog box is opened by clicking the Info button seen in the menu: Tools, Hardware.
- Operating System that BusView is running under.
- Status of all LEDs on the board.
- Status of the dot matrix LED display.
- Power and Temperature reading.

  These can be found by clicking Voltage and Temperature in the View menu.
- Log files and trace files

**North and South America**

| | |
|---|---|
| Telephone Support | (281) 584-0728 |
| Fax | (281) 584-9034 |
| Website | www.cwcdefense.com/support |

**Europe and the rest of the world**

| | |
|---|---|
| Telephone Support | +47 22 10 60 90 |
| Fax | +47 22 10 62 02 |
| Website | www.cwcdefense.com/support |

# *References*

*PCI Local Bus Specification*, Revision 2.3

*PCI-X Local Bus Specification*, Revision 1.0a

PCI SIG Web Page: http://www.pcisig.com

Tom Shanley and Don Anderson. *PCI System Architecture*, 4th ed.: Addison-Wesley Pub Co, 1999.

Tom Shanley. *PCI-X System Architecture:* Addison-Wesley Pub Co, 2001.

Ed Solari and George Willse. *PCI Hardware and Software*, 4th ed.: Annabooks, 1998.