CURTISS-WRIGHT

VANGUARD Bus Analyzer

User Guide
VME Analyzer and Exerciser
for
BusView5

Document No. A-T-MU-BAVME###-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
- Appendixes: Specifications

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

CWDS 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.

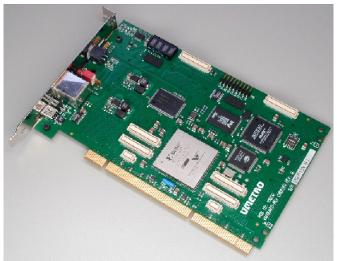
- **PCI** - A PCI carrier card contains the Exerciser. A SAM module is added to provide Analyzer, Protocol Checker and Statistics functions. Also available is the 0-slot PCI adapter which allows a PCI card (Device Under Test) to be inserted into the same slot as the PCI Vanguard.
- **Compact PCI** - 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.

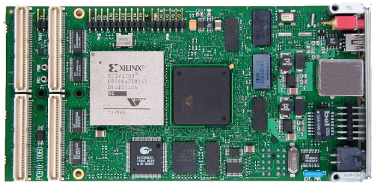
It can be used in any 32 or 64 bit slot in a PCI/PCI-X motherboard.

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

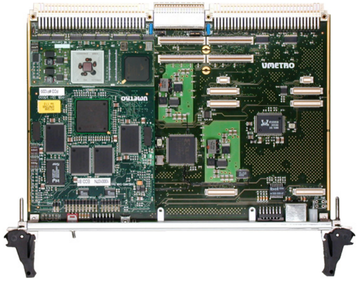
The CompactPCI Vanguard is a 3U Eurocard based form connected via a passive PCI backplane.

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.

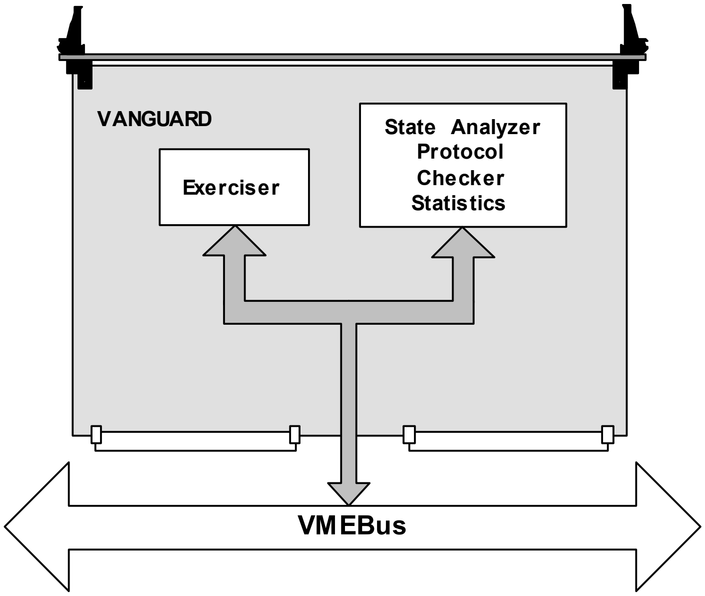


FIGURE 1-1 *Analyzer acquires bus signals*

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 313.

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 VME Assembly
- 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 VME Assembly

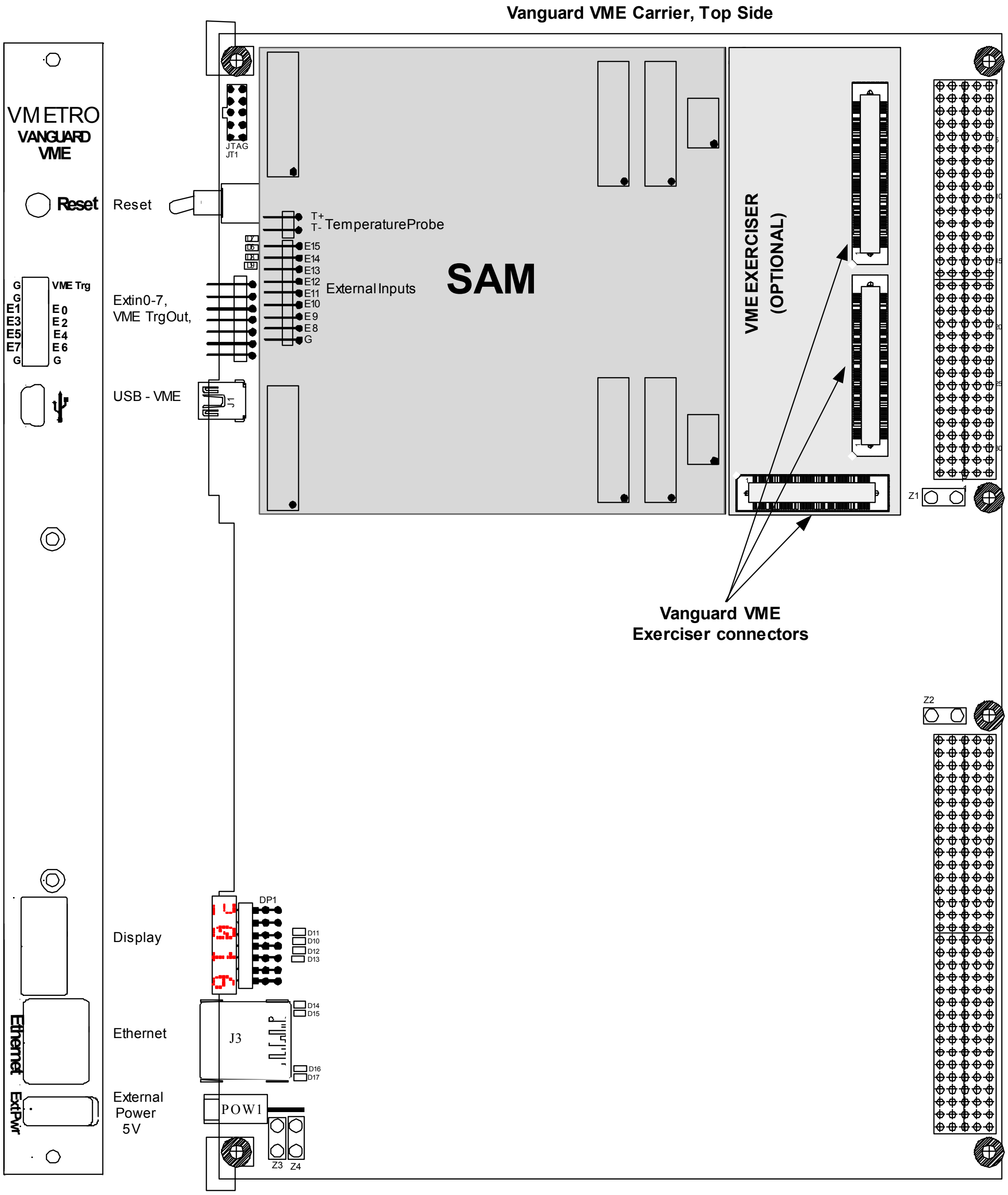


FIGURE 2-1 *Vanguard VME board layout*

Vanguard Hardware Features

LED Dot Matrix Display

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
- Bus Type

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 Options tab in the Hardware Menu dialog. If no options have been set, then the display will show the voltage.

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 jumpers must be moved from Z1 and Z2 and placed in positions Z3 and Z4.

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 “VME Trg”.

External Inputs

There are 16 inputs available which are marked E0-E15. These correspond to the Ext[15:0] 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 59)

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

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.

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.

2.2 Exerciser

The optional Exerciser module offers a programmable VMEbus Master, Target and System controller.

The Exerciser is supplied as a small circuit board to be mounted onto the VME carrier board.

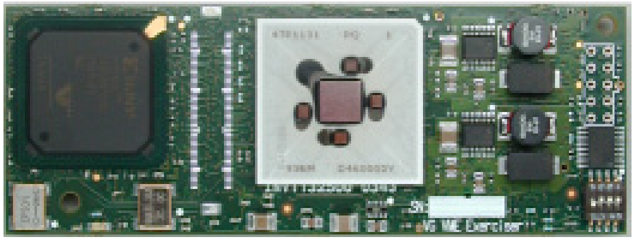


FIGURE 2-2 *Vanguard VME Exerciser module*

Exerciser DIP switch settings

SW1 Not In Use (Default Off)

SW2 Not In Use (Default Off)

SW3 **On:** SYSRES* and SYSFAIL* are disabled at power-up (Default).
Off: SYSRES* and SYSFAIL* are enabled.

SW4 **On:** Auto System Controller is enabled. The Vanguard VME Exerciser will act as system controller as long as BG3IN is low at power-up. However, if another device drives BG3IN high then the Vanguard VME Exerciser will not act as system controller.
Off: The Vanguard VME Exerciser will not act as system controller (Default).

Installing the Exerciser Module

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

If the Exerciser is purchased separately from the Vanguard VME carrier board use these instructions to mount the Exerciser module onto the VME carrier board.

1. Place the Vanguard VME board on a surface with controlled static characteristics.
2. Remove the Exerciser module from its anti-static bag.
3. Locate the three connectors on the bottom of the Exerciser module and line them up with the three connectors shown in Figure 2-1.
4. Firmly but gently, press the Exerciser into the connectors on the carrier board.

2.3 External Power

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

To change to external power, the jumpers placed in positions Z1 and Z2 must be moved to positions Z3 and Z4.

2.4 Miscellaneous Parts

External Temperature Probe

Connects to the Temperature Input pins.

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

Patch Cords

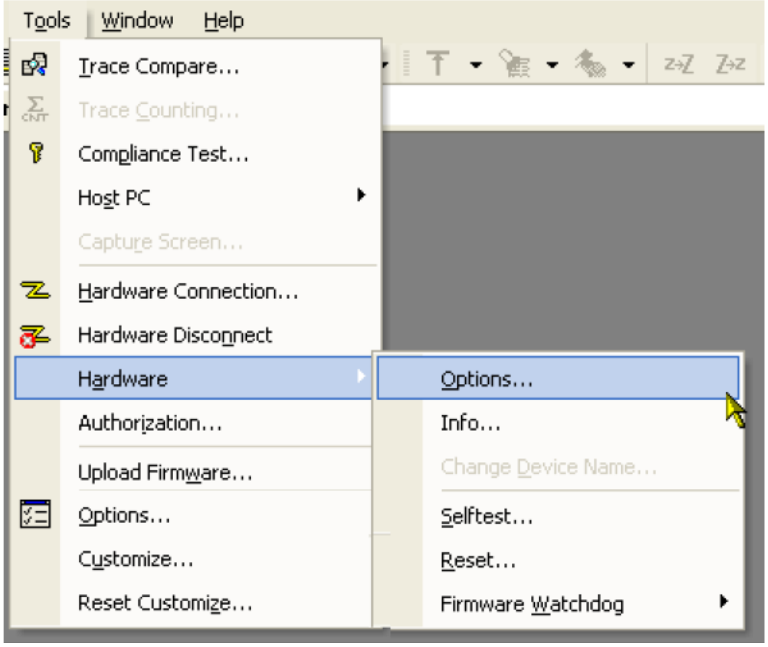
These cables are necessary for connecting the External Inputs.

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.

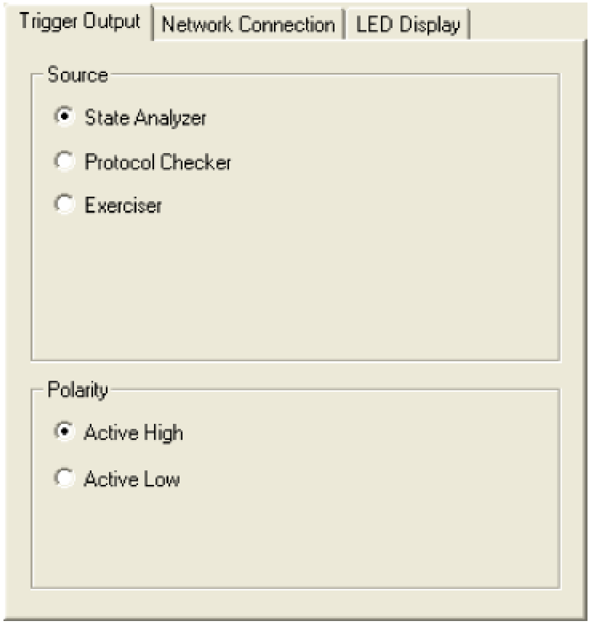
2.5 Hardware Menu

The Hardware sub menu is found under Hardware in the Tools menu



Options

Trigger Output


Trigger Output | Network Connection | LED Display

Source

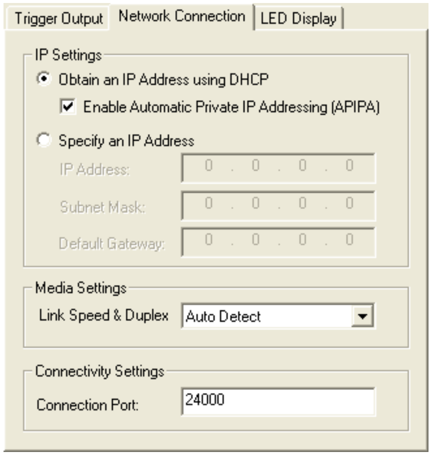
● State Analyzer
○ Protocol Checker
○ Exerciser

Polarity

● Active High
○ Active Low

- 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

● Obtain an IP Address using DHCP

☑ Enable Automatic Private IP Addressing (APIPA)

○ Specify an IP Address

IP Address: 0 . 0 . 0 . 0

Subnet Mask: 0 . 0 . 0 . 0

Default Gateway: 0 . 0 . 0 . 0

Media Settings

Link Speed & Duplex: Auto Detect

Connectivity Settings

Connection Port: 24000

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)

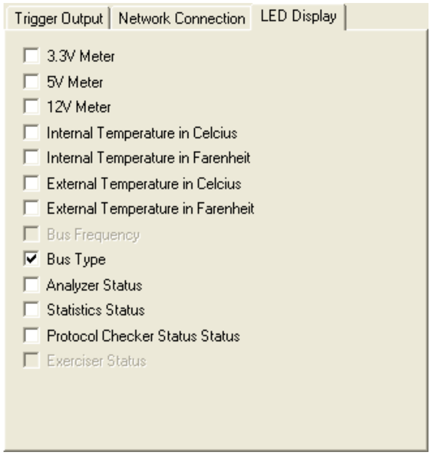
  10Mbps - Half Duplex (10bT HD - slowest)

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

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



□ 3.3V Meter
□ 5V Meter
□ 12V Meter
□ Internal Temperature in Celcius
□ Internal Temperature in Farenheit
□ External Temperature in Celcius
□ External Temperature in Farenheit
□ Bus Frequency
☑ Bus Type
□ Analyzer Status
□ Statistics Status
□ Protocol Checker Status Status
□ Exerciser Status

- 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.
- Internal/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 "Unkn" (Unknown) is displayed.
- Internal/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 "Unkn" (Unknown) is displayed.
- Bus frequency - Frequency is displayed as "33M2", "12k7", i.e. one decimal with an uppercase "M" (Mega Hertz) or lowercase "k" (kilo Hertz) as the decimal point.

  A bus frequency of 0Hz will be displayed as “NOCL”
- Bus type - "VME"
- 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:
  "Unkn" - Not configured
  "PDIS" - Prot. check. Disabled
  "PENA" - Prot. check. Enabled

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 315 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 315 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.

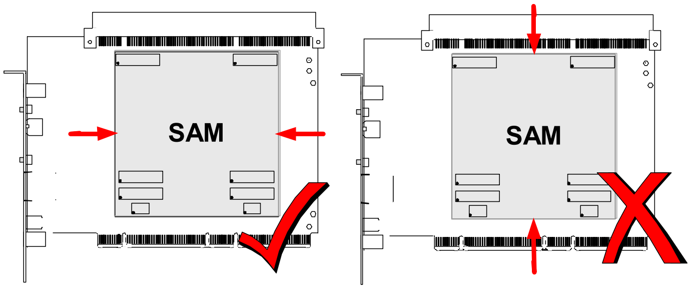
2.6 Moving the SAM

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

5. Wear an anti-static wrist strap or follow the static electricity precautions listed in the Installation Guide.
6. Ensure the carrier board containing the SAM is placed on a surface with suitably controlled anti-static properties.
7. 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:.

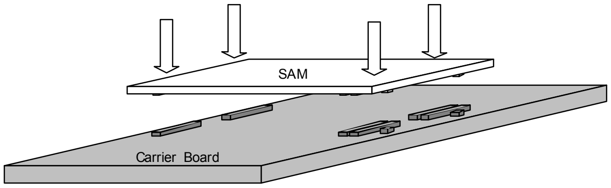
Note – Be careful when removing the SAM. The connectors may become damaged if they are forced at an angle.



8. 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.


SAM
Carrier Board

2.7 VME P0 Carrier

PMC V(I/O) Voltage Keys

The PMC voltage keys must be mounted in either the 3.3V or 5V position on the VMEP0 Carrier according to the V(I/O) voltage used by the Host system for the P0 PCI bus.

Make sure the voltage key screw is tightened firmly, since the V(I/O) voltage is electrically connected the V(I/O) pins on the PMC slot on the VMEP0 Carrier using the voltage key pin.

TABLE 2-2. J4, PMC-PO CLK Select

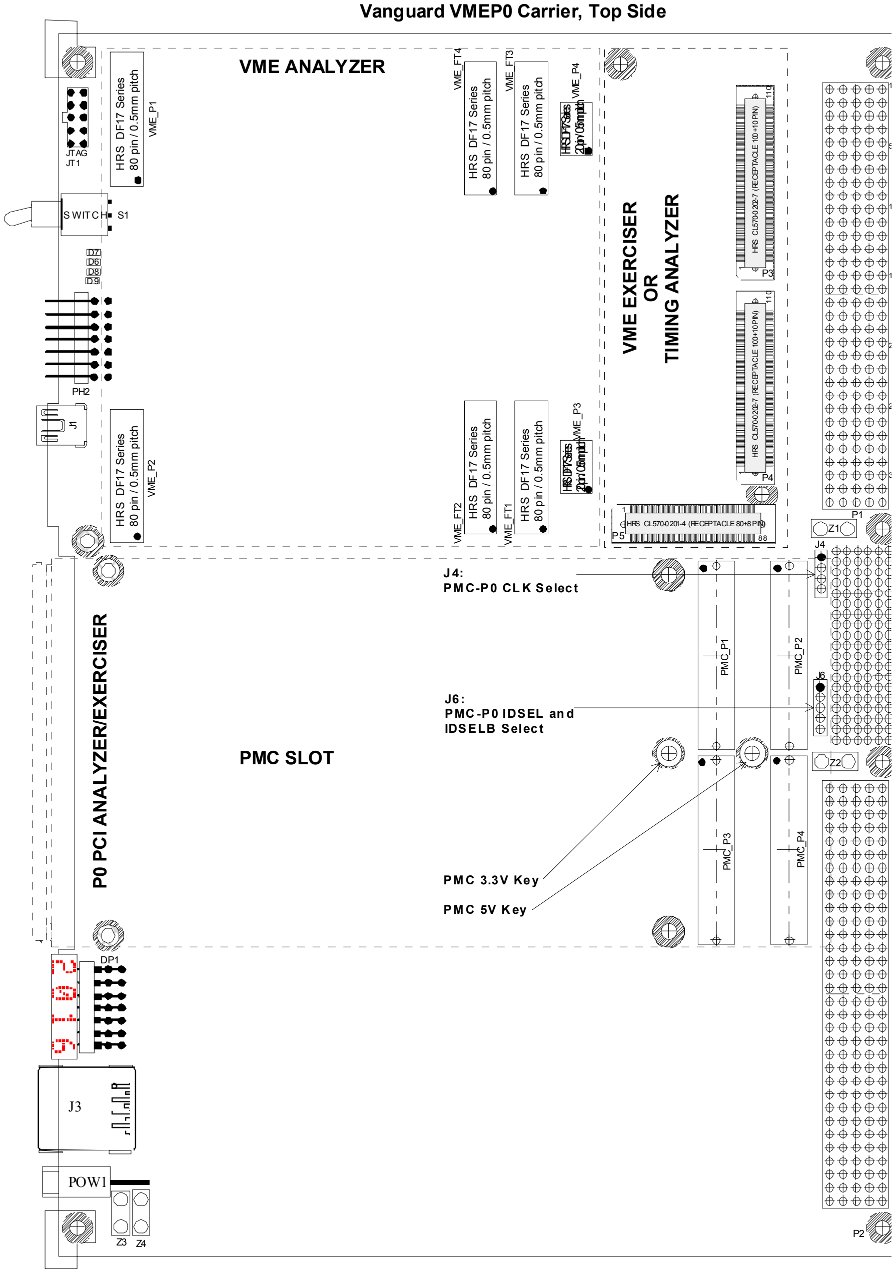
Shunt position	PMC CLK pin Function	Comment
1-2	n.a.	Not Used
2-3 (default)	CLK Input	P0 pin 4D is used as clock source to PMC slot
3-4	n.a.	Not Used

TABLE 2-3. J6, PMC-PO IDSEL and IDSELB Select

Shunt position	PMC CLK pin Function	Comment
1-2 (default)	IDSEL = AD12	IDSEL is connected to AD12 on P0 connector
2-3	IDSEL = AD13	IDSEL is connected to AD13 on P0 connector
3-4 (default)	IDSELB = AD13	IDSELB is connected to AD13 on P0 connector
4-5	IDSELB = AD14	IDSELB is connected to AD14 on P0 connector

Board Layout

Vanguard VMEP0 Carrier, Top Side



P0 Connector Pin Assignment (PCI) on VG-VMEP0

TABLE 2-4. **P0 Connector Pin Assignment (PCI) on VG-VMEP0**

Pin Number	P0 Connector					
	Row a	**Row b**	**Row c**	**Row d**	**Row e**	**Row f**
1	AD32	AD38	AD45	AD52	AD58	GND
2	AD33	AD39	AD46	AD53	AD59	GND
3	AD34	AD40	AD47	AD54	AD60	GND
4	VIO	AD41	AD48	CLK	PERR#	GND
5	REQ2#(*)	GNT2#(*)	REQ#	GNT#	C/BE4#	GND
6	C/BE1#	ACK64#	REQ64#	LOCK#	SERR#	GND
7	RST#	FRAME#	C/BE2#	C/BE3#	C/BE0#	GND
8	PAR	TRDY#	IRDY#	DEVSEL#	C/BE6#	GND
9	AD35	AD42	AD49	AD55	AD61	GND
10	AD36	AD43	AD50	AD56	AD62	GND
11	AD37	AD44	AD51	AD57	AD63	GND
12	AD0	AD1	PAR64	C/BE7#	STOP#	GND
13	AD5	AD2	AD3	INTA#	INTB#	GND
14	AD8	AD9	AD6	AD7	AD4	GND
15	AD15	AD12	AD13	AD10	AD11	GND
16	AD18	AD19	AD16	AD17	AD14	GND
17	AD25	AD22	AD23	AD20	AD21	GND
18	INTD#	INTC#	AD26	AD27	AD24	GND
19	C/BE5#	AD30	AD31	AD28	AD29	GND

(*) REQ2# and GNT2# are the request and grant signals from an additional PCI slot.

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:

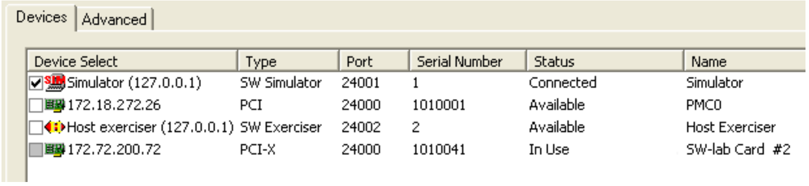
1. Open the Device Information dialog (press Hardware Detection button from the Tools Menu), and select the Advanced tab.
2. Enter the IP address assigned to your Vanguard and press OK, the Name field is optional.
3. Return to the Hardware Connection dialog (press Tools - Hardware - Options).
4. Select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



Device Select	Type	Port	Serial Number	Status	Name
☑ Simulator (127.0.0.1)	SW Simulator	24001	1	Connected	Simulator
□ 172.18.272.26	PCI	24000	1010001	Available	PMC0
□ Host exerciser (127.0.0.1)	SW Exerciser	24002	2	Available	Host Exerciser
▨ 172.72.200.72	PCI-X	24000	1010041	In Use	SW-lab Card #2

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 38 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 26.

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.

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

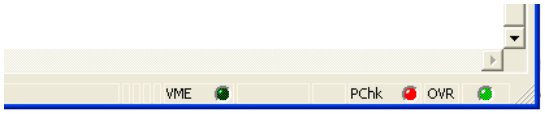


FIGURE 3-2 *Status Bar*

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.

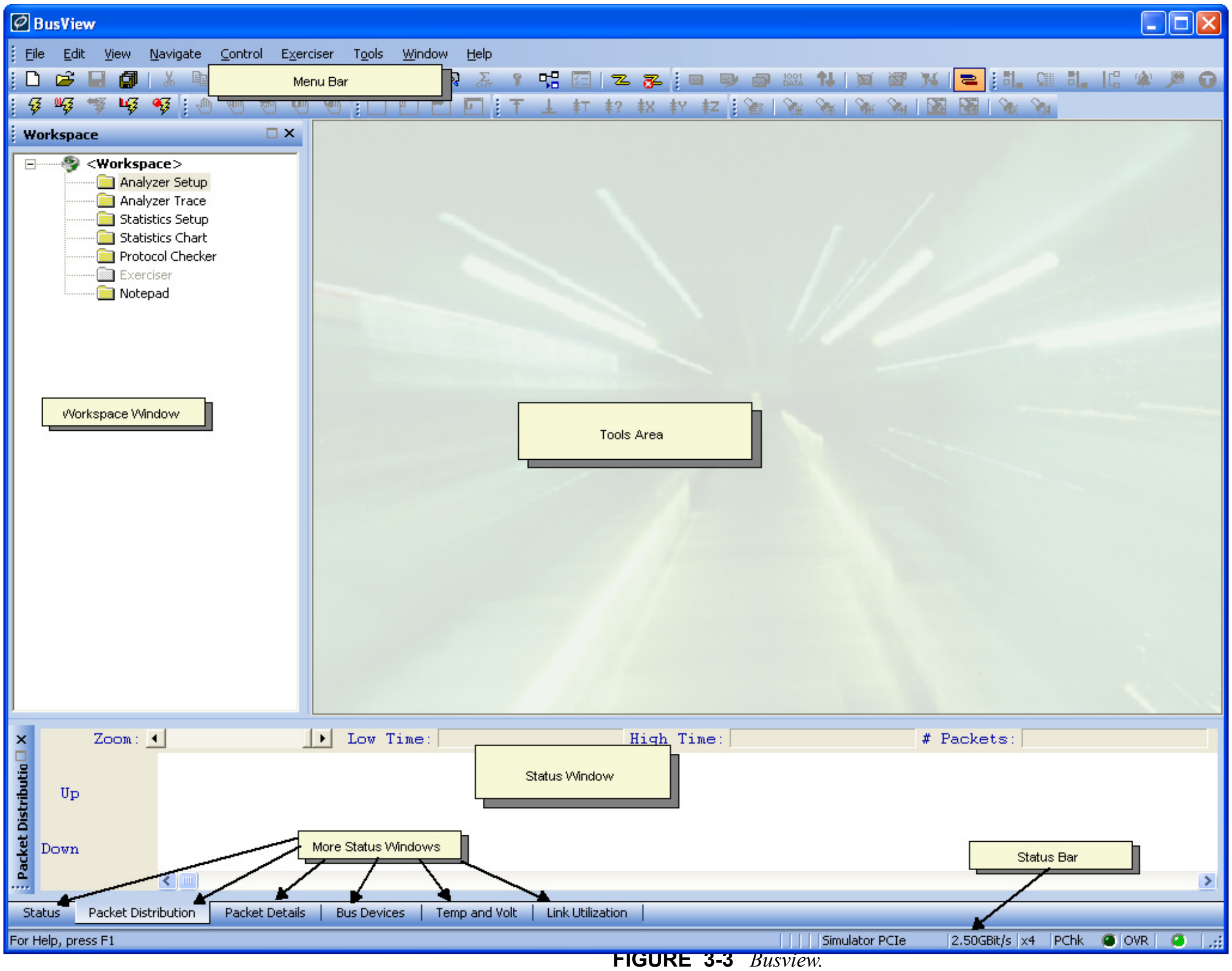


FIGURE 3-3 *Busview.*

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:

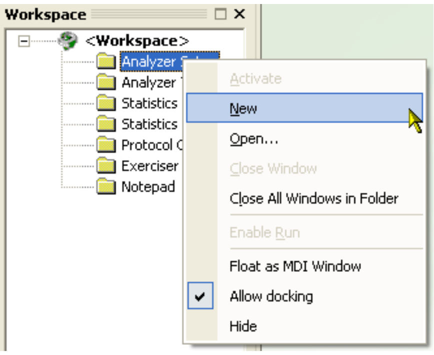


FIGURE 3-4 *Using the Workspace Window*

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

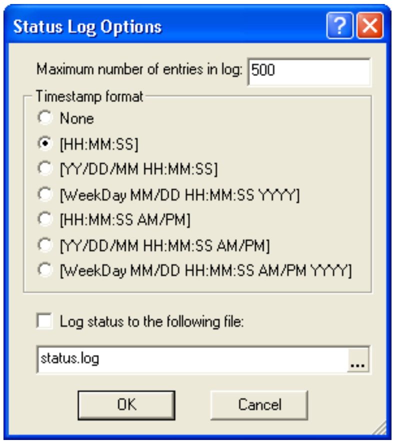
This window is a running list of all activity in the tools. Each entry is time stamped. This log can be saved to a file for future reference.

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.



Status Log Options

Maximum number of entries in log: 500

Timestamp format

○ None
● [HH:MM:SS]
○ [YY/DD/MM HH:MM:SS]
○ [WeekDay MM/DD HH:MM:SS YYYY]
○ [HH:MM:SS AM/PM]
○ [YY/DD/MM HH:MM:SS AM/PM]
○ [WeekDay MM/DD HH:MM:SS AM/PM YYYY]

□ Log status to the following file:

status.log

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 41.

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 87 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.

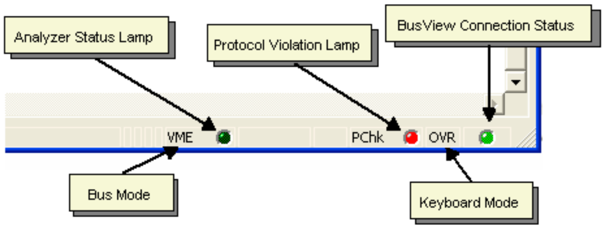


FIGURE 3-5 *Status Bar*

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

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.

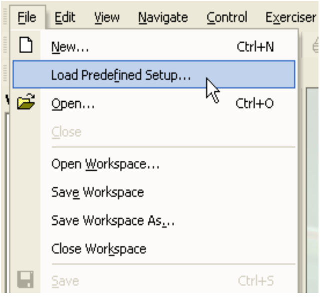
Undo, Copy, Cut and Paste

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.

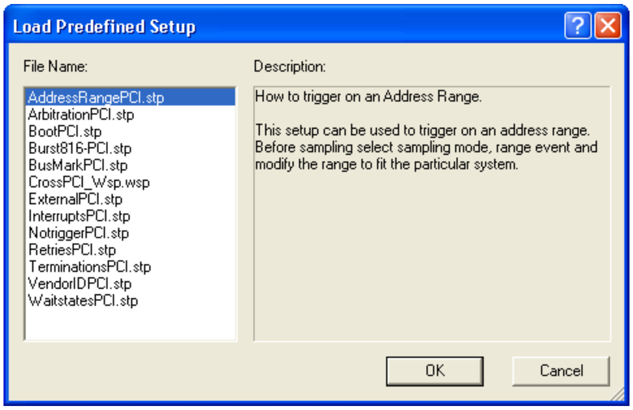
3.4 Predefined Setups

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.



AddressRangePCI.stp
ArbitrationPCI.stp
BootPCI.stp
Burst816-PCI.stp
BusMarkPCI.stp
CrossPCI_Wsp.wsp
ExternalPCI.stp
InterruptsPCI.stp
NotriggerPCI.stp
RetriesPCI.stp
TerminationsPCI.stp
VendorIDPCI.stp
WaitstatesPCI.stp

How to trigger on an Address Range.

This setup can be used to trigger on an address range. Before sampling select sampling mode, range event and modify the range to fit the particular system.

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. The Decode Option for each Field. 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 one Vanguard VME and one Vanguard VME P0 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\VMEComm.ini"
   or

   (for 64 bit systems)
   "C:\Program Files (x 86)\Vmetro\BusView5\BusView5.exe" "-CC:\Documents and Settings\User\My Documents\BusView\Documents\VMEComm.ini"

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

4. Start BusView from this shortcut and in the Device Information dialog, select the Vanguard Express. When BusView next closes, a file called **VMEComm.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 VME whenever this shortcut is used.

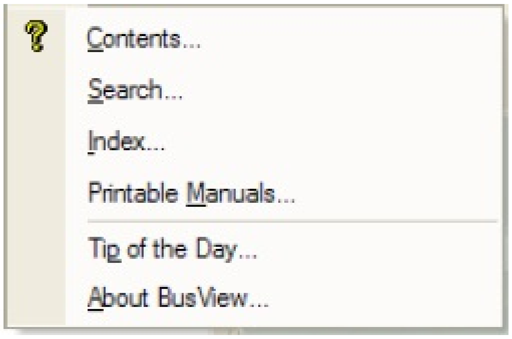
5. Repeat the above steps to create another shortcut called **BusViewVMEP0** for the VanguardVME P0.

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\VMEComm.ini"
“C:\Documents and Settings\User\My Documents\BusView\Documents\VME.wsp"
will connect to the Vanguard VME we stated in the above example as well as open BusView using the Workspace Files **VME.wsp**.

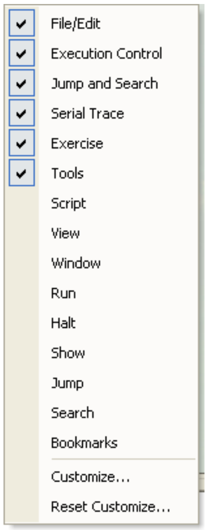
3.7 Updating Software and Getting Help

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


☑ File/Edit
☑ Execution Control
☑ Jump and Search
☑ Serial Trace
☑ Exercise
☑ Tools
Script
View
Window
Run
Halt
Show
Jump
Search
Bookmarks
Customize...
Reset Customize...

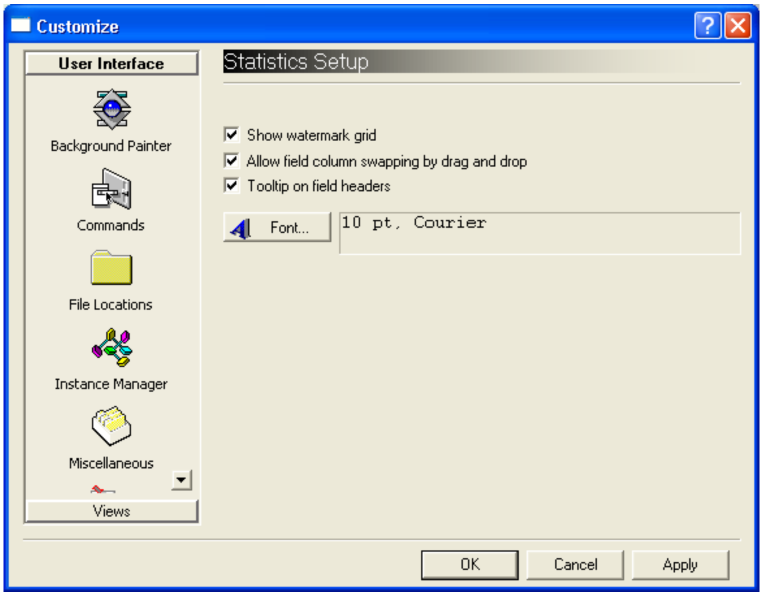
The appearance of BusView can be changed using the Customize menu. To access the Customize menu, right click anywhere on the Menu Bar or the Tools area as shown in Figure 3-3

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.



Statistics Setup

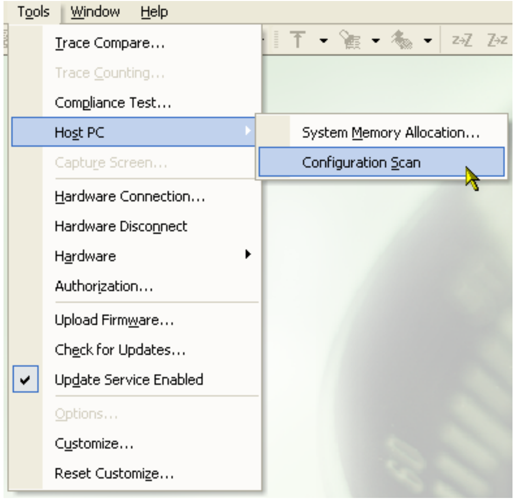
☑ Show watermark grid
☑ Allow field column swapping by drag and drop
☑ Tooltip on field headers

Font: 10 pt, Courier

FIGURE 3-7 *Customize Dialog*

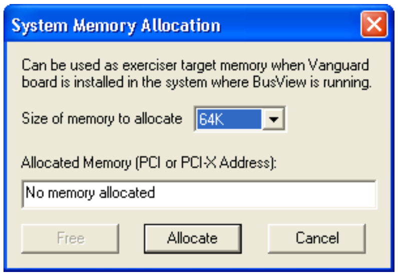
3.9 Host PC

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.



System Memory Allocation

Can be used as exerciser target memory when Vanguard board is installed in the system where BusView is running.

Size of memory to allocate: 64K

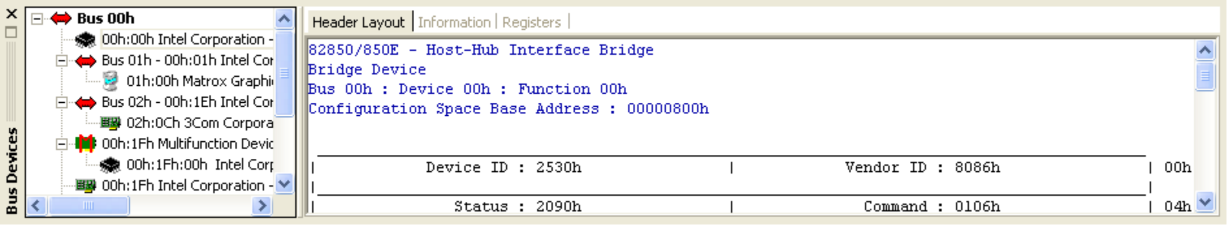
Allocated Memory (PCI or PCI-X Address):

No memory allocated

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.


Bus Devices
Bus 00h
00h:00h Intel Corporation -
Bus 01h - 00h:01h Intel Cor
01h:00h Matrox Graphi
Bus 02h - 00h:1Eh Intel Cor
02h:0Ch 3Com Corpora
00h:1Fh Multifunction Devic
00h:1Fh:00h Intel Corp
00h:1Fh Intel Corporation -
Header Layout
Information
Registers

```
82850/850E - Host-Hub Interface Bridge
Bridge Device
Bus 00h : Device 00h : Function 00h
Configuration Space Base Address : 00000800h

 _________________________________________________________________________________________
|            Device ID : 2530h               |               Vendor ID : 8086h            | 00h
|____________________________________________|____________________________________________|
|               Status : 2090h               |                 Command : 0106h            | 04h
```

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 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 to data.

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

4.1 Introduction

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

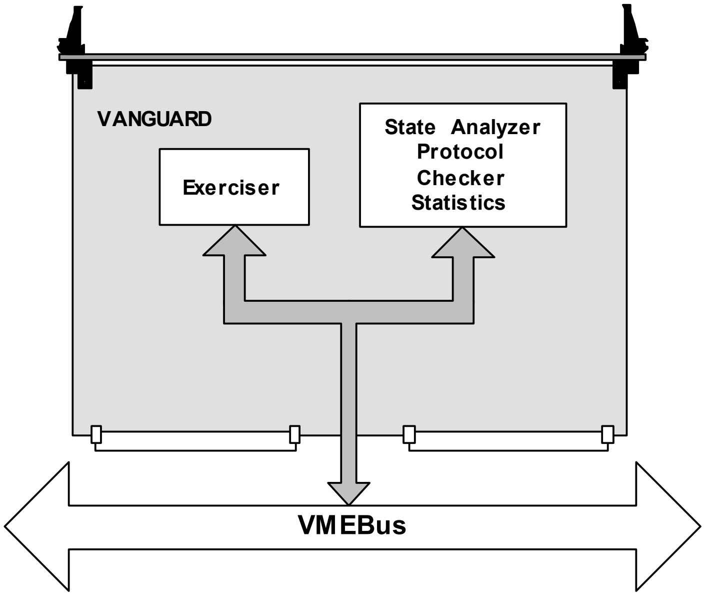


FIGURE 4-1 *Block diagram of the State Analyzer*

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.

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.

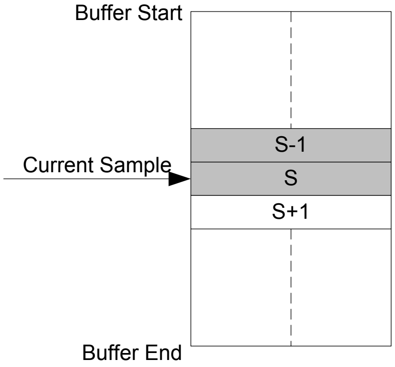
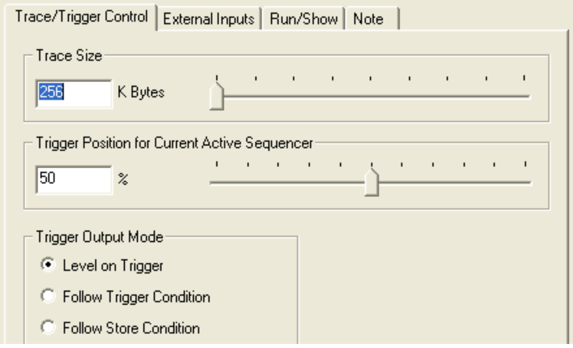


FIGURE 4-2 *Trace Buffer as circular memory*

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.



Trace Size

256 K Bytes

Trigger Position for Current Active Sequencer

50 %

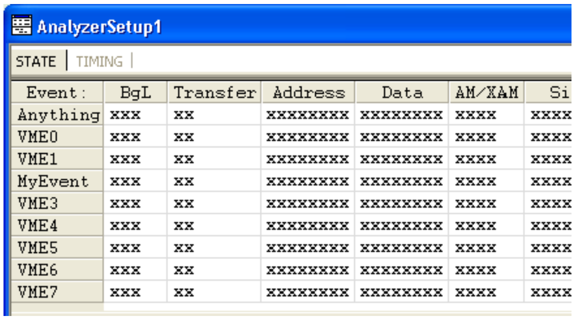
Trigger Output Mode

● Level on Trigger

○ Follow Trigger Condition

○ Follow Store Condition

FIGURE 4-3 *Trace buffer settings*



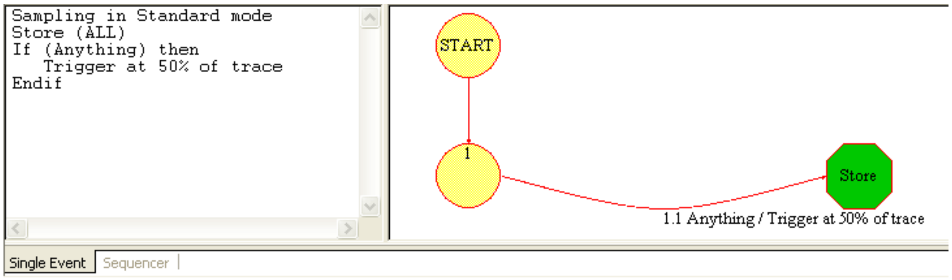
Event:	BgL	Transfer	Address	Data	AM/XAM	Si
Anything	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
VME0	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
VME1	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
MyEvent	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
VME3	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
VME4	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
VME5	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
VME6	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
VME7	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx

FIGURE 4-4 *Example of VME 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.



```
Sampling in Standard mode
Store (ALL)
If (Anything) then
   Trigger at 50% of trace
Endif
```

FIGURE 4-5 *The Sequencer*

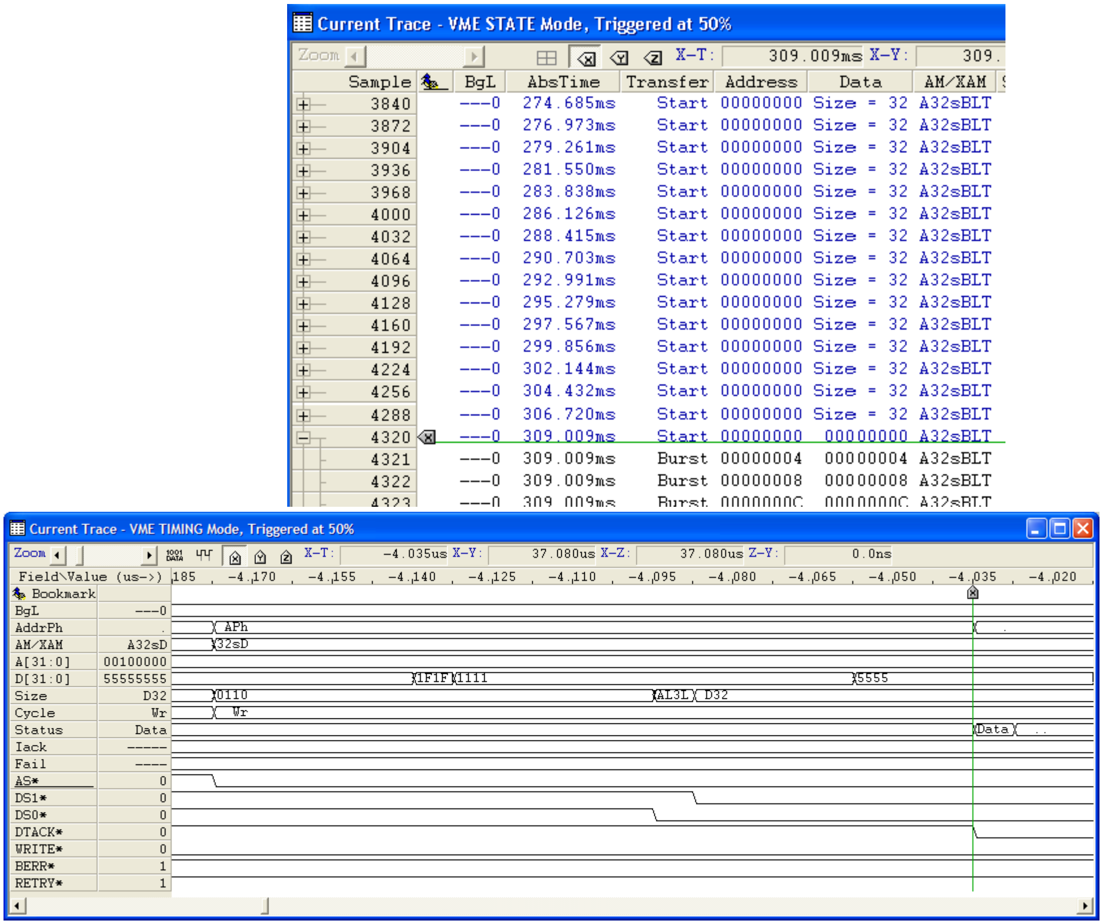
A diagrammatic representation of the sequencer is displayed in parallel to the sequencer. The diagram is useful for visualizing the current sequence. It is possible to move elements of the diagram to improve readability.

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.

VME Trace



Current Trace - VME STATE Mode, Triggered at 50%

Zoom X-T: 309.009ms X-Y: 309.

Sample		BgL	AbsTime	Transfer	Address	Data	AM/XAM
3840		---0	274.685ms	Start	00000000	Size = 32	A32sBLT
3872		---0	276.973ms	Start	00000000	Size = 32	A32sBLT
3904		---0	279.261ms	Start	00000000	Size = 32	A32sBLT
3936		---0	281.550ms	Start	00000000	Size = 32	A32sBLT
3968		---0	283.838ms	Start	00000000	Size = 32	A32sBLT
4000		---0	286.126ms	Start	00000000	Size = 32	A32sBLT
4032		---0	288.415ms	Start	00000000	Size = 32	A32sBLT
4064		---0	290.703ms	Start	00000000	Size = 32	A32sBLT
4096		---0	292.991ms	Start	00000000	Size = 32	A32sBLT
4128		---0	295.279ms	Start	00000000	Size = 32	A32sBLT
4160		---0	297.567ms	Start	00000000	Size = 32	A32sBLT
4192		---0	299.856ms	Start	00000000	Size = 32	A32sBLT
4224		---0	302.144ms	Start	00000000	Size = 32	A32sBLT
4256		---0	304.432ms	Start	00000000	Size = 32	A32sBLT
4288		---0	306.720ms	Start	00000000	Size = 32	A32sBLT
4320	☒	---0	309.009ms	Start	00000000	00000000	A32sBLT
4321		---0	309.009ms	Burst	00000004	00000004	A32sBLT
4322		---0	309.009ms	Burst	00000008	00000008	A32sBLT
4323		---0	309.009ms	Burst	0000000C	0000000C	A32sBLT

Current Trace - VME TIMING Mode, Triggered at 50%

Zoom X-T: -4.035us X-Y: 37.080us X-Z: 37.080us Z-Y: 0.0ns

Field\Value (us->)	
Bookmark	
BgL	---0
AddrPh	.
AM/XAM	A32sD
A[31:0]	00100000
D[31:0]	55555555
Size	D32
Cycle	Wr
Status	Data
Iack	-----
Fail	----
AS*	0
DS1*	0
DS0*	0
DTACK*	0
WRITE*	0
BERR*	1
RETRY*	1

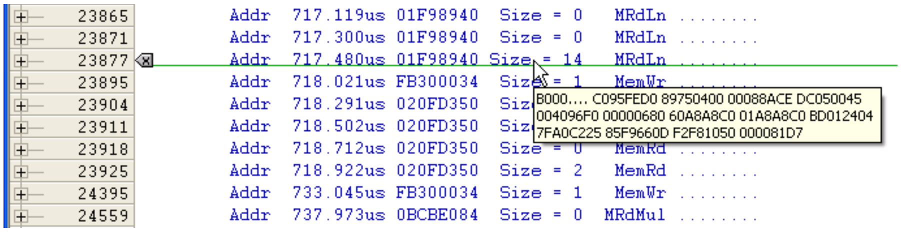
FIGURE 4-6 *Example VME trace in alphanumeric and waveform formats*

To help identify data phases, the Vanguard will increment the address field in the trace buffer for each data cycle. The start address of a data block is highlighted.

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.
- 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.



```
23865   Addr  717.119us 01F98940  Size = 0   MRdLn ........
23871   Addr  717.300us 01F98940  Size = 0   MRdLn ........
23877   Addr  717.480us 01F98940 Size = 14   MRdLn ........
23895   Addr  718.021us FB300034  Size = 1   MemWr ........
23904   Addr  718.291us 020FD350  Siz
23911   Addr  718.502us 020FD350  Siz
23918   Addr  718.712us 020FD350  Size = 0   MemRd ........
23925   Addr  718.922us 020FD350  Size = 2   MemRd ........
24395   Addr  733.045us FB300034  Size = 1   MemWr ........
24559   Addr  737.973us 0BCBE084  Size = 0  MRdMul ........
```

```
B000.... C095FED0 89750400 00088ACE DC050045
004096F0 00000680 60A8A8C0 01A8A8C0 BD012404
7FA0C225 85F9660D F2F81050 000081D7
```

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:

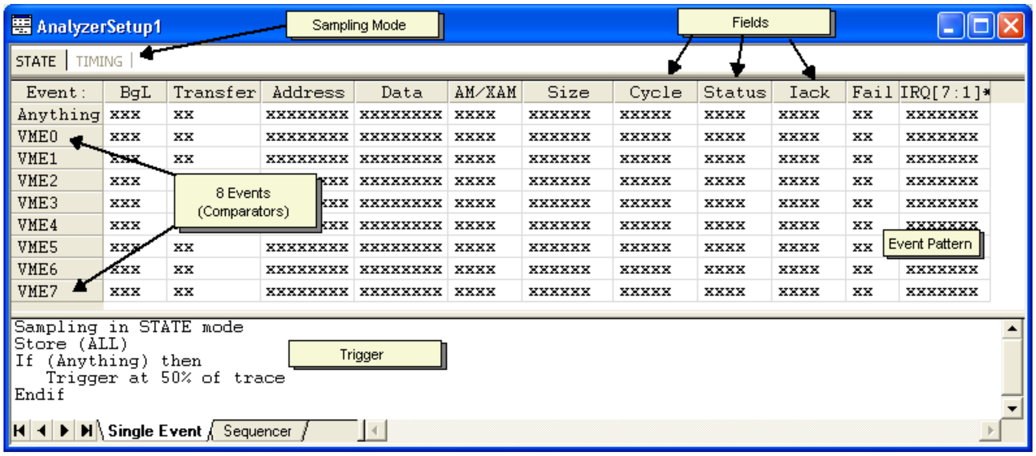
1. A new instance of the Analyzer is opened.
2. Event Patterns are set in one or more Comparators.
3. A trigger condition is set to stop sampling in the trace buffer.

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.



Event:	BgL	Transfer	Address	Data	AM/XAM	Size	Cycle	Status	Iack	Fail	IRQ[7:1]*
Anything	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
VME0	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
VME1	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
VME2	xxx		xxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
VME3	xxx		xxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
VME4	xxx		xxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
VME5	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	
VME6	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
VME7	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx

```
Sampling in STATE mode
Store (ALL)
If (Anything) then
   Trigger at 50% of trace
Endif
```

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

EventsAny 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.

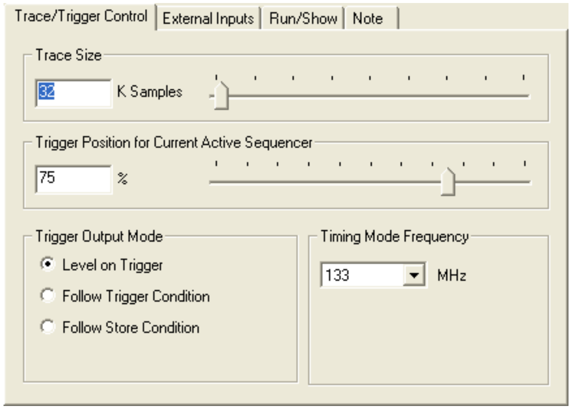
Event Patterns This area is used to specify what signals the comparators should use when looking at the activity. The Fields shown depend on which Sampling Mode is used. The default events do not have any customizable fields inserted. Once the Format or Type fields are specified, the most common user defined fields are displayed. More Fields can be inserted, See “Manipulating Field Columns” on page 59.

TriggerUsing 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



Trace Size
32 K Samples

Trigger Position for Current Active Sequencer
75 %

Trigger Output Mode
● Level on Trigger
○ Follow Trigger Condition
○ Follow Store Condition

Timing Mode Frequency
133 MHz

FIGURE 4-8 *Trace Setup Options window*

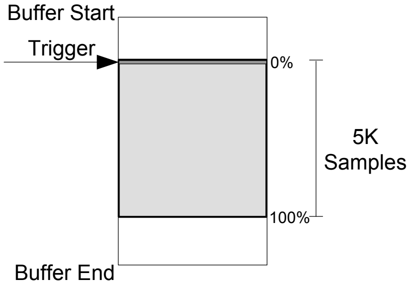
Trace Size

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.

Trigger Position

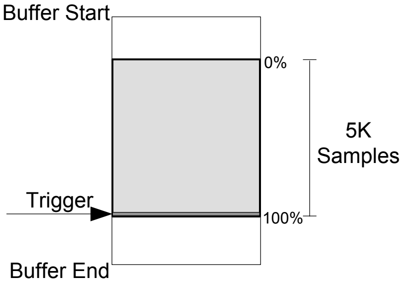
The trigger position value determines where the start and end of the stored trace is. In the following examples, we assume the size of the trace buffer has been set to 5 K Samples.

Example a Trigger position at 0%



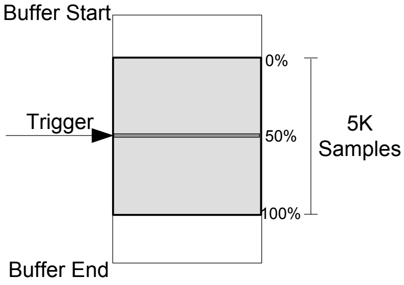
When the trigger condition has occurred, the next 5000 samples are stored.

Example b Trigger at 100%



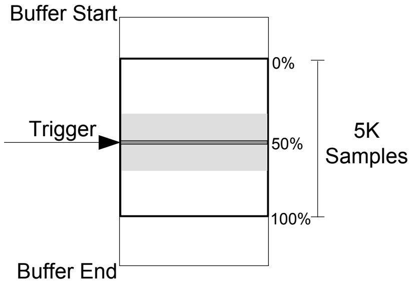
When the trigger condition has occurred, the previous 5000 samples are stored.

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:

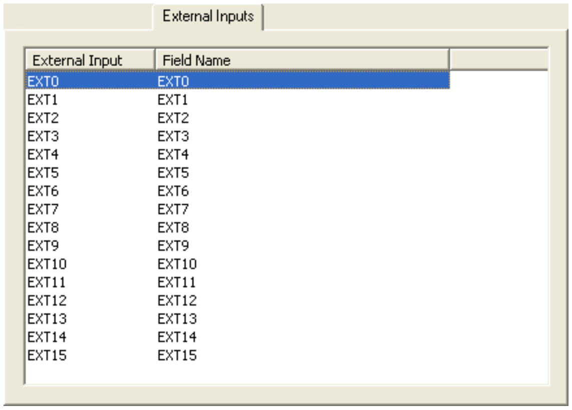
- Level on Trigger - The TRIG output is active once the trigger condition occurs and stays active for the duration of the trace.
- Follow Trigger - The TRIG output is active once the trigger condition occurs and stays active until the next sample that does not meet the trigger condition.
- Follow Store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

Timing Mode Frequency

Frequency used in Timing Mode sampling.

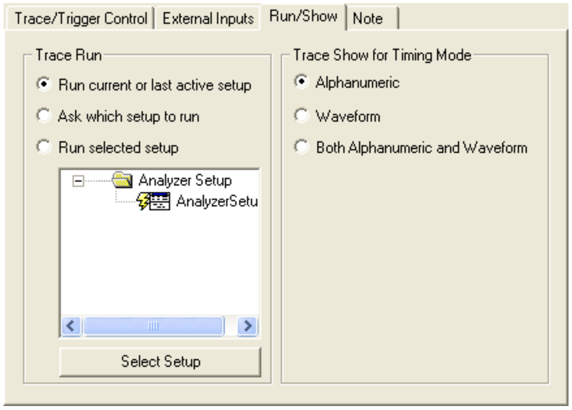
External Inputs



External Input	Field Name
EXT0	EXT0
EXT1	EXT1
EXT2	EXT2
EXT3	EXT3
EXT4	EXT4
EXT5	EXT5
EXT6	EXT6
EXT7	EXT7
EXT8	EXT8
EXT9	EXT9
EXT10	EXT10
EXT11	EXT11
EXT12	EXT12
EXT13	EXT13
EXT14	EXT14
EXT15	EXT15

The Field Name can be changed on this menu by double clicking on the Field Name. See "External Inputs" on page 16 for information about the external inputs.

Trace Run



Trace Run
- ● Run current or last active setup
- ○ Ask which setup to run
- ○ Run selected setup

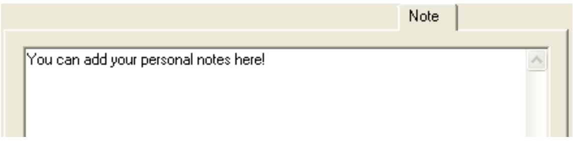
Trace Show for Timing Mode
- ● Alphanumeric
- ○ Waveform
- ○ Both Alphanumeric and Waveform

- 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 TimingMode

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

Note



You can add your personal notes here!

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

4.3 Sampling Modes

State Mode (Synchronous)

Synchronous sampling is used for STATE analysis, and captures cycles from the target system one by one, so that each collected cycle forms one line in the trace buffer. This sampling mode requires that the sampling logic extract sampling clocks from the target bus at the correct times in order to store information like address, data, transfer size and status in a compact form in the trace buffer.

Timing Mode (Asynchronous)

Asynchronous sampling is used for TIMING analysis, i.e. the bus is sampled at a fixed rate that can be selected at fourteen different speeds as fast as 133MHz or as slow as 6.25MHz.

With a sampling rate of 133MHz the bus is sampled every 7.5ns, which is sufficient to show the general timing relation between signals on VME.

4.4 Event Patterns

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

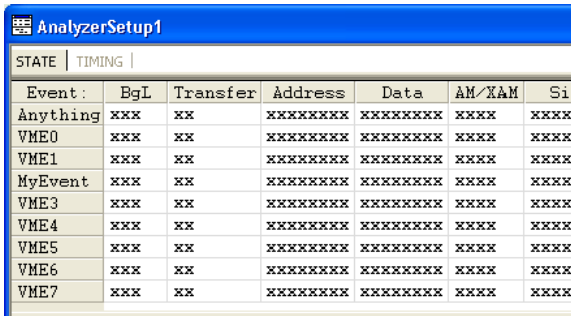
There are three steps to setting up a Single Event trigger.

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

Events

VME0-VME7

The eight predefined editable patterns are, by default, labeled VME0-VME7



Event :	BgL	Transfer	Address	Data	AM/XAM	Si
Anything	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
VME0	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
VME1	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
MyEvent	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
VME3	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
VME4	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
VME5	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
VME6	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx
VME7	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxx

FIGURE 4-9 *Analyzer Setup Screen*

Each of these events can also be renamed. In Figure 4-9,VME2 has been renamed “MyEvent” (See “Rename” on page 63).

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 59), as well as patterns with user-defined labels (See “Manipulating Events” on page 62).

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:AS*). 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.

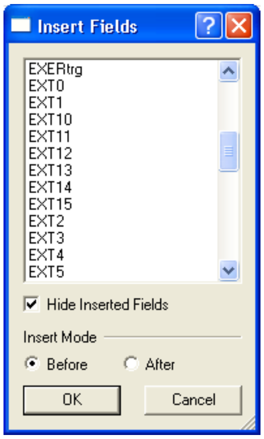
LINK# is either PAD or a number. To specify an number just type it in. This is used when multiple links are used on the same connector. This is not supported by the Vanguard but you may trigger when the Link number changes from PAD to, typically, 0.

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).



Insert Fields

EXERtrg
EXT0
EXT1
EXT10
EXT11
EXT12
EXT13
EXT14
EXT15
EXT2
EXT3
EXT4
EXT5

☑ Hide Inserted Fields

Insert Mode

● Before ○ 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 37 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.

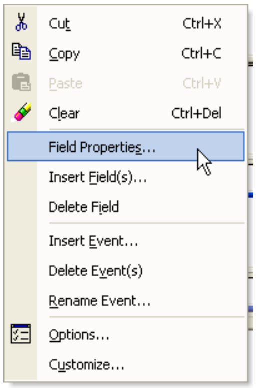


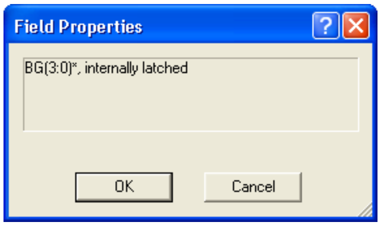
FIGURE 4-10 *Right-click Field menu*

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.



BG(3:0)*, internally latched

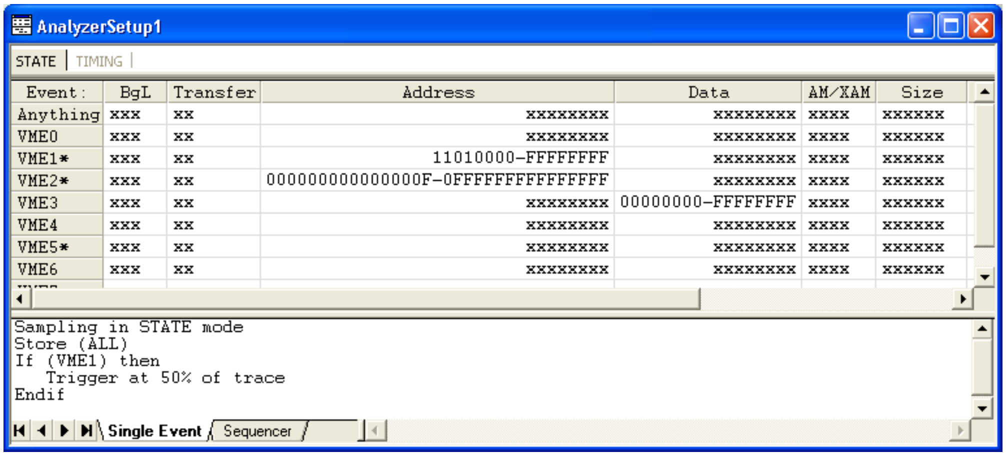
Negating the range is done by typing an exclamation mark(!).

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.

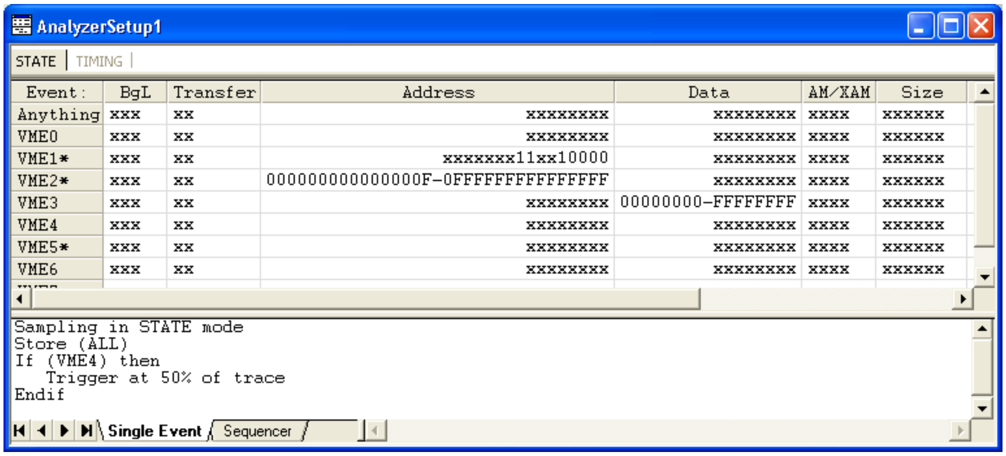


Event:	BgL	Transfer	Address	Data	AM/XAM	Size
Anything	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx
VME0	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx
VME1*	xxx	xx	11010000-FFFFFFFF	xxxxxxxx	xxxx	xxxxxx
VME2*	xxx	xx	000000000000000F-0FFFFFFFFFFFFFFF	xxxxxxxx	xxxx	xxxxxx
VME3	xxx	xx	xxxxxxxx	00000000-FFFFFFFF	xxxx	xxxxxx
VME4	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx
VME5*	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx
VME6	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx

```
Sampling in STATE mode
Store (ALL)
If (VME1) then
   Trigger at 50% of trace
Endif
```

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.



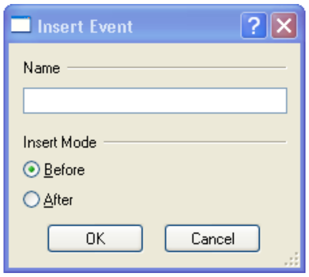
Event:	BgL	Transfer	Address	Data	AM/XAM	Size
Anything	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx
VME0	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx
VME1*	xxx	xx	xxxxxxx11xx10000	xxxxxxxx	xxxx	xxxxxx
VME2*	xxx	xx	000000000000000F-0FFFFFFFFFFFFFF	xxxxxxxx	xxxx	xxxxxx
VME3	xxx	xx	xxxxxxxx	00000000-FFFFFFFF	xxxx	xxxxxx
VME4	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx
VME5*	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx
VME6	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx

```
Sampling in STATE mode
Store (ALL)
If (VME4) then
   Trigger at 50% of trace
Endif
```

FIGURE 4-12 *'Don't Care' binary address range*

Manipulating Events

Add an Event



Insert Event

Name

Insert Mode

● Before

○ After

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 37 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 (VME2) 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 . 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 the 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, but (A+B)*C ≠ 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 ≠ !A+B*C*

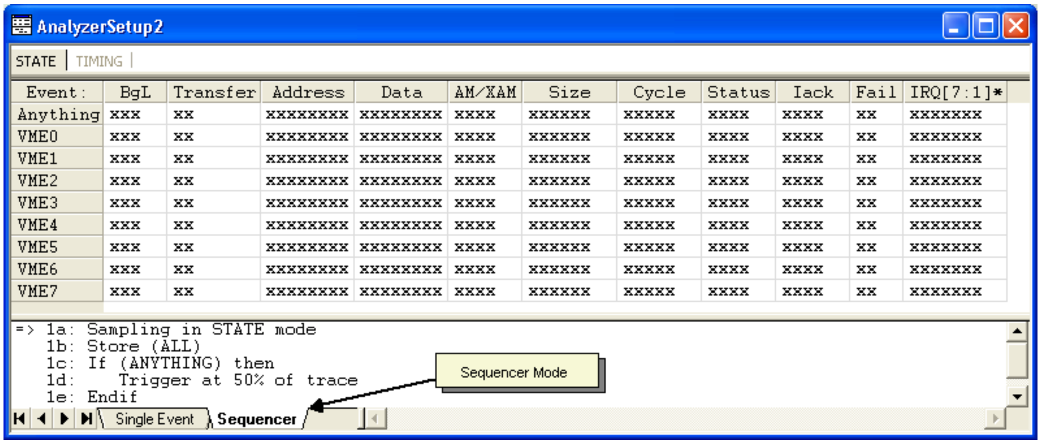
The logical NOT operator can be used on single event names, or on sub-expressions within brackets. The logical NOT operator “!” is always evaluated first.Summary of implicit Actions and Transitions

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.



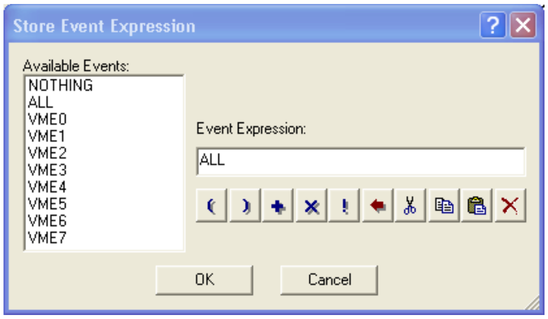
Event:	BgL	Transfer	Address	Data	AM/XAM	Size	Cycle	Status	Iack	Fail	IRQ[7:1]*
Anything	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
VME0	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
VME1	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
VME2	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
VME3	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
VME4	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
VME5	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
VME6	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
VME7	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx

```
=> 1a: Sampling in STATE mode
   1b: Store (ALL)
   1c: If (ANYTHING) then
   1d:    Trigger at 50% of trace
   1e: Endif
```

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.

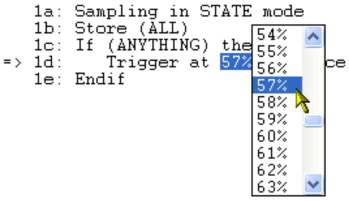


Event Expression:

```
ALL
```

FIGURE 4-14 *Event Expression dialog box*

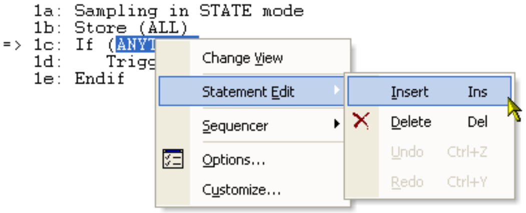
Some parameters, such as the Trigger Position, use a drop-down menu from which to choose values.



```
   1a: Sampling in STATE mode
   1b: Store (ALL)
   1c: If (ANYTHING) the
=> 1d:    Trigger at 57%                ce
   1e: Endif
```

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.

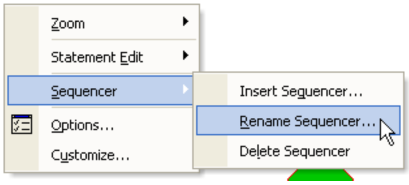


```
   1a: Sampling in STATE mode
   1b: Store (ALL)
=> 1c: If (ANYT
   1d:    Trigg
   1e: Endif
```

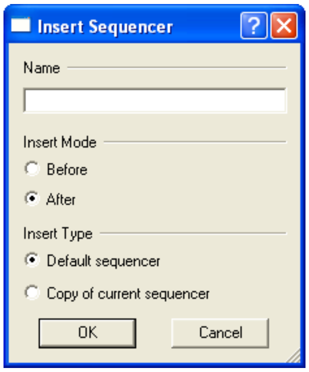
Alternatively press the Insert key on your keyboard.

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.



Insert Sequencer

Name

Insert Mode

○ Before

● After

Insert Type

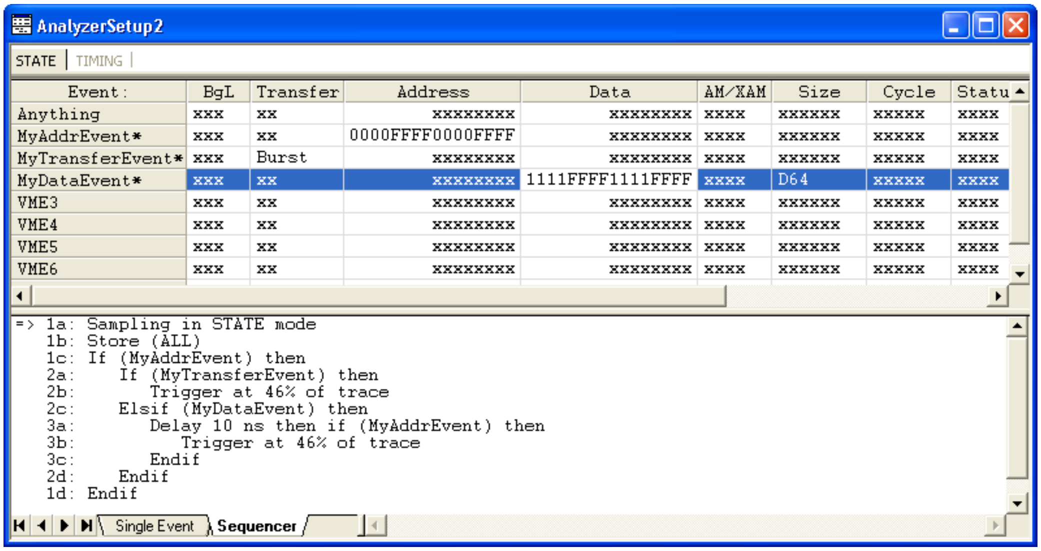
● Default sequencer

○ Copy of current sequencer

Graphical View

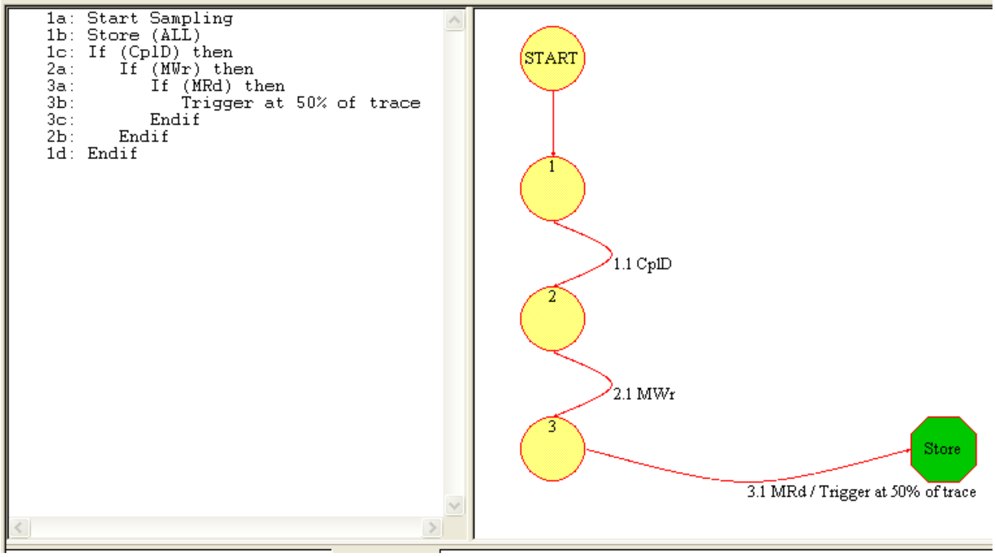
The graphical representation of the sequencer is useful where the sequence of events becomes complex. Each of the icons in the graphical view can be moved around by selecting it and holding down the left mouse button while moving the icon.

Sequencer Example



Event :	BgL	Transfer	Address	Data	AM/XAM	Size	Cycle	Statu
Anything	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx
MyAddrEvent*	xxx	xx	0000FFFF0000FFFF	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx
MyTransferEvent*	xxx	Burst	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx
MyDataEvent*	xxx	xx	xxxxxxxx	1111FFFF1111FFFF	xxxx	D64	xxxxx	xxxx
VME3	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx
VME4	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx
VME5	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx
VME6	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx

```
=> 1a: Sampling in STATE mode
   1b: Store (ALL)
   1c: If (MyAddrEvent) then
   2a:     If (MyTransferEvent) then
   2b:         Trigger at 46% of trace
   2c:     Elsif (MyDataEvent) then
   3a:         Delay 10 ns then if (MyAddrEvent) then
   3b:             Trigger at 46% of trace
   3c:         Endif
   2d:     Endif
   1d: Endif
```

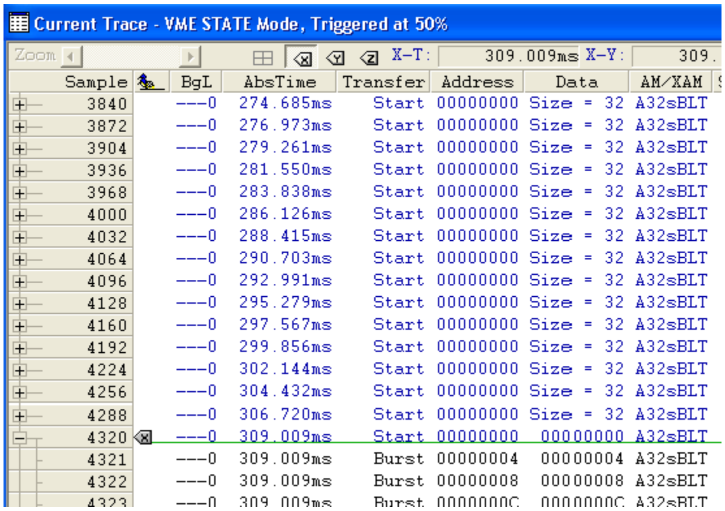


```
1a: Start Sampling
1b: Store (ALL)
1c: If (CplD) then
2a:     If (MWr) then
3a:         If (MRd) then
3b:             Trigger at 50% of trace
3c:         Endif
2b:     Endif
1d: Endif
```

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.

AlphanumericThe Alphanumeric display format is used by default.



Current Trace - VME STATE Mode, Triggered at 50%

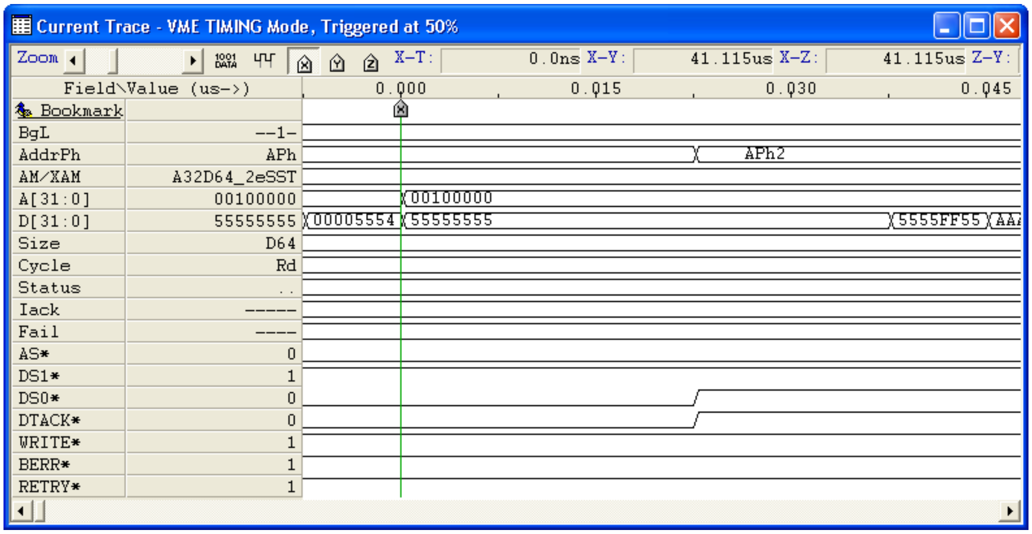
X-T: 309.009ms X-Y: 309.

Sample	BgL	AbsTime	Transfer	Address	Data	AM/XAM
3840	---0	274.685ms	Start	00000000	Size = 32	A32sBLT
3872	---0	276.973ms	Start	00000000	Size = 32	A32sBLT
3904	---0	279.261ms	Start	00000000	Size = 32	A32sBLT
3936	---0	281.550ms	Start	00000000	Size = 32	A32sBLT
3968	---0	283.838ms	Start	00000000	Size = 32	A32sBLT
4000	---0	286.126ms	Start	00000000	Size = 32	A32sBLT
4032	---0	288.415ms	Start	00000000	Size = 32	A32sBLT
4064	---0	290.703ms	Start	00000000	Size = 32	A32sBLT
4096	---0	292.991ms	Start	00000000	Size = 32	A32sBLT
4128	---0	295.279ms	Start	00000000	Size = 32	A32sBLT
4160	---0	297.567ms	Start	00000000	Size = 32	A32sBLT
4192	---0	299.856ms	Start	00000000	Size = 32	A32sBLT
4224	---0	302.144ms	Start	00000000	Size = 32	A32sBLT
4256	---0	304.432ms	Start	00000000	Size = 32	A32sBLT
4288	---0	306.720ms	Start	00000000	Size = 32	A32sBLT
4320	---0	309.009ms	Start	00000000	00000000	A32sBLT
4321	---0	309.009ms	Burst	00000004	00000004	A32sBLT
4322	---0	309.009ms	Burst	00000008	00000008	A32sBLT
4323	---0	309.009ms	Burst	0000000C	0000000C	A32sBLT

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.

WaveformThe 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.



Current Trace - VME TIMING Mode, Triggered at 50%

X-T: 0.0ns X-Y: 41.115us X-Z: 41.115us Z-Y:

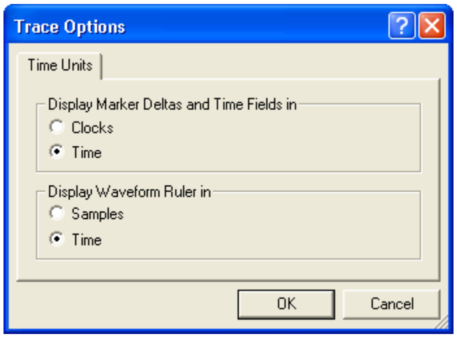
Field\Value (us->)	
Bookmark	
BgL	--1-
AddrPh	APh
AM/XAM	A32D64_2eSST
A[31:0]	00100000
D[31:0]	55555555
Size	D64
Cycle	Rd
Status	. .
Iack	------
Fail	----
AS*	0
DS1*	1
DS0*	0
DTACK*	0
WRITE*	1
BERR*	1
RETRY*	1

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



Trace Options

Time Units

Display Marker Deltas and Time Fields in
○ Clocks
● Time

Display Waveform Ruler in
○ Samples
● Time

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.

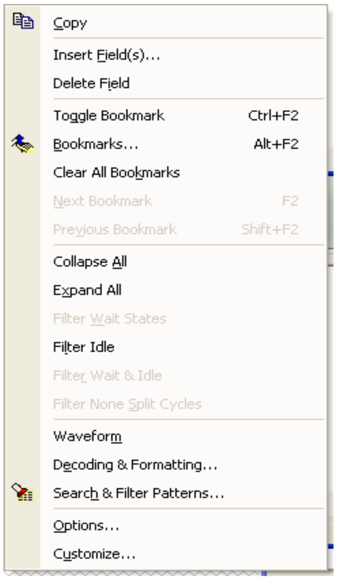


FIGURE 4-16 *Trace Display menu*

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 59).

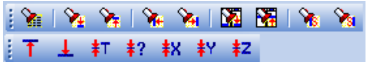
Formatting Options

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

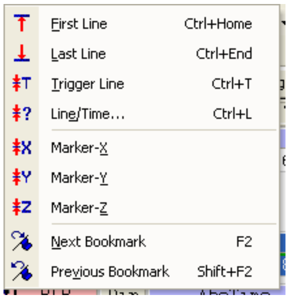
Use the mouse and left click parts of the trace display to select a field.

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

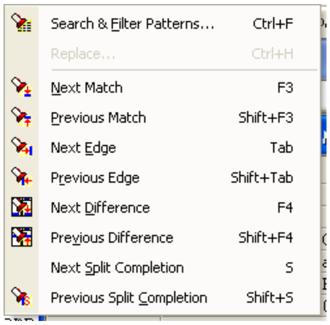
Marker-Z: Jump to the Z marker

Next Bookmark: Jump to the next bookmark. (See “Bookmarks” on page 78)

Previous Bookmark: Jump to the previous bookmark. (See “Bookmarks” on page 78)

The Search tools

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



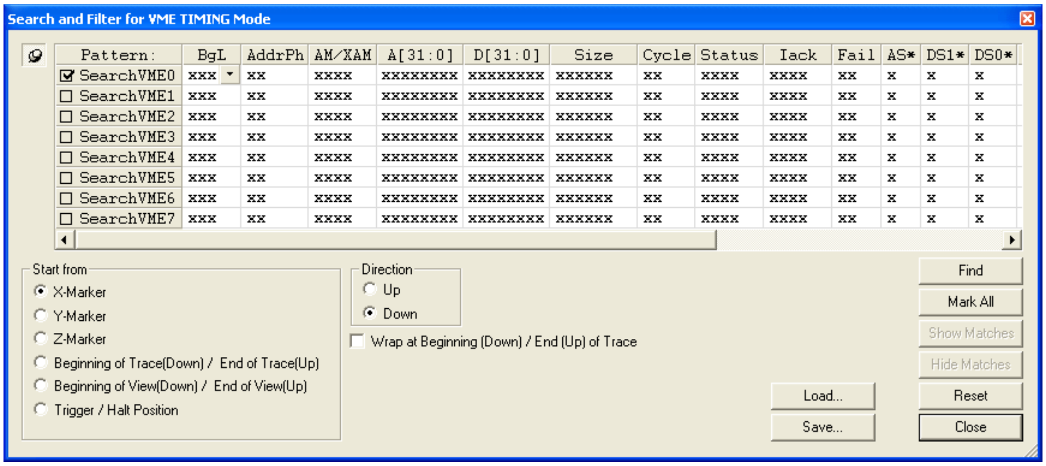
Search locates a particular pattern in the trace buffer. Extract provides a qualified presentation of samples from the trace buffer, so that only samples matching the specified pattern are displayed.

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 58).

Event Searching

This is where you define which patterns to search for.



Search and Filter for VME TIMING Mode

Pattern:	BgL	AddrPh	AM/XAM	A[31:0]	D[31:0]	Size	Cycle	Status	Iack	Fail	AS*	DS1*	DS0*
☑ SearchVME0	xxx	xx	xxxx	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	x	x	x
□ SearchVME1	xxx	xx	xxxx	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	x	x	x
□ SearchVME2	xxx	xx	xxxx	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	x	x	x
□ SearchVME3	xxx	xx	xxxx	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	x	x	x
□ SearchVME4	xxx	xx	xxxx	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	x	x	x
□ SearchVME5	xxx	xx	xxxx	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	x	x	x
□ SearchVME6	xxx	xx	xxxx	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	x	x	x
□ SearchVME7	xxx	xx	xxxx	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	x	x	x

Start from
● X-Marker
○ Y-Marker
○ Z-Marker
○ Beginning of Trace(Down) / End of Trace(Up)
○ Beginning of View(Down) / End of View(Up)
○ Trigger / Halt Position

Direction
○ Up
● Down

□ Wrap at Beginning (Down) / End (Up) of Trace

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.

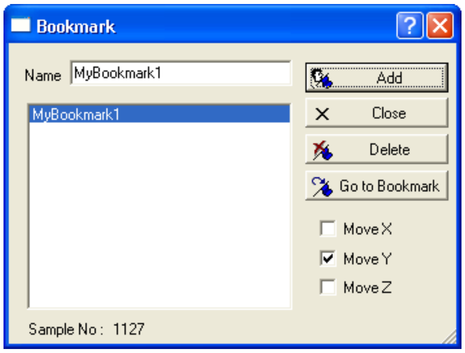
Previous Difference Move to the previous difference between two traces when using Trace Compare.

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.



Name: MyBookmark1

MyBookmark1

□ Move X
☑ Move Y
□ Move Z

Sample No : 1127

BookmarksOpens 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.

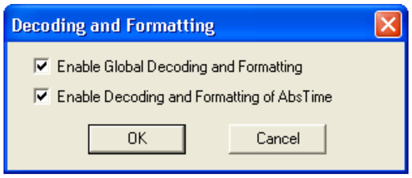
Next Bookmark Jump to the next bookmark.

Previous Bookmark Jump to the previous bookmark.

4.7 Alphanumeric Trace Display

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.



☑ Enable Global Decoding and Formatting

☑ Enable Decoding and Formatting of AbsTime

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

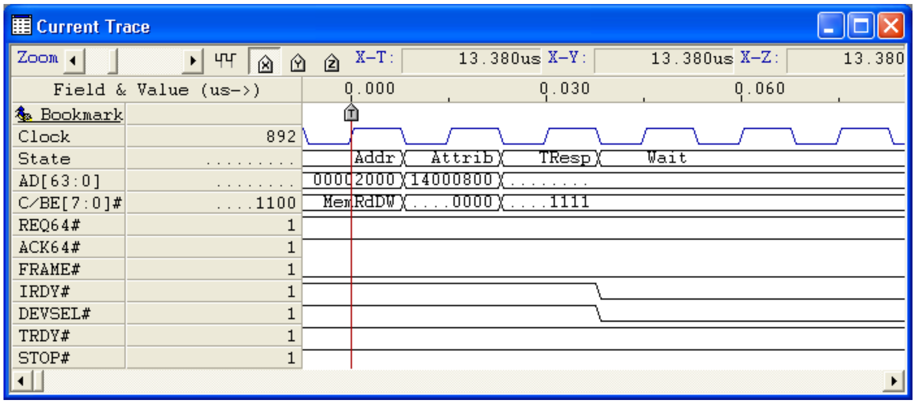
By default, global decoding and formatting is on in Standard and Transfer mode, and off in Clock mode.

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 AS* is set to zero, because there is always traffic when AS* is active.



Field & Value (us->)	
Bookmark	
Clock	892
State	....
AD[63:0]	....
C/BE[7:0]#	1100
REQ64#	1
ACK64#	1
FRAME#	1
IRDY#	1
DEVSEL#	1
TRDY#	1
STOP#	1

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.

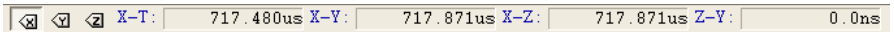
Jump tools

The Jump tools work the same way as in alphanumeric mode (“Jump Tools” on page 75), 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.



X-T: 717.480us X-Y: 717.871us X-Z: 717.871us Z-Y: 0.0ns

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 C:\Documents and Setting ... in the Location text box. The file can also be saved after the editing is complete.

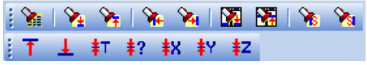
Loading an Analyzer Setup File

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 51

Navigation



Jump Tools

The Search tools

Bookmark options

Trigger

z→Z Zoom in on Waveform

Z→z Zoom out of Waveform

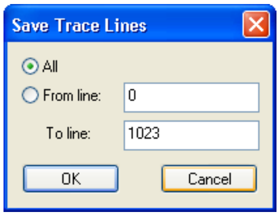
Trace Files

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.



Save Trace Lines

● All
○ From line: 0
To line: 1023

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.

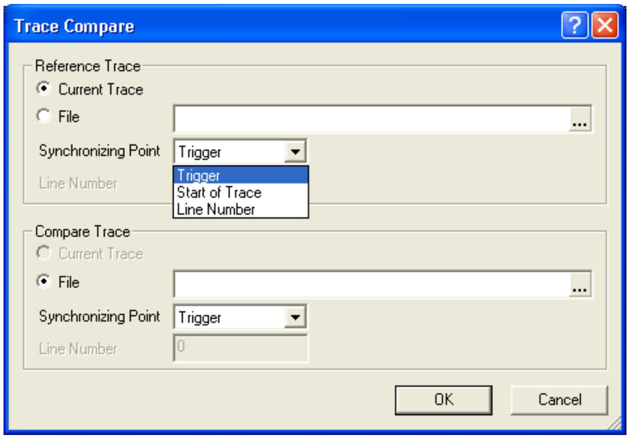
In general, it is reasonable to expect that a full 2M Sample uncompressed trace file will exceed 100MB in size. A temporary file is used to hold the current trace. Trace files are compressed when saved to disk.

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.



Trace Compare

Reference Trace

● Current Trace

○ File ...

Synchronizing Point: Trigger

- Trigger
- Start of Trace
- Line Number

Line Number

Compare Trace

○ Current Trace

● File ...

Synchronizing Point: Trigger

Line Number: 0

OK | Cancel

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:

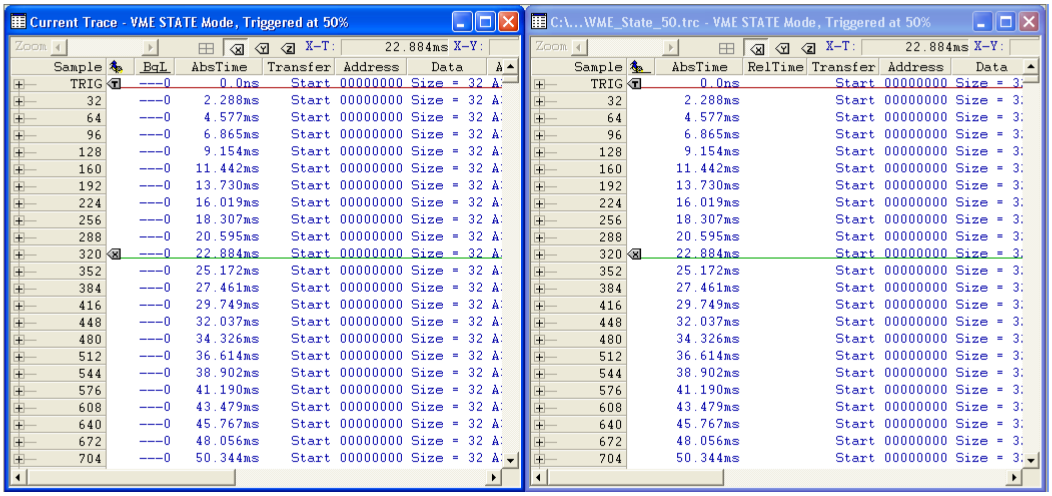
Current Trace Use the trace currently in the Vanguard trace buffer

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.



Current Trace - VME STATE Mode, Triggered at 50%

X-T: 22.884ms

Sample	BgL	AbsTime	Transfer	Address	Data	A
TRIG	---0	0.0ns	Start	00000000	Size = 32	A:
32	---0	2.288ms	Start	00000000	Size = 32	A:
64	---0	4.577ms	Start	00000000	Size = 32	A:
96	---0	6.865ms	Start	00000000	Size = 32	A:
128	---0	9.154ms	Start	00000000	Size = 32	A:
160	---0	11.442ms	Start	00000000	Size = 32	A:
192	---0	13.730ms	Start	00000000	Size = 32	A:
224	---0	16.019ms	Start	00000000	Size = 32	A:
256	---0	18.307ms	Start	00000000	Size = 32	A:
288	---0	20.595ms	Start	00000000	Size = 32	A:
320	---0	22.884ms	Start	00000000	Size = 32	A:
352	---0	25.172ms	Start	00000000	Size = 32	A:
384	---0	27.461ms	Start	00000000	Size = 32	A:
416	---0	29.749ms	Start	00000000	Size = 32	A:
448	---0	32.037ms	Start	00000000	Size = 32	A:
480	---0	34.326ms	Start	00000000	Size = 32	A:
512	---0	36.614ms	Start	00000000	Size = 32	A:
544	---0	38.902ms	Start	00000000	Size = 32	A:
576	---0	41.190ms	Start	00000000	Size = 32	A:
608	---0	43.479ms	Start	00000000	Size = 32	A:
640	---0	45.767ms	Start	00000000	Size = 32	A:
672	---0	48.056ms	Start	00000000	Size = 32	A:
704	---0	50.344ms	Start	00000000	Size = 32	A:

C:\...\VME_State_50.trc - VME STATE Mode, Triggered at 50%

X-T: 22.884ms

Sample	AbsTime	RelTime	Transfer	Address	Data
TRIG	0.0ns		Start	00000000	Size = 3:
32	2.288ms		Start	00000000	Size = 3:
64	4.577ms		Start	00000000	Size = 3:
96	6.865ms		Start	00000000	Size = 3:
128	9.154ms		Start	00000000	Size = 3:
160	11.442ms		Start	00000000	Size = 3:
192	13.730ms		Start	00000000	Size = 3:
224	16.019ms		Start	00000000	Size = 3:
256	18.307ms		Start	00000000	Size = 3:
288	20.595ms		Start	00000000	Size = 3:
320	22.884ms		Start	00000000	Size = 3:
352	25.172ms		Start	00000000	Size = 3:
384	27.461ms		Start	00000000	Size = 3:
416	29.749ms		Start	00000000	Size = 3:
448	32.037ms		Start	00000000	Size = 3:
480	34.326ms		Start	00000000	Size = 3:
512	36.614ms		Start	00000000	Size = 3:
544	38.902ms		Start	00000000	Size = 3:
576	41.190ms		Start	00000000	Size = 3:
608	43.479ms		Start	00000000	Size = 3:
640	45.767ms		Start	00000000	Size = 3:
672	48.056ms		Start	00000000	Size = 3:
704	50.344ms		Start	00000000	Size = 3:

FIGURE 4-19 *Trace Compare Example*

The highlighting can only be turned off by closing one of the compared trace files.

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.

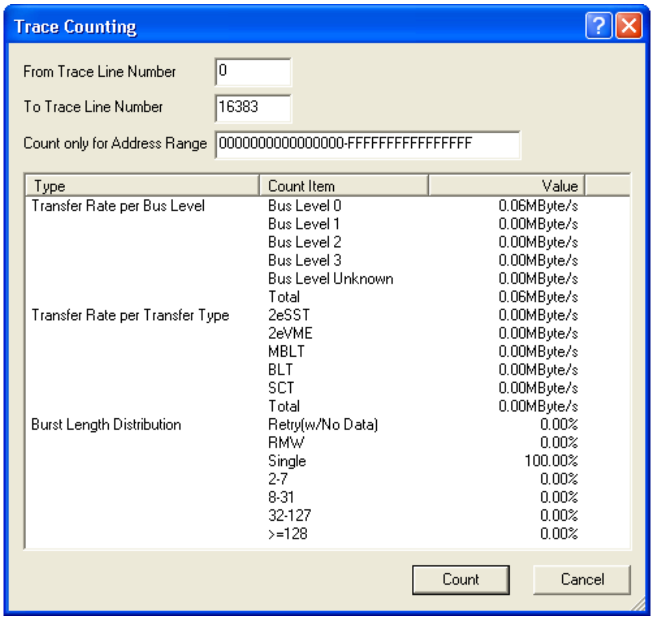
From Trace Line Number: Trace Line from which counting should commence.

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 96 with one exception, the ‘Transfer Rate’. In this case, Transfer Rate shown is for the whole bus.

FIGURE 4-20 *Trace Counting*



From Trace Line Number: 0

To Trace Line Number: 16383

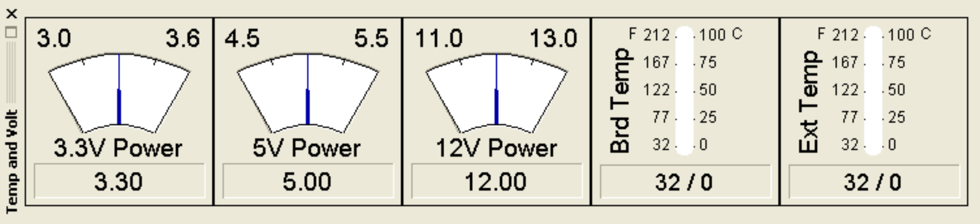
Count only for Address Range: 0000000000000000-FFFFFFFFFFFFFFFF

Type	Count Item	Value
Transfer Rate per Bus Level	Bus Level 0	0.06MByte/s
	Bus Level 1	0.00MByte/s
	Bus Level 2	0.00MByte/s
	Bus Level 3	0.00MByte/s
	Bus Level Unknown	0.00MByte/s
	Total	0.06MByte/s
Transfer Rate per Transfer Type	2eSST	0.00MByte/s
	2eVME	0.00MByte/s
	MBLT	0.00MByte/s
	BLT	0.00MByte/s
	SCT	0.00MByte/s
	Total	0.00MByte/s
Burst Length Distribution	Retry(w/No Data)	0.00%
	RMW	0.00%
	Single	100.00%
	2-7	0.00%
	8-31	0.00%
	32-127	0.00%
	>=128	0.00%

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.

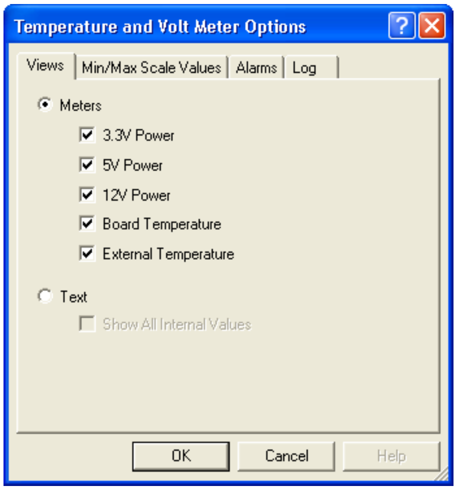
The voltage measured is dependent on how the power jumpers are installed.



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.



Temperature and Volt Meter Options

Views | Min/Max Scale Values | Alarms | Log

● Meters

☑ 3.3V Power

☑ 5V Power

☑ 12V Power

☑ Board Temperature

☑ External Temperature

○ Text

□ Show All Internal 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

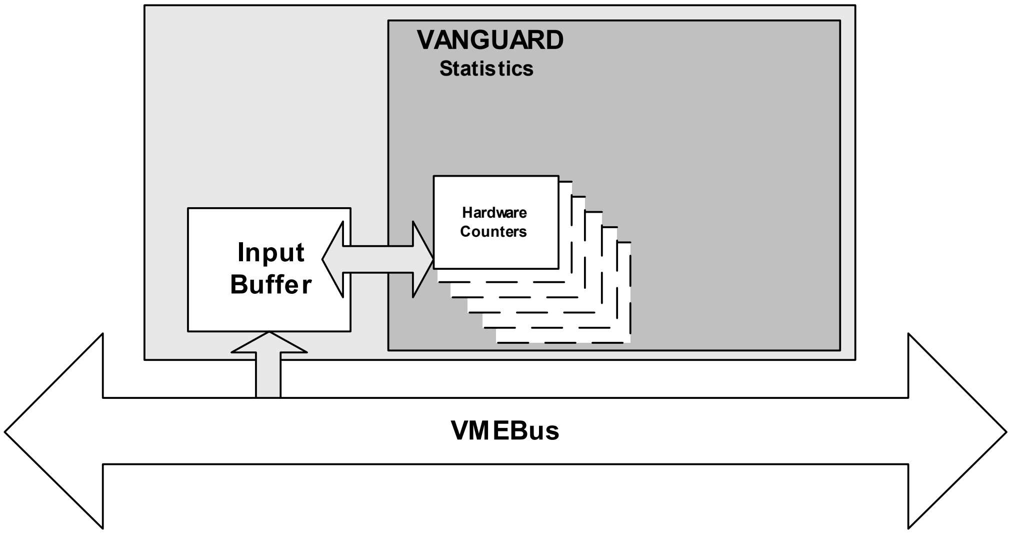


FIGURE 5-1 *Block diagram of the Statistics module*

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.

- Interrupt Distribution
- Interrupt Service Time Distribution
- Arbiter Latency

Pre-defined Statistics for VME

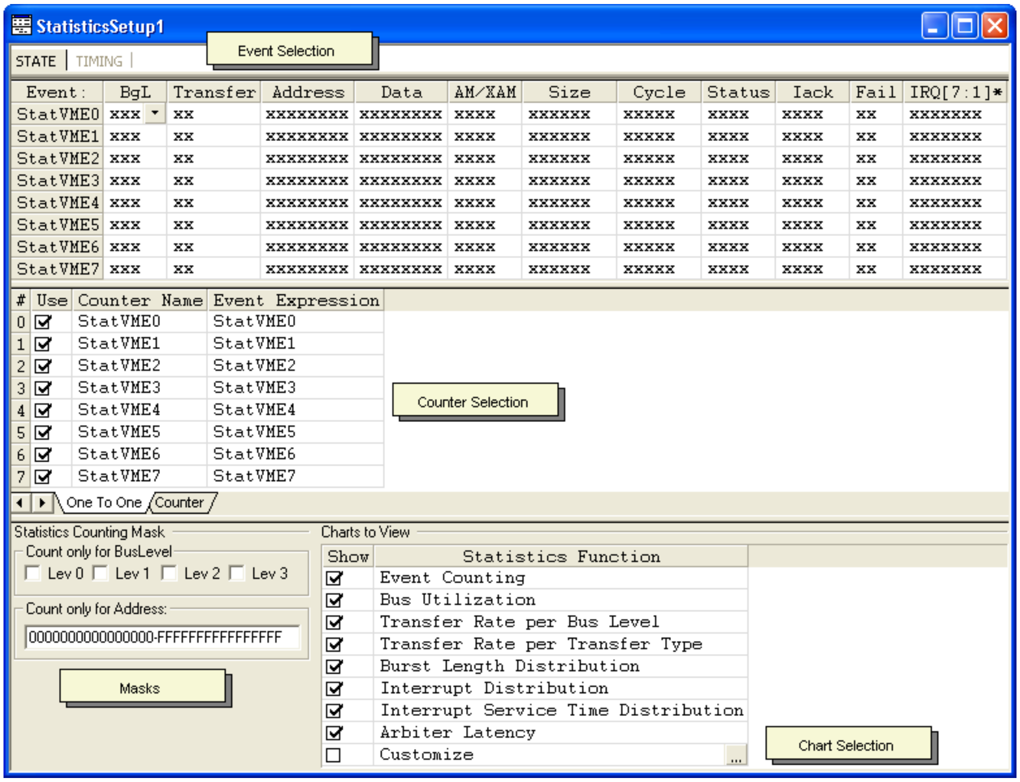
- Bus Utilization - Displays parameters that measure bus traffic performance.
- Transfer Rate - Gives a measurement in MTransfers/Sec and MB/Sec according to Bus Level and Transfer type.
- Block Length Distribution - Counts the number of bytes in each transaction, and displays them according to block length.
- Interrupt Distribution - Displays how often interrupts are occurring.
- Interrupt Service Time Distribution - Measures the time taken for each interrupt to be serviced by the system.
- Arbiter Latency - Counts the average number of clock cycles from REQ# asserted to GNT# asserted.

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:



Event:	BgL	Transfer	Address	Data	AM/XAM	Size	Cycle	Status	Iack	Fail	IRQ[7:1]*
StatVME0	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
StatVME1	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
StatVME2	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
StatVME3	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
StatVME4	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
StatVME5	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
StatVME6	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx
StatVME7	xxx	xx	xxxxxxxx	xxxxxxxx	xxxx	xxxxxx	xxxxx	xxxx	xxxx	xx	xxxxxxx

#	Use	Counter Name	Event Expression
0	☑	StatVME0	StatVME0
1	☑	StatVME1	StatVME1
2	☑	StatVME2	StatVME2
3	☑	StatVME3	StatVME3
4	☑	StatVME4	StatVME4
5	☑	StatVME5	StatVME5
6	☑	StatVME6	StatVME6
7	☑	StatVME7	StatVME7

Statistics Counting Mask

Count only for BusLevel

□ Lev 0 □ Lev 1 □ Lev 2 □ Lev 3

Count only for Address:

0000000000000000-FFFFFFFFFFFFFFFF

Charts to View

Show	Statistics Function
☑	Event Counting
☑	Bus Utilization
☑	Transfer Rate per Bus Level
☑	Transfer Rate per Transfer Type
☑	Burst Length Distribution
☑	Interrupt Distribution
☑	Interrupt Service Time Distribution
☑	Arbiter Latency
□	Customize ...

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 58). 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 37 for details.

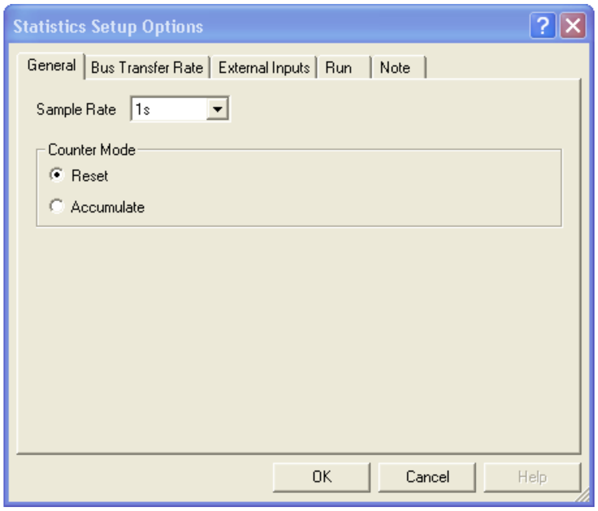
Statistics Counting Masks: Used to reject certain results from the statistics.

- **Count Only for BusLevel** - Select BusLevel to count.
- **Count only for Address** - Specify Address range to count.

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.



Sample Rate 1s

Counter Mode

● Reset

○ Accumulate

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.

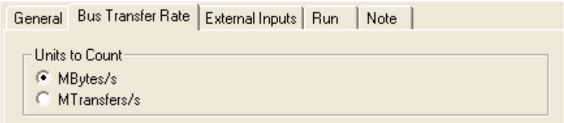
Selection of the Accumulate versus Reset mode is typically driven by the total number of samples to be observed in the measurement. Measurements made in Clock Sampling mode typically require the use of the Accumulate mode to yield significant results because the counters reach terminal count very rapidly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

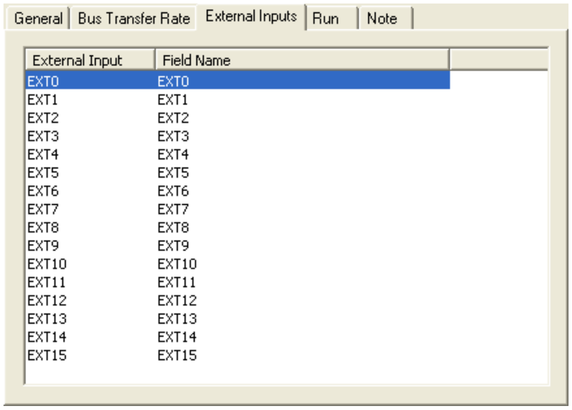


Units to Count
● MBytes/s
○ MTransfers/s

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.

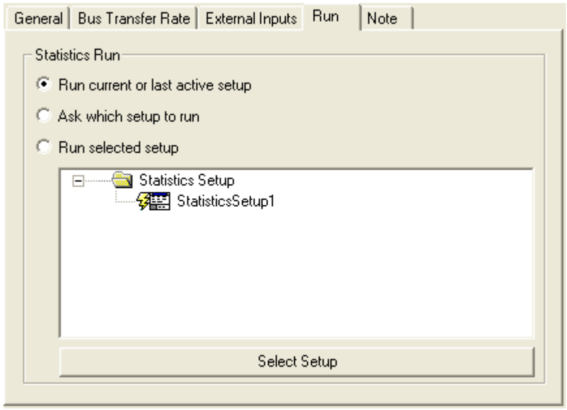
External Inputs



External Input	Field Name
EXT0	EXT0
EXT1	EXT1
EXT2	EXT2
EXT3	EXT3
EXT4	EXT4
EXT5	EXT5
EXT6	EXT6
EXT7	EXT7
EXT8	EXT8
EXT9	EXT9
EXT10	EXT10
EXT11	EXT11
EXT12	EXT12
EXT13	EXT13
EXT14	EXT14
EXT15	EXT15

The Field Name can be changed on this menu by double clicking on the Field Name. See “External Inputs” on page 16 for information about the external inputs.

Statistics Run



Statistics Run

● Run current or last active setup

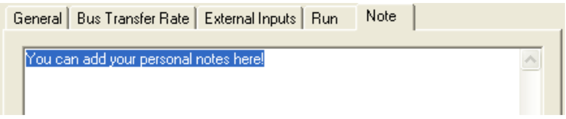
○ Ask which setup to run

○ Run selected setup

Statistics Setup
StatisticsSetup1

- 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



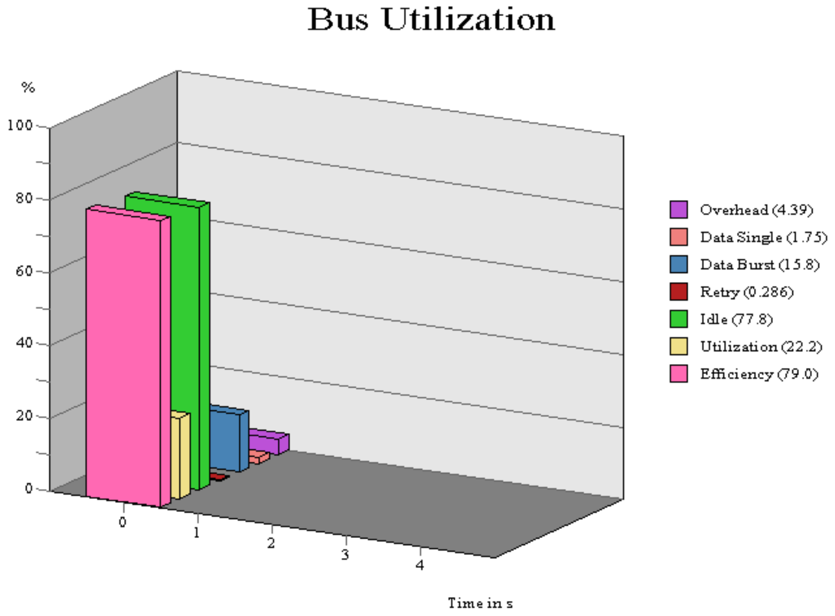
You can add your personal notes here!

- 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 block transfers relative to the total time, i.e. how much time is spent transferring single data across the bus.

DataBlock- The Data Block column measure the duration of block data transfers relative to the total time, i.e. how much time is spent transferring block data across the bus.

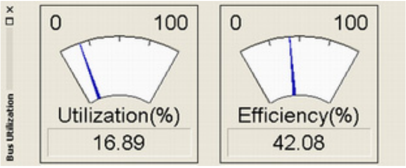
Retries - Number of Retry and BusError cycles on the bus relative to the total time.

Idle - Number of Idle cycles relative to the total time. It is calculated by dividing the number of samples with NOT (AS* OR DTACK* OR DS1* OR DS0* or BERR* OR RETRY*).

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.

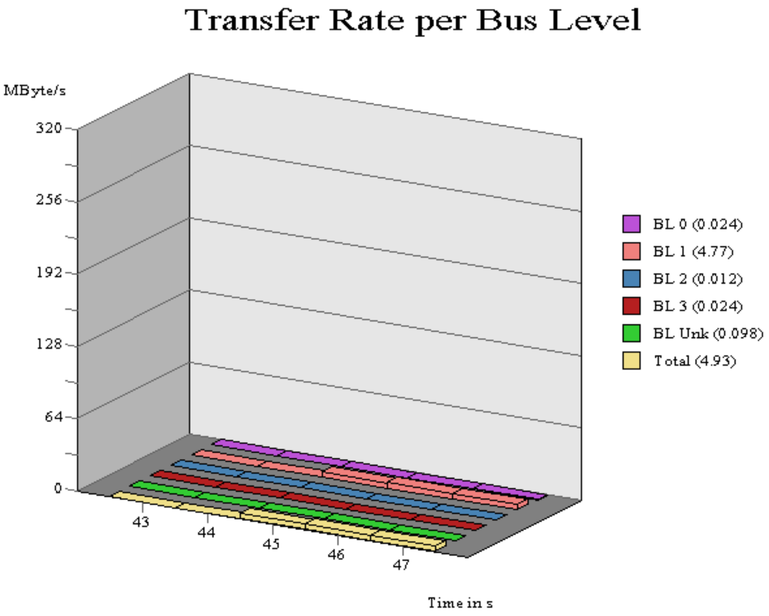


Utilization(%)
16.89

Efficiency(%)
42.08

Transfer Rate per Bus Level

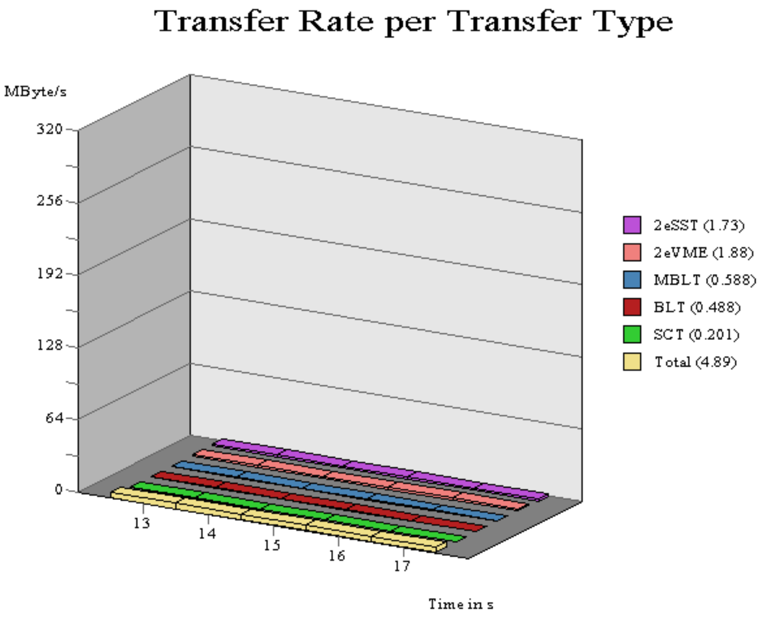
The Transfer Rate statistics calculates the transfer rate in MTransfers/Sec and MB/Sec.



The *total*-bar will always display the sum of all the Bus Levels. It is recommended to run this mode over some time. This will give a more representative average of the systems transfer rates.

Transfer Rate per Transfer Type

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



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

Block Length Distribution

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

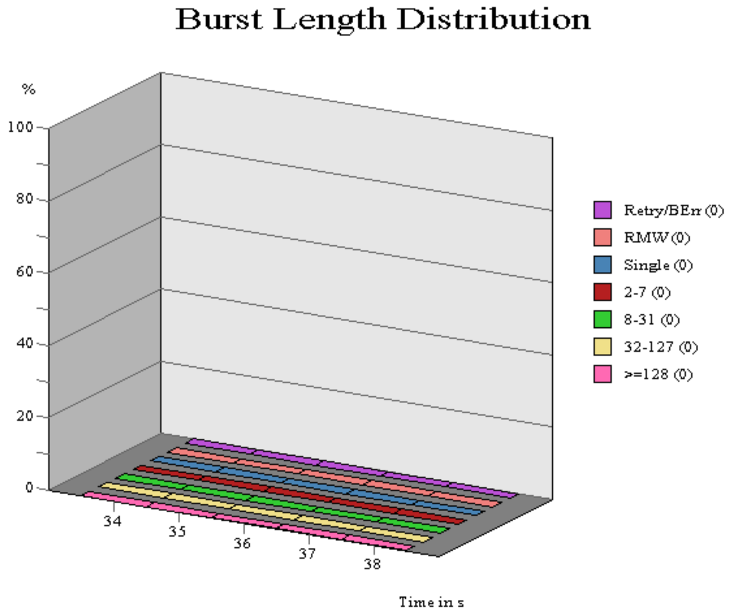


TABLE 5-1. **Block Length Distribution explanation**

Abbreviation	**Block length/Termination**
Retry/BErr	Target Retry, i.e. no data transferred
RMW	Read Modify Write data transfers
Single	Single Cycles, i.e. one data phase
2-7	Block length from 2 to 7
8-31	Block length from 8 to 31
32-63	Block length from 32 to 63
64-127	Block length from 64 to 127
>=128	Block length longer than or equal to 128

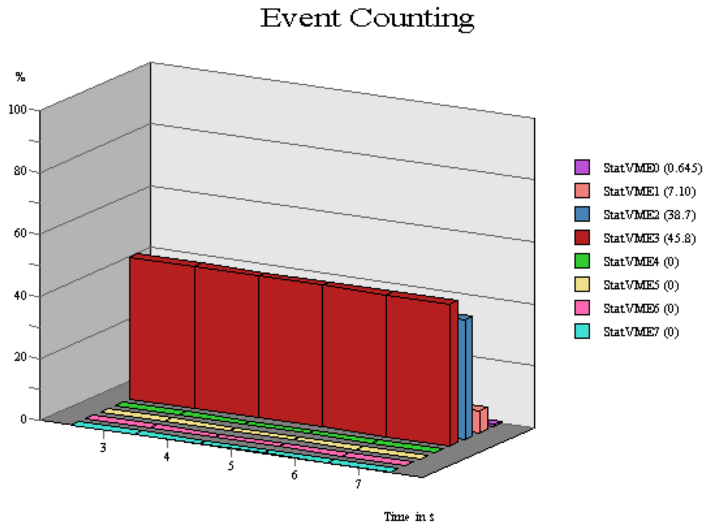
5.3 User-defined Statistics

Event Counting

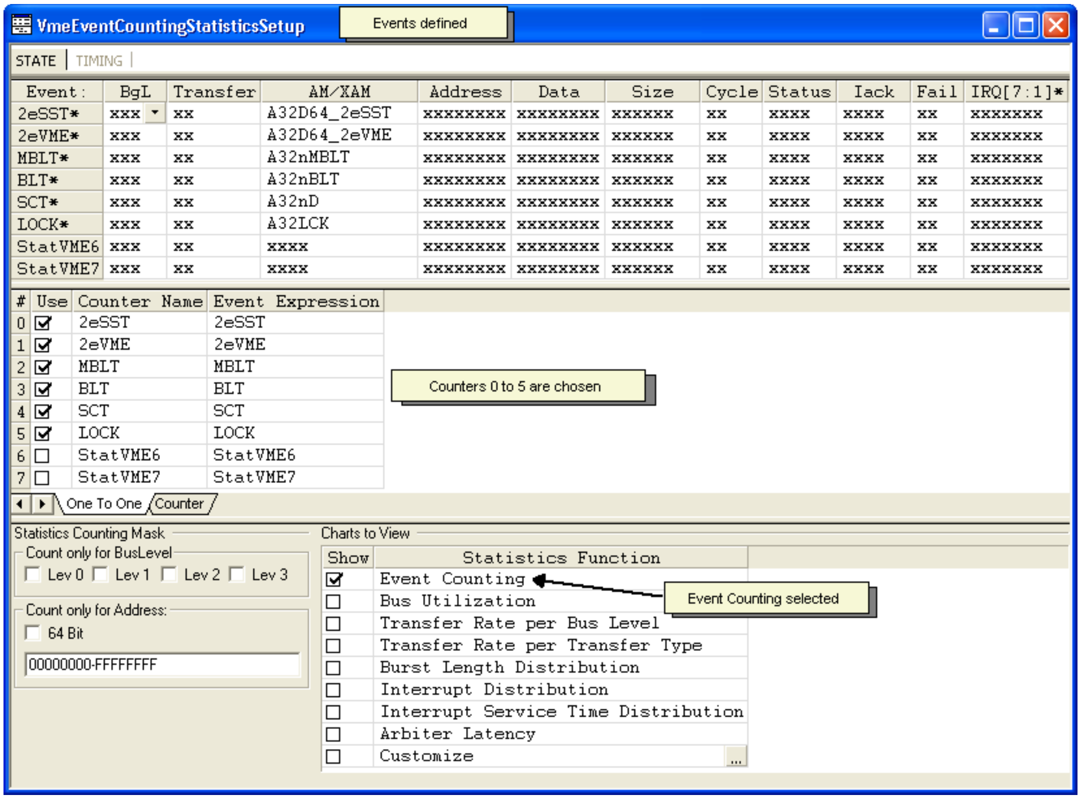
The Event Counting statistic can be used in all sampling modes. Event Counting is useful for counting the occurrences of different types of cycles with a specific AM code, like A32D64,2eVME etc.

Note – The Analyzer and the Statistics must be use the same sampling mode.

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 58).



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.



VmeEventCountingStatisticsSetup

Events defined

STATE | TIMING

Event:	BgL	Transfer	AM/XAM	Address	Data	Size	Cycle	Status	Iack	Fail	IRQ[7:1]*
2eSST*	xxx	xx	A32D64_2eSST	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	xxxxxxx
2eVME*	xxx	xx	A32D64_2eVME	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	xxxxxxx
MBLT*	xxx	xx	A32nMBLT	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	xxxxxxx
BLT*	xxx	xx	A32nBLT	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	xxxxxxx
SCT*	xxx	xx	A32nD	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	xxxxxxx
LOCK*	xxx	xx	A32LCK	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	xxxxxxx
StatVME6	xxx	xx	xxxx	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	xxxxxxx
StatVME7	xxx	xx	xxxx	xxxxxxxx	xxxxxxxx	xxxxxx	xx	xxxx	xxxx	xx	xxxxxxx

#	Use	Counter Name	Event Expression
0	☑	2eSST	2eSST
1	☑	2eVME	2eVME
2	☑	MBLT	MBLT
3	☑	BLT	BLT
4	☑	SCT	SCT
5	☑	LOCK	LOCK
6	□	StatVME6	StatVME6
7	□	StatVME7	StatVME7

Counters 0 to 5 are chosen

Statistics Counting Mask

Count only for BusLevel: □ Lev 0 □ Lev 1 □ Lev 2 □ Lev 3

Count only for Address: □ 64 Bit

00000000-FFFFFFFF

Charts to View

Show	Statistics Function
☑	Event Counting
□	Bus Utilization
□	Transfer Rate per Bus Level
□	Transfer Rate per Transfer Type
□	Burst Length Distribution
□	Interrupt Distribution
□	Interrupt Service Time Distribution
□	Arbiter Latency
□	Customize ...

Event Counting selected

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

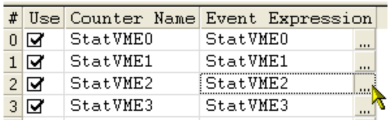
In One to One mode, each counter is represented by one event.

Counter 0 shows that we are using the Event called 2eSST, which has its event pattern defined as AM=A32D64_2eSST. Counter 0 will therefore count A32D64_2eSST 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 58).

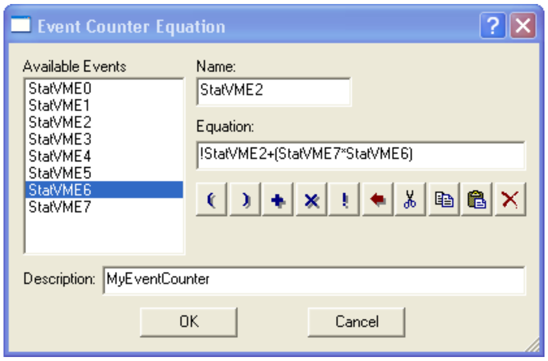
Counter mode

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



#	Use	Counter Name	Event Expression
0	☑	StatVME0	StatVME0 …
1	☑	StatVME1	StatVME1 …
2	☑	StatVME2	StatVME2 …
3	☑	StatVME3	StatVME3 …

FIGURE 5-5 *Edit an Event Counter Equation*



Event Counter Equation

Available Events:
StatVME0
StatVME1
StatVME2
StatVME3
StatVME4
StatVME5
StatVME6
StatVME7

Name: StatVME2

Equation: !StatVME2+(StatVME7*StatVME6)

Description: MyEventCounter

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

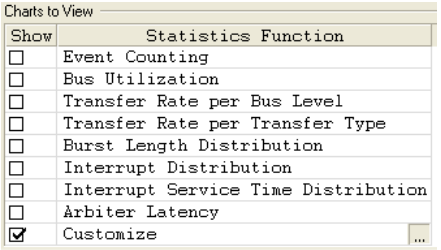
! Logical NOT

← Erase

✂ ⧉ 📋 ✕ Cut, Copy, Paste and Delete.

Customized Charts

Customized charts can be made using one or more of the hardware counters. (See “Summary of Hardware Counters” on page 140 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.



Charts to View

Show	Statistics Function
□	Event Counting
□	Bus Utilization
□	Transfer Rate per Bus Level
□	Transfer Rate per Transfer Type
□	Burst Length Distribution
□	Interrupt Distribution
□	Interrupt Service Time Distribution
□	Arbiter Latency
☑	Customize ...

Adding, Deleting and Renaming Customized Charts.

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

Insert a new chart

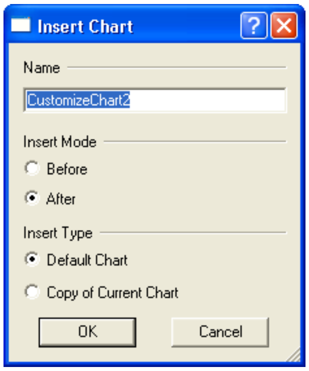
Note – When inserting a new Customize function using the right click menu, the mouse cursor must be below the Wait States function. It is not possible to insert a customized chart between the pre-defined statistics functions.

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.



Insert Chart

Name: CustomizeChart2

Insert Mode
○ Before
● After

Insert Type
● Default Chart
○ Copy of Current Chart

Delete a chart

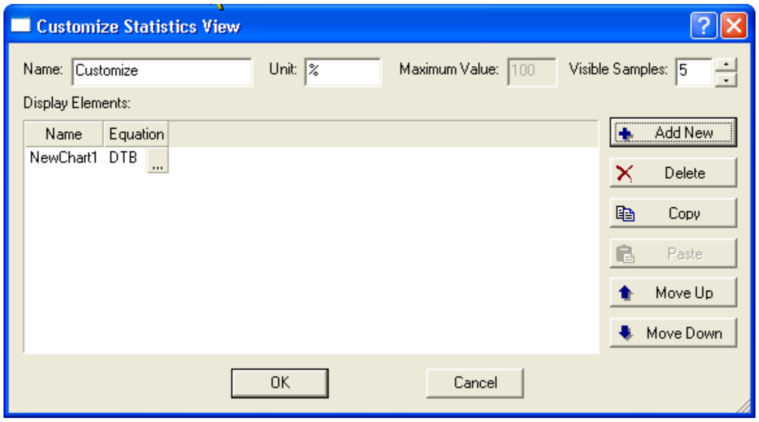
To delete a customized chart, right click on the chart you wish to delete and click Delete Chart from the menu that appears.

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.



Name: Customize Unit: % Maximum Value: 100 Visible Samples: 5

Display Elements:

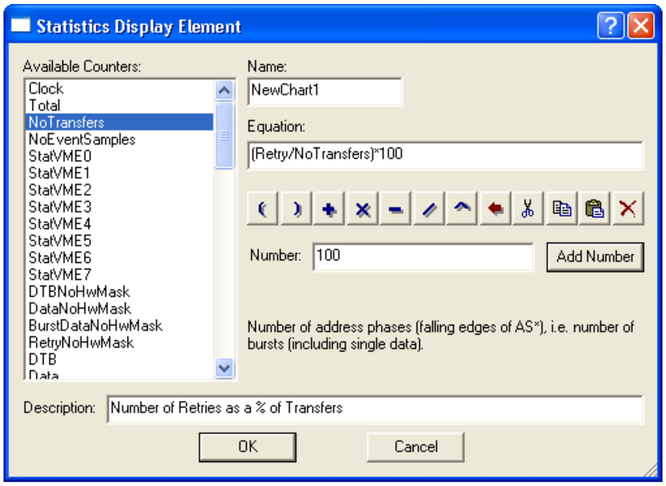
Name	Equation
NewChart1	DTB

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 DTB 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.



Available Counters:

Clock
Total
NoTransfers
NoEventSamples
StatVME0
StatVME1
StatVME2
StatVME3
StatVME4
StatVME5
StatVME6
StatVME7
DTBNoHwMask
DataNoHwMask
BurstDataNoHwMask
RetryNoHwMask
DTB
Data

Name: NewChart1

Equation: (Retry/NoTransfers)*100

Number: 100

Number of address phases (falling edges of AS*), i.e. number of bursts (including single data).

Description: Number of Retries as a % of Transfers

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:

(Retry/NoTransfers)*100

This example will display a chart showing the percentage of Retries 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 5-2 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 5-2. Summary of Hardware Counters

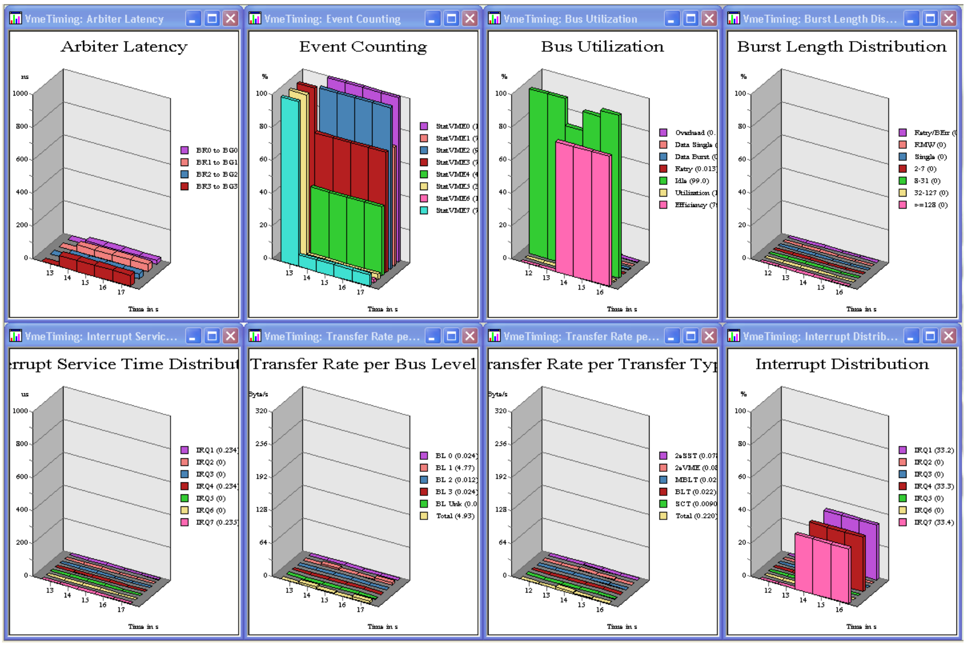
Clock	The Timetag and Timing sampling clock period in pico-seconds (ps).
Total	Total number of clocks per statistics sample.
NoTransfers	Number of address phases (falling edges of AS*), i.e. number of blocks including singla data.
NoEventSamples	Number of samples in trace using the selected sampling mode and its options.
StatVME0	Event expression counter 0. Name specified in the counter expression.
StatVME1	Event expression counter 1. Name specified in the counter expression.
StatVME2	Event expression counter 2. Name specified in the counter expression.
StatVME3	Event expression counter 3. Name specified in the counter expression.
StatVME4	Event expression counter 4. Name specified in the counter expression.
StatVME5	Event expression counter 5. Name specified in the counter expression.
StatVME6	Event expression counter 6. Name specified in the counter expression.
StatVME7	Event expression counter 7. Name specified in the counter expression.
DTBNoHwMask	Total DTB. NOTE: This counter ignores the hardware mask.
DataNoHwMask	Number of clocks with data. NOTE: This counter ignores the hardware mask.
BlockDataNoHwMask	Number of clocks with block data. NOTE: This counter ignores the hardware mask.
RetryNoHwMask	Number of clocks with RETRY* (or RESP*) active. NOTE: This counter ignores the hardware mask.
DTB	Total DTB.
Data	Number of clocks with data.
BlockData	Number of clocks with block data.
Retry	Number of clocks with RETRY* (or RESP*) active.
AStoDTACK	Number of clocks between AS* and DTACK*
BlockLenGT128	Number of transfers with block length greater than or equal to 128 data cycles.
BlockLen32_127	Number of transfers with block length between 32 and 127 data cycles.
BlockLen8_31	Number of transfers with block length between 8 and 31 data cycles.
BlockLen2_7	Number of transfers with block length between 2 and 7 data cycles.
BlockLenSingle	Number of single data transfers.
BlockLenNull	Number of transfers with no data (Retry, Suspend or Terminate).
RMW	Number of transfers with RMW cycles.
SlaveSuspend	Number of slave suspended transfers.
SlaveTerminate	Number of slave terminated transfers.
BusError	Number of bus error terminated transfers.
BusXferRate2ESST	Transfer rate using 2eSST. Note: The counter ignores the Bus Level hardware mask.
BusXferRate2EVME	Transfer rate using 2eVME. Note: The counter ignores the Bus Level hardware mask.
BusXferRateMBLT	Transfer rate using MBLT. Note: The counter ignores the Bus Level hardware mask.
BusXferRateBLT	Transfer rate using BLT. Note: The counter ignores the Bus Level hardware mask.
BusXferRateSCT	Transfer rate using SCT. Note: The counter ignores the Bus Level hardware mask.

TABLE 5-2. Summary of Hardware Counters (Continued)

BusXferRateBL0	Transfer rate for Bus Level 0. Note: The counter ignores the Bus Level hardware mask.
BusXferRateBL1	Transfer rate for Bus Level 1. Note: The counter ignores the Bus Level hardware mask.
BusXferRateBL2	Transfer rate for Bus Level 2. Note: The counter ignores the Bus Level hardware mask.
BusXferRateBL3	Transfer rate for Bus Level 3. Note: The counter ignores the Bus Level hardware mask.
BusXferRateUnknown	Transfer rate for unknown Bus Levels. Note: The counter ignores the Bus Level hardware mask.
IntacksOnIRQ1	Number of interrupt acknowledges on IRQ1.
IntacksOnIRQ2	Number of interrupt acknowledges on IRQ2.
IntacksOnIRQ3	Number of interrupt acknowledges on IRQ3.
IntacksOnIRQ4	Number of interrupt acknowledges on IRQ4.
IntacksOnIRQ5	Number of interrupt acknowledges on IRQ5.
IntacksOnIRQ6	Number of interrupt acknowledges on IRQ6.
IntacksOnIRQ7	Number of interrupt acknowledges on IRQ7.
IRQ1Clocks	Number of clocks IRQ1 is active.
IRQ2Clocks	Number of clocks IRQ2 is active.
IRQ3Clocks	Number of clocks IRQ3 is active.
IRQ4Clocks	Number of clocks IRQ4 is active.
IRQ5Clocks	Number of clocks IRQ5 is active.
IRQ6Clocks	Number of clocks IRQ6 is active.
IRQ7Clocks	Number of clocks IRQ7 is active.
BusRequest0	Number of bus requests 0.
BusRequest1	Number of bus requests 1.
BusRequest2	Number of bus requests 2.
BusRequest3	Number of bus requests 3.
BR0toGNT0	Number of clocks from bus request 0 to grant 0.
BR1toGNT1	Number of clocks from bus request 0 to grant 1.
BR2toGNT2	Number of clocks from bus request 0 to grant 2.
BR3toGNT3	Number of clocks from bus request 0 to grant 3.
NoBlocks	Number of address phases with a complete block.

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

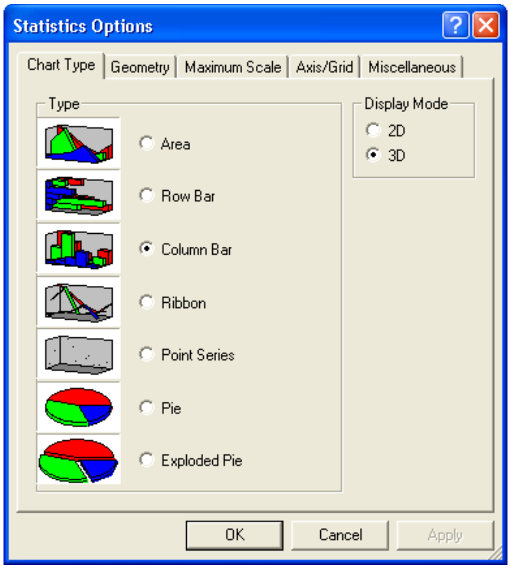

Statistics Options

Chart Type | Geometry | Maximum Scale | Axis/Grid | Miscellaneous

Type
- ○ Area
- ○ Row Bar
- ● Column Bar
- ○ Ribbon
- ○ Point Series
- ○ Pie
- ○ Exploded Pie

Display Mode
- ○ 2D
- ● 3D

Chart Type and Display Mode

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

Geometry

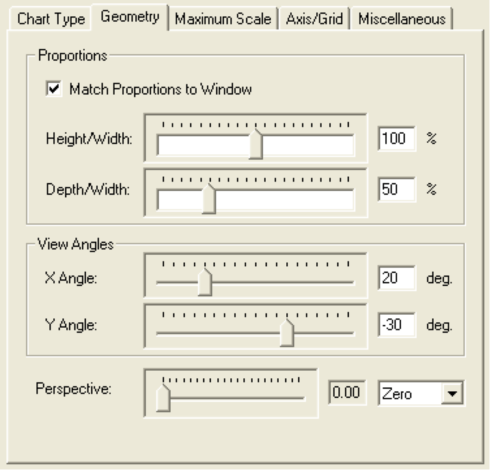


Chart Type | Geometry | Maximum Scale | Axis/Grid | Miscellaneous

Proportions

☑ Match Proportions to Window

Height/Width: 100 %

Depth/Width: 50 %

View Angles

X Angle: 20 deg.

Y Angle: -30 deg.

Perspective: 0.00 Zero

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

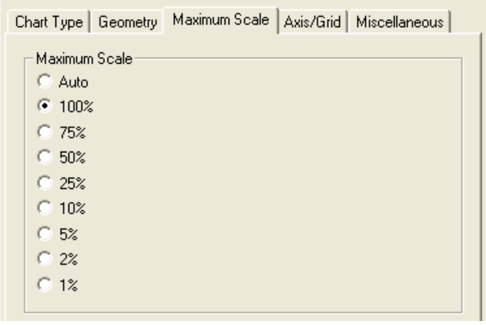
View Angles

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

Perspective

- Adjusts the perspective view of the chart.

Maximum Scale



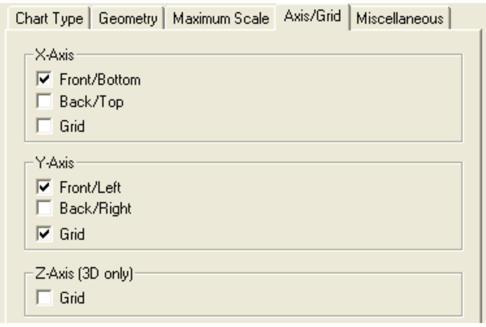
Maximum Scale

○ Auto
● 100%
○ 75%
○ 50%
○ 25%
○ 10%
○ 5%
○ 2%
○ 1%

- 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
☐ Back/Top
☐ Grid

Y-Axis

☑ Front/Left
☐ Back/Right
☑ Grid

Z-Axis (3D only)

☐ 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.

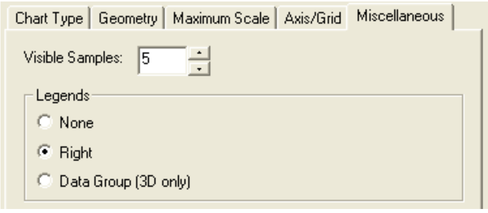
Y Axis

- 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: 5

Legends
○ None
● Right
○ Data Group (3D only)

Visible Samples

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

Legends

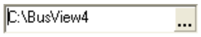
- The position where chart legends should be shown.

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.



C:\BusView4 ...

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.

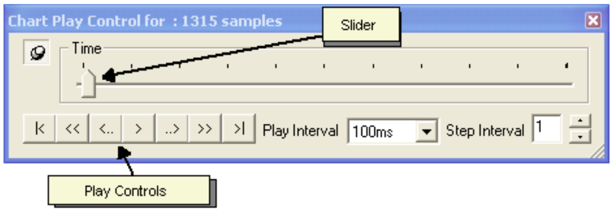


Chart Play Control for : 1315 samples

Play Interval 100ms

Step Interval 1

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 VME specification. This chapter explains the use of the Protocol Checker, and describes each violation in detail.

- Introduction
- Operation
- VME Protocol Violations

Use of the Protocol Checker requires the “VG-VP” license to have been purchased. See “Ordering Information” on page 313.

6.1 Introduction

The Vanguard Protocol Checker is a parallel trigger module which recognizes violations of the VME 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.

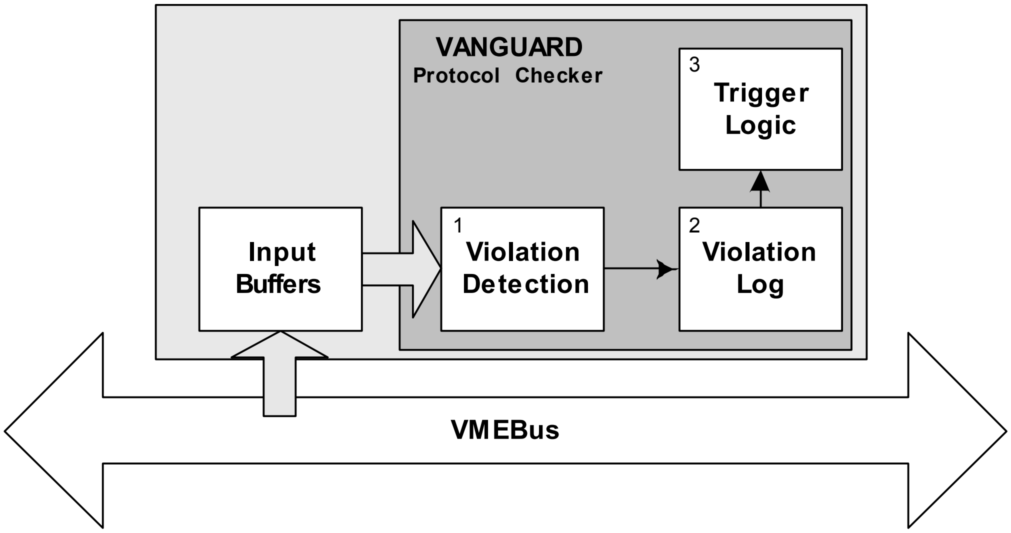


FIGURE 6-1 *The main blocks of the VANGUARD Protocol Checker.*

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 each occurrence of a violation is time stamped.

3. **Trigger Logic**
   On each occurrence of a bus 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.

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.

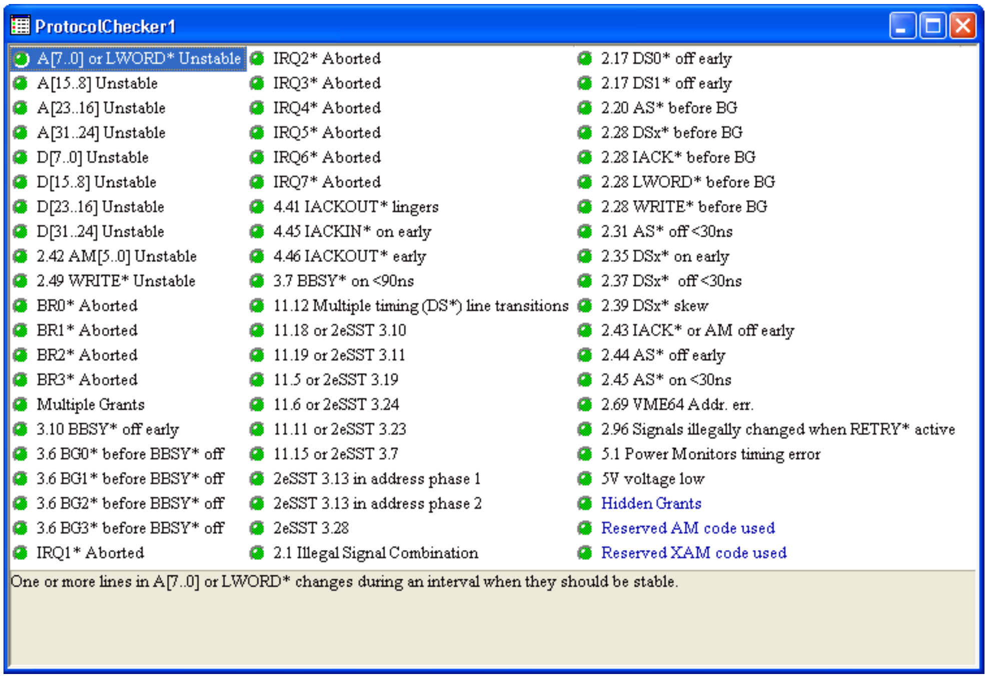
6.2 Operation

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*



A[7..0] or LWORD* Unstable
A[15..8] Unstable
A[23..16] Unstable
A[31..24] Unstable
D[7..0] Unstable
D[15..8] Unstable
D[23..16] Unstable
D[31..24] Unstable
2.42 AM[5..0] Unstable
2.49 WRITE* Unstable
BR0* Aborted
BR1* Aborted
BR2* Aborted
BR3* Aborted
Multiple Grants
3.10 BBSY* off early
3.6 BG0* before BBSY* off
3.6 BG1* before BBSY* off
3.6 BG2* before BBSY* off
3.6 BG3* before BBSY* off
IRQ1* Aborted

IRQ2* Aborted
IRQ3* Aborted
IRQ4* Aborted
IRQ5* Aborted
IRQ6* Aborted
IRQ7* Aborted
4.41 IACKOUT* lingers
4.45 IACKIN* on early
4.46 IACKOUT* early
3.7 BBSY* on <90ns
11.12 Multiple timing (DS*) line transitions
11.18 or 2eSST 3.10
11.19 or 2eSST 3.11
11.5 or 2eSST 3.19
11.6 or 2eSST 3.24
11.11 or 2eSST 3.23
11.15 or 2eSST 3.7
2eSST 3.13 in address phase 1
2eSST 3.13 in address phase 2
2eSST 3.28
2.1 Illegal Signal Combination

2.17 DS0* off early
2.17 DS1* off early
2.20 AS* before BG
2.28 DSx* before BG
2.28 IACK* before BG
2.28 LWORD* before BG
2.28 WRITE* before BG
2.31 AS* off <30ns
2.35 DSx* on early
2.37 DSx* off <30ns
2.39 DSx* skew
2.43 IACK* or AM off early
2.44 AS* off early
2.45 AS* on <30ns
2.69 VME64 Addr. err.
2.96 Signals illegally changed when RETRY* active
5.1 Power Monitors timing error
5V voltage low
Hidden Grants
Reserved AM code used
Reserved XAM code used

One or more lines in A[7..0] or LWORD* changes during an interval when they should be stable.

FIGURE 6-3 *Violations in "Classic" 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 three 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.

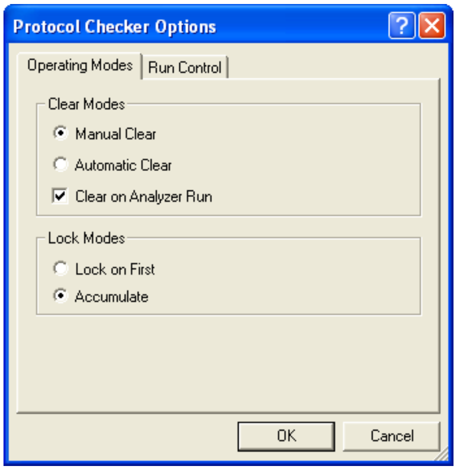
- A green lamp indicates that the corresponding violation is "on" and will be tested for.
- A dark green lamp indicates that this violation has been turned "off" and will not be tested for.
- A red lamp indicates that this violation has been detected.

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.

Protocol Checker Options

Operating Modes



Protocol Checker Options

Clear Modes

● Manual Clear

○ Automatic Clear

☑ Clear on Analyzer Run

Lock Modes

○ Lock on First

● Accumulate

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.

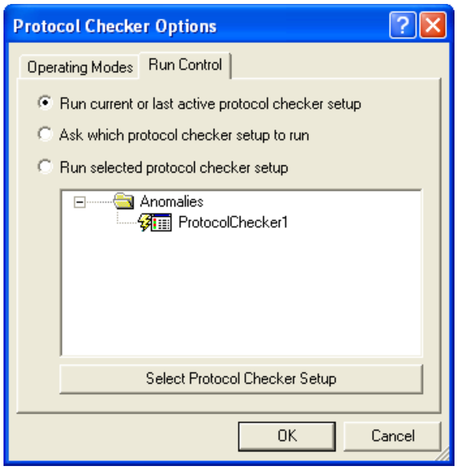
Clear on Trace Run: The Protocol Checker output trigger is cleared each time a trace is run.

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

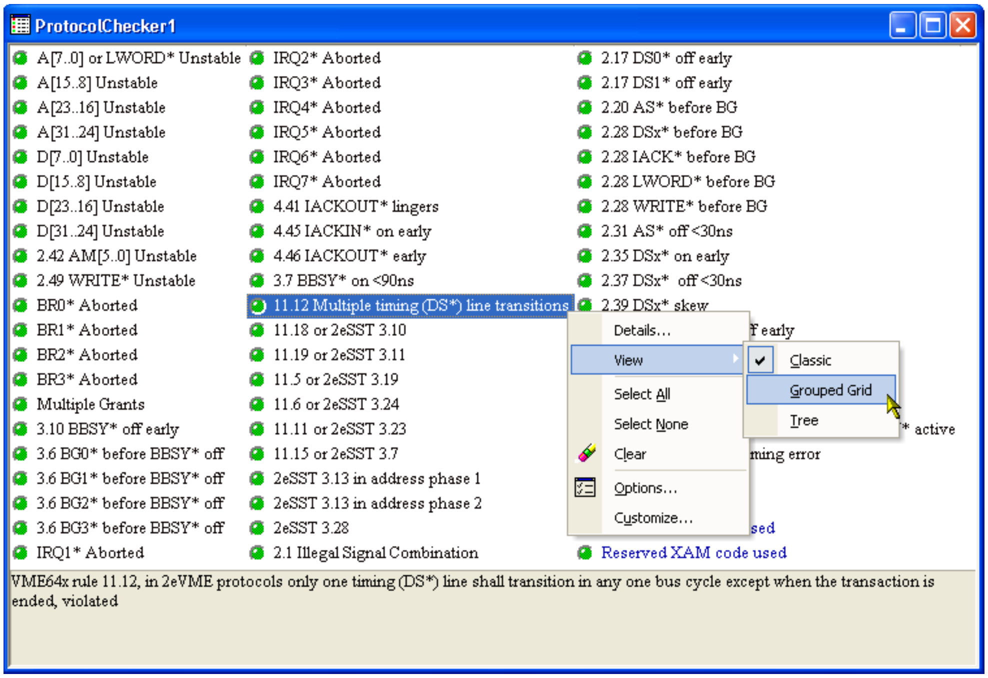


● Run current or last active protocol checker setup
○ Ask which protocol checker setup to run
○ Run selected protocol checker setup

FIGURE 6-5 *Protocol Checker Run Control*

Using the Protocol Checker

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



A[7..0] or LWORD* Unstable
A[15..8] Unstable
A[23..16] Unstable
A[31..24] Unstable
D[7..0] Unstable
D[15..8] Unstable
D[23..16] Unstable
D[31..24] Unstable
2.42 AM[5..0] Unstable
2.49 WRITE* Unstable
BR0* Aborted
BR1* Aborted
BR2* Aborted
BR3* Aborted
Multiple Grants
3.10 BBSY* off early
3.6 BG0* before BBSY* off
3.6 BG1* before BBSY* off
3.6 BG2* before BBSY* off
3.6 BG3* before BBSY* off
IRQ1* Aborted

IRQ2* Aborted
IRQ3* Aborted
IRQ4* Aborted
IRQ5* Aborted
IRQ6* Aborted
IRQ7* Aborted
4.41 IACKOUT* lingers
4.45 IACKIN* on early
4.46 IACKOUT* early
3.7 BBSY* on <90ns
11.12 Multiple timing (DS*) line transitions
11.18 or 2eSST 3.10
11.19 or 2eSST 3.11
11.5 or 2eSST 3.19
11.6 or 2eSST 3.24
11.11 or 2eSST 3.23
11.15 or 2eSST 3.7
2eSST 3.13 in address phase 1
2eSST 3.13 in address phase 2
2eSST 3.28
2.1 Illegal Signal Combination

2.17 DS0* off early
2.17 DS1* off early
2.20 AS* before BG
2.28 DSx* before BG
2.28 IACK* before BG
2.28 LWORD* before BG
2.28 WRITE* before BG
2.31 AS* off <30ns
2.35 DSx* on early
2.37 DSx* off <30ns
2.39 DSx* skew
f early
* active
ming error
sed
Reserved XAM code used

VME64x rule 11.12, in 2eVME protocols only one timing (DS*) line shall transition in any one bus cycle except when the transaction is ended, violated

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

Note – When screening the VME 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.

DetailsGive 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 categorised 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 bus 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 59).

The PCHKTrg signal field can have the following values:

TABLE 6-1. PCHKtrg values

Symbol	Value	Explanation
-	1	No violation
TRIG	0	VME bus violation
x	x	Don’t care

Trigger External Instruments on Violations

An oscilloscope or external logic analyzer can be triggered by the Protocol Checker in different ways. One way is to use the output signal available on the pin header of the Vanguard main board. A more convenient way is to use the front panel Trigger Output

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 VME clocks after the actual violation is detected on the bus.

6.3 VME Protocol Violations

Summary of VME protocol violations

The following classes of VME bus specification violations are detected.

Summary of detected VME protocol violations

Unstable			
A[7..0] or LWORD* Unstable	page 124	A[15..8] Unstable	page 125
A[23..16] Unstable	page 125	A[31..24] Unstable	page 126
D[7..0] Unstable	page 128	D[15..8] Unstable	page 131
D[23..16] Unstable	page 131	D[31..24] Unstable	page 132
2.42 AM [5..0] Unstable	page 132	2.49 WRITE* Unstable	page 133
Request/Grant			
BR0* Aborted	page 134	3.10 BBSY* off early	page 136
Multiple Grants	page 135	3.6 BG[0:3]* before BBSY* off	page 138
Interrupt			
IRQ(7:1)* aborted	page 140	4.46 IACKOUT* early	page 143
4.45 IACKOUT* on early	page 142		
Master			
3.7 BBSY* on <90ns	page 145	11.12 Multiple timing (DS*) line transitions	page 146
11.18 or 2eSST 3.10	page 147	11.19 or 2eSST 3.11	page 147
11.5 or 2eSST 3.19	page 148	11.6 or 2eSST 3.24	page 148
11.11 or 2eSST 3.23	page 149	11.15 or 2eSST 3.7	page 149
2eSST 3.13 in address phase 1	page 150	2eSST 3.13 in address phase 2	page 150
2eSST 3.28	page 151	2.1 Illegal Signal Combination	page 151
2.17 DS0* off early	page 152	2.17 DS1* off early	page 153
2.20 AS* before BG	page 154	2.28 DSx* before BG	page 155
2.28 IACK* before BG	page 156	2.28 LWORD* before BG	page 157
2.28 WRITE* before BG	page 158	2.31 AS* off<30ns	page 159
2.35 DSx* on early	page 160	2.37 DSx* off <30ns	page 161
2.39 DSx* skew	page 162	2.44 AS* off early	page 164
2.43 IACK*, AM off early	page 163	2.45 AS* on <30ns	page 165
2.69 VME64 Address error	page 166	2.96 Signals illegally changed when RETRY* active	page 166
5.1 Power Monitors timing error	page 167	5V voltage low	page 167
Warnings			
Hidden Grants	page 168	Reserved AM code used	page 169
Reserved XAM code used	page 169		

References used in protocol violation descriptions:

[1] *American National Standard for 2eSST, ANSI/VITA 1.5-2003*

[2] *American National Standard for VME64 Extensions, ANSI/VITA 1.1-1997*

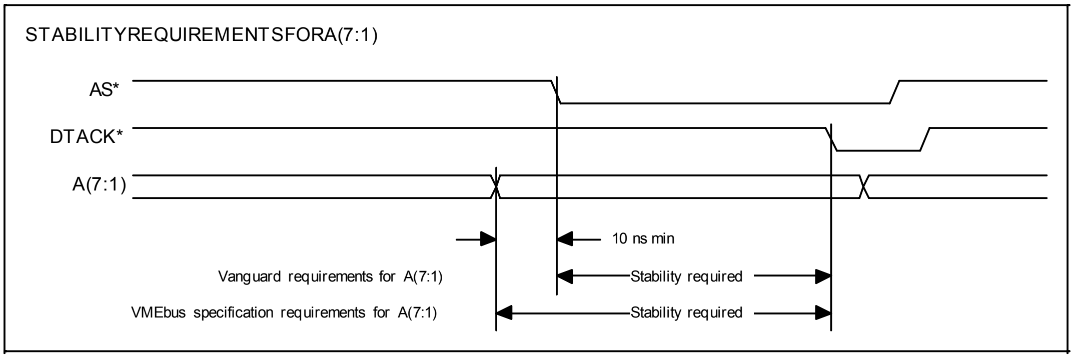
[3] *American National Standard for VME64, ANSI/VITA 1-1994*

A[7..0] or LWORD* Unstable

This violation is triggered when a change is detected on address lines A(7:1) or LWORD* during an interval where these lines should be stable. To verify this error, the following signals must be inspected : A(7:1), LWORD*, AS* and DTACK*.

In 64-bit data (D64) block cycles, during the data transfer phase, the 31 address lines plus LWORD* (as well as the data lines) contain data information, and therefore must be seen as stable at the same times as data lines (D31:00) are required to be stable. In other words, during the data transfer portions of a D64 write cycle data must be stable before the start of the data strobes, and must be held stable by the master until each falling edge of DTACK*. During the data transfer portions of a D64 read cycle data must be stable prior to 25 ns after each falling edge of DTACK*, and must be held stable until each set of data strobes are rescinded. During the data transfer phase of D64 cycles, lines A(7:1) and LWORD* carry data bits D(39:32) (BYTE(3)).

The following diagram illustrates the key signals present during a normal VMEbus cycle, and shows the time period during which address lines A(7:1) and LWORD* are required to be stable. (LWORD* is not explicitly shown in the diagram, but has the same requirements as A(7:1).

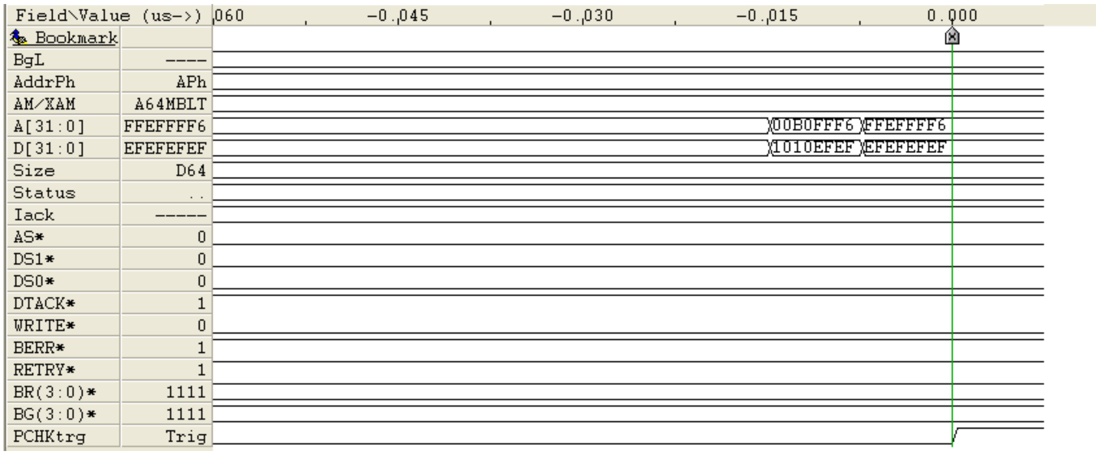

STABILITYREQUIREMENTSFORA(7:1)
AS*
DTACK*
A(7:1)
10 ns min
Vanguard requirements for A(7:1)
Stability required
VMEbus specification requirements for A(7:1)
Stability required

Conditions which could cause violation of this rule :

1. Design error in a bus master, in which address is not set up sufficiently in advance of the assertion of AS* or so that address does not remain until DTACK* is detected.
2. Design error in bus arbitration in one or more masters, such that two or more masters take control of the bus simultaneously. This condition is most common in systems in which overlapped arbitration (early release of BBSY*) is employed, due to slightly increased opportunity for design error.

A[15..8] Unstable

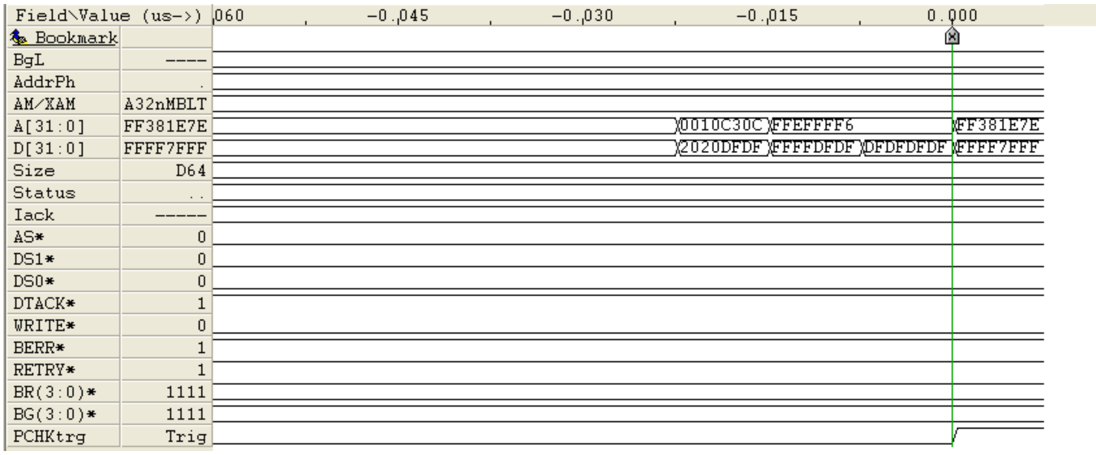
The "A(15:8) unstable" violation is triggered when a change is detected on address lines A(15:8) during an interval where these lines should be stable. Stability is checked during all cycles except interrupt acknowledge VMEbus cycles, as these cycles do not use lines A(15:8). During the data transfer phase of D64 cycles these lines are checked for stability, as lines A(15:8) carry data bits D(47:40) (BYTE(2)).



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	APh
AM/XAM	A64MBLT
A[31:0]	FFEFFFF6
D[31:0]	EFEFEFEF
Size	D64
Status	..
Iack	-----
AS*	0
DS1*	0
DS0*	0
DTACK*	1
WRITE*	0
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

A[23..16] Unstable

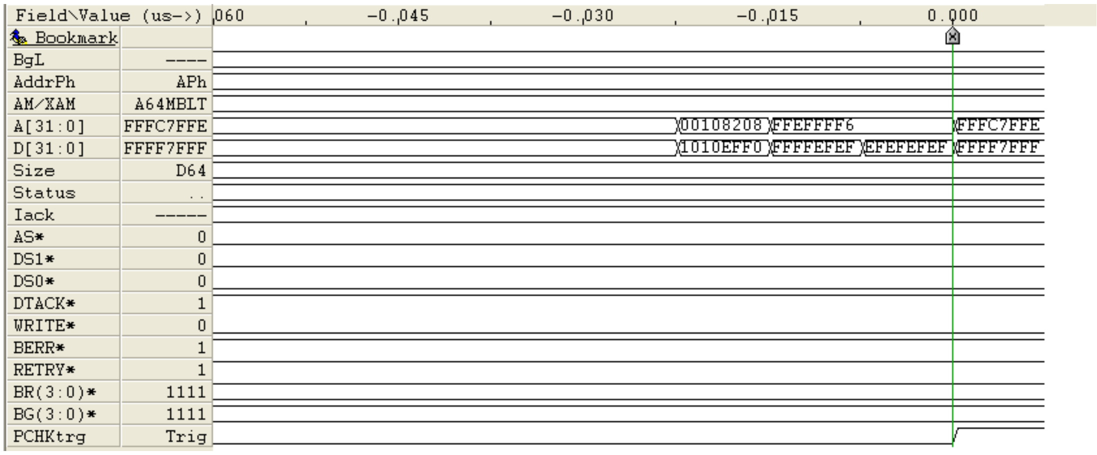
The "A(23:16) unstable" violation is triggered when a change is detected on address lines A(23:16) during an interval where these lines should be stable. Stability is only checked during cycles in which "standard" or "extended" addressing is specified by the address modifiers, as short address and interrupt acknowledge cycles do not use lines A(23:16). During the data transfer phase of D64 cycles these lines are checked for stability, as lines A(23:16) carry data bits D(55:48) (BYTE(1)).



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	.
AM/XAM	A32nMBLT
A[31:0]	FF381E7E
D[31:0]	FFFF7FFF
Size	D64
Status	..
Iack	-----
AS*	0
DS1*	0
DS0*	0
DTACK*	1
WRITE*	0
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

A[31..24] Unstable

The "A(31:24) unstable" violation is triggered when a change is detected on address lines A(31:24) during an interval where these lines should be stable. Stability is only checked during "extended" addressing cycles as specified by the address modifiers, as other cycles do not use lines A(31:24). During the data transfer phase of D64 cycles these lines are checked for stability, as lines A(31:24) carry data bits D(63:56) (BYTE(0)).



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	APh
AM/XAM	A64MBLT
A[31:0]	FFFC7FFE
D[31:0]	FFFF7FFF
Size	D64
Status	..
Iack	-----
AS*	0
DS1*	0
DS0*	0
DTACK*	1
WRITE*	0
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

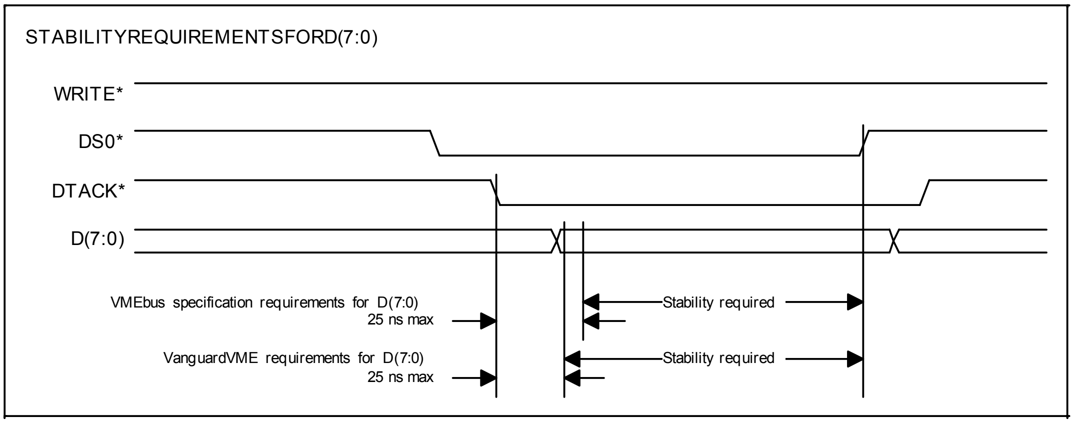
D[7..0] Unstable

Triggered when a change is detected on data lines D(7:0) during an interval where these lines should be stable. The conditions under which they are checked for stability depend upon the type of bus cycle in progress. Each type of cycle is discussed separately below. To verify this error, the following signals must be inspected : D(7:0), DS0*, DTACK* and WRITE*.

D[7..0] Unstable (Continued)

D(7:0) unstable during VMEbus read cycle

The following diagram illustrates the key signals present during a normal VMEbus read cycle, and shows the time period during which data lines D(7:0) are required to be stable.


STABILITYREQUIREMENTSFORD(7:0)
WRITE*
DS0*
DTACK*
D(7:0)
VMEbus specification requirements for D(7:0)
25 ns max
Stability required
VanguardVME requirements for D(7:0)
25 ns max
Stability required

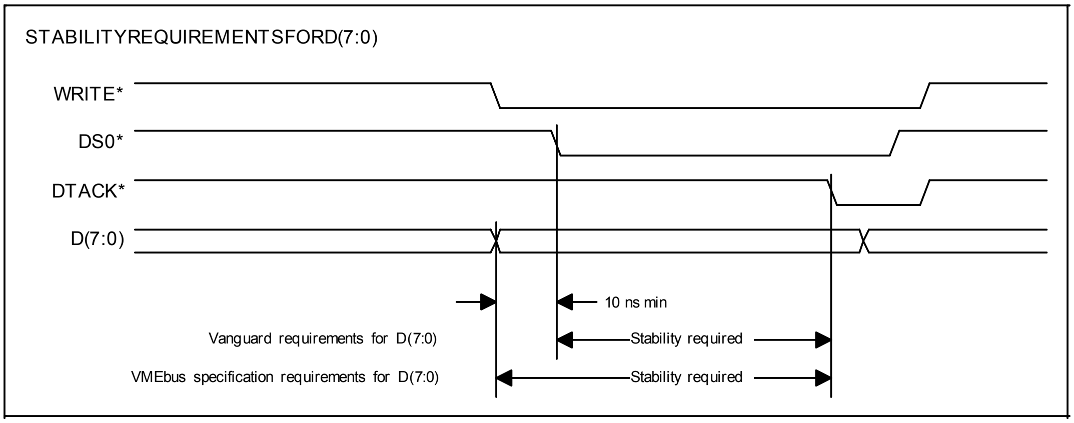
Conditions which could cause violation of this rule :

1. Memory board or other slave asserts DTACK* before it is supplying valid data.
2. Design error in slave, in which data is not latched and therefore does not remain stable and valid even after addresses go away and until data strobe DS0* is rescinded.
3. Design error in slave, in which a board selects itself using address strobe (AS*) only. In this way the slave may be falsely responding to a bus cycle when it is really not being addressed. This will result in two slaves responding to the same bus cycle.
4. Design error in master, in which the data transceivers of a master are illegally changed in directions after DTACK*, and consequently "fight" with the buffers of slave boards.

D[7..0] Unstable (Continued)

D(7:0) unstable during VMEbus write cycle

The following diagram illustrates the key signals present during a normal VMEbus write cycle, and shows the time period during which data lines D(7:0) are required to be stable.


STABILITYREQUIREMENTSFORD(7:0)
WRITE*
DS0*
DTACK*
D(7:0)
10 ns min
Vanguard requirements for D(7:0)
Stability required
VMEbus specification requirements for D(7:0)
Stability required

Conditions which could cause violation of this rule :

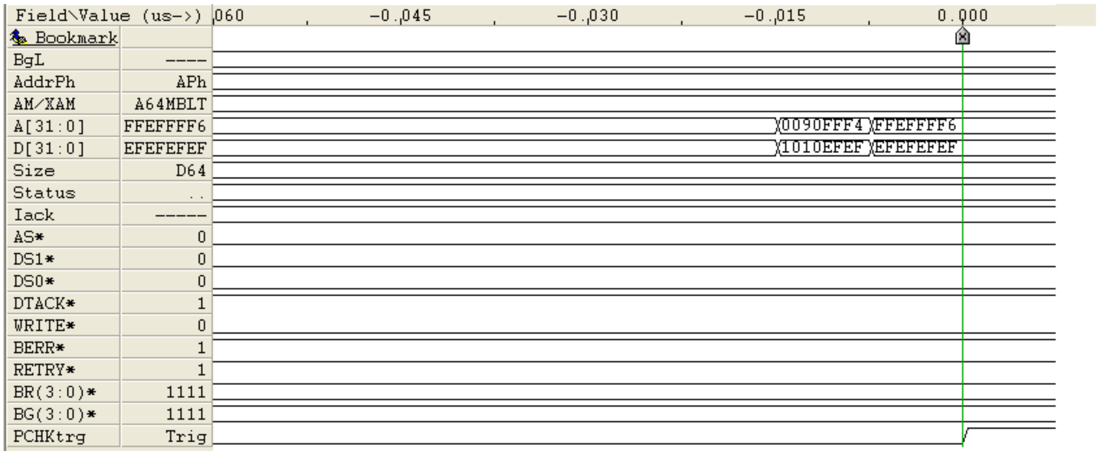
1. Design error in a bus master, in which data is not set up sufficiently in advance of the assertion of DS0*, or so that data does not remain until DTACK* is detected.
2. Design error in bus arbitration in one or more masters, resulting in two or more masters taking control of the bus simultaneously. This condition is most common in systems in which overlapped arbitration (early release of BBSY*) is employed, due to increased opportunity for design error.

D(7:0) unstable during address phase of A64/A40 cycles

In 64-bit address (A64) cycles the 32 data lines (as well as the address lines) contain address information, and therefore must be seen as stable from the time 10 ns before AS* is asserted until the first falling edge of DTACK*, BERR* or RETRY*. During the address phase of A64 or A40 cycles, lines D(7:0) carry address bits A(39:32). The Vanguard checks all applicable lines for stability, and at times is checking all 64 address and data lines simultaneously.

D[15..8] Unstable

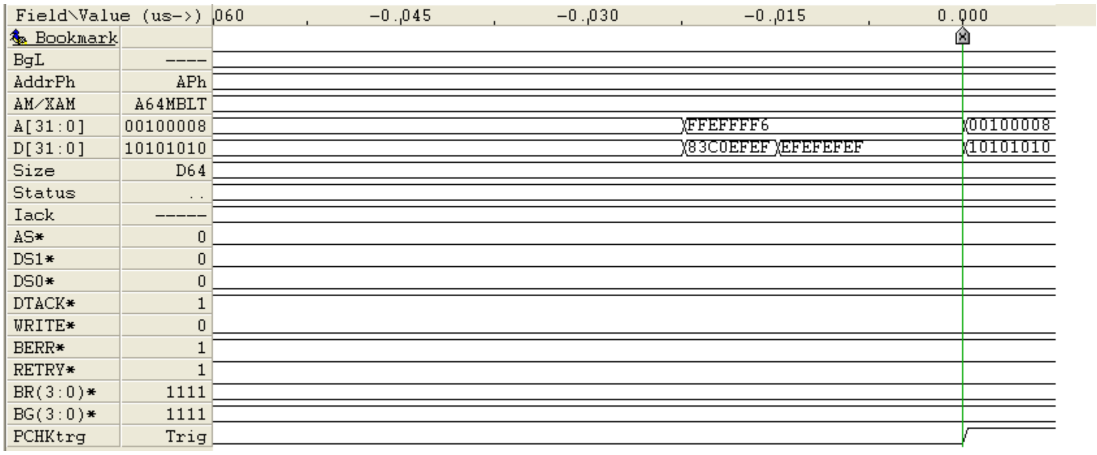
The "D(15:8) unstable" violation is triggered when a change is detected on data lines D(15:8) during an interval where these lines should be stable. The conditions for stability are essentially the same as for "D(7:0) unstable" above, but with DS1* substituted for DS0* in the above diagram. During the address phase of A64 cycles, lines D(15:8) carry address bits A(47:40). During A40 cycles, lines D(15:8) carry address bits A(31:24).



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	APh
AM/XAM	A64MBLT
A[31:0]	FFEFFFF6
D[31:0]	EFEFEFEF
Size	D64
Status	..
Iack	-----
AS*	0
DS1*	0
DS0*	0
DTACK*	1
WRITE*	0
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

D[23..16] Unstable

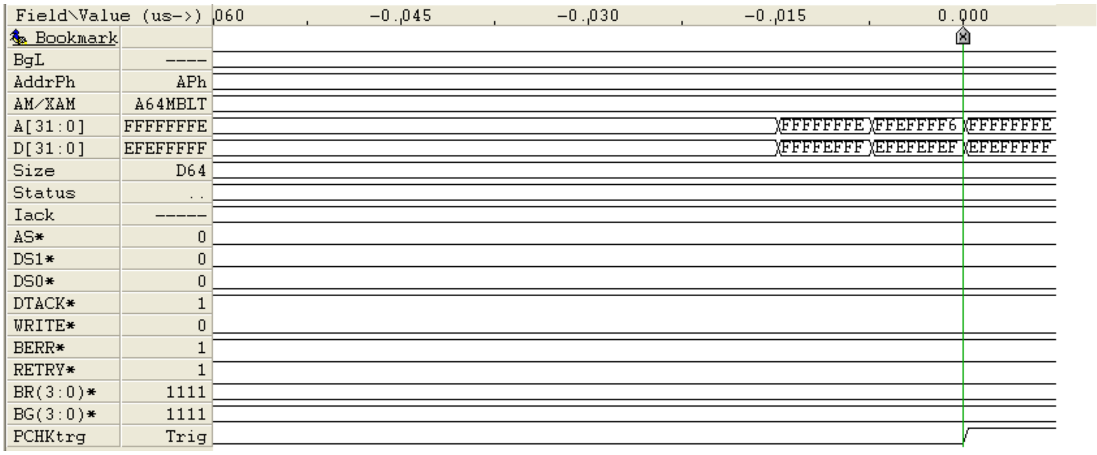
The "D(23:16) unstable" violation is triggered when a change is detected on data lines D(23:16) during an interval where these lines should be stable. The conditions for stability are essentially the same as for "D(7:0) unstable". Stability is only checked for D(23:16) during longword cycles and unaligned 2-byte and 3-byte cycles. During the address phase of A64 cycles, lines D(23:16) carry address bits A(55:48).



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	APh
AM/XAM	A64MBLT
A[31:0]	00100008
D[31:0]	10101010
Size	D64
Status	..
Iack	-----
AS*	0
DS1*	0
DS0*	0
DTACK*	1
WRITE*	0
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

D[31..24] Unstable

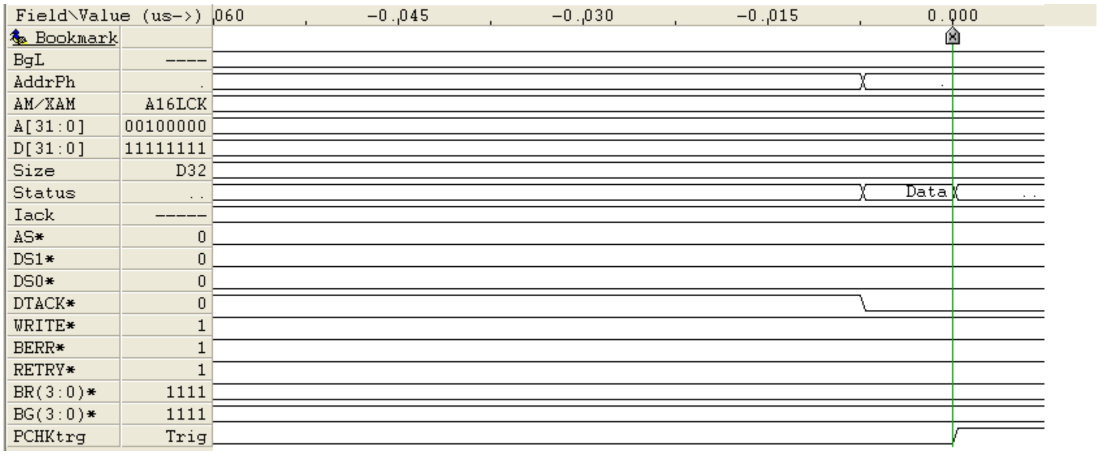
The "D(31:24) unstable" violation is triggered when a change is detected on data lines D(31:24) during an interval where these lines should be stable. The conditions for stability are essentially the same as for "D(15:8) unstable". Stability is only checked for D(31:24) during longword cycles and left-shifted unaligned 3-byte cycles. During the address phase of A64 cycles, lines D(31:24) carry address bits A(63:56).



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	APh
AM/XAM	A64MBLT
A[31:0]	FFFFFFFE
D[31:0]	EFEFFFFF
Size	D64
Status	..
Iack	-----
AS*	0
DS1*	0
DS0*	0
DTACK*	1
WRITE*	0
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

2.42 AM [5..0] Unstable

During all data transfer cycles, the master must maintain a valid AM code and ensure that IACK* stays high until it detects the last falling edge on DTACK* or BERR*, rule 2.42 [3, page 86]. An error will be reported if the AM code or IACK* is unstable from the time AS* is asserted until DTACK* or BERR* is asserted.



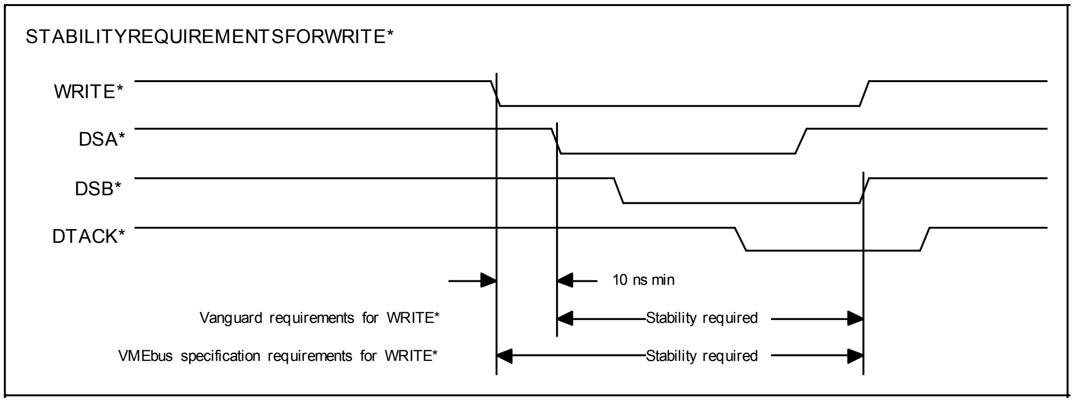
Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	.
AM/XAM	A16LCK
A[31:0]	00100000
D[31:0]	11111111
Size	D32
Status	..
Iack	-----
AS*	0
DS1*	0
DS0*	0
DTACK*	0
WRITE*	1
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

2.49 WRITE* Unstable

Once a master has driven DSA* low, it must not change the level of WRITE* until DSB* is high, rule 2.49 [3, page 87]. WRITE* will be tested for stability from the falling edge of DSA* until the rising edge of DSB*. If the signal changes polarity during that interval, an error is reported.

To verify this error, the following signals must be inspected : WRITE*, DS0*, DS1* and DTACK*.

The following diagram illustrates the key signals present during a normal VMEbus write cycle, and shows the time period during which WRITE* is required to be stable.

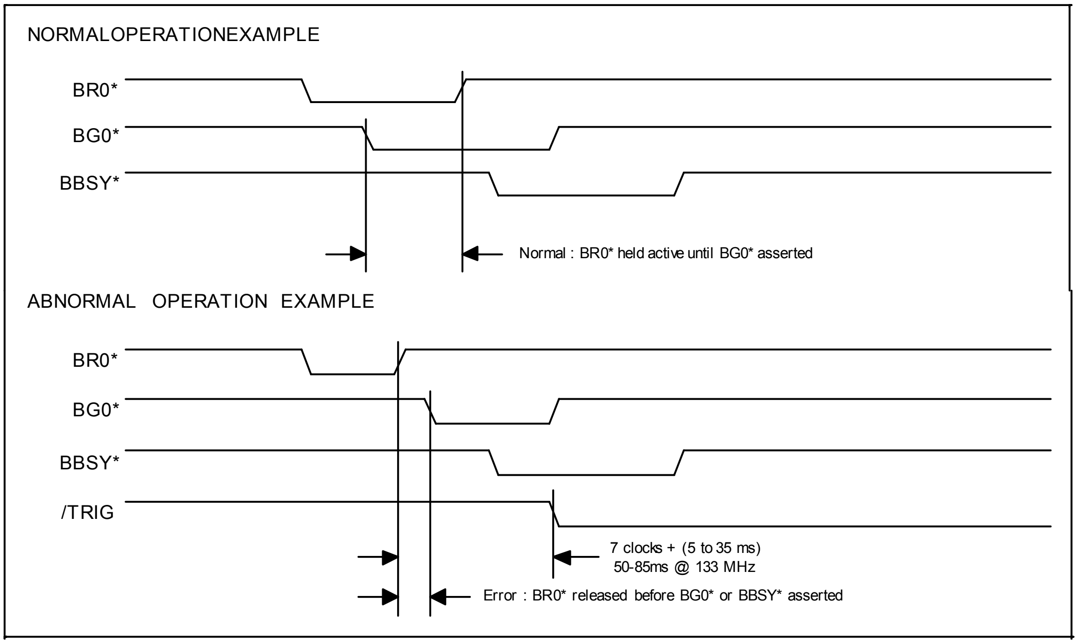

STABILITYREQUIREMENTSFORWRITE*
WRITE*
DSA*
DSB*
DTACK*
10 ns min
Vanguard requirements for WRITE*
Stability required
VMEbus specification requirements for WRITE*
Stability required

Note that DSA* and DSB* refer, respectively, to the first and last transition of data strobes DS0* and DS1*.

BR0* Aborted

BR1*, BR2*, BR3* are the same as "BR0* aborted" violation, but pertains to BR1*/BG1*, BR2*/BG2* and BR3*/BG3* instead of BG0*/BG0*.

The "BR0* aborted" violation is triggered when bus request BR0* is prematurely aborted (rescinded) before BG0* or BBSY* occurs. To verify this error, the following signals must be inspected : BR0*, BG0* and BBSY*. If using an oscilloscope or external logic analyzer, then BG0* must be measured directly at the Vanguard slot. The following diagram illustrates normal operation as well as an error condition.


NORMALOPERATIONEXAMPLE
BR0*
BG0*
BBSY*
Normal : BR0* held active until BG0* asserted
ABNORMAL OPERATION EXAMPLE
BR0*
BG0*
BBSY*
/TRIG
7 clocks + (5 to 35 ms)
50-85ms @ 133 MHz
Error : BR0* released before BG0* or BBSY* asserted

Note that this violation will not be seen if the Vanguard is located at a higher slot number than a master which is taking over the bus and therefore does not pass a bus grant down the daisy chain.

In State mode the clock will run at 133 MHz. In Timing mode the clock frequency will depend on settings made in BusView.

Conditions which could cause this error:

1. Design error in a bus requester, in which a request is made and then aborted.
2. Bus noise (usually due to ringing or ground bounce) on the bus request line, which causes it to momentarily go inactive (high).

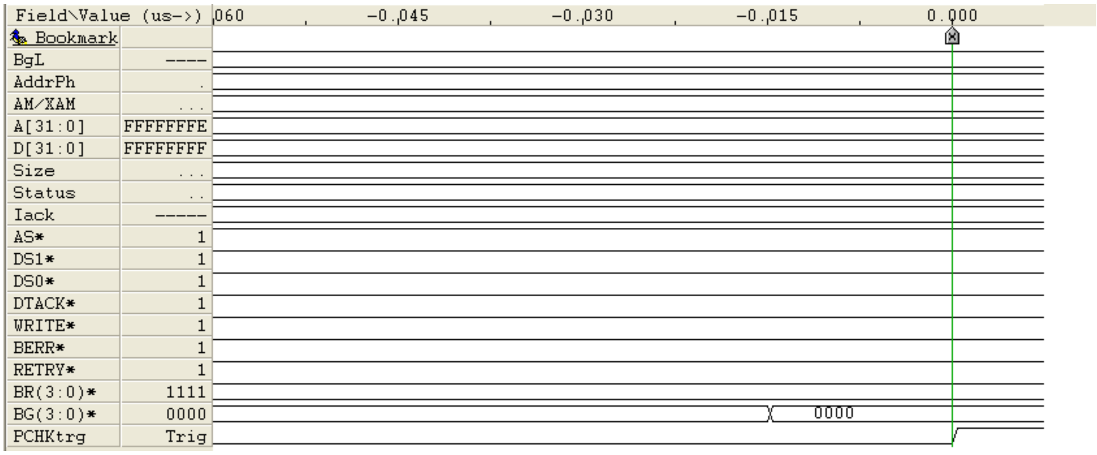
Multiple Grants

The "Multiple grants" violation is triggered when more than one bus grant BG(3:0)* occurs at the same time. The VMEbus specification requires that only one grant ever occurs at any time. To verify this error, the following signals must be inspected : BG(3:0)*.

Note that this violation will not be seen if the Vanguard is located at a higher slot number than a master which is not passing a grant down the daisy chain.

Conditions which could cause this error:

1. Design error in which the bus arbiter simultaneously generates grants at two or more levels.
2. Design error in bus grant daisy chain circuitry in a master which resides to the left of the Vanguard, where the master produces a BGxOUT* signal even when no BGxIN* is issued to it.

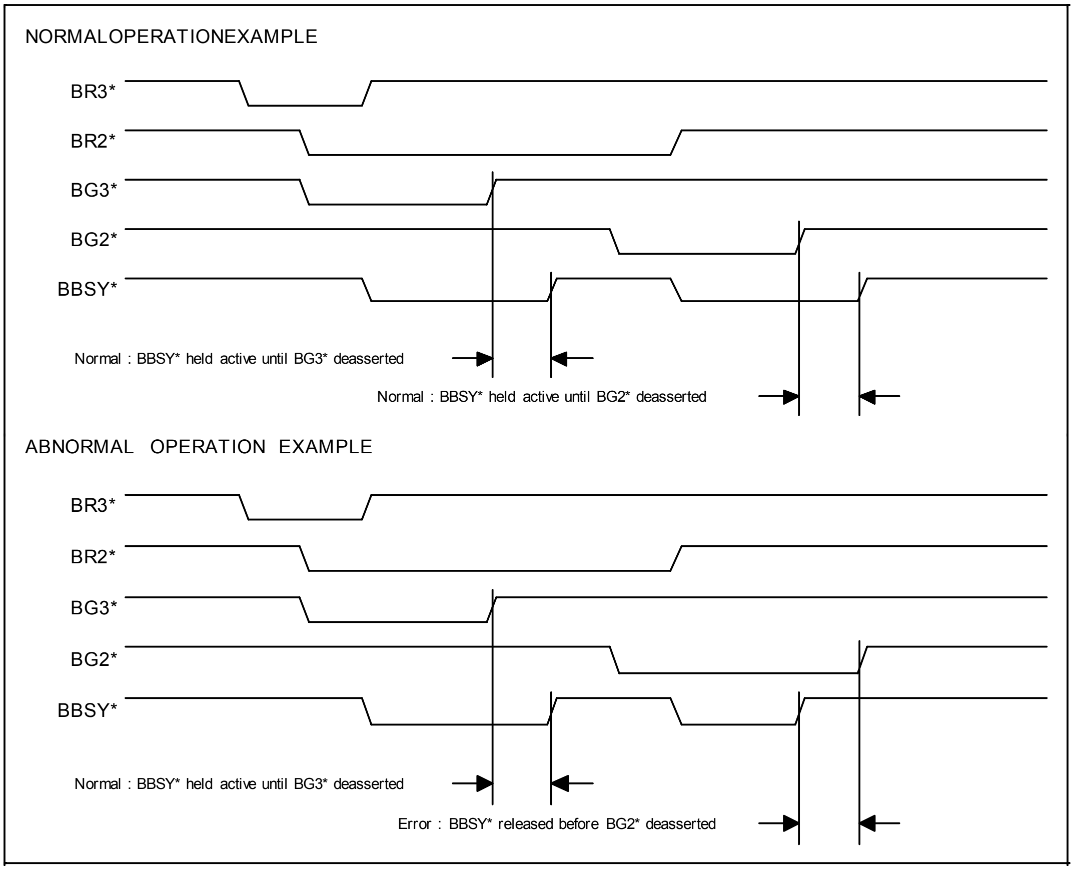


Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	.
AM/XAM	...
A[31:0]	FFFFFFFE
D[31:0]	FFFFFFFF
Size	...
Status	..
Iack	-----
AS*	1
DS1*	1
DS0*	1
DTACK*	1
WRITE*	1
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	0000
PCHKtrg	Trig

3.10 BBSY* off early

The "Rule 3.10 BBSY* off early" violation is triggered when a requester rescinds BBSY* before the bus grant goes high. The VMEbus specification requires that BBSY* be held active until the grant which led to BBSY* is rescinded. To verify this error, the following signals must be inspected : BG(3:0)* and BBSY*. If using an oscilloscope or external logic analyzer, then BG(3:0)* must be measured directly at the Vanguard slot.

The following diagram illustrates normal operation as well as an error condition.


NORMALOPERATIONEXAMPLE
BR3*
BR2*
BG3*
BG2*
BBSY*
Normal : BBSY* held active until BG3* deasserted
Normal : BBSY* held active until BG2* deasserted
ABNORMAL OPERATION EXAMPLE
BR3*
BR2*
BG3*
BG2*
BBSY*
Normal : BBSY* held active until BG3* deasserted
Error : BBSY* released before BG2* deasserted

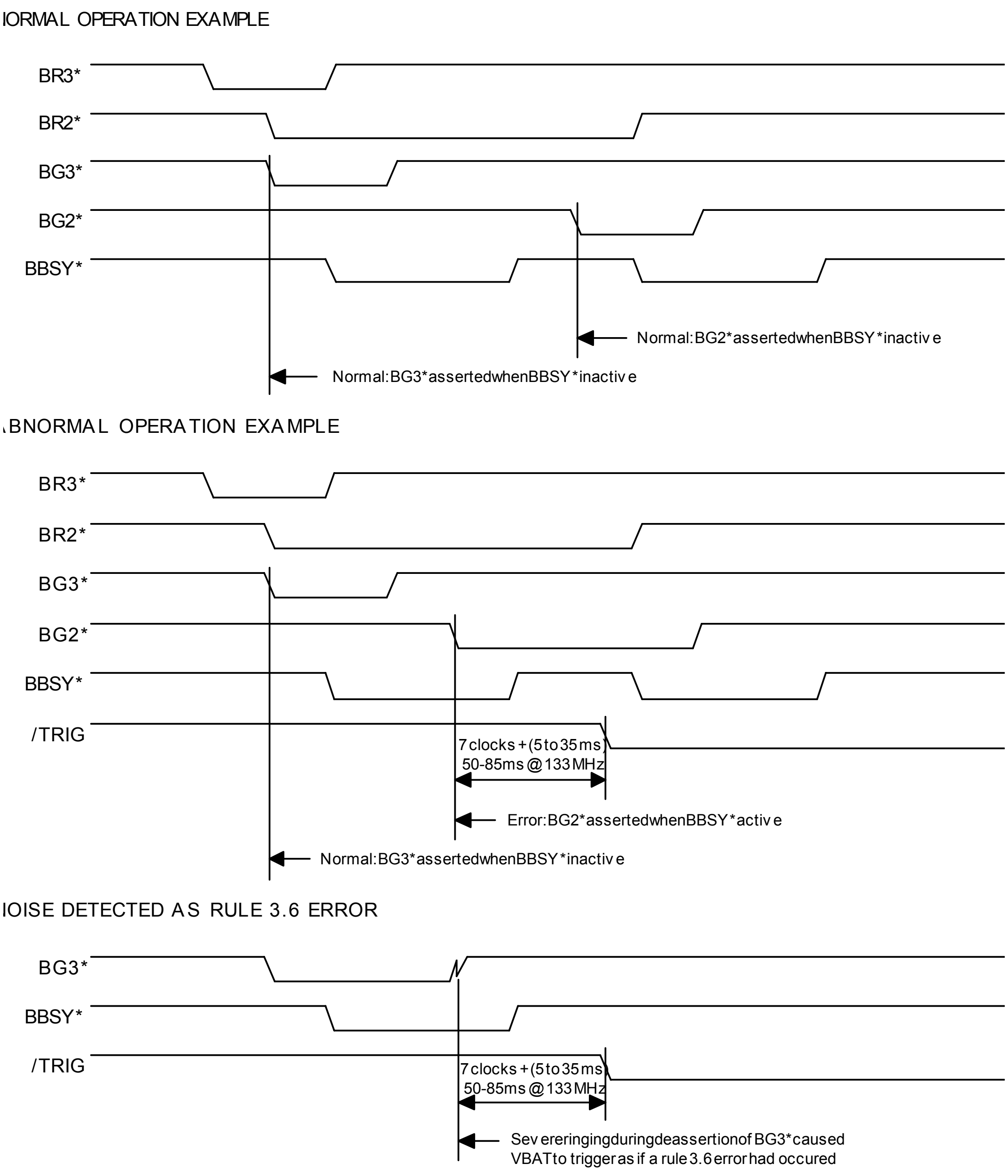
Note that this violation will not be seen if the Vanguard is located at a higher slot number than a master which is taking over the bus and therefore does not pass a grant down the daisy chain.

3.6 BG[0:3]* before BBSY* off

The "Rule 3.6 : Bus grant before BBSY* off" violation is triggered when the bus is granted to a new master (BGx* goes active) before BBSY* goes away. A new bus grant BGx* is prohibited from being issued until BBSY* from the previous master goes away, as illustrated below. To verify this error, the following signals must be inspected : BG(3:0)* and BBSY*. If using an oscilloscope or external logic analyzer, then BG(3:0)* must be measured directly at the Vanguard slot.

3.6 BG[0:3]* before BBSY* off (Continued)

The following diagram illustrates normal operation, an error condition, and an example of noise detected as this error.

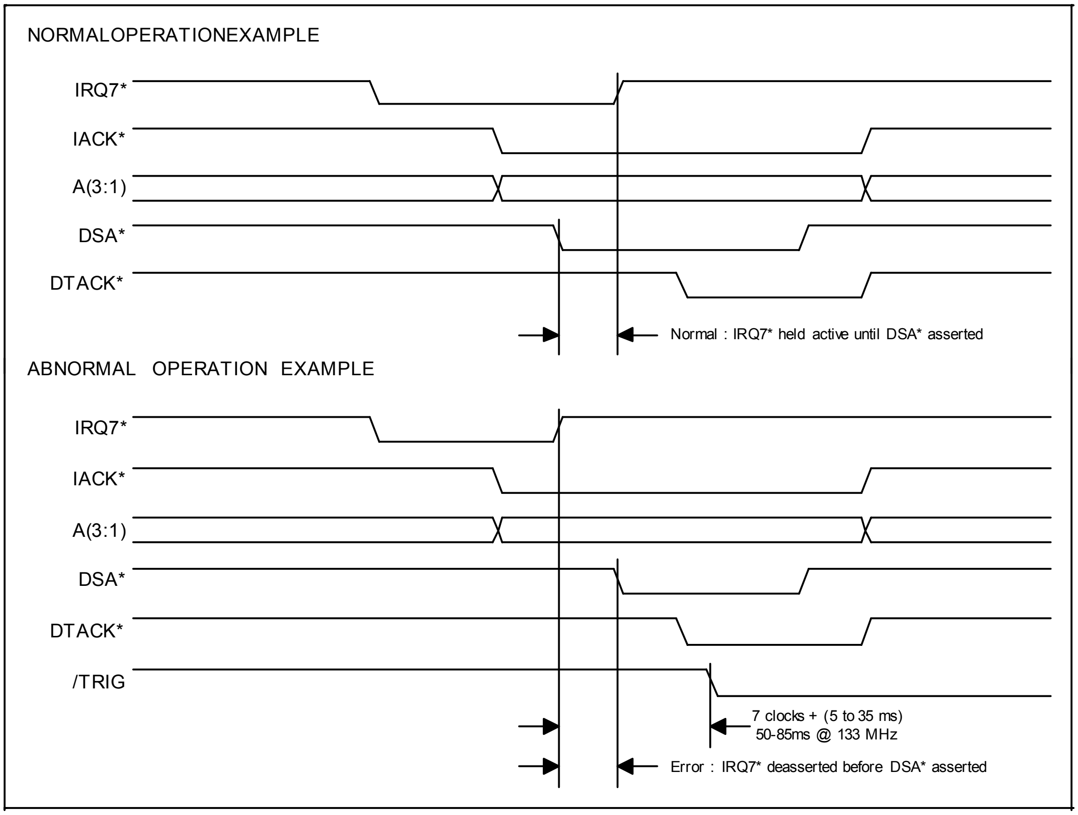

IORMAL OPERATION EXAMPLE
BR3*
BR2*
BG3*
BG2*
BBSY*
Normal:BG2*assertedwhenBBSY*inactive
Normal:BG3*assertedwhenBBSY*inactive
ıBNORMAL OPERATION EXAMPLE
BR3*
BR2*
BG3*
BG2*
BBSY*
/TRIG
7clocks+(5to35ms)
50-85ms@133MHz
Error:BG2*assertedwhenBBSY*active
Normal:BG3*assertedwhenBBSY*inactive
IOISE DETECTED AS RULE 3.6 ERROR
BG3*
BBSY*
/TRIG
7clocks+(5to35ms)
50-85ms@133MHz
Severeringingduringdeassertionof BG3*caused
VBATto triggerasif a rule3.6errorhad occured

Note that this violation will not be seen if the Vanguard is located at a higher slot number than a master which is taking over the bus and therefore does not pass a bus grant down the daisy chain.

IRQ(7:1)* aborted

The "IRQ(7:1)* aborted" violation is triggered when one of the interrupt lines IRQ(7:1)* is aborted before the appropriate interrupt acknowledge cycle is started. Rules 4.5 and 4.6 of the VMEbus specification prohibit an interrupter from releasing an interrupt request line IRQx* before it detects a falling edge on DSA* during the interrupt acknowledge cycle which acknowledges its interrupt. To verify this error, the following signals must be inspected : DS0*, DS1*, IRQ(7:1)*, IACK*, A(3:1) and DTACK*.

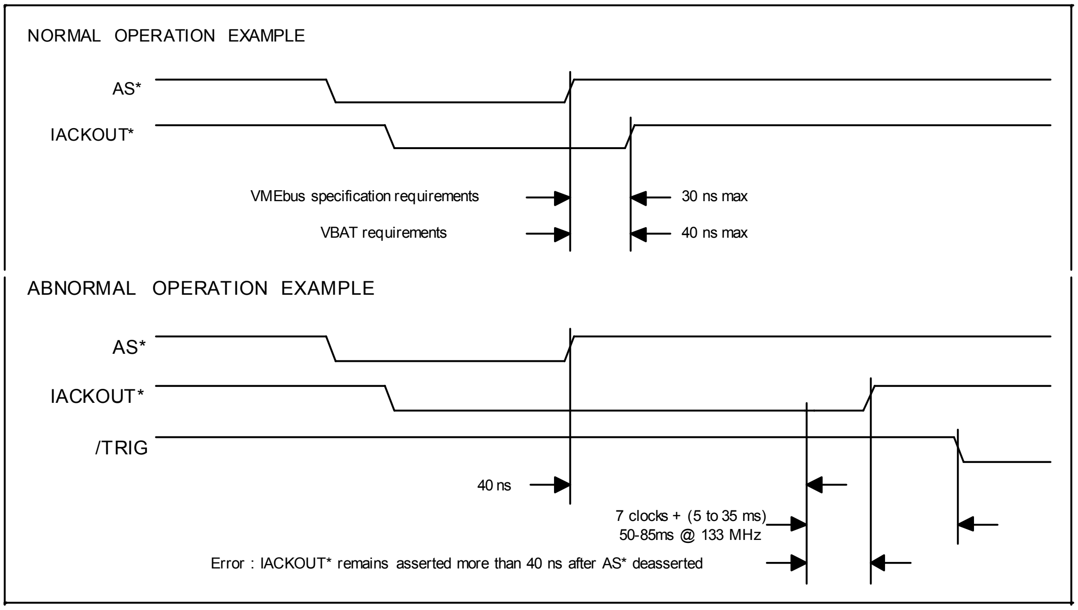
The following diagram illustrates normal operation as well as an error condition.



4.41 IACKOUT* lingers

The "Rule 4.41 : IACKOUT* lingers" violation is triggered when, at the end of an interrupt acknowledge cycle, the IACKOUT* line of a master was not driven high within 40 ns after AS* was driven high. To verify this error, the following signals must be inspected : AS*, IACK* and IACKOUT*. If using an oscilloscope or external logic analyzer, then IACKOUT* must be measured directly at the Vanguard slot.

The following diagram illustrates normal operation as well as an error condition.

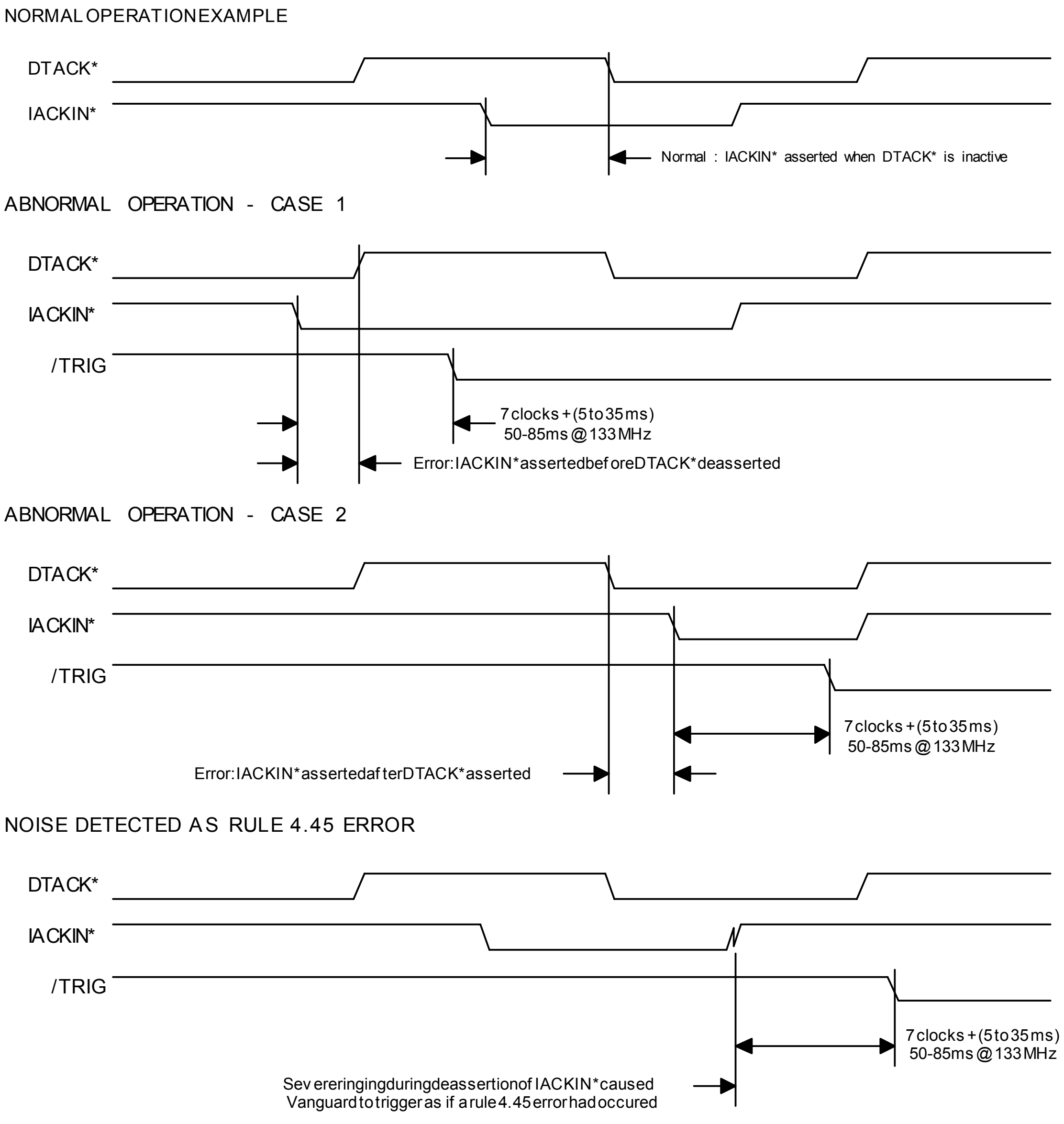


Note that this violation will not be seen if the Vanguard is located at a higher slot number than the failing board, because IACKIN*/IACKOUT* is a daisy-chain line.

4.45 IACKOUT* on early

The "Rule 4.45 : IACKIN* on early" violation is triggered when IACKIN* goes low before DTACK* and BERR* have gone high (been rescinded after the previous cycle), or if IACKIN* goes low after DTACK* is driven low during the present cycle. To verify this error the following signals must be inspected : DTACK*, BERR*, AS* and IACKIN*. If using an oscilloscope or external logic analyzer, then IACKIN* must be measured directly at the Vanguard slot.

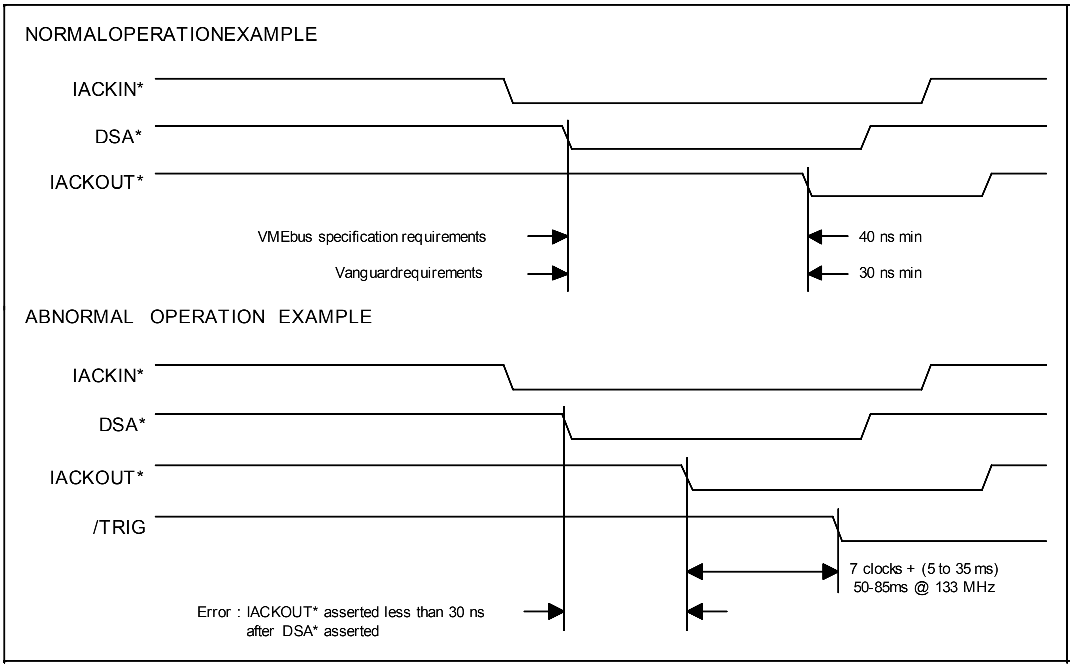
The following diagram illustrates normal operation, two different error conditions, and an example of noise detected as this error.


NORMAL OPERATION EXAMPLE
DTACK*
IACKIN*
Normal : IACKIN* asserted when DTACK* is inactive
ABNORMAL OPERATION - CASE 1
DTACK*
IACKIN*
/TRIG
7 clocks + (5 to 35 ms)
50-85ms @ 133 MHz
Error: IACKIN* asserted before DTACK* deasserted
ABNORMAL OPERATION - CASE 2
DTACK*
IACKIN*
/TRIG
7 clocks + (5 to 35 ms)
50-85ms @ 133 MHz
Error: IACKIN* asserted after DTACK* asserted
NOISE DETECTED AS RULE 4.45 ERROR
DTACK*
IACKIN*
/TRIG
7 clocks + (5 to 35 ms)
50-85ms @ 133 MHz
Severe ringing during deassertion of IACKIN* caused
Vanguard to trigger as if a rule 4.45 error had occured

4.46 IACKOUT* early

The "Rule 4.46 : IACKOUT* early" violation is triggered when the IACKOUT* line of the board to the left of the Vanguard is driven low less than 30 ns after the falling edge of AS*, when the IACKIN* line is low when the IACK daisy-chain driver detects a falling edge on DSA*. To verify this error, the following signals must be inspected : AS*, DS0*, DS1*, IACKIN* and IACKOUT*. If using an oscilloscope or external logic analyzer, then IACKOUT* must be measured at the slot preceding the Vanguard slot.

The following diagram illustrates normal operation as well as an error condition.



Conditions which could cause this error:

1. Design error in which IACKOUT* is issued by an interrupter in direct response to IACKIN* without qualifying the output with AS* and DS*, and without providing enough delay for settling of the IACK daisy-chain. This error is made frequently, since this rule was not present in early VMEbus specifications.

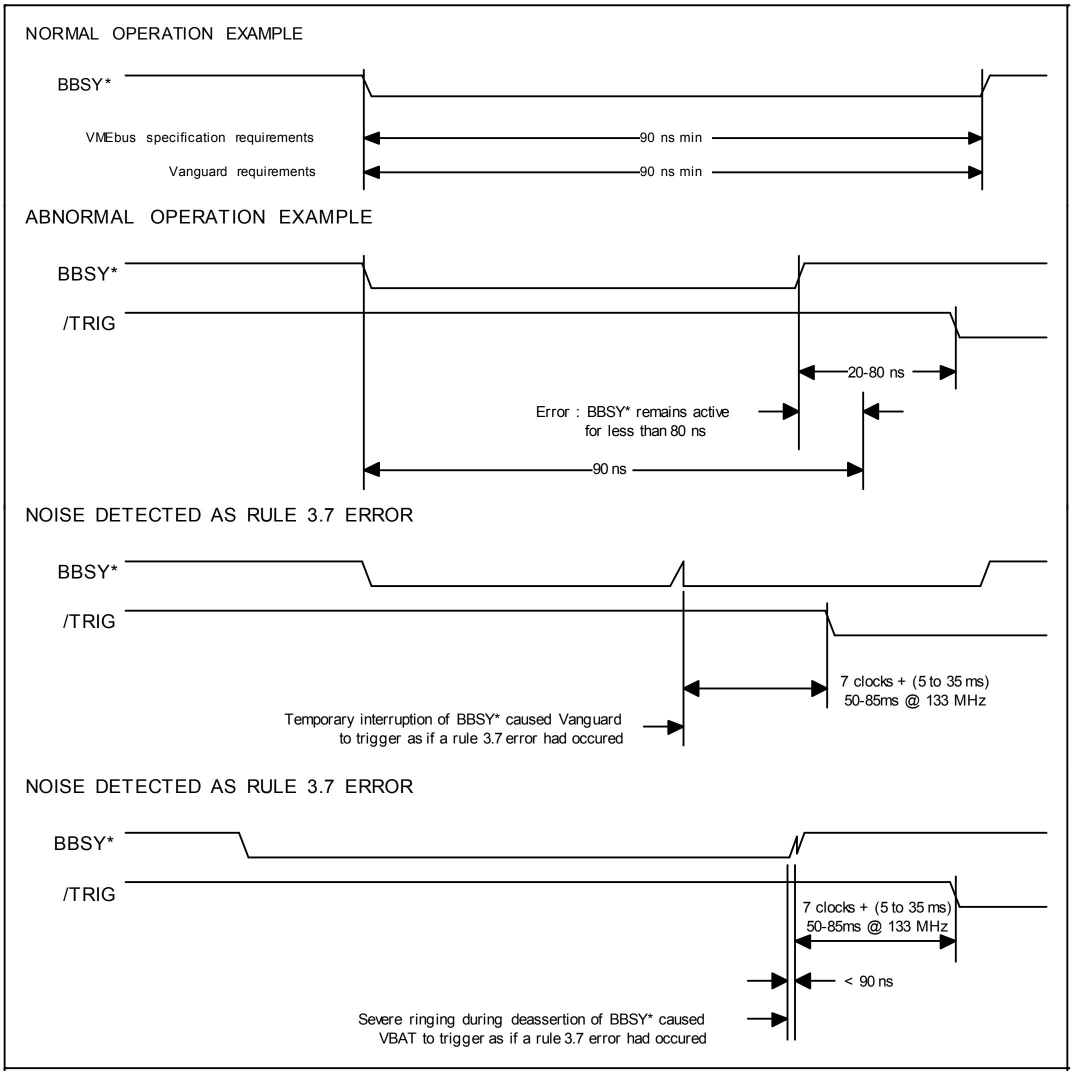
Condition which would cause a false error, when no error actually exists :

1. The Vanguard is installed in a slot to the left of where the first interrupter in the system is located. Since IACKIN* is derived from IACK* at slot 1, it will necessarily violate this rule prior to travelling through an interrupter. For this reason this error should be checked out and then ignored if the above case is true.

3.7 BBSY* on <90ns

The "Rule 3.7 : BBSY* on < 90 ns" violation is triggered when BBSY* is active for too short a time (< 90 ns). To verify this error, the following signals must be inspected : BBSY*.

The following diagram illustrates normal operation, an error condition, and two examples of noise detected as this error.


NORMAL OPERATION EXAMPLE
BBSY*
VMEbus specification requirements
90 ns min
Vanguard requirements
90 ns min
ABNORMAL OPERATION EXAMPLE
BBSY*
/TRIG
20-80 ns
Error : BBSY* remains active
for less than 80 ns
90 ns
NOISE DETECTED AS RULE 3.7 ERROR
BBSY*
/TRIG
7 clocks + (5 to 35 ms)
50-85ms @ 133 MHz
Temporary interruption of BBSY* caused Vanguard
to trigger as if a rule 3.7 error had occured
NOISE DETECTED AS RULE 3.7 ERROR
BBSY*
/TRIG
7 clocks + (5 to 35 ms)
50-85ms @ 133 MHz
< 90 ns
Severe ringing during deassertion of BBSY* caused
VBAT to trigger as if a rule 3.7 error had occured

3.7 BBSY* on <90ns (Continued)

Conditions which could cause this error:

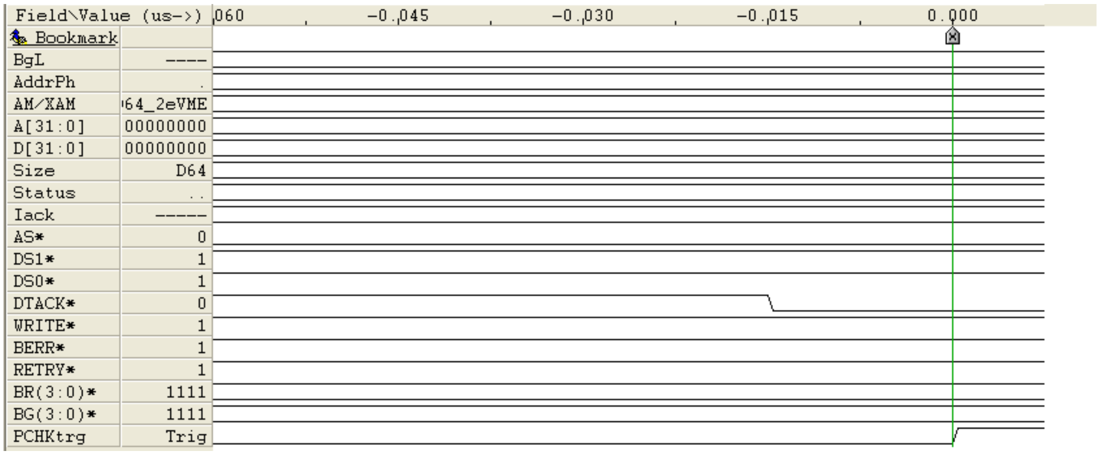
1. Design error in an "overlapped arbitration" bus master, in which BBSY* is rescinded immediately after AS* is asserted, without giving due consideration to ensuring that BBSY* has been held active for long enough.

2. Bus noise, usually due to ground bounce from simultaneous assertion of 31 address lines (plus 32 data lines in case of a write cycle) which causes BBSY* to go momentarily inactive (high).

3. Bus noise (usually aggravated by the slow rise time of this open collector line) in which BBSY* is asserted, then rescinded, and noise causes it to momentarily go active (low) again. This situation is common, due to the slow and gradual de-assertion of BBSY*, due to the fact that BBSY* is normally an open collector output.

11.12 Multiple timing (DS*) line transitions

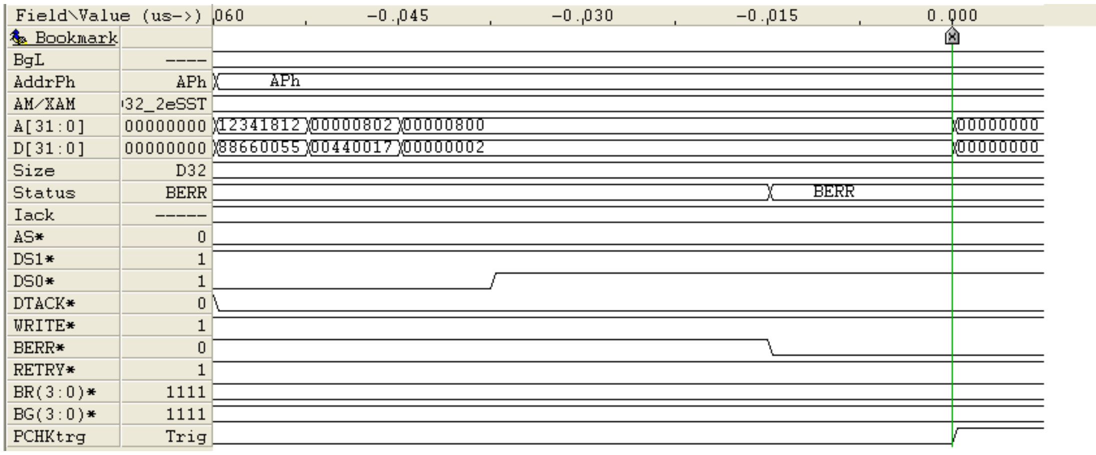
In 2eVME protocols only one timing (DS*) line shall transition in any one bus cycle except when the transaction is ended, VME64x rule 11.12 [2, page 61]. Error 11.12 from the protocol checker indicates that this rule is broken.



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	.
AM/XAM	'64_2eVME
A[31:0]	00000000
D[31:0]	00000000
Size	D64
Status	..
Iack	-----
AS*	0
DS1*	1
DS0*	1
DTACK*	0
WRITE*	1
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

11.18 or 2eSST 3.10

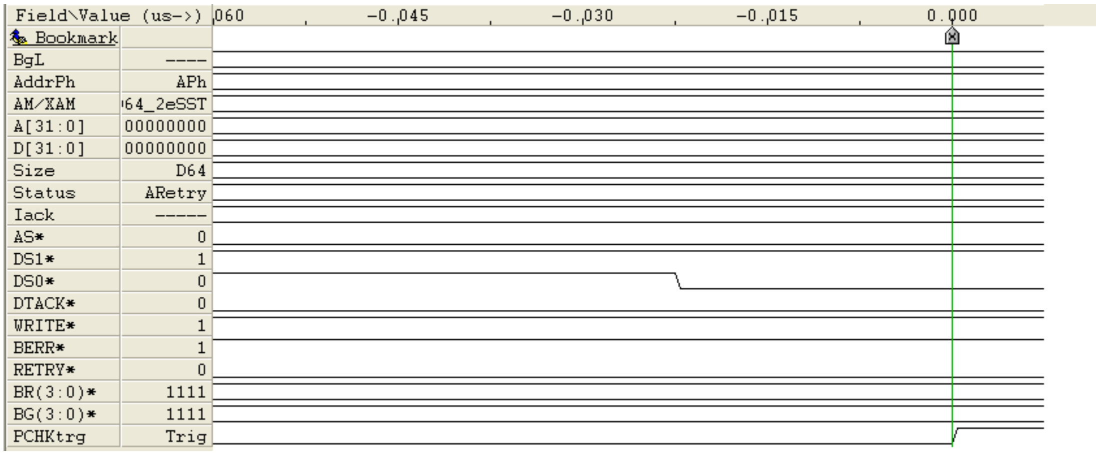
According to VMEx rule 11.18 [2, page 61] and 2eSST rule 3.10 [1, page 18], BERR* shall only be asserted on the first or third address phase instead of toggling DTACK*. If is asserted during address phase 2, an error is reported.



Field\Value	(us->)
Bookmark	
BgL	----
AddrPh	APh
AM/XAM	32_2eSST
A[31:0]	00000000
D[31:0]	00000000
Size	D32
Status	BERR
Iack	-----
AS*	0
DS1*	1
DS0*	1
DTACK*	0
WRITE*	1
BERR*	0
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

11.19 or 2eSST 3.11

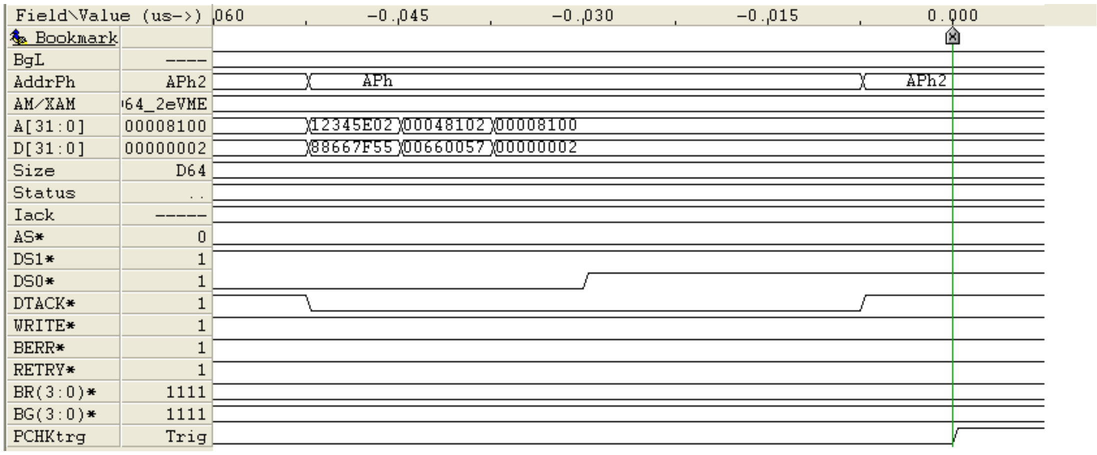
DTACK* shall be the only valid response during address phase two, 2eVME rule 11.19 [2, page 61] and 2eSST rule 3.11 [1, 18]. Since DTACK* is the only valid response in address phase two, DTACK* must be high at the end of address phase two. If this is not the case, an error is reported.



Field\Value	(us->)
Bookmark	
BgL	----
AddrPh	APh
AM/XAM	64_2eSST
A[31:0]	00000000
D[31:0]	00000000
Size	D64
Status	ARetry
Iack	-----
AS*	0
DS1*	1
DS0*	0
DTACK*	0
WRITE*	1
BERR*	1
RETRY*	0
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

11.5 or 2eSST 3.19

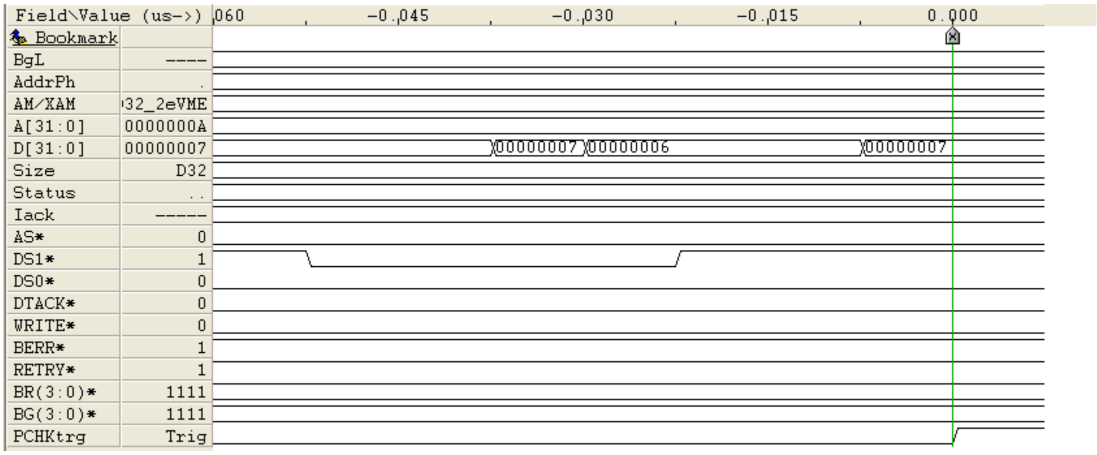
In the second address phase, the cycle count is transferred on A[15:8]. The cycle count is the number of data beats to be transferred divided by two. As the maximum number of data beats is 256, the maximum allowed cycle count is 128. For a 2eSST transfer, the protocol checker will issue an error if the cycle count is greater than 128. This is a violation of 2eSST rule 3.19 [1, page 20]. For 2eVME transfers, the protocol checker will issue an error if the cycle count is greater than 128, but not equal to 255. This is a violation of VME64x rule 11.15 [2, page 60]. A cycle count of 255 indicates that block length is unknown and slave termination expected, VME64x rule 11.10 [2, page 60].



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	APh2
AM/XAM	64_2eVME
A[31:0]	00008100
D[31:0]	00000002
Size	D64
Status	..
Iack	-----
AS*	0
DS1*	1
DS0*	1
DTACK*	1
WRITE*	1
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

11.6 or 2eSST 3.24

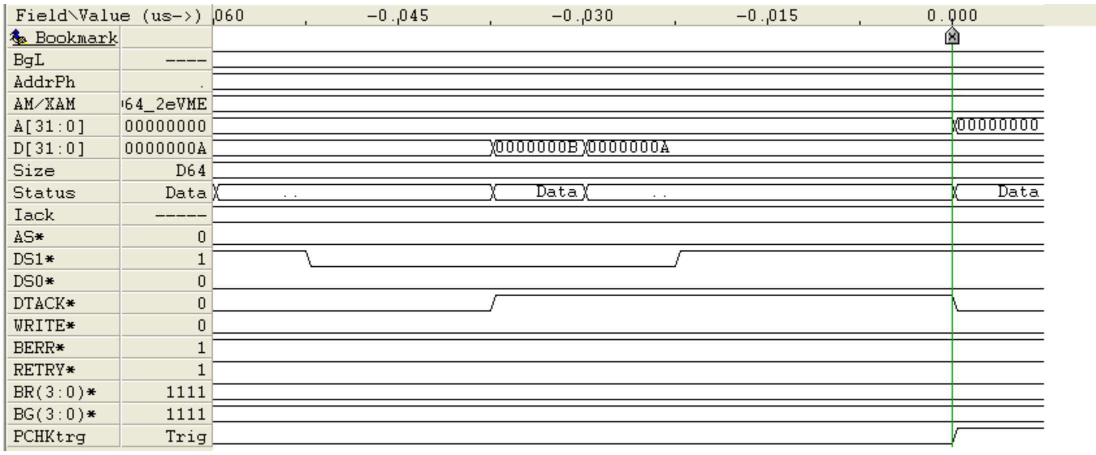
During 2eSST transfers the transmitter shall not cross any 2048 byte boundary, rule 3.23 [1, page 22]. 6U 2eVME transfers have the same requirement, but 3U 2eVME transfers must not cross any 1024 byte boundary, rule 11.6 [2, page 60]. The protocol checker stores the start address, counts the number of bytes transferred and reports any violation to these rules.



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	.
AM/XAM	32_2eVME
A[31:0]	0000000A
D[31:0]	00000007
Size	D32
Status	..
Iack	-----
AS*	0
DS1*	1
DS0*	0
DTACK*	0
WRITE*	0
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

11.11 or 2eSST 3.23

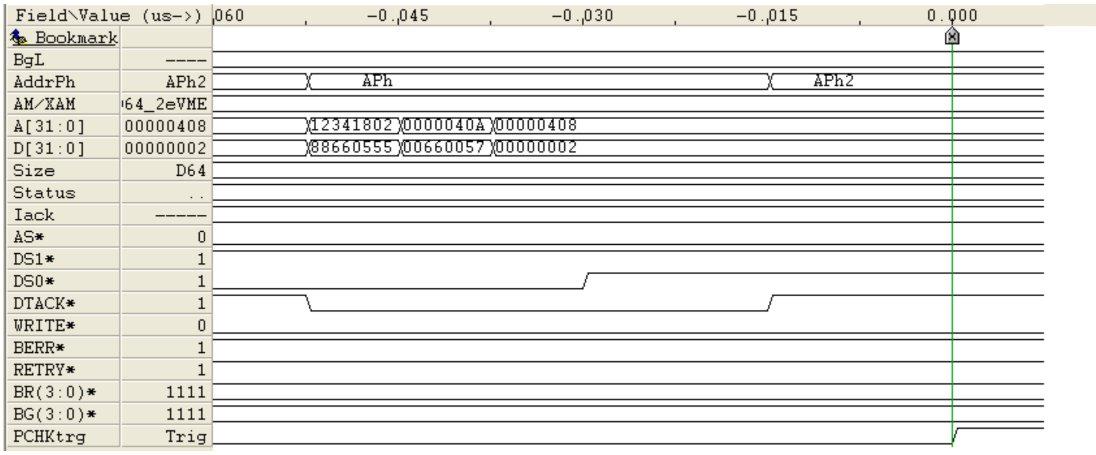
2eSST rule 3.23 [1, page 22] and VME64x rule 11.11 [1, page 60] specify that the transmitter shall not attempt to transfer more data during the address phase than is specified by the cycle count. If such an attempt is done, a error will be indicated by the protocol checker.



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	.
AM/XAM	'64_2eVME
A[31:0]	00000000
D[31:0]	0000000A
Size	D64
Status	Data
Iack	-----
AS*	0
DS1*	1
DS0*	0
DTACK*	0
WRITE*	0
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

11.15 or 2eSST 3.7

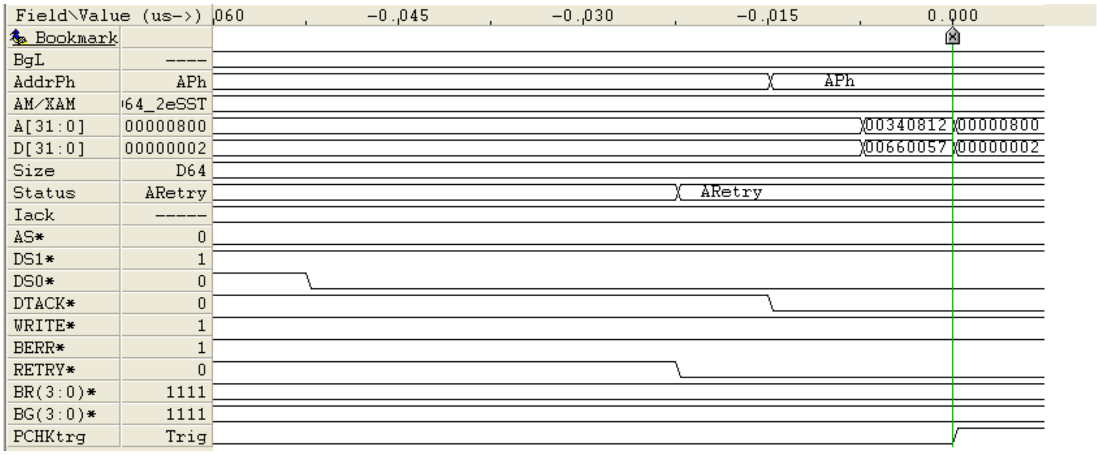
For both 2eSST and 2eVME transfers it is required that the starting address is aligned on 16-byte boundaries, 2eSST rule 3.7 [1, page 18] and VME64x rule 11.15 [2, page 61]. This requirement is equivalent to requiring the four least significant bits of A to be zero during address phase 2. An error is reported if they are not equal to zero.



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	APh2
AM/XAM	'64_2eVME
A[31:0]	00000408
D[31:0]	00000002
Size	D64
Status	. .
Iack	-----
AS*	0
DS1*	1
DS0*	1
DTACK*	1
WRITE*	0
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

2eSST 3.13 in address phase 1

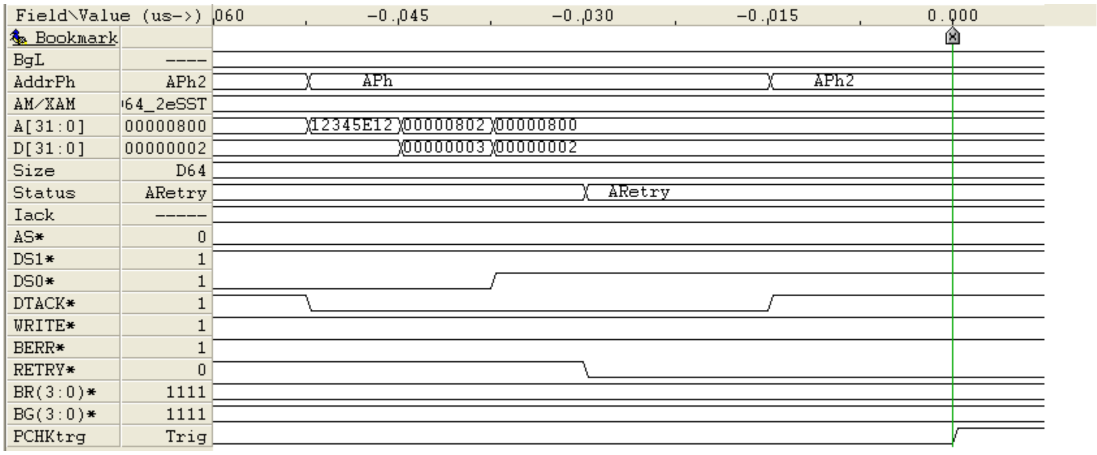
RETRY*/RESP* must not be asserted during address phase one or two of 2eSST transfers, rule 3.13 [1, page 18]. If the rule is violated, the protocol checker will report the error and in which address phase the error occurred.



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	APh
AM/XAM	'64_2eSST
A[31:0]	00000800
D[31:0]	00000002
Size	D64
Status	ARetry
Iack	-----
AS*	0
DS1*	1
DS0*	0
DTACK*	0
WRITE*	1
BERR*	1
RETRY*	0
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

2eSST 3.13 in address phase 2

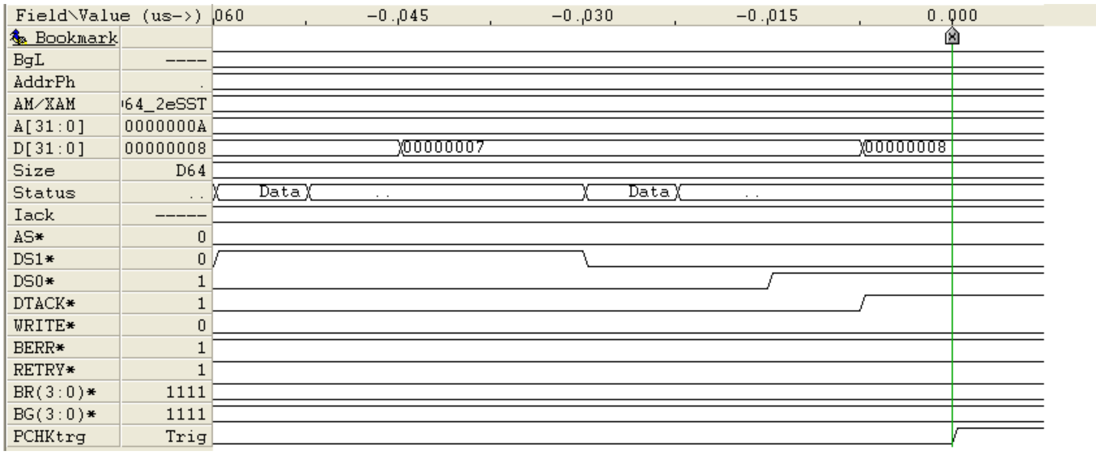
RETRY*/RESP* must not be asserted during address phase one or two of 2eSST transfers, rule 3.13 [1, page 18]. If the rule is violated, the protocol checker will report the error and in which address phase the error occurred.



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	APh2
AM/XAM	'64_2eSST
A[31:0]	00000800
D[31:0]	00000002
Size	D64
Status	ARetry
Iack	-----
AS*	0
DS1*	1
DS0*	1
DTACK*	1
WRITE*	1
BERR*	1
RETRY*	0
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

2eSST 3.28

For 2eSST transfers, rule 3.28 and 3.29 [1, pp 22-23] states that the master shall terminate the transfer by ending the data transfer on an even data beat. If the master attempts to transfer using an odd number of beats, an error will be reported.



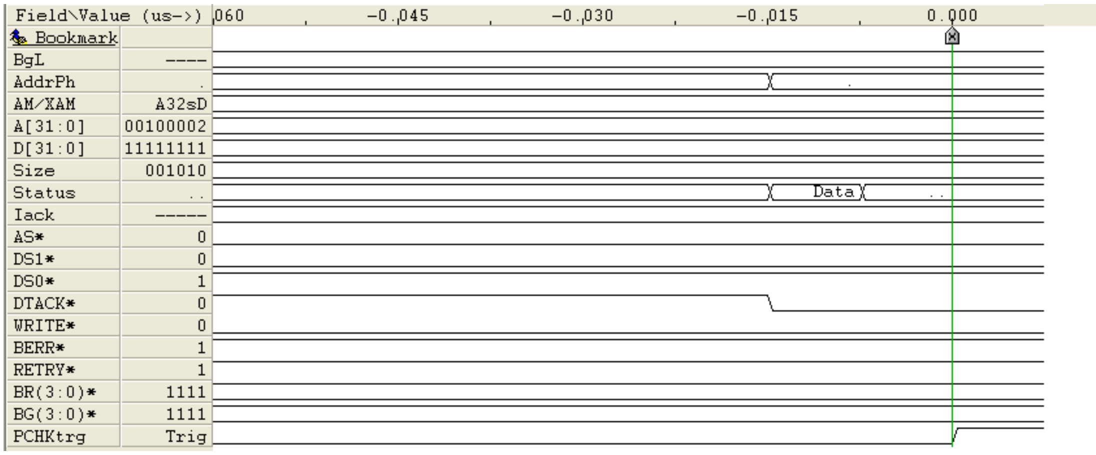
Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	.
AM/XAM	'64_2eSST
A[31:0]	0000000A
D[31:0]	00000008
Size	D64
Status	..
Iack	-----
AS*	0
DS1*	0
DS0*	1
DTACK*	1
WRITE*	0
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

2.1 Illegal Signal Combination

The "Rule 2.1 : Illegal signal combination" violation is triggered when either of the following illegal combinations occur :

DS1*=high, DS0*=low, A01=high, LWORD*=low
or
DS1*=low, DS0*=high, A01=high, LWORD*=low

To verify this error, the following signals must be inspected : DS0*, DS1*, A01, LWORD* and DTACK*.

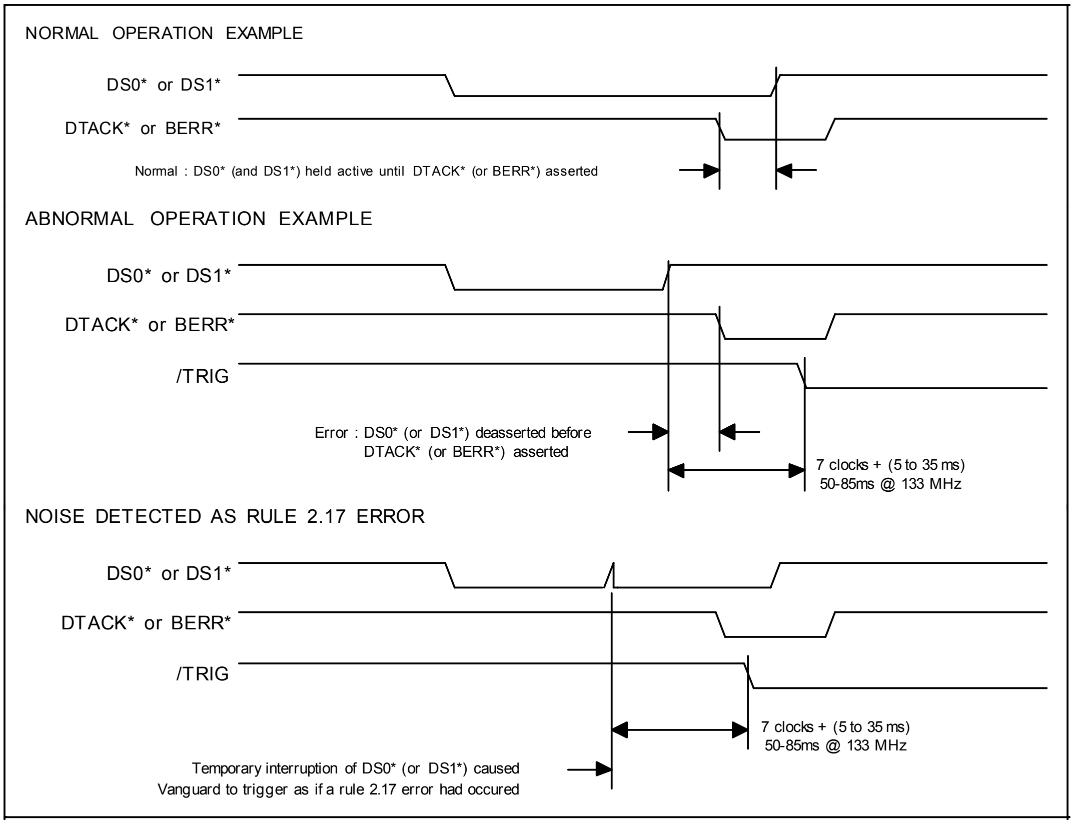


Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	.
AM/XAM	A32sD
A[31:0]	00100002
D[31:0]	11111111
Size	001010
Status	..
Iack	-----
AS*	0
DS1*	0
DS0*	1
DTACK*	0
WRITE*	0
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

2.17 DS0* off early

The "Rule 2.17 : DSx* off early" violation is triggered when DS0* or DS1* is rescinded before DTACK* or BERR* is received. To verify this error, the following signals must be inspected : DS0*, DS1*, DTACK* and BERR*.

The following diagram illustrates normal operation, an error condition, and a situation where noise is detected as this error.


NORMAL OPERATION EXAMPLE
DS0* or DS1*
DTACK* or BERR*
Normal : DS0* (and DS1*) held active until DTACK* (or BERR*) asserted
ABNORMAL OPERATION EXAMPLE
DS0* or DS1*
DTACK* or BERR*
/TRIG
Error : DS0* (or DS1*) deasserted before DTACK* (or BERR*) asserted
7 clocks + (5 to 35 ms) 50-85ms @ 133 MHz
NOISE DETECTED AS RULE 2.17 ERROR
DS0* or DS1*
DTACK* or BERR*
/TRIG
7 clocks + (5 to 35 ms) 50-85ms @ 133 MHz
Temporary interruption of DS0* (or DS1*) caused Vanguard to trigger as if a rule 2.17 error had occured

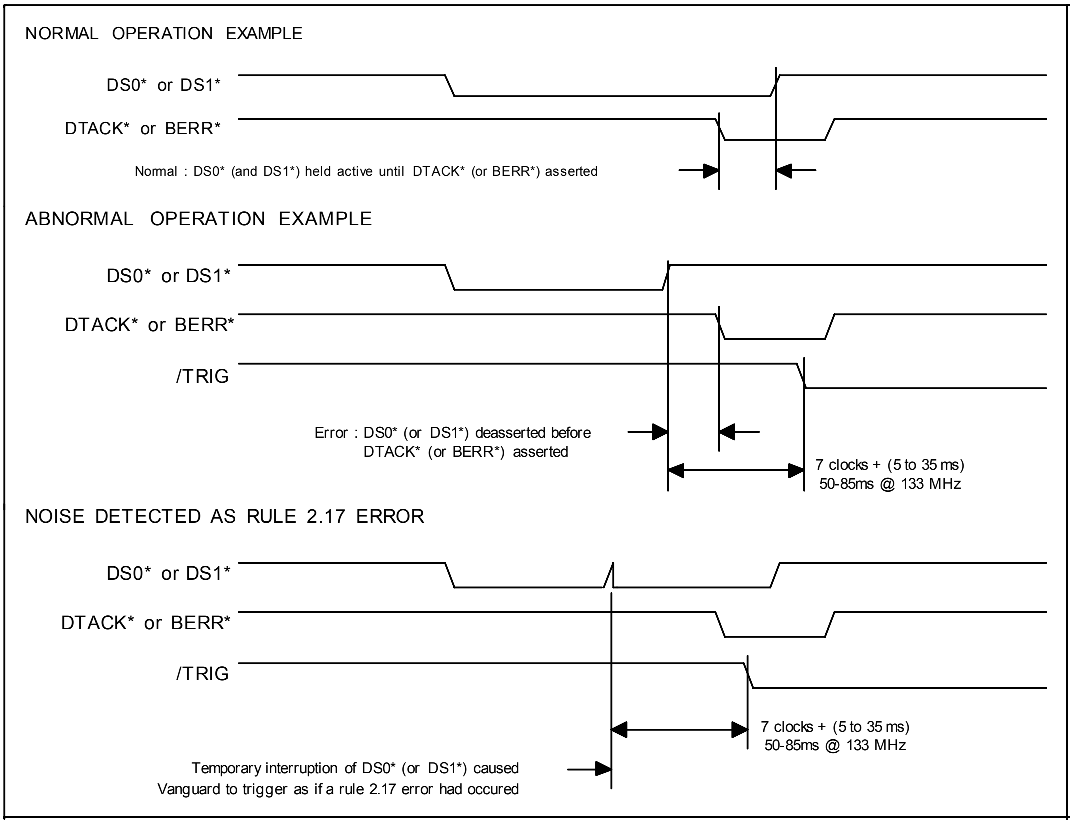
Conditions which could cause this error:

1. Design error in a bus master, in which one or both data strobes are rescinded before DTACK* is received.
2. Bus noise (usually due to ringing or ground bounce) which causes one of the data strobes to go momentarily inactive (high).
3. Wrong timeout settings. If, for example, a CPU board (master) has a local bus timeout shorter than the VMEbus timeout and attempts to access a non-existing VME address, a timeout on the local bus may illegally terminate the cycle before BERR* is received on VMEbus.

2.17 DS1* off early

The "Rule 2.17 : DSx* off early" violation is triggered when DS0* or DS1* is rescinded before DTACK* or BERR* is received. To verify this error, the following signals must be inspected : DS0*, DS1*, DTACK* and BERR*.

The following diagram illustrates normal operation, an error condition, and a situation where noise is detected as this error.



NORMAL OPERATION EXAMPLE
DS0* or DS1*
DTACK* or BERR*
Normal : DS0* (and DS1*) held active until DTACK* (or BERR*) asserted
ABNORMAL OPERATION EXAMPLE
DS0* or DS1*
DTACK* or BERR*
/TRIG
Error : DS0* (or DS1*) deasserted before DTACK* (or BERR*) asserted
7 clocks + (5 to 35 ms)
50-85ms @ 133 MHz
NOISE DETECTED AS RULE 2.17 ERROR
DS0* or DS1*
DTACK* or BERR*
/TRIG
7 clocks + (5 to 35 ms)
50-85ms @ 133 MHz
Temporary interruption of DS0* (or DS1*) caused Vanguard to trigger as if a rule 2.17 error had occured

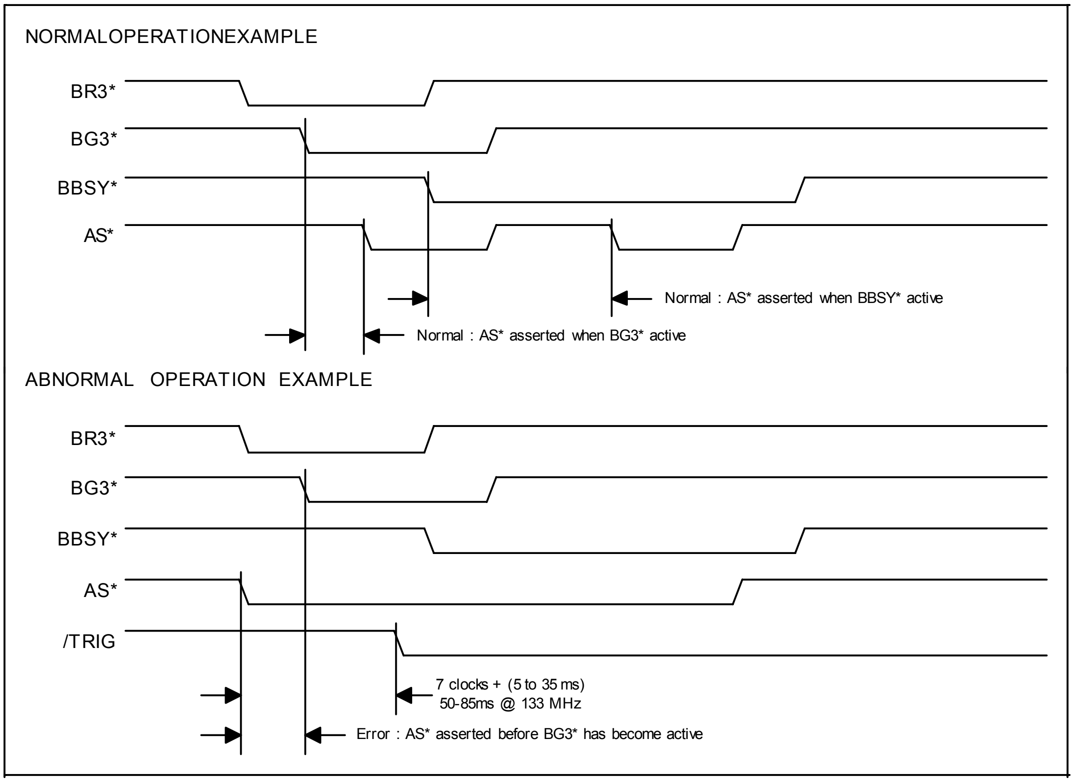
Conditions which could cause this error:

1. Design error in a bus master, in which one or both data strobes are rescinded before DTACK* is received.
2. Bus noise (usually due to ringing or ground bounce) which causes one of the data strobes to go momentarily inactive (high).
3. Wrong timeout settings. If, for example, a CPU board (master) has a local bus timeout shorter than the VMEbus timeout and attempts to access a non-existing VME address, a timeout on the local bus may illegally terminate the cycle before BERR* is received on VMEbus.

2.20 AS* before BG

The "Rule 2.20 : AS* before bus grant" violation is triggered when AS* has been driven low before the bus has been properly granted to the master. No master is permitted to drive AS* (as well as other signals dealt with below) until it receives it's bus grant, or unless BBSY* is already active. To verify this error, the following signals must be inspected : BG(3:0)*, BBSY*, AS* and DTACK*. If using an oscilloscope or external logic analyzer, then BG(3:0)* must be measured directly at the Vanguard slot.

The following diagram illustrates normal operation as well as an error condition. (BR3* and BG3* is shown in the diagram, but all pairs of Bus Request and Bus Grant lines have the same requirements).

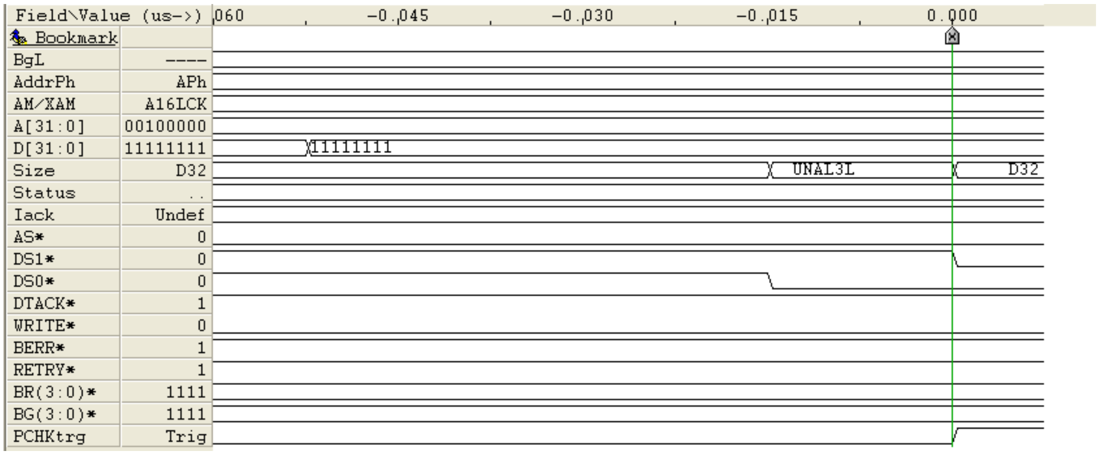

NORMALOPERATIONEXAMPLE
BR3*
BG3*
BBSY*
AS*
Normal : AS* asserted when BBSY* active
Normal : AS* asserted when BG3* active
ABNORMAL OPERATION EXAMPLE
BR3*
BG3*
BBSY*
AS*
/TRIG
7 clocks + (5 to 35 ms)
50-85ms @ 133 MHz
Error : AS* asserted before BG3* has become active

Note that this violation will not be seen if the Vanguard is located at a higher slot number than a master which is taking over the bus and therefore does not pass a bus grant down the daisy chain.

2.28 DSx* before BG

The "Rule 2.28 : DSx* before bus grant" violation is triggered when DS0* or DS1* has been driven low before the bus has been properly granted to the master. Operation is essentially the same as for "Rule 2.20 : AS* before bus grant" violation above, except that DS0* or DS1* should be substituted for AS* in the diagram. To verify this error, the following signals must be inspected : BG(3:0)*, BBSY*, DS0*, DS1* and DTACK*. If using an oscilloscope or external logic analyzer, then BG(3:0)* must be measured directly at the Vanguard slot.

Note that this violation will not be seen if the Vanguard is located at a higher slot number than a master which is taking over the bus and therefore does not pass a bus grant down the daisy chain.

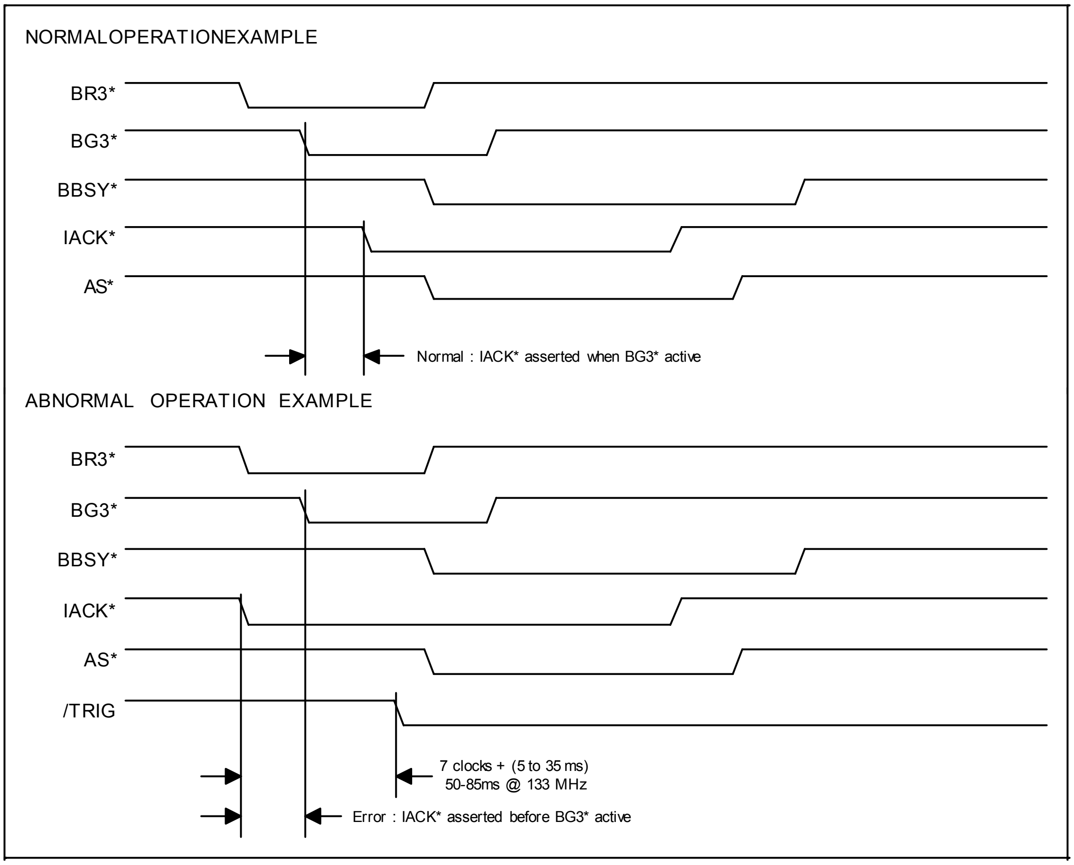


Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	APh
AM/XAM	A16LCK
A[31:0]	00100000
D[31:0]	11111111
Size	D32
Status	..
Iack	Undef
AS*	0
DS1*	0
DS0*	0
DTACK*	1
WRITE*	0
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

2.28 IACK* before BG

The "Rule 2.28 : IACK* before bus grant" violation is triggered when IACK* has been driven low before the bus has been properly granted to the master. Operation is essentially the same as for "Rule 2.20 : AS* before bus grant" violation above, except that IACK* should be substituted for AS* in the diagram. To verify this error, the following signals must be inspected : BG(3:0)*, BBSY*, IACK*, AS* and DTACK*. If using an oscilloscope or external logic analyzer, then BG(3:0)* must be measured directly at the Vanguard slot.

The following diagram illustrates normal operation as well as an error condition. (BR3* and BG3* is shown in the diagram, but all pairs of Bus Request and Bus Grant lines have the same requirements).

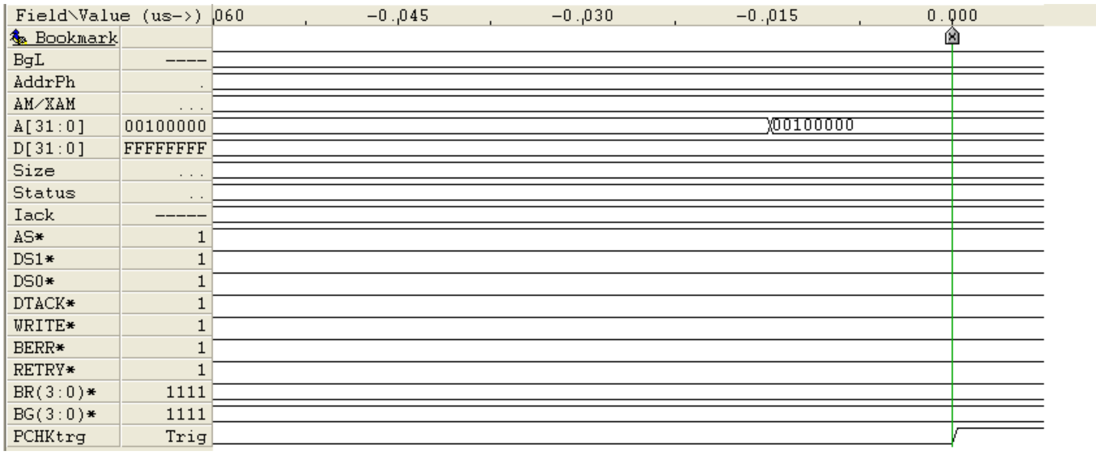


Note that this violation will not be seen if the Vanguard is located at a higher slot number than a master which is taking over the bus and therefore does not pass a bus grant down the daisy chain.

2.28 LWORD* before BG

The "Rule 2.28 : LWORD* before bus grant" violation is triggered when LWORD* has been driven low before the bus has been properly granted to the master. Operation is essentially the same as for "Rule 2.20 : AS* before bus grant" violation above, except that LWORD* should be substituted for AS* in the diagram. To verify this error, the following signals must be inspected : BG(3:0)*, BBSY*, LWORD*, AS* and DTACK*. If using an oscilloscope or external logic analyzer, then BG(3:0)* must be measured directly at the Vanguard slot.

Note that this violation will not be seen if the Vanguard is located at a higher slot number than a master which is taking over the bus and therefore does not pass a bus grant down the daisy chain.

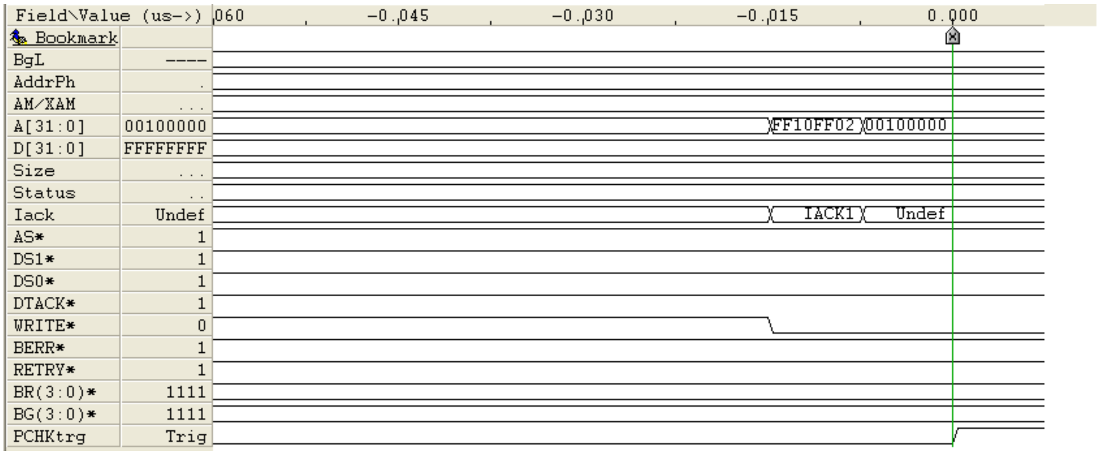


Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	.
AM/XAM	...
A[31:0]	00100000
D[31:0]	FFFFFFFF
Size	...
Status	..
Iack	-----
AS*	1
DS1*	1
DS0*	1
DTACK*	1
WRITE*	1
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

2.28 WRITE* before BG

The "Rule 2.28 : WRITE* before bus grant" violation is triggered when WRITE* has been driven low before the bus has been properly granted to the master. Operation is essentially the same as for "Rule 2.20 : AS* before bus grant" violation above, except that WRITE* should be substituted for AS* in the diagram. To verify this error, the following signals must be inspected : BG(3:0)*, BBSY*, WRITE*, AS* and DTACK*. If using an oscilloscope or external logic analyzer, then BG(3:0)* must be measured directly at the Vanguard slot.

Note that this violation will not be seen if the Vanguard is located at a higher slot number than a master which is taking over the bus and therefore does not pass a bus grant down the daisy chain.

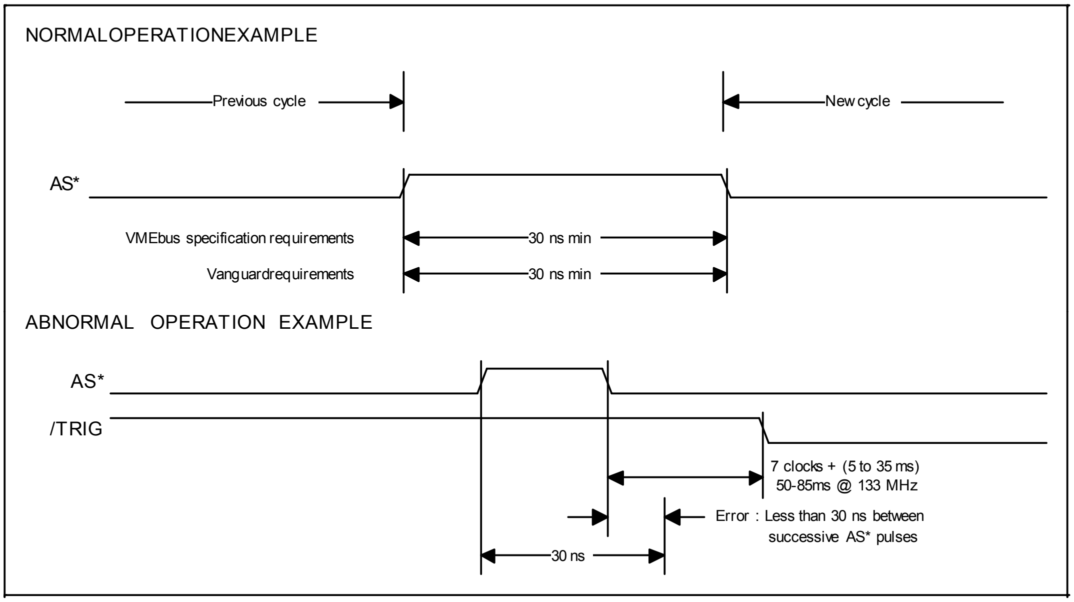


Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	.
AM/XAM	...
A[31:0]	00100000
D[31:0]	FFFFFFFF
Size	...
Status	..
Iack	Undef
AS*	1
DS1*	1
DS0*	1
DTACK*	1
WRITE*	0
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

2.31 AS* off<30ns

The "Rule 2.31 : AS* off < 30 ns" violation is triggered when AS* is inactive for too short a time period (< 30 ns) between successive bus cycles. To verify this error, the following signals must be inspected : AS* and DTACK*.

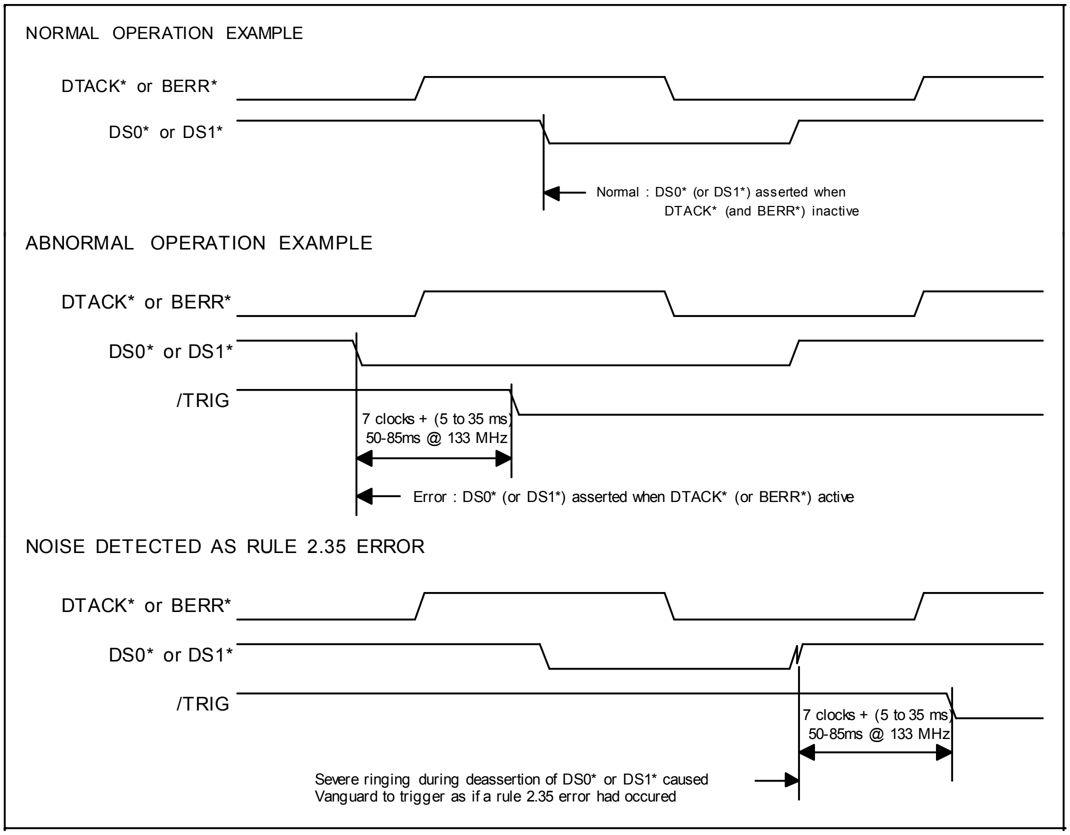
The following diagram illustrates normal operation as well as an error condition.



2.35 DSx* on early

The "Rule 2.35 : DSx* on early" violation is triggered when DS0* or DS1* are driven low before DTACK* and BERR* have gone high (been rescinded after the previous cycle). To verify this error, the following signals must be inspected : DS0*, DS1*, DTACK* and BERR*.

The following diagram illustrates normal operation, an error condition, and a situation where noise is detected as this error.


NORMAL OPERATION EXAMPLE
DTACK* or BERR*
DS0* or DS1*
Normal : DS0* (or DS1*) asserted when
DTACK* (and BERR*) inactive
ABNORMAL OPERATION EXAMPLE
DTACK* or BERR*
DS0* or DS1*
/TRIG
7 clocks + (5 to 35 ms)
50-85ms @ 133 MHz
Error : DS0* (or DS1*) asserted when DTACK* (or BERR*) active
NOISE DETECTED AS RULE 2.35 ERROR
DTACK* or BERR*
DS0* or DS1*
/TRIG
7 clocks + (5 to 35 ms)
50-85ms @ 133 MHz
Severe ringing during deassertion of DS0* or DS1* caused
Vanguard to trigger as if a rule 2.35 error had occured

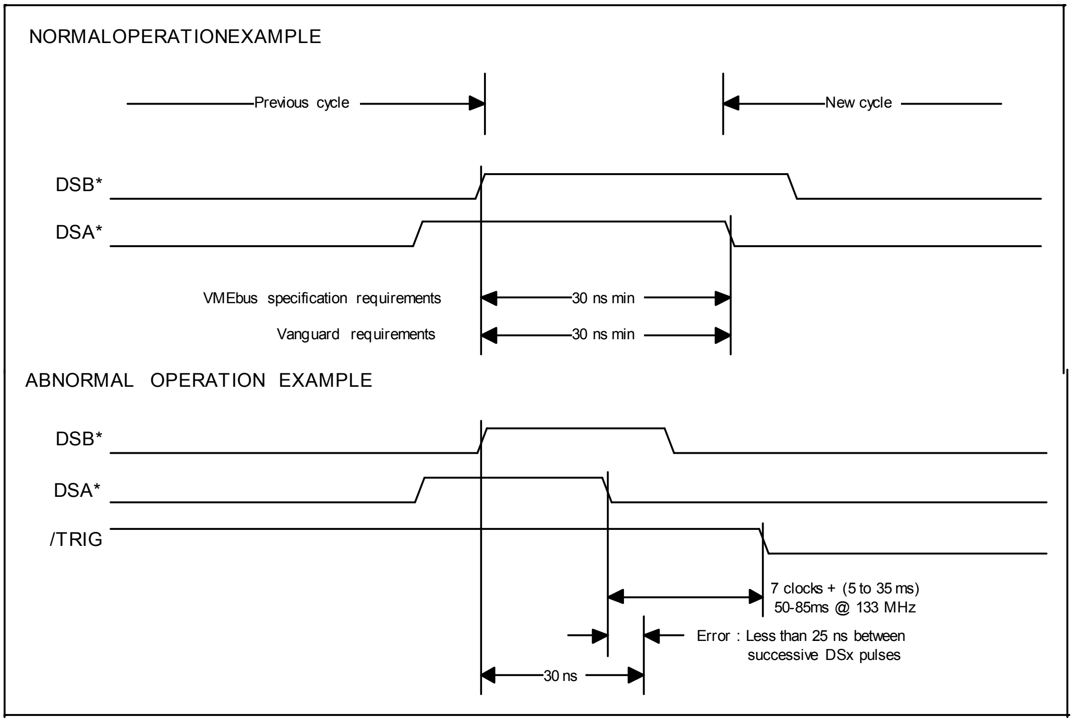
2.37 DSx* off <30ns

The "Rule 2.37 : DSx* off < 30 ns" violation is triggered when DS0* or DS1* are inactive for an insufficient time period between successive bus cycles (< 30 ns). To verify this error, the following signals must be inspected : DS0*, DS1* and DTACK*.

The following diagram illustrates normal operation as well as an error condition.

Note that DSA* and DSB* refer to the first and last transition, respectively, of data strobes DS0* and DS1*.

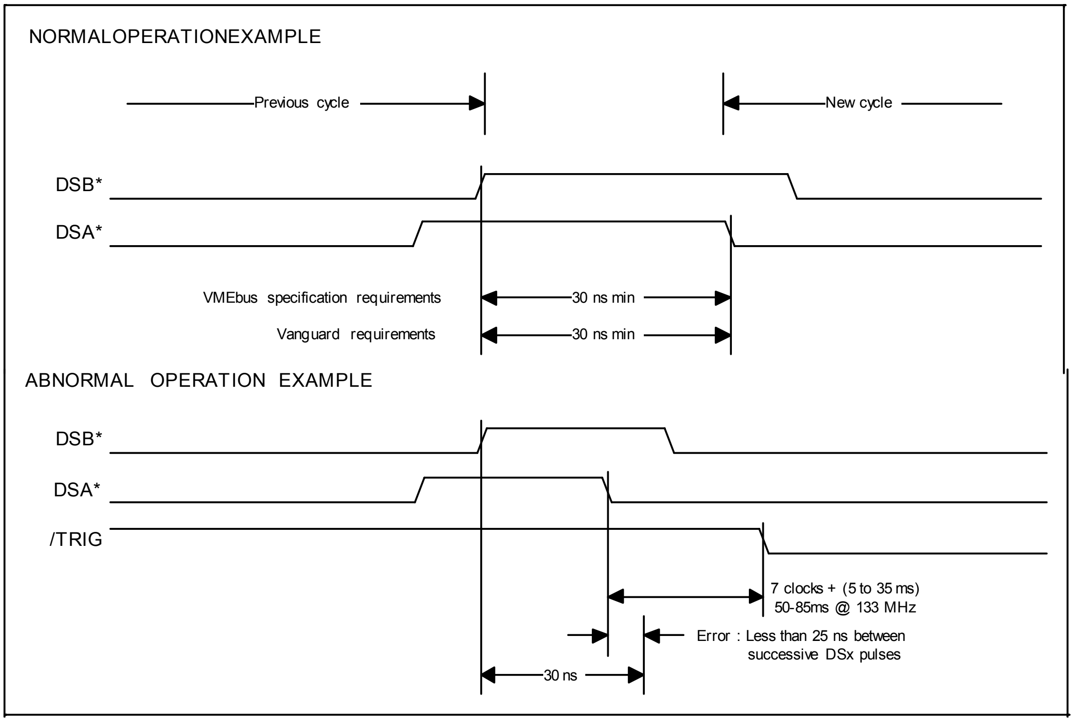
Note: In addition to its normal function, the "Rule 2.37 : DSx* off < 30 ns" violation may often be triggered by non-monotonic rising (ending) edges of DS0* or DS1*. Refer to the description of "Rule 2.39 : DSx* skew" violation which discussed a similar situation.



2.39 DSx* skew

The "Rule 2.39 : DSx* skew" violation is triggered when, during cycles where both data strobes are driven low, there is excessive skew (> 20 ns) between the starting edges of the first and second strobe. To verify this error, the following signals must be inspected : DS0*, DS1* and DTACK*.

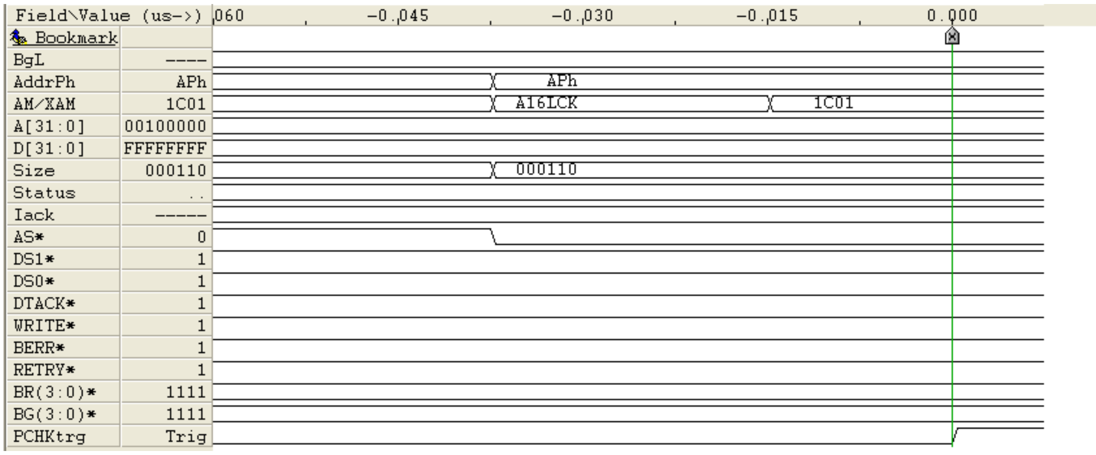
The following diagram illustrates normal operation and an error condition.



In addition to its normal function, the "Rule 2.39 : DSx* skew" violation may often be triggered by non-monotonic rising (ending) edges of DS0* or DS1*. Such behaviour is usually the result of a slow-rising DS0* or DS1* waveform which dwells near the threshold region for a long time, making it susceptible to multiple transitions through the receiver's threshold due to any coupled noise. Slow rise time is often caused by excessive bus capacitance due to board having PCB traces for DS0* and DS1* that are much longer than the specified 2 inches. Such noise is potentially serious since some slaves use the ending edges of DS0* or DS1* for incrementing counters etc. Multiple edges could cause trouble in that and other cases. Therefore, this alternative activator of this violation trigger should be investigated carefully.

2.43 IACK*, AM off early

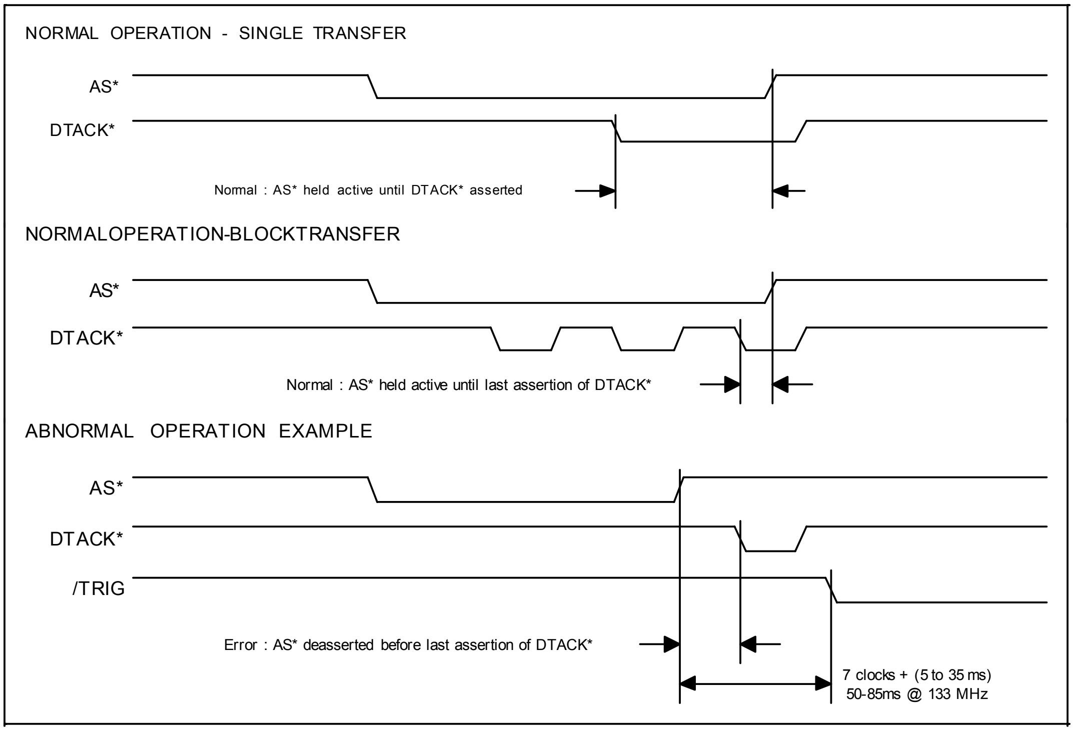
The master must not change the levels on IACK*, A, AM or LWORD* until 30 ns after AS* is driven low, rule 2.43 [3, page 86]. The protocol checker will only report an error if AM or IACK* changes from the time AS* is asserted and 30 ns thereafter, instabilities on LWORD* and A is not detected.



2.44 AS* off early

The "Rule 2.44 : AS* off early" violation is triggered when AS* is rescinded before the last falling edge on DTACK*, BERR* or RETRY* occurs. To verify this error, the following signals must be investigated : AS*, DTACK*, BERR*, DS0* and DS1*.

The following diagram illustrates the correct timing relationship between AS* and DTACK*/BERR* for single transfer and block transfer, as well as an error condition. (BERR* is not explicitly shown in the diagram, but has the same timing requirements as DTACK*).


NORMAL OPERATION - SINGLE TRANSFER
AS*
DTACK*
Normal : AS* held active until DTACK* asserted
NORMALOPERATION-BLOCKTRANSFER
AS*
DTACK*
Normal : AS* held active until last assertion of DTACK*
ABNORMAL OPERATION EXAMPLE
AS*
DTACK*
/TRIG
Error : AS* deasserted before last assertion of DTACK*
7 clocks + (5 to 35 ms)
50-85ms @ 133 MHz

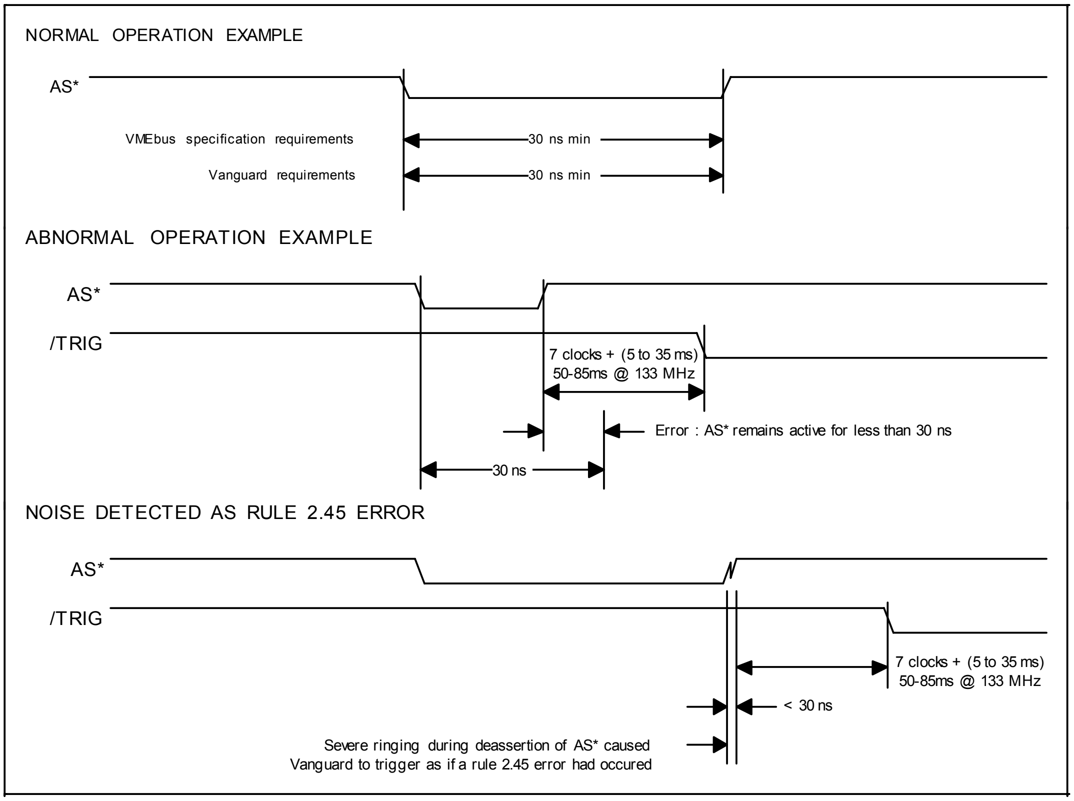
Conditions which could cause this error:

1. Design error in a bus master, in which AS* is rescinded before DTACK* is received.
2. Bus noise (usually due to ringing or ground bounce) which causes AS* to go momentarily inactive (high).
3. Wrong timeout settings. If, for example, a CPU board (master) has a local bus timeout shorter than the VMEbus timeout and attempts to access a non-existing VME address, a timeout on the local bus may illegally terminate the cycle before BERR* is received on VMEbus.

2.45 AS* on <30ns

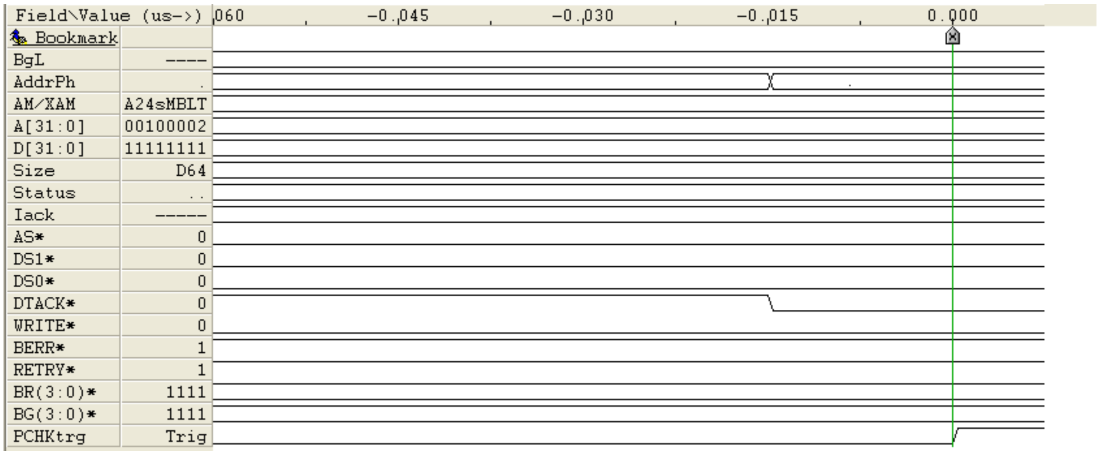
The "Rule 2.45 : AS* on < 30 ns" violation is triggered when AS* is active for too short a time (< 30 ns). To verify this error, inspect AS*.

The following diagram illustrates normal operation, an error condition, and an example of noise detected as this error.


NORMAL OPERATION EXAMPLE
AS*
VMEbus specification requirements
30 ns min
Vanguard requirements
30 ns min
ABNORMAL OPERATION EXAMPLE
AS*
/TRIG
7 clocks + (5 to 35 ms)
50-85ms @ 133 MHz
Error : AS* remains active for less than 30 ns
30 ns
NOISE DETECTED AS RULE 2.45 ERROR
AS*
/TRIG
7 clocks + (5 to 35 ms)
50-85ms @ 133 MHz
< 30 ns
Severe ringing during deassertion of AS* caused
Vanguard to trigger as if a rule 2.45 error had occured

2.69 VME64 Address error

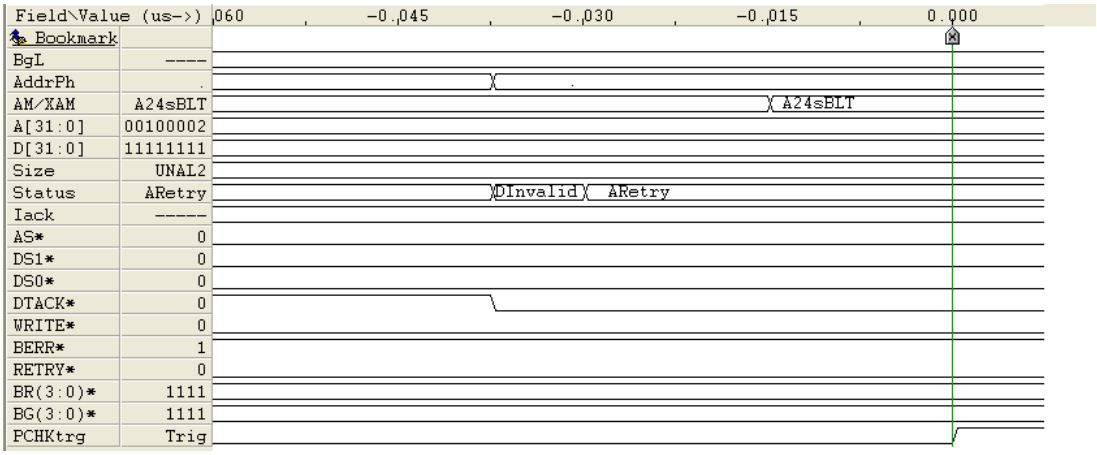
The "Rule 2.69 : VME64 address error" violation is triggered when A02, A01, DS1*, DS0* and LWORD* are not all driven low during the address phase of a 64-bit data cycle.



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	.
AM/XAM	A24sMBLT
A[31:0]	00100002
D[31:0]	11111111
Size	D64
Status	..
Iack	-----
AS*	0
DS1*	0
DS0*	0
DTACK*	0
WRITE*	0
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

2.96 Signals illegally changed when RETRY* active

A master that responds to RETRY* must not change the levels on IACK* and AM until it detects the rising edge of RETRY*, rule 2.96 [3, page 88]. If AM or IACK* change during the time RETRY* is low, an error is reported. Other properties described in rule 2.96 are not tested.

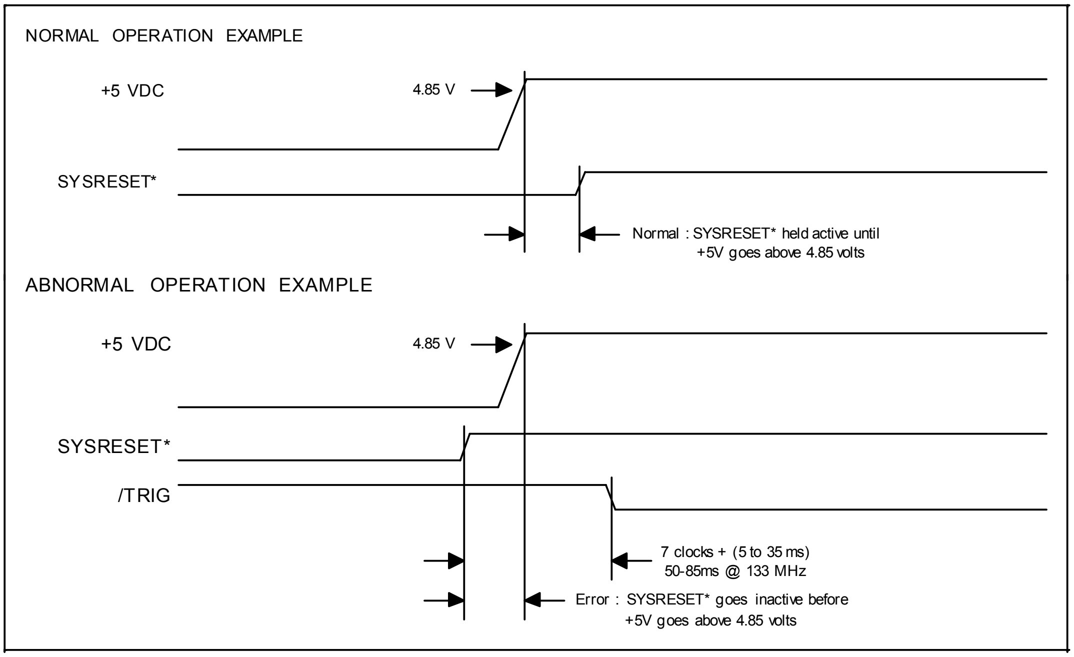


Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	.
AM/XAM	A24sBLT
A[31:0]	00100002
D[31:0]	11111111
Size	UNAL2
Status	ARetry
Iack	-----
AS*	0
DS1*	0
DS0*	0
DTACK*	0
WRITE*	0
BERR*	1
RETRY*	0
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

5.1 Power Monitors timing error

The "Rule 5.1 : SYSRESET* off < 4.85 V" violation is triggered when, during power up, the SYSRESET* signal goes away before the +5V power lines go above 4.85 volts. The VMEbus specification requires that SYSRESET* be held low until 200 ms after the +5V power goes to 4.875 volts. The Vanguard is more lenient, requiring the +5V lines to reach at least 4.85 volts before SYSRESET* is released.

The following diagram illustrates normal operation as well as an error condition.


NORMAL OPERATION EXAMPLE
+5 VDC
4.85 V
SYSRESET*
Normal : SYSRESET* held active until
+5V goes above 4.85 volts
ABNORMAL OPERATION EXAMPLE
+5 VDC
4.85 V
SYSRESET*
/TRIG
7 clocks + (5 to 35 ms)
50-85ms @ 133 MHz
Error : SYSRESET* goes inactive before
+5V goes above 4.85 volts

5V voltage low

The "5 V supply < 4.85 V" violation is triggered when the +5V power on the bus dips below 4.85 volts. This trigger is capable of detecting even very brief power dropouts. This type of problem is far more common than is generally realized.

Conditions which could cause this error:

1. The +5V power at the power supply may be too low a value.
2. The wiring which supplies the +5V power between the power supply and the backplane may be too thin, resulting in excessive voltage drop. This problem may generally be solved by connecting remote-sense power supply inputs at the backplane.

Hidden Grants

The "Grant hidden (Not error)" violation is triggered when the first error condition is detected, if the Vanguard was unable to fully detect errors on the bus grant daisy chain signals BG(3-0)*, because it was located to the right of the active master, and the daisy chain was therefore blocked.

This violation does not in itself indicate an error condition, but is a warning that one or more of the following error conditions may have been hidden from being detected by the Vanguard.

Multiple Grants: Rule 2.20 : AS* Before Bus Grant

BR0* Aborted: Rule 2.28 : IACK* Before Bus Grant

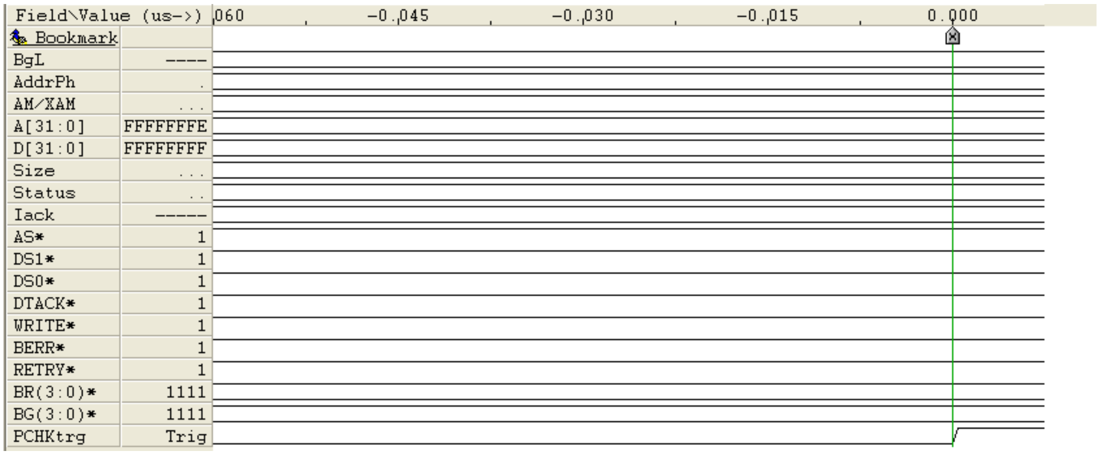
BR1* Aborted: Rule 2.28 : WRITE* Before Bus Grant

BR2* Aborted: Rule 2.28 : LWORD* Before Bus Grant

BR3* Aborted: Rule 2.28 : DSx* Before Bus Grant

Rule 3.6   : BG Before BBSY* Off

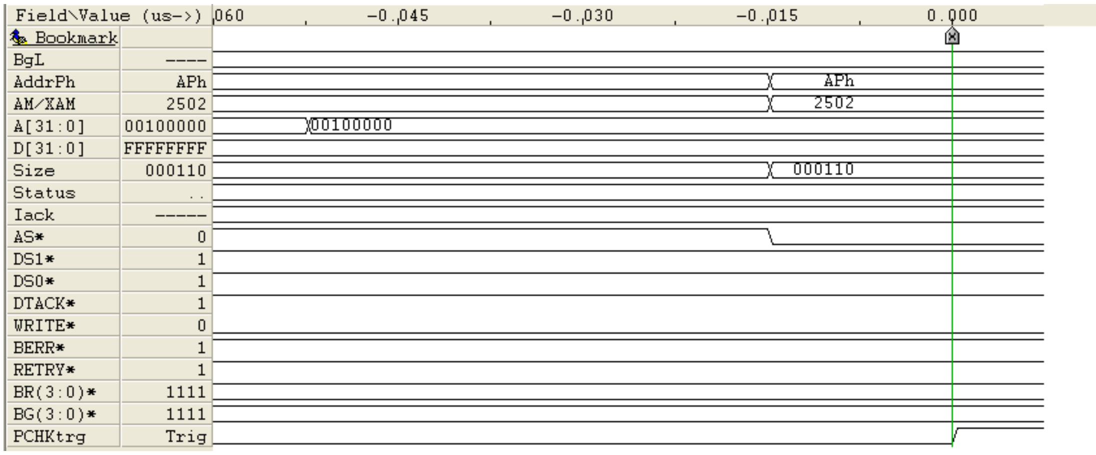
Rule 3.10 : BBSY* Off Early



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	.
AM/XAM	...
A[31:0]	FFFFFFFE
D[31:0]	FFFFFFFF
Size	...
Status	..
Iack	-----
AS*	1
DS1*	1
DS0*	1
DTACK*	1
WRITE*	1
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

Reserved AM code used

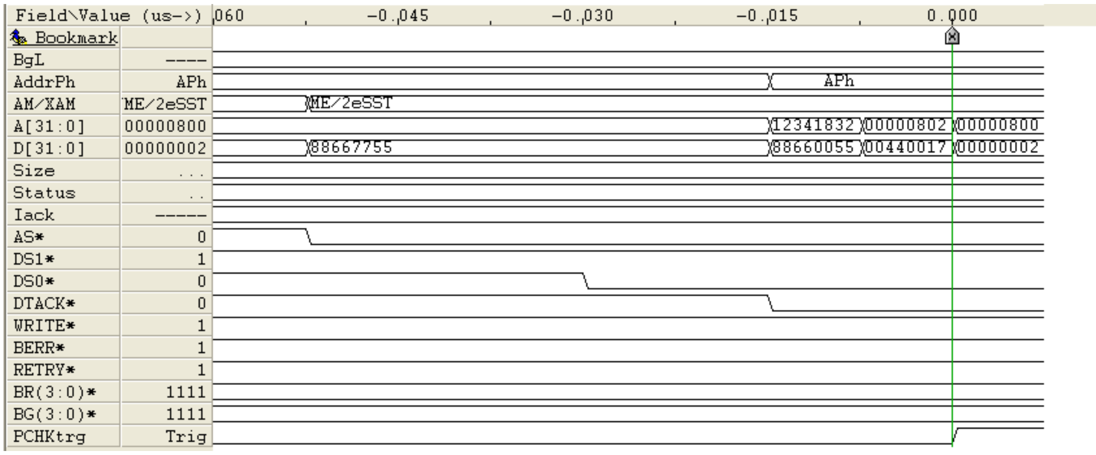
This is not a violation, it is just a warning that one of the reserved AM codes in table 2-3 in the VME64 specification [3, pp 21-22] has been used in a VME transfer that was not an interrupt acknowledge.



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	APh
AM/XAM	2502
A[31:0]	00100000
D[31:0]	FFFFFFFF
Size	000110
Status	..
Iack	-----
AS*	0
DS1*	1
DS0*	1
DTACK*	1
WRITE*	0
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

Reserved XAM code used

The XAM code is checked on the first falling edge of DTACK* in 2eSST and 2eVME cycles. If the code is not among the ones defined in the specifications for 2eSST[1], page 12, or VME64x[2], page 58, a warning is issued.



Field\Value (us->)	
Bookmark	
BgL	----
AddrPh	APh
AM/XAM	ME/2eSST
A[31:0]	00000800
D[31:0]	00000002
Size	...
Status	..
Iack	-----
AS*	0
DS1*	1
DS0*	0
DTACK*	0
WRITE*	1
BERR*	1
RETRY*	1
BR(3:0)*	1111
BG(3:0)*	1111
PCHKtrg	Trig

7 Exerciser

The Exerciser is used to generate cycles with known characteristics on the VME 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 313.

7.1 Overview

Features

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 (VME)

Data can be written to and read from this memory. This way the Exerciser can emulate a device or VME board. The Exerciser also has the following properties:

- VMEbus System Controller
- Autoslot ID mechanism
- VMEbus Requestor (Level 0-3)
- VMEbus Interrupter and Interrupt handler (IRQ1-7)
- VMEbus master/slave A32, A24, A16:D32, D16, D8, UAT
- VMEbus master/slave A32, A24:D32BLT, D64MBLT
- VMEbus master/slave A64, D32, D32BLT, D64MBLT
- VMEbus master/slave 2eSST: A64, A32
- VMEbus master/slave 2eSST Broadcast: A64, A32
- BLT/MBLT cycle

Generate Interrupts

The Exerciser can generate VME Interrupts.

7.2 Introduction

The main features of the Exerciser

- Provides D32, D16, D8 and UAT under A32, A24 and A16 addressing modes plus D64 BLT and D32BLT under A64, A32 and A24 addressing modes.
- As a VME master in DMA mode, the Exerciser provides A64, A32, broadcast A32 and broadcast A64 2eSST transaction.
- Provides real VME 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 VME board and has 8 MByte slave memory available for use.
- DMA transfers can be set up with the Exerciser acting as a “slave” or a “master”.
- Can generate any VME interrupts.

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.

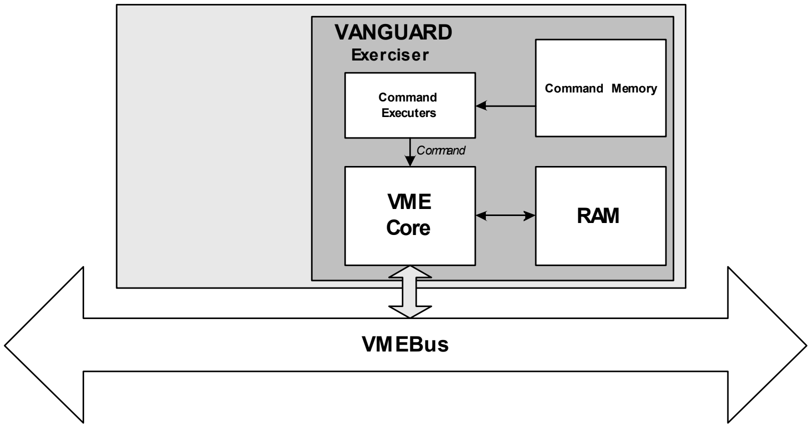


FIGURE 7-1 *Exerciser Block Diagram*

Command Memory This area contains the list of commands that will be executed onto the VME bus.

Command Executer Controls the VME Core. It reads commands from the Command Memory and sends status reports back to BusView.

VME Core These are the FPGA images used to act as a VME Exerciser.

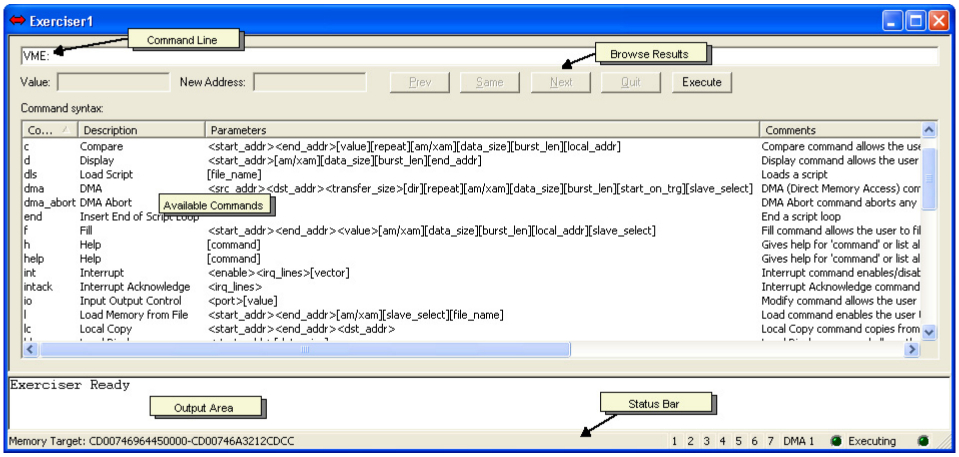
RAM This memory area is for use in DMA transfers and in mapping target memory.

7.3 Operation

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.

Exerciser Window

The Exerciser Window provides a command line interface to issue commands, and an area where the results of issued commands are displayed.



VME:

Command syntax:

Co...	Description	Parameters	Comments
c	Compare	<start_addr><end_addr>[value][repeat][am/xam][data_size][burst_len][local_addr]	Compare command allows the use
d	Display	<start_addr>[am/xam][data_size][burst_len][end_addr]	Display command allows the user
dls	Load Script	[file_name]	Loads a script
dma	DMA	<src_addr><dst_addr><transfer_size>[dir][repeat][am/xam][data_size][burst_len][start_on_trg][slave_select]	DMA (Direct Memory Access) com
dma_abort	DMA Abort		DMA Abort command aborts any
end	Insert End of Script Loop		End a script loop
f	Fill	<start_addr><end_addr><value>[am/xam][data_size][burst_len][local_addr][slave_select]	Fill command allows the user to fil
h	Help	[command]	Gives help for 'command' or list al
help	Help	[command]	Gives help for 'command' or list al
int	Interrupt	<enable><irq_lines>[vector]	Interrupt command enables/disab
intack	Interrupt Acknowledge	<irq_lines>	Interrupt Acknowledge command
io	Input Output Control	<port>[value]	Modify command allows the user
l	Load Memory from File	<start_addr><end_addr>[am/xam][slave_select][file_name]	Load command enables the user
lc	Local Copy	<start_addr><end_addr><dst_addr>	Local Copy command copies from

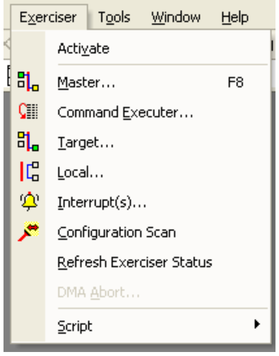
```
Exerciser Ready
```

Memory Target: CD00746964450000-CD00746A3212CDCC   1 2 3 4 5 6 7 DMA 1 ● Executing ●

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 (Slave) 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 215.

Help

The Exerciser provides several ways of getting help.

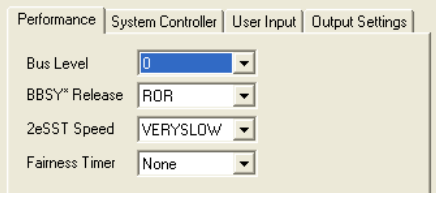
- When typing a command in the command line, the text color will change. When the text is red, this means that the command does not yet have all necessary parameters to be executed.
- Context sensitive help. Type a command in the Exerciser window, and press Ctrl-F1 keys. The on-line help opens at the 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

Right-click on the Exerciser Window and click Options to open the Exerciser Options Window.

Performance Options



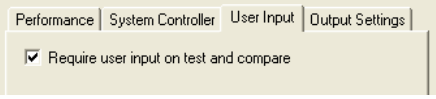
Bus Level 0
BBSY* Release ROR
2eSST Speed VERYSLOW
Fairness Timer None

FIGURE 7-3 *Exerciser Performance Options*

- Bus Level -Bus Level to use for the commands 0 - 3
- BBSY* Release - BBSY* Release type. Available options are:
  - ROR -Release On Request.
  - RWD - Release When Done.
  - ROC - Release On Clear.
  - RNE - Release NEver.
- 2eSST Speed - Options are:
  - VERYSLOW - SST160, effective rate 146MBytes/s.
  - SLOW - SST267, effective rate 171MBytes/s.
  - MEDIUM - SST267, effective rate 205MBytes/s.
  - FAST - SST267, effective rate 256MBytes/s (default).
  - ULTRAFAST - SST320+, effective rate 341MBytes/s.
- Fairness Timer - Options are:
  - Infinite - Vanguard does not assert a VME request if another request is already pending on the same level.
  - 20 us - Assert request after 20 micro seconds or when released by other devices.
  - None - Vanguard asserts the request immediately.

These options can also be set via the command line prompt with the *opt* command:

User Input



☑ Require user input on test and compare

opt <enable>[bus_level] [BBSY_release] [2eSST_speed] [fairness_timer]

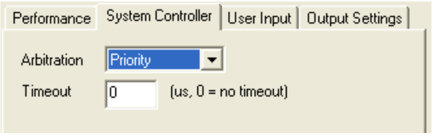
Argument			**Description**	**Default**
Required	**Optional**	**Type**		
enable		BOOL	1= Enable Exerciser, 0= Disable Exerciser	0
	bus_level	number	Bus Level to use for the commands 0 - 3	3

	BBSY_release	text	BBSY* release type
			ROR = Release On Request.
			RWD = Release When Done.
			ROC = Release On Clear.
			RNE = Release NEver.
				ROR

	2eSST_speed	text	VERYSLOW = SST160, effective rate 146MBytes/s.
			SLOW = SST267, effective rate 171MBytes/s.
			MEDIUM = SST267, effective rate 205MBytes/s.
			FAST = SST267, effective rate 256MBytes/s (default).
			ULTRAFAST = SST320+, effective rate 341MBytes/s.
				FAST

	fairness_timer	number	0= None 1= 20us 2= Infinite	0

System Controller



Arbitration Priority

Timeout 0 (us, 0 = no timeout)

- Arbitration - Select the type of VME bus arbitration. Round Robin, or Priority.

- Timout - Timeout for data transfers on the VME bus in micro seconds.

These options can also be set via the command line prompt with the *optx* command:
optx <enable>[arbitration_type] [timeout]

Argument			**Description**	**Default**
Required	**Optional**	**Type**		
	enable	BOOL	1= set parameters 0= reset to default	0

	arbitration_type	text	rr= Round Robin
			p = Priority	

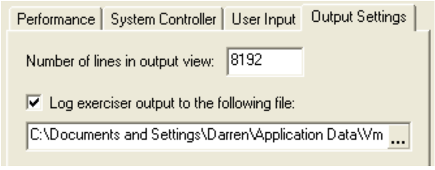
	timeout	number	Data to bus error timout in microseconds 0 (default) - 255us. Zero means timer is inactive (no timeout)	

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



Number of lines in output view: 8192

☑ Log exerciser output to the following file:

C:\Documents and Settings\Darren\Application Data\Vm

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.

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:

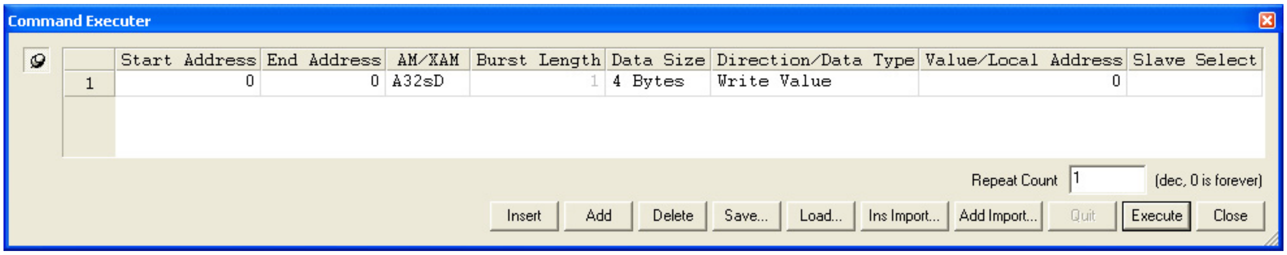
- <argument> = required.
- [<argument>] = optional.

When using the command line interface, all required arguments have to be specified. When using dialog boxes, the arguments have default values.

Executing Multiple Commands

Command Executer

The Command Executer is used to issue a list of Cycle Sequences. When executed, the whole list is transferred to the Vanguard, where it is run in hardware - as fast as possible.



	Start Address	End Address	AM/XAM	Burst Length	Data Size	Direction/Data Type	Value/Local Address	Slave Select
1	0	0	A32sD	1	4 Bytes	Write Value	0	

Repeat Count 1 (dec, 0 is forever)

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 215.

7.5 Summary of Commands

Master Commands		Local Commands		Miscellaneous Commands		Script Commands	
Modify	page 183	Local Display	page 202	Refresh	page 208	Start Recording	page 216
Write	page 184	Local Modify	page 203	Target	page 209	Stop Recording	page 216
Fill	page 185	Local Fill	page 204	Interrupt	page 210	Load Script	page 217
DMA	page 187	Local Copy	page 205	Interrupt Acknowledge	page 213	Run Script	page 217
Test	page 190	Local Load	page 206	Exerciser Options	page 177	Wait	page 217
Compare	page 192	Local Save	page 207	Help	page 176	Pause	page 217
Cycle Sequence	page 194			Intexe	page 211	Loop	page 218
Exercise	page 196			IO	page 212	End	page 218
Load Memory	page 198					Show	page 218
Save Memory	page 200					Comment	page 218

Where AM/XAM codes are used, the following table is valid:

TABLE 7-1. Mnemonics for AM/XAM hex codes

Mnemonic	Value(hex)	Mnemonic	Value(hex)
A32LCK	05	CRCSR	2F
A32nMBLT*	08	---------------	----
A32nD	09	A24LCK	32
A32nP	0A	A24nMBLT*	38
A32nBLT*	0B	A24nD	39
A32sMBLT*	0C	A24nP	3A
A32sD	0D	A24nBLT*	3B
A32sP	0E	A24sMBLT*	3C
A32sBLT*	0F	A24sD	3D
---------------	------	A24sP	3E
A64MBLT*	00	A24sBLT*	3F
A64D	01	-------------	------
A64BLT*	03	A16nD	29
A64LCK	04	A16LCK	2C
-----------------	----	A16sD	2D
A32D64_2eSST	2011		
A64D64_2eSST	2012		
A32D64_2eSSTBC **	2021		
A64D64_2eSSTBC **	2022		

* Not available in Modify

** Only used in commands that perform a Write operation.

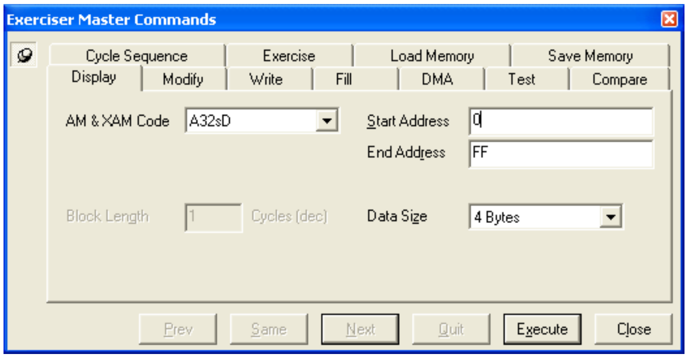
Display

Displays Memory space in 256 Byte blocks.

Command Line format:

d <start_addr>[am/xam][<data_size>][<block_len>] [<end_addr>]

Argument			**Description**	**Default**
Required	**Optional**	**Type**		
start_addr		Hex number	Start address of area to display.	
	am/xam	char	Address modifier hex value in range 00-3F if am = 20 the value should be a 4 digit hexadecimal number where the 1. byte is 20 and the 2. byte is the xam code. Valid xam codes are 11, 12, 21 or 22. The modifier can also be given as mnemonic instead of hex, see Table 7-1 on page 181	0D or A32sP
	data_size	unsigned int	Size of each data object 1, 2, 4 or 8 bytes	4
	block_len	unsigned int	block length in cycles 1-256	1
	end_addr	hex number	End address of the area to display. Maximum is start_addr + FFFF	FF



AM & XAM Code: A32sD

Start Address: 0

End Address: FF

Block Length: 1 Cycles (dec)

Data Size: 4 Bytes

Example:`d 80001000 A32sD 4 0`

`d` = Display command
`1000` = start address
A32sD = am code
`4` = 4 byte accesses (DWORD)
`0` = Number of Cycles

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 `start_addr+0x7f`, only `0x80` bytes of data are displayed in each block.

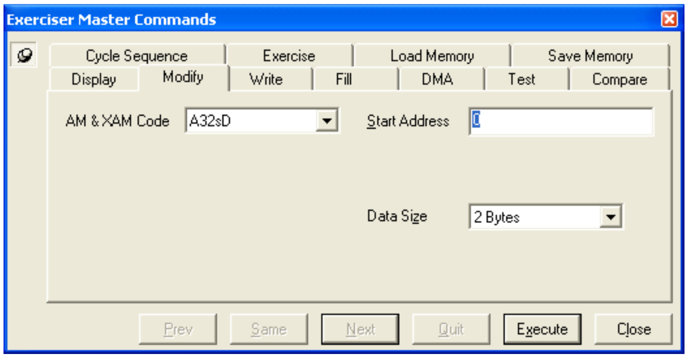
Modify

Read and Modify data in memory space.

Command Line format:

m <start_addr>[<am>] [<data_size>]

Argument			Description	Default
Required	**Optional**	**Type**		
start_addr		Hex number	Start address of area to display.	
	am	char	Address modifier hex value in range 00-3F The modifier can also be given as mnemonic instead of hex, see Table 7-1 on page 181	0D or A32sP
	data_size	unsigned int	Size of each data object 1, 2, 4 or 8 bytes	4



AM & XAM Code: A32sD

Start Address: 0

Data Size: 2 Bytes

Example:`m 1000 A32sD 4`

`m` = Modify command
`1000` = start address
A32sD = am code
`4` = 4 byte accesses

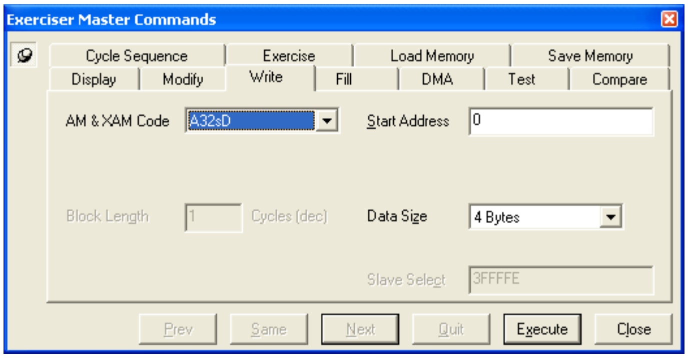
Write

Used to write data into Memory space.

Command Line format:

w <start_addr>[<am/xam>][<data_size>] [<block_len>] [<slave_select>]

Argument			Description	Default
Required	Optional	Type		
start_addr		Hex number	Start address of area to display.	
	am/xam	char	Address modifier hex value in range 00-3F if am = 20 the value should be a 4 digit hexadecimal number where the 1. byte is 20 and the 2. byte is the xam code. Valid xam codes are 11, 12, 21 or 22. The modifier can also be given as mnemonic instead of hex, see Table 7-1 on page 181	0D or A32sP
	data_size	unsigned int	Size of each data object 1, 2, 4 or 8 bytes	4
	block_len	unsigned int	block length in cycles 1-256	1
	slave_select	Hex number	The slave select address for 2eSST broadcast. Third address phase as 32 bits value where A[21:1] is used.	0x3FFFFE (all selected)



AM & XAM Code: A32sD

Start Address: 0

Block Length: 1 Cycles (dec)

Data Size: 4 Bytes

Slave Select: 3FFFFE

Example:`w 1000`

`w` = Write command
`1000` = start address.

When the block 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 block length.

Fill

Fills Memory, I/O space with a given pattern or value.

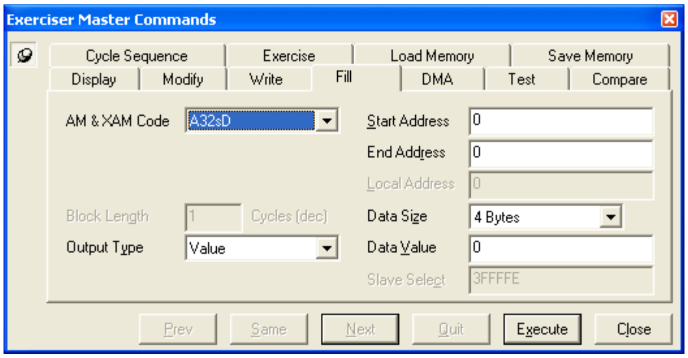
Command Line format:

f <start_addr><end_addr><value>
[<am/xam>][<data_size>][<block_len>][<local_addr>][<slave_select>]]

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
			char values:
			o = Walking one pattern
			z = Walking zero pattern
			s = address as data
			r = random data
			l = use data in “local addr”
				o

	am/xam	char	Address modifier hex value in range 00-3F if am = 20 the value should be a 4 digit hexadecimal number where the 1. byte is 20 and the 2. byte is the xam code. Valid xam codes are 11, 12, 21 or 22. The modifier can also be given as mnemonic instead of hex, see Table 7-1 on page 181	0D or A32sP
	data_size	unsigned int	Size of each data object 1, 2, 4 or 8 bytes	4
	block_len	unsigned int	block length in cycles 1-256	1
	local_addr	hex number	Local memory address of data pattern	
	slave_select	Hex number	The slave select address for 2eSST broadcast. Third address phase as 32 bits value where A[21:1] is used.	0x3FFFFE (all selected)



AM & XAM Code: A32sD
Start Address: 0
End Address: 0
Local Address: 0
Block Length: 1 Cycles (dec)
Data Size: 4 Bytes
Output Type: Value
Data Value: 0
Slave Select: 3FFFFE

Example:`f 1000 10ff o A32sD 4 8`

`f` = Fill command
`1000` = start address
`10ff` = end address
`o` = Walking One fill pattern
A32sD = am code
`4` = 4 bytes data size and 8 = block length in cycles

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.

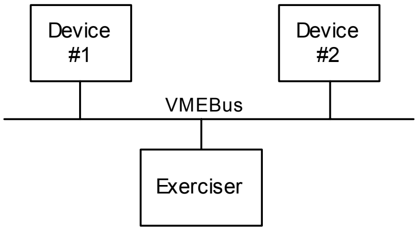
DMA

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 1 DMA channel available.

DMA Transfers can be made in three different ways:

- Local to VME - Transfers data from Local (Target) memory to a VME location. i.e from Exerciser to Device #1 in the diagram shown.
- VME to Local - Transfers data from a VME location to local memory. i.e. from Device #1 to the Exerciser in the diagram shown.
- VME to VME - Transfers data from a VMElocation to local memory, and then from local memory to another VMElocation. 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.

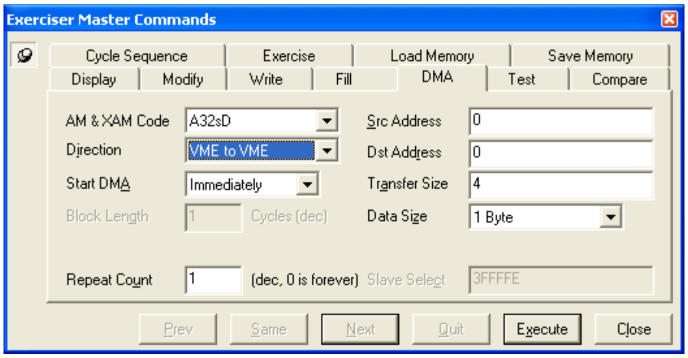
Command Line format:

dma <src_addr> <dst_addr> <transfer_size> <dir> [<repeat>] [<am/xam>] [<data_size>] [<block_len>] [start_on_trg] [<slave_select>]]

Argument			**Description**	**Default**
Required	**Optional**	**Type**		
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]	vv = VME to VME memory
			vl = VME to local memory
			lv = local to VME memory	

	repeat	unsigned_int	Number of times to repeat the DMA transfer 0 = forever	1
	am/xam	char	Address modifier hex value in range 00-3F if am = 20 the value should be a 4 digit hexadecimal number where the 1. byte is 20 and the 2. byte is the xam code. Valid xam codes are 11, 12, 21 or 22. The modifier can also be given as mnemonic instead of hex, see Table 7-1 on page 181	0D or A32sP
	data_size	unsigned int	Size of each data object 1, 2, 4 or 8 bytes depending on am/xm code.	4
	block_len	unsigned int	block length in cycles 1-256	1
	start_on_trg	BOOL	Start DMA transfer on trigger from the Analyzer 1= Enable 0= Disabled	0
	slave_select	Hex number	The slave select address for 2eSST broadcast. Third address phase as 32 bits value where A[21:1] is used.	0x3FFFFE (all selected)



AM & XAM Code: A32sD

Direction: VME to VME

Start DMA: Immediately

Block Length: 1 Cycles (dec)

Repeat Count: 1 (dec, 0 is forever)

Src Address: 0

Dst Address: 0

Transfer Size: 4

Data Size: 1 Byte

Slave Select: 3FFFFE

Example: `dma 10000 20000 1000  vv 4`

`dma` = DMA command
`10000` = source start address
`20000` = destination start address
`1000` = bytes to transfer
`vv` = VME to VME transfer
`4` = repeat 4 times.

A DMA transfer of 0x1000 bytes from address 0x10000 to address 0x20000 is executed 4 times.

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.

Test

Repeatedly fills Memory space with a given pattern, reads it back, and checks for errors.

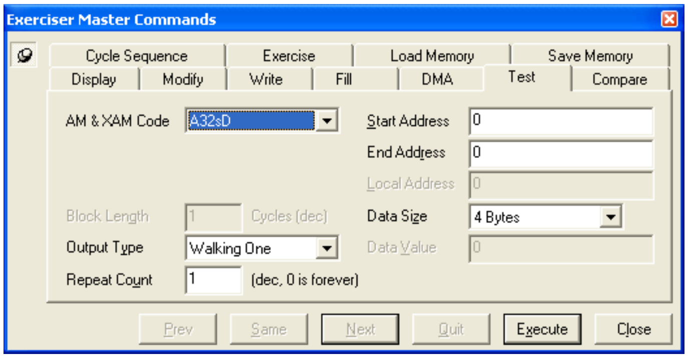
Command Line format:

t <start_addr> <end_addr> [<value>] [<repeat>] [<am/xam>][<data_size>] [<block_len>] [<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
			char values:
			o = Walking one pattern
			z = Walking zero pattern
			s = address as data
			r = random data
			l = use data in "local addr"
				o

	repeat	unsigned_int	Number of times to repeat the DMA transfer 0 = forever	1
	am/xam	char	Address modifier hex value in range 00-3F if am = 20 the value should be a 4 digit hexadecimal number where the 1. byte is 20 and the 2. byte is the xam code. Valid xam codes are 11, 12, 21 or 22. The modifier can also be given as mnemonic instead of hex, see Table 7-1 on page 181	0D or A32sP
	data_size	unsigned int	Size of each data object 1, 2, 4 or 8 bytes	4
	block_len	unsigned int	block length in cycles 1-256	1
	local_addr	hex number	Local memory address of data pattern	0



AM & XAM Code: A32sD

Start Address: 0

End Address: 0

Local Address: 0

Block Length: 1 Cycles (dec)

Data Size: 4 Bytes

Output Type: Walking One

Data Value: 0

Repeat Count: 1 (dec, 0 is forever)

Example: `t 10000 1ffff o 32 A32sD 4 1`

`t` = Test command
`10000` = start address
`1ffff` = end address
`o` = Walking One pattern
`32` = repeat 32 times
A32sD = am code
`4` = 4 byte accesses
`1` = block length.

The result is a test that writes `0xffff` bytes of walking a one pattern to VME 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=l` 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 VME 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 block cycles are used, an error can occur in the first transfer of the block, but the trigger signal will not be asserted until the block is finished and the verify process has started.

Compare

Repeatedly reads Memory space, and compares the data with a given pattern.

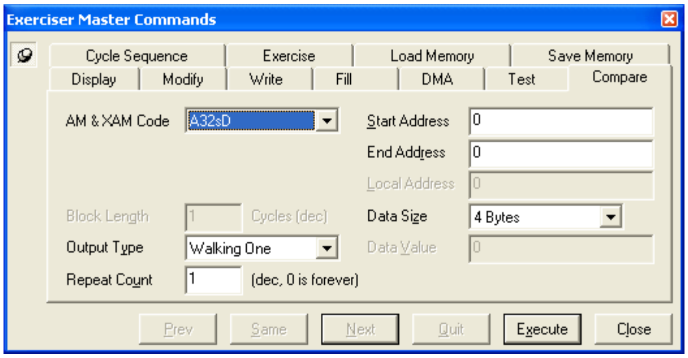
Command Line format:

c <start_addr> <end_addr> [<value>] [<repeat>] [<am/xam>] [<data_size>] [<block_len>][<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
			char values:
			o = Walking one pattern
			z = Walking zero pattern
			s = address as data
			r = random data
			l = use data in “local addr”
				o

	repeat	unsigned_int	Number of times to repeat the DMA transfer 0 = forever	1
	am/xam	char	Address modifier hex value in range 00-3F if am = 20 the value should be a 4 digit hexadecimal number where the 1. byte is 20 and the 2. byte is the xam code. Valid xam codes are 11, 12, 21 or 22. The modifier can also be given as mnemonic instead of hex, see Table 7-1 on page 181	0D or A32sP
	data_size	unsigned int	Size of each data object 1, 2, 4 or 8 bytes	4
	block_len	unsigned int	block length in cycles 1-256	1
	local_addr	hex number	Local memory address of data pattern	0



AM & XAM Code: A32sD

Start Address: 0

End Address: 0

Local Address: 0

Block Length: 1 Cycles (dec)

Data Size: 4 Bytes

Output Type: Walking One

Data Value: 0

Repeat Count: 1 (dec, 0 is forever)

Example:`c 10000 1ffff o 1 A32sD 4 4`

`c` = Compare command
`10000` = start address
`1ffff` = end address
`o` = Walking one
`1` = do not repeat
A32sD = am code
`4` = 4 bytes accesses
`4` = 4 cycles block length

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 VME 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 block cycles are used, an error can occur in the first transfer of the block, but the trigger signal will not be asserted until the block is finished and the verify process has started.

Cycle Sequence

Generates a sequence of VME cycles. Only error messages are output. This command is useful to put traffic on the bus.

Command Line format:

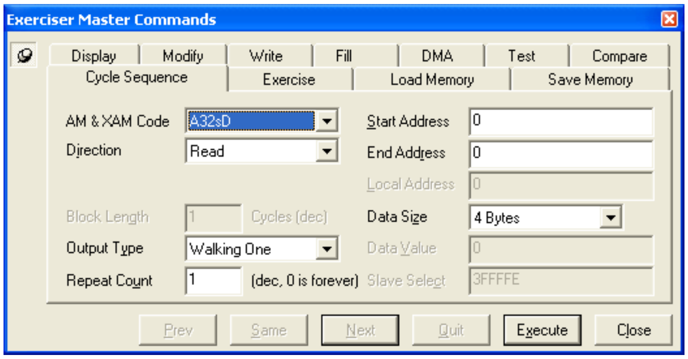
q <start_addr> <end_addr> [<value>] [<dir>][<repeat>] [<am/xam>] [<data_size>] [<block_len>] [<local_addr>][<slave_select>]

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
			char values:
			o = Walking one pattern
			z = Walking zero pattern
			s = address as data
			r = random data
			l = use data in "local addr"
				o

	dir	char	Transfer direction:
			r = read
			w = write
				r

	repeat	unsigned_int	Number of times to repeat the sequence 0 = forever	1
	am/xam	char	Address modifier hex value in range 00-3F if am = 20 the value should be a 4 digit hexadecimal number where the 1. byte is 20 and the 2. byte is the xam code. Valid xam codes are 11, 12, 21 or 22. The modifier can also be given as mnemonic instead of hex, see Table 7-1 on page 181	0D or A32sP
	data_size	unsigned int	Size of each data object 1, 2, 4 or 8 bytes	4
	block_len	unsigned int	block length in cycles 1-256	1
	local_addr	hex number	Local memory address of data pattern	0
	slave_select	Hex number	The slave select address for 2eSST broadcast. Third address phase as 32 bits value where A[21:1] is used.	0x3FFFFE (all selected)



AM & XAM Code: A32sD

Direction: Read

Block Length: 1 Cycles (dec)

Output Type: Walking One

Repeat Count: 1 (dec, 0 is forever)

Start Address: 0

End Address: 0

Local Address: 0

Data Size: 4 Bytes

Data Value: 0

Slave Select: 3FFFFE

Example:`q 20000 200ff 1`

`q` = Cycle Sequence command
`20000` = VME start address
`200ff` = VME end address
`1` = never repeat (repeat count 1)

Place a user-defined pattern on the bus:

The value argument specifies which kind of pattern to be used. Using `value=1` 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

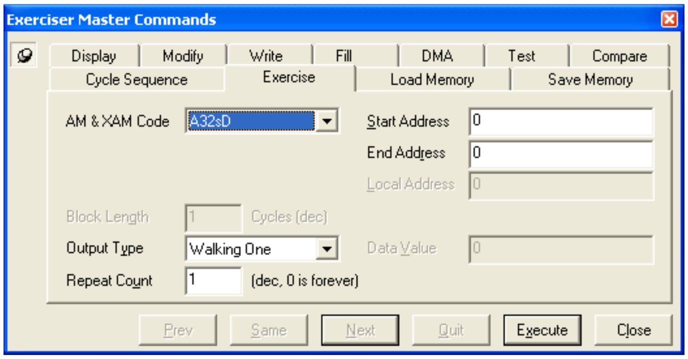
Reads and writes cycles with varying data sizes.

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>] [<block_len>][<am/xam>] [<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 char values: z= Walking zero pattern is filled into the area o = Walking one pattern is filled into the area s = address as data is filled into the area r = random data is filled into the area l = use data in “local addr”	o
	repeat	unsigned int	Number of times to repeat the DMA transfer 0 = forever	1
	am/xam	char	Address modifier hex value in range 00-3F if am = 20 the value should be a 4 digit hexadecimal number where the 1. byte is 20 and the 2. byte is the xam code. Valid xam codes are 11, 12, 21 or 22. The modifier can also be given as mnemonic instead of hex, see Table 7-1 on page 181	0D or A32sP
	block_len	unsigned int	block length in cycles 1-256	1
	local_addr	hex number	Local memory address of data pattern	0



AM & XAM Code: A32sD
Start Address: 0
End Address: 0
Local Address: 0
Block Length: 1 Cycles (dec)
Output Type: Walking One
Data Value: 0
Repeat Count: 1 (dec, 0 is forever)

Example:`x 0 8000000 ABBACAFE 0`

`x` = Exercise command
`0` = VME start address
`8000000` = VME 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=1` 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.

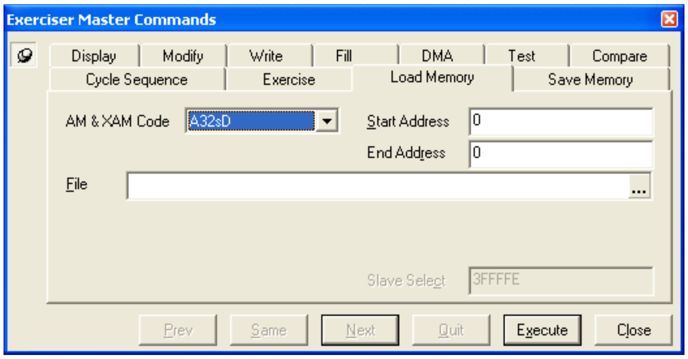
Load Memory

Data files previously generated with the Save or Local Save commands, can be loaded into VME memory using the Load command.

Command Line format:

l <start_addr><end_addr> [<am/xam>][<slave_select>][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	
	am/xam	char	Address modifier hex value in range 00-3F if am = 20 the value should be a 4 digit hexadecimal number where the 1. byte is 20 and the 2. byte is the xam code. Valid xam codes are 11, 12, 21 or 22. The modifier can also be given as mnemonic instead of hex, see Table 7-1 on page 181	0D or A32sP
	slave_select	Hex number	The slave select address for 2eSST broadcast. Third address phase as 32 bits value where A[21:1] is used.	0x3FFFFE (all selected)
	[<filename>]	string	Name of the file to load.	



Exerciser Master Commands

AM & XAM Code: A32sD

Start Address: 0

End Address: 0

File:

Slave Select: 3FFFFE

Example: l 0 100

`0x100` bytes of data will be loaded from the file to VME memory, starting from VME 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 VME 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.

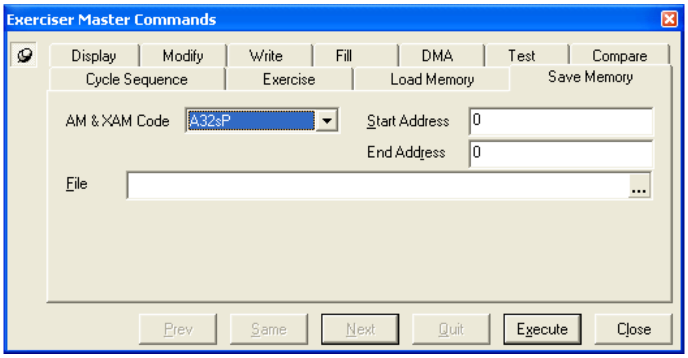
Save Memory

Save VME memory to file.

Command Line format:

s<start_addr><end_addr> [<am/xam>][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	
	am/xam	char	Address modifier hex value in range 00-3F if am = 20 the value should be a 4 digit hexadecimal number where the 1. byte is 20 and the 2. byte is the xam code. Valid xam codes are 11, 12, 21 or 22. The modifier can also be given as mnemonic instead of hex, see Table 7-1 on page 181	0D or A32sP
	[<filename>]	string	Name of the file to save the data in.	



AM & XAM Code: A32sP

Start Address: 0

End Address: 0

File:

Example: s 0 ffc

The VME Exerciser will perform Memory Read cycles on the VME 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 VME Exerciser window, or any of the ‘Q’, ‘q’, ‘Esc’, or ‘.’ keys, to abort the Save command.

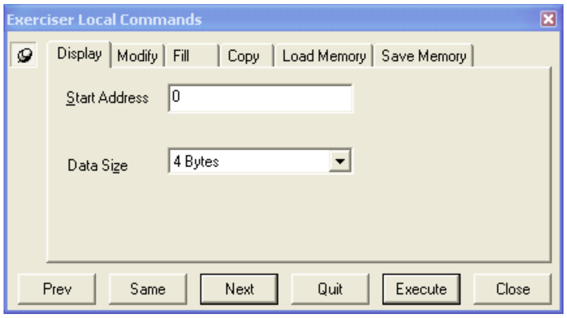
Local Display

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.

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



Start Address 0

Data Size 4 Bytes

Example: `ld 200000 2`

`ld` = Local Display command
`200000` = Local User Memory start address
`2` = 2 bytes display format.

Local Modify

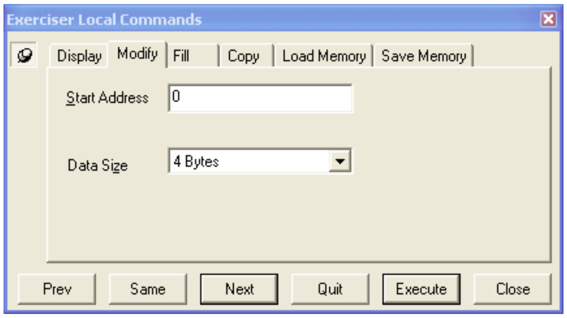
Displays data with data size 1, 2, 4, or 8 bytes, at a given local user address, and optionally allows Local User Memory modification.

The local memory addresses range from 0x0 to 0x7FFFFF.
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



Start Address: 0

Data Size: 4 Bytes

Example: `lm a000 1`

`lm` = Local Modify command
`a000` = Local User Memory start address
`1` = 1 byte display format.

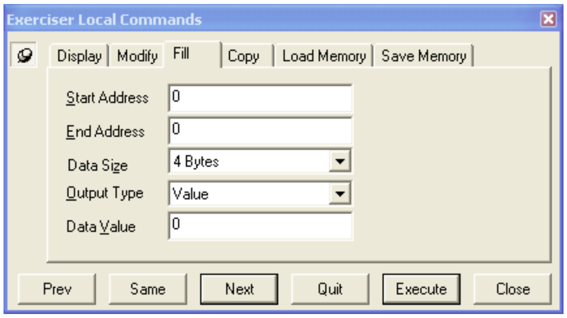
Local Fill

Fills local user memory on the Exerciser, with a given data pattern or value

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 char values: z = Walking zero pattern is filled into the area o = Walking one pattern is filled into the area s = address as data is filled into the area 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



Start Address 0
End Address 0
Data Size 4 Bytes
Output Type Value
Data Value 0

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.

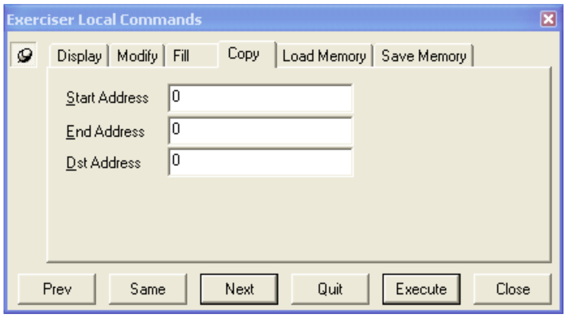
Local Copy

Copies local user Exerciser memory from one location to another.

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	



Start Address: 0
End Address: 0
Dst Address: 0

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.

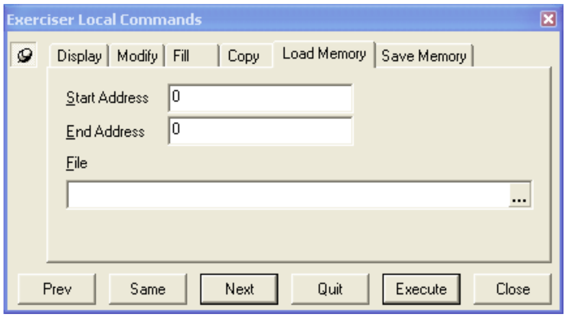
Local Load

Data files previously generated with the Save or Local Save commands, can be loaded into Local User Memory using the Local Load command.

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.	



Start Address 0

End Address 0

File

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 VME Exerciser window, or any of the ‘Q’, ‘q’, ‘Esc’, or ‘.’ keys, to abort the Local Load command.

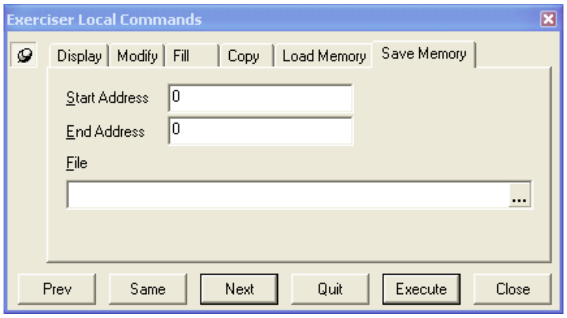
Local Save

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.	



Start Address: 0

End Address: 0

File:

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 VME 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.

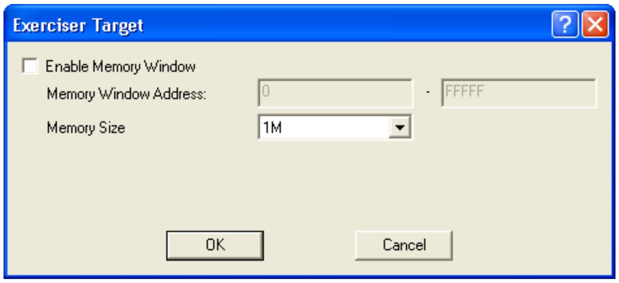
Target

Maps VME target window into Local User Memory. When activated, parts of the Local User Memory can be accessed by other VME bus masters. This is how the Exerciser can emulate a VME slave memory device.

Command Line format:

target <enable>[<vme_base>][size]

Argument			Description	Default
Required	**Optional**	**Type**		
	enable	BOOL	Enable/Disable: 1 = Enable 0 = Disable	0
	vme_base	hex number	VME start address of the window Value must be specified if enabled.	
	size	hex number	Size in hex bytes 100000 - 800000	800000



Exerciser Target

□ Enable Memory Window

Memory Window Address: 0 - FFFFF

Memory Size 1M

Example 1:target 1 80000000

target = Target command
1 = enable a target window
80000000 = VME start address of window

Example 2:target 0

target = Target command
0 = disable the target window

Status Line Information

The status line at the bottom of the BusView window displays the current target window status.

Interrupt

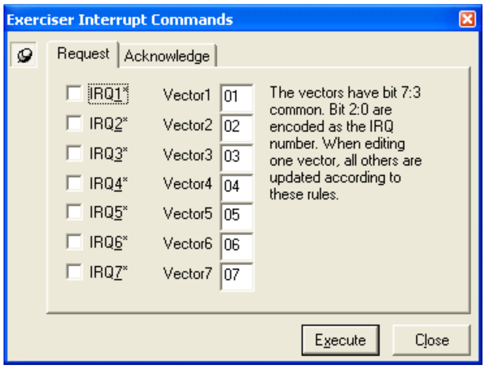
The Interrupt command generates VME interrupts.

Command Line format:

int <enable> <irq_line> [vector]

Argument			Description	Default
Required	**Optional**	**Type**		
enable		BOOL	1 = Enable Interrupt 0 = Disable Interrupt	
irq_line		char	VME interrupt request line. Valid values are 1 to 7.	
	vector	int	The interrupt vectors (0x00-0xFF). Vector to be returned on interrupt acknowledge is bit 7:3 from this vector and the irq level encoded as bit 2:0	

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:



□ IRQ1* Vector1 01
□ IRQ2* Vector2 02
□ IRQ3* Vector3 03
□ IRQ4* Vector4 04
□ IRQ5* Vector5 05
□ IRQ6* Vector6 06
□ IRQ7* Vector7 07

The vectors have bit 7:3 common. Bit 2:0 are encoded as the IRQ number. When editing one vector, all others are updated according to these rules.

Example 1: int 1 2

int = int command
1 = enable interrupt
2 = interrupt line.

Example 2: int 0 2

int = int command
0 = disable interrupt
2 = interrupt line.

Status Line Information

The status line at the bottom of the BusView window identifies the active interrupt line. LEDs marked 1-7are active if the corresponding interrupt line is enabled.

The same information can be found by simply typing “refresh” at the VME 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. Valid value is 'a', 'b', 'c', 'd' or '0'. Set this to '0' to stop exercising. 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. 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. 1 - drive the port high. If no value parameter specified, read and return current value from the port.	20ms

Interrupt Acknowledge

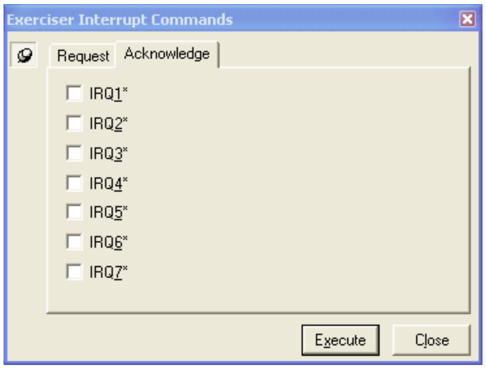
The Interrupt Acknowledge command generates interrupt acknowledge cycles on the VME bus.

Command Line format:

intack <irq_lines>

where:

irq_lines - Interrupt line to generate (up to 7 digits each different and in range 1-7).



7.6 Configuration Scan

The Exerciser can perform a complete scan of VME configuration space. It systematically probes for all possible devices on the bus that the Vanguard is located on.Performing a Configuration Scan

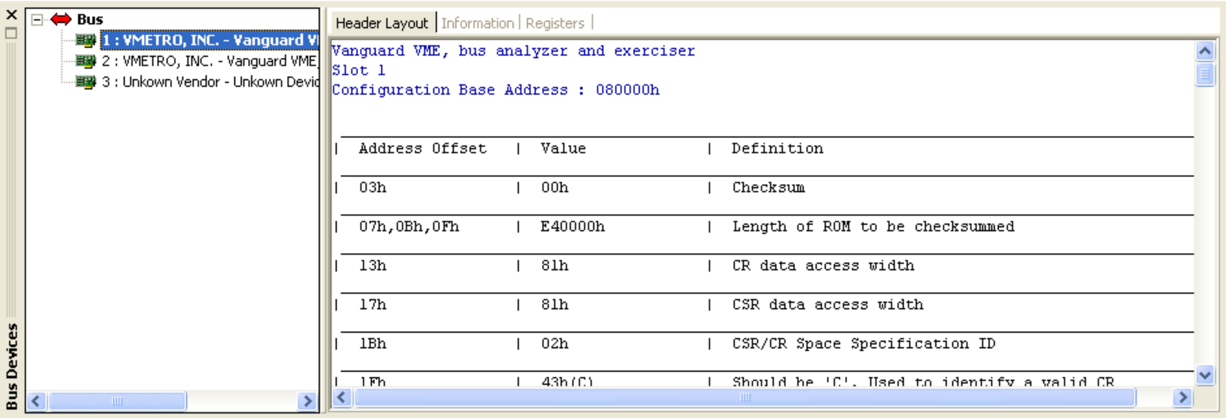
The Configuration Scan can be started using one of the following methods:

- 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.


Bus Devices
Bus
1 : VMETRO, INC. - Vanguard V
2 : VMETRO, INC. - Vanguard VME
3 : Unkown Vendor - Unkown Devic
Header Layout
Information
Registers

```
Vanguard VME, bus analyzer and exerciser
Slot 1
Configuration Base Address : 080000h

|  Address Offset   |  Value            |  Definition

|  03h              |  00h              |  Checksum

|  07h,0Bh,0Fh      |  E40000h          |  Length of ROM to be checksummed

|  13h              |  81h              |  CR data access width

|  17h              |  81h              |  CSR data access width

|  1Bh              |  02h              |  CSR/CR Space Specification ID

|  1Fh              |  43h(C)           |  Should be 'C'. Used to identify a valid CR
```

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 VME 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 41 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

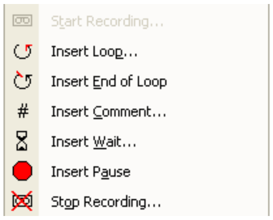
1. Click Start Recording from the Exerciser, Script menu or type `rec_start` in the command line.
2. Type a filename for the script file and choose a location to store it.
3. Every Exerciser command executed now will be recorded into this script file. The Script Commands are also now available.
4. Click 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*

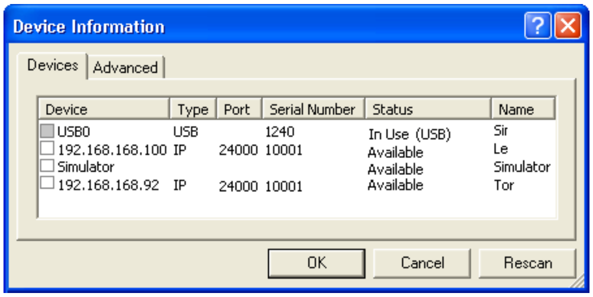
The Simulator can be used where the Vanguard Hardware is not available. It allows most of the features in BusView to be used without having a connection to the Vanguard Hardware.

- 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.



Device	Type	Port	Serial Number	Status	Name
☐ USB0	USB		1240	In Use (USB)	Sir
☐ 192.168.168.100	IP	24000	10001	Available	Le
☐ Simulator				Available	Simulator
☐ 192.168.168.92	IP	24000	10001	Available	Tor

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.

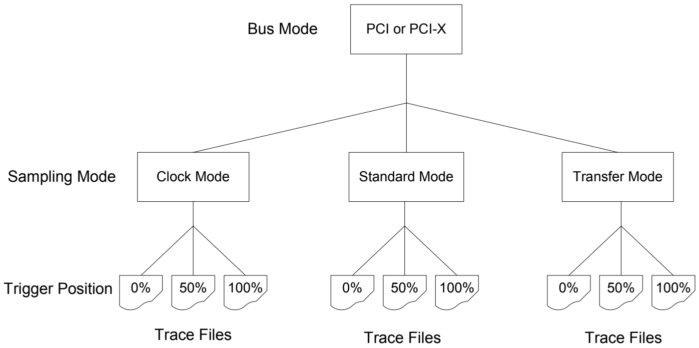
It is important to realize that the trace shown after the Analyzer has triggered in Simulator mode, bears no relation to the Event Patterns defined in the Analyzer Setup Window. However, there are some conditions used to determine which trace file is displayed.

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

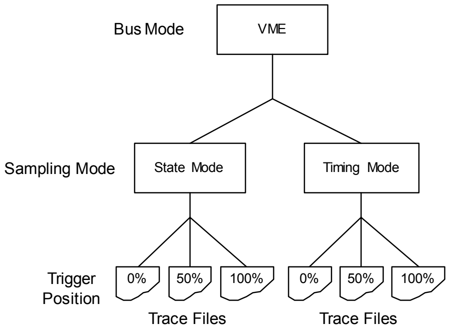
Each of the trace files have a different Trigger Position, 0%, 50% and 100%. When the Analyzer is run, the trace file chosen for display is determined as that which is closest to the chosen trigger position defined in the Analyzer Setup Options. To summarize:

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.

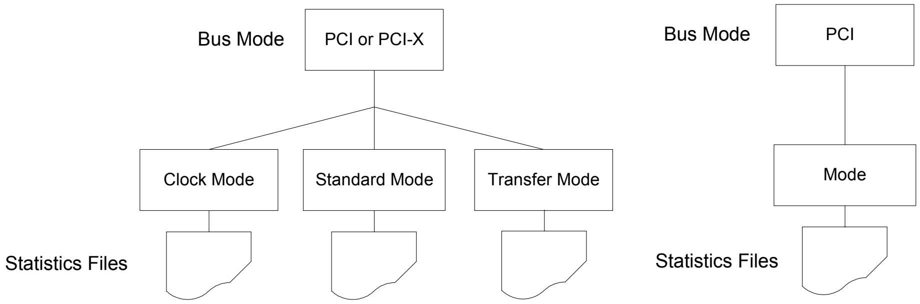
.

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.

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.

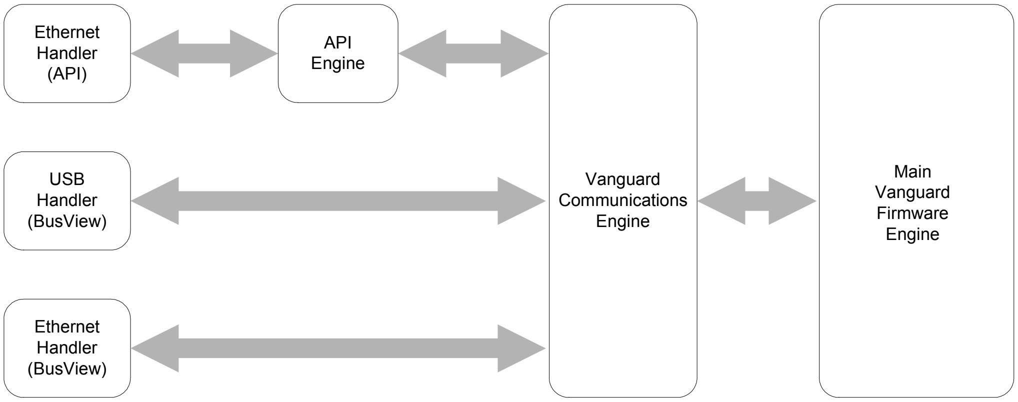

Ethernet Handler (API)
API Engine
USB Handler (BusView)
Ethernet Handler (BusView)
Vanguard Communications Engine
Main Vanguard Firmware Engine

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 **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 233). 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: 0x01: VME System Reset Assert event 0x02: VME System Reset De-assert event 0x04: Bus Frequency Changed event 0x08: Analyzer Triggered event 0x10: Analyzer Trace Full event 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 229. 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: “<address>: access failed”, 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

Command Name	Page
vanguard_version_info	241
vanguard_device_info	242
vanguard_api_option	243
vanguard_device_option	244
vanguard_reset	245
vanguard_selftest	246

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.

Command Name	Page
analyzer_setup	285
analyzer_run	248
analyzer_halt	249
analyzer_status	250
analyzer_trace_data	251

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.

Command Name	Page
pcheck_setup	255
pcheck_option	258
pcheck_run	259
pcheck_halt	260
pcheck_status	261
pcheck_clear	262

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.

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. 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. 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]. The binary sequence $0x1234ABBA_{16}$ would e.g. be formatted as the string “1234ABBA”. 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.

Where AM/XAM codes are used, the following codes are valid:

TABLE 9-6. Valid AM/XAM codes

Value (hex)	Value (decimal)[a]	Description
00	0	A64 64-bit block transfer (MBLK)
01	1	A64 single access transfer
03	3	A64 block transfer (BLT)
04	4	A64 lock command (LCK)
05	5	A32 lock command (LCK)
08	8	A32 non-privileged 64-bit block transfer (MBLT)
09	9	A32 non-privileged data access

TABLE 9-6. Valid AM/XAM codes (Continued)

Value (hex)	Value (decimal)[a]	Description
0A	10	A32 non-privileged program access
0B	11	A32 non-privileged block transfer (BLT)
0C	12	A32 supervisory 64-bit block transfer (MBLT)
0D	13	A32 supervisory data access
0E	14	A32 supervisory program access
0F	15	A32 supervisory block transfer (BLT)
29	41	A64 non-privileged access
2C	44	A16 lock command (LCK)
2D	45	A16 supervisory access
2F	47	CR / CSR space
32	50	A24 lock command (LCK)
38	56	A24 non-privileged 64-bit block transfer (MBLT)
39	57	A24 non-privileged data access
3A	58	A24 non-privileged program access
3B	59	A24 non-privileged block transfer (BLT)
3C	60	A24 supervisory 64-bit block transfer (MBLT)
3D	61	A24 supervisory data access
3E	62	A24 supervisory program access
3F	63	A24 supervisory block transfer (BLT)
2011	8209	A32/D64 2eSST
2012	8210	A64/D64 2eSST
2021[b]	8225	A32/D64, Broadcast 2eSST
2022[b.]	8226	A64/D64, Broadcast 2eSST
10-1F	16-31	

a. Use the decimal value in the API functions

b. Only used in commands that perform a write operation

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: -1 = Unknown 0 = PCI 1 = PCI-X 2 = VME *3 = Reserved* 4 = PCI
	int	Unused for VME, always returns 0.
vg_type	int	Vanguard type: 0 = Vanguard PCI 1 = Vanguard CompactPCI 2 = Vanguard PMC 3 = Vanguard VME *4 = Reserved* 5 = Vanguard PCI0SL 6 = Vanguard XMC 7 = Vanguard SAE
sys_con	boolean	`true` if the Vanguard acts as a system controller

The API forces the user to call this function initially after a connection has been established to the API server, or after a VME 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 9-7 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 9-7. **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: 0x01: Enable VME System Reset Assert event 0x02: Enable VME System Reset De-assert event 0x04: Enable Bus Frequency Changed event 0x08: Enable Analyzer Triggered event 0x10: Enable Analyzer Trace Full event 0x20: Enable Protocol Checker Triggered event	0
1 = event_format	1 = Format event notification messages as XML 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 9-8 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.

TABLE 9-8. **Vanguard device options**

Option	Description
0 - 4	*Reserved*
5 = Trigger Output	Trigger output option. This setting is lost when the Vanguard is powered off. 0 = State Analyzer Source, Active High Polarity 1 = State Analyzer Source, Active Low Polarity 2 = Protocol Checker Source, Active High Polarity 3 = Protocol Checker Source, Active Low Polarity 4 = Exerciser Source, Active High Polarity 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 9-9 below for available alternatives.	

Returned data:

None

TABLE 9-9. **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 = System Reset	Trigger a VME System Reset. This command is only valid if an Exerciser is present. 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: 0: Trace is Empty 1: Analyzer is running 2: Analyzer has triggered 3: Trace is full 4: Analyzer has halted 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: 0 = State Mode 1 = Timing Mode *2 = Reserved* 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 9-11 below.

TABLE 9-11. VME 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 VME clock count of the trace line, relative to the trigger line. See also bit 287 (TT_overflow). The corresponding clock frequency is given by the `samp_speed` value from the `analyzer_status` command.
127-64 (64)	A[63:0]	64-bit address with address increment, stored in a little endian format. Address[63:0] is fetched from the following VME signal lines: Address[31:0]: A[31:1], LWORD*/DS1* (LWORD* is used in 2eVME/SST, DS1* in VME64) Address[63:32]: D[31:0]
191-128 (64)	D[63:0]	64-bit data, stored in a little endian format. Data[63:0] is fetched from the following VME signal lines Data[31:0]: D[31:0] Data[63:32]: A[31:1], LWORD*
197-192 (6)	AM[5:0]	Address Modifier code
198	AS*	Address Strobe signal
199	DS1*	Data Strobe 1 signal
200	DS0*	Data Strobe 0 signal
201	DTACK*	Data Acknowledge signal
202	BERR*	Bus Error signal
203	RETRY*	Retry signal (6U systems)
204	RESP*	Retry signal (3U systems)
205	WRITE*	Read/Write signal
206	LWORD*	Long-Word signal
210-207 (4)	BG[3:0]*	Bus Grant signals
214-211 (4)	BR[3:0]*	Bus Request signals
215	BBSY*	Bus Busy signal
216	BCLR*	Bus Clear signal
223-217 (7)	IRQ[7:1]*	Interrupt Request signals
224	IACK*	Interrupt Acknowledge signal
225	IACKIO*	Interrupt Acknowledge Daisy Chain signal
226	ACFAIL*	AC Failure signal
227	SYSFAIL*	System Failure signal
228	SYSCLOCK	System Clock signal (16 MHz)
229	SYSRESET*	System Reset signal
230	SERA	Serial Bus line A signal
231	SERB	Serial Bus line B signal
232	D64	Set if 64 bit transfer
233	D32	Set if VME64 long-word and 2eVME/SST 32-bit data transfer
234	Data Valid	Set when there is a data cycle on the bus

TABLE 9-11. **VME trace line format (Continued)**

Bits	Name	Description
235	RMW	Set when RMW cycle
236	BLK (Burst)	Set when Block/Burst transfer
238-237 (2)	Address Phase[1:0]	In Timing Mode, these signals have the following meaning: 00 = Not Address Phase 01 = 2eVME/SST Address Phase 1 / Regular Address Phase 10 = 2eVME/SST Address Phase 2 11 = 2eVME/SST Address Phase 3 In State Mode only bit 0 is used, and it has the following meaning: 1 = First Data Cycle in a Block/Burst transfer. Will be set in cycles with only one data phase, and in both the read and the write phase of RMW. Will also be set when RETRY or BERR is active in the 2eVME/SST address phase. 0 = Remaining Data Cycles in a Block/Bursts transfer
246-239 (8)	XAM[7:0]	Extended Address Modifier Code. XAM[7:0] is fetched from the following VME signal lines in Address Phase 1 during 2eVME/SST cycles: XAM[7:1]: A[7:1] XAM[0]: LWORD*
251-247 (5)	GA[4:0]	Geographical Address for the Master. GA[4:0] is fetched from the following VME signal lines in Address Phase 2 during 2eVME/SST cycles: GA[4:0]: A[23:16]
255-252 (4)	Trf_Rate[3:0]	2eSST Transfer Rate. Trf_Rate[3:0] is fetched from the following VME signal lines in Address Phase 2 during 2eVME/SST cycles: Trf_Rate[3:0]: D[3:0]
263-256 (8)	SubUnit[7:0]	SubUnit number in Master. SubUnit[7:0] is fetched from the following VME signal lines in Address Phase 2 during 2eVME/SST cycles: SubUnit[7:0]: A[31:24]
264	2eSST_Odd	2eSST Odd bit. 2eSST_Odd is fetched from the following VME signal lines in Address Phase 2 during 2eVME/SST cycles: 2eSST_Odd: D[4]
266-265 (2)	BGL[1:0]	Bus Grant Level. BGL is a coded version of the Bus Grant signals BG[3:0]*, and is fetched in the Arbitration Phase of the bus cycle
267	BG_Hidden	Bus Grant Hidden. BG_Hidden is set if the Bus Grant signals BG[3:0]* is hidden for the analyzer (i.e. BGL[1:0] not valid)
273-268 (6)	AS2DATA	Counts the latency from falling edge of AS* to the first cycle when data is valid (access time)

TABLE 9-11. VME trace line format (Continued)

Bits	Name	Description
281-274 (8)	CycleCount	Counts the number of data cycles in a 2eVME/SST transfer. A data cycle contains 2 data beats (rising/falling edge). The counter counts down from the loaded cycle count available in Address Phase 2. The maximum CycleCount loaded is 0x80 (128), and since there are 2 data beats per CycleCount, the counter is decremented each 2nd data beat.
282	*Spare*	*Reserved*
283	P2Trg	Trigger signal from P2 SAM Analyzer
284	PChkTrg	Trigger signal from Protocol Checker
285	ExerTrg	Trigger signal from Exerciser
286	*Spare*	*Reserved*
287	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.
295-288 (8)	ExtInput[7:0]	External inputs [7:0] from pin header on Carrier front panel
303-296 (8)	ExtInput[15:8]	External input [15:8] from pin header on VME SAM
319-304 (16)	*Spare*	*Reserved*

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, which is 8. The bytes must be ordered in ascending order, i.e. the first byte in the array represents mask byte 0. The possible violations are given in Table 9-12.	

Returned data:

None

TABLE 9-12. VME violations

Index	Byte:Bit	Description
0	0:0	A[7:0] or LWORD* unstable
1	0:1	A[15:8] unstable
2	0:2	A[23:16] unstable
3	0:3	A[31:24] unstable
4	0:4	D[7:0] unstable
5	0:5	D[15:8] unstable
6	0:6	D[23:16] unstable
7	0:7	D[31:24] unstable
8	1:0	BR0* aborted
9	1:1	BR1* aborted
10	1:2	BR2* aborted
11	1:3	BR3* aborted
12	1:4	Multiple grants
13	1:5	3.10 BBSY* off early
14	1:6	3.6 BG0* before BBSY* off
15	1:7	3.6 BG1* before BBSY* off
16	2:0	3.6 BG2* before BBSY* off
17	2:1	3.6 BG3* before BBSY* off

TABLE 9-12. VME violations (Continued)

Index	Byte:Bit	Description
18	2:2	3.7 BBSY* on < 90ns
19	2:3	Hidden grants
20	2:4	IRQ1* aborted
21	2:5	IRQ2* aborted
22	2:6	IRQ3* aborted
23	2:7	IRQ4* aborted
24	3:0	IRQ5* aborted
25	3:1	IRQ6* aborted
26	3:2	IRQ7* aborted
27	3:3	4.41 IACKOUT* lingers
28	3:4	4.45 IACKIN* on early
29	3:5	4.46 IACKOUT* early
30	3:6	11.12 Multiple timing (DS*) line transitions
31	3:7	11.18 or 2eSST 3.10
32	4:0	11.19 or 2eSST 3.11
33	4:1	11.5 or 2eSST 3.19
34	4:2	11.6 or 2eSST 3.24
35	4:3	11.11 or 2eSST 3.23
36	4:4	11.15 or 2eSST 3.7
37	4:5	2eSST 3.13 in address phase 1
38	4:6	2eSST 3.13 in address phase 2
39	4:7	2eSST 3.28
40	5:0	2.1 Illegal signal combination
41	5:1	2.17 DS0* off early
42	5:2	2.17 DS1* off early
43	5:3	2.20 AS* before BG
44	5:4	2.28 DSx* before BG
45	5:5	2.28 IACK* before BG
46	5:6	2.28 LWORD* before BG
47	5:7	2.28 WRITE* before BG
48	6:0	2.31 AS* off < 30ns
49	6:1	2.35 DSx* on early
50	6:2	2.37 DSx* off < 30ns
51	6:3	2.39 DSx* skew
52	6:4	2.42 AM[5:0] unstable
53	6:5	2.43 IACK* or AM off early

TABLE 9-12. **VME violations (Continued)**

Index	**Byte:Bit**	**Description**
54	6:6	2.44 AS* off early
55	6:7	2.45 AS* on < 30ns
56	7:0	2.49 WRITE* unstable
57	7:1	2.69 VME64 addr. err.
58	7:2	2.96 Signals illegally changed when RETRY* active
59	7:3	Reserved AM code used
60	7:4	Reserved XAM code used
61	7:5	5V voltage low
62	7:6	5.1 Power monitors timing error
63	*7:7*	*Reserved*

pcheck_option

Set/get Protocol Checker options.

Arguments:

Argument				Description	Default
	Required	Optional	Type		
0	option		int	The selected option. See Table 9-13 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 9-13. Protocol checker options

Option	Values	Default
0 = lock_on_first	1 = Set the lock on first option 0 = Clear the lock on first option	0
1 = auto_clear	1 = Set the auto clear option 0 = Clear the auto clear option	0
2 = clear_on_run	1 = Set the clear on run option 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: 0 = Protocol Checker is running 1 = Protocol Checker has triggered 2 = Protocol Checker is halted 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, which is 8. 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: -2 = Exerciser failed to load -1 = No exerciser available 0 = Normal exerciser 1 = Enhanced exerciser
exer_init_status	int	Exerciser initialization status: 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. 1 = Normal 2 = No Grant: The exerciser does not respond, possibly due to no grant.

`exerciser_read`

Reads data from Memory space.

Arguments:

Argument				Description	Default
	Required	**Optional**	**Type**		
0	start_addr		int64	Start address of area to read.	
1		am_xam	int	Address modifier. See Table 9-6 on page 239 for valid codes.	13 (0x0D)
2		data_size	int	Size of each data object 1, 2, 4 or 8 bytes	4
3		block_len	int	block length in cycles 1-256	1
4		end_addr	int64	End address of the area to read. 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 `start_addr+0x7f`, only `0x80` bytes of data are read in each block.

exerciser_write

Writes data into Memory 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		am_xam	int	Address modifier. See Table 9-6 on page 239 for valid codes.	13 (0x0D)
3		data_size	int	Size of each data object 1, 2, 4 or 8 bytes	4
4		block_len	int	block length in cycles 1-256	1
5		slave_select	int64	The slave select address for 2eSST broadcast. Third address phase as 32 bits value where A[21:1] is used.	3FFFFE (all selected)

Returned data:

None

When the block 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 block length.

exerciser_fill

Fills Memoryspace 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: 0= Walking One pattern 1= Walking Zero pattern 2= address as data 3= random data 4= use pattern in `pattern_data` 5= use local data from `pattern_data`	0
3		am_xam	int	Address modifier. See Table 9-6 on page 239 for valid codes.	13 (0x0D)
4		data_size	int	Size of each data object 1, 2, 4 or 8 bytes	4
5		block_len	int	block length in cycles 1-256	1
6		pattern_data	int64	If `pattern_id` = 4, this value represents the pattern to use. If `pattern_id` = 5, this value is the Local memory address of data pattern.	0
7		slave_select	int64	The slave select address for 2eSST broadcast. Third address phase as 32 bits value where A[21:1] is used.	3FFFFE (all selected)

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	
2		am_xam	int	Address modifier. See Table 9-6 on page 239 for valid codes.	13 (0x0D)
3		slave_select	int64	The slave select address for 2eSST broadcast. Third address phase as 32 bits value where A[21:1] is used.	3FFFFE (all selected)

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: 0= Walking One pattern 1= Walking Zero pattern 2= address as data 3= random data 4= use pattern in `pattern_data` 5= use local data from `pattern_data`	0
3		repeat	int	Number of times to repeat the DMA transfer Must be >= 1	1
4		am_xam	int	Address modifier. See Table 9-6 on page 239 for valid codes.	13 (0x0D)
5		block_len	int	block length in cycles 1-256	1
6		pattern_data	int64	If `pattern_id` = 4, this value represents the pattern to use. If `pattern_id` = 5, this value is the Local memory address of the data pattern.	0

Returned data:

None

exerciser_test

Repeatedly fills Memoryspace 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: 0= Walking One pattern 1= Walking Zero pattern 2= address as data 3= random data 4= use pattern in `pattern_data` 5= use local data from `pattern_data`	0
3		repeat	int	Number of times to repeat the DMA transfer Must be >= 1	1
4		am_xam	int	Address modifier. See Table 9-6 on page 239 for valid codes.	13 (0x0D)
5		data_size	int	Size of each data object 1, 2, 4 or 8 bytes	4
6		block_len	int	block length in cycles 1-256	1
7		pattern_data	int64	If `pattern_id` = 4, this value represents the pattern to use. 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 VME Bus Analyzer when a verify 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 block cycles are used, an error can occur in the first transfer of the block, but the trigger signal will not be asserted until the block is finished and the verify process has started.

exerciser_compare

Repeatedly reads Memory 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: 0= Walking One pattern 1= Walking Zero pattern 2= address as data 3= random data 4= use pattern in `pattern_data` 5= use local data from `pattern_data`	0
3		repeat	int	Number of times to repeat the DMA transfer Must be >= 1	1
4		am_xam	int	Address modifier. See Table 9-6 on page 239 for valid codes.	13 (0x0D)
5		data_size	int	Size of each data object 1, 2, 4 or 8 bytes	4
6		block_len	int	block length in cycles 1-256	1
7		pattern_data	int64	If `pattern_id` = 4, this value represents the pattern to use. 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 VME 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 block cycles are used, an error can occur in the first transfer of the block, but the trigger signal will not be asserted until the block is finished and the verify process has started.

exerciser_cycle_sequence

Generates a sequence of VME 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: 0= Walking One pattern 1= Walking Zero pattern 2= address as data 3= random data 4= use pattern in `pattern_data` 5= use local data from `pattern_data`	0
3		dir	int	Transfer direction: 0= read 1 = write	0
4		repeat	int	Number of times to repeat the DMA transfer Must be >= 1	1
5		am_xam	int	Address modifier. See Table 9-6 on page 239 for valid codes.	13 (0x0D)
6		data_size	int	Size of each data object 1, 2, 4 or 8 bytes	4
7		block_len	int	block length in cycles 1-256	1
8		pattern_data	int64	If `pattern_id` = 4, this value represents the pattern to use. If `pattern_id` = 5, this value is the Local memory address of the data pattern.	0
9		slave_select	int64	The slave select address for 2eSST broadcast. Third address phase as 32 bits value where A[21:1] is used.	3FFFFE (all selected)

Returned data:

None

exerciser_dma

Runs DMA transfers on the bus.

Arguments:

Argument				Description	Default
	Required	**Optional**	**Type**		
0	src_addr		int64	Source start address	
1	dst_addr		int64	Destination start address	
2	transfer_size		int	Number of bytes to transfer	
3		dir	int	0 = VME to VME memory 1 = VME to local memory 2 = local to VME memory	0
4		repeat	int	Repeat count for DMA transfer (0 = forever)	1
5		am_xam	int	Address modifier. See Table 9-6 on page 239 for valid codes.	13 (0x0D)
6		data_size	int	Size of each data object. 4 or 8 bytes, depending on the `am_xam` code	4
7		block_len	int	block length in cycles 1-256	1
8		start_on_trg	boolean	Start DMA transfer on trigger from the Analyzer `true` = Enabled `false` = Disabled	`false`
9		slave_select	int64	The slave select address for 2eSST broadcast. Third address phase as 32 bits value where A[21:1] is used.	3FFFFE (all selected)

Note – If no parameters are specified, DMA status is returned. If any parameters are given, the first three parameters are required.

Returned data:

	Type	Description
dma_run_status	int	Status of DMA controller: 0 = Running 1 = Waiting for trigger 2 = Done 3 = Aborted 4 = Error
dma_exer_status	int	Exerciser status for DMA controller: 0 = Status OK 1 = Master Abort 2 = Target Abort *3 = Reserved* 4 = Data Parity error 5 = Address Parity error *6 = Reserved* 7 = Retry expired 8 = Target Disconnect 9 = Split Completion error 10 = General error *11 = Reserved* 12 = Exerciser running 13 = Out of tags 14 = VME bus error *15 = Reserved* 16 = Out of DMA descriptors

exerciser_dma_abort

Aborts ongoing DMA transfer started with the `exerciser_dma` command.

Arguments:

None

Return format:

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: 0= Walking One pattern 1= Walking Zero pattern 2= address as data 3= random data 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. If the `value` argument is specified, the selected `port` is set as follows: `true` = Drive the port high `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:

None

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 3 values.
vendor_id	int64	Vendor ID for this device
board_id	int64	Board ID for this device
slot	int	Slot number for this device

exerciser_target

Maps VME target window into Local User Memory. When activated, parts of the Local User Memory can be accessed by other VME bus masters. This is how the Exerciser can emulate a VME slave memory device.

Arguments:

Argument				Description	Default
	Required	**Optional**	**Type**		
0		enable	boolean	`true` = Enable `false` = Disable	
1		vme_base	int64	VME start address of the window Value must be specified if `enable` is set to `true`.	0
2		size	int	Size in Mbytes of this target. Valid options are 1, 2, 4 and 8	8
3		addr_space	int	Address space to activate. Could be a logical OR combination of the following flags: 0x1: A64 0x2: A32 0x4: A24 0x8: A16 Address space is only activated if base address is inside maximum address of the respective address space.	15 (all)

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
addr_space	int	If `mem_bar_en` is `true`, holds the enabled address spaces. A logical OR combination of the following flags: 0x1: A64 0x2: A32 0x4: A24 0x8: A16

`exerciser_interrupt`

Generates VME interrupts.

Arguments:

Argument				Description	Default
	Required	**Optional**	**Type**		
0	enable		boolean	`true` = Enable Interrupt `false` = Disable Interrupt	
1	irq_lines		int	VME interrupt request line. Valid values are a combination of: 0x01 = irq level 1 0x02 = irq level 2 0x04 = irq level 3 0x08 = irq level 4 0x10 = irq level 5 0x20 = irq level 6 0x40 = irq level 7	
2	vector		int	The interrupt vectors (0x00-0xFF). Vector to be returned on interrupt acknowledge is bit 7:3 from this vector and the irq level encoded as bit 2:0. If `enable` is `false`, this parameter is ignored.	

Note – If no parameters are specified, interrupt status is returned. If any parameters are given, all parameters are required.

Returned data:

	Type	Description
level1	boolean	irq level 1 status: `true` = Interrupt is enabled `false` = Interrupt is disabled
level2	boolean	irq level 2 status
level3	boolean	irq level 3 status
level4	boolean	irq level 4 status
level5	boolean	irq level 5 status
level6	boolean	irq level 6status
level7	boolean	irq level 7status

exerciser_interrupt_acknowledge

Generates interrupt acknowledge cycles on the VME bus.

Arguments:

Argument				Description	Default
	Required	Optional	Type		
0	irq_lines		int	Interrupt request line. Valid values are a combination of: 0x01 = irq level 1 0x02 = irq level 2 0x04 = irq level 3 0x08 = irq level 4 0x10 = irq level 5 0x20 = irq level 6 0x40 = irq level 7	

Returned data:

	Type	Description
level1	int	irq level 1 status: -2 = Interrupt acknowledged with bus error -1 = Status for this level is invalid (interrupt acknowledge not requested) >0 = Interrupt acknowledged with this vector returned
level2	int	irq level 2 status
level3	int	irq level 3 status
level4	int	irq level 4 status
level5	int	irq level 5 status
level6	int	irq level 6 status
level7	int	irq level 7 status

exerciser_option

Set/get VME exerciser options.

Arguments:

Argument				Description	Default
	Required	Optional	Type		
0	option		int	The selected option. See 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 9-14. VME options

Option	Values	Default
0 = bus_level	Bus Level to use for the commands 0 - 3	3
1 = BBSY_release_type	0 = Release On Request (ROR) 1 = Release When Done (RWD) 2 = Release On Clear (ROC) 3 = Release Never (RNE)	0
2 = 2eSST_speed	0 = SST160, effective rate 146MBytes/s. 1 = SST267, effective rate 171MBytes/s. 2 = SST267, effective rate 205MBytes/s. 3 = SST267, effective rate 256MBytes/s (default). 4 = SST320+, effective rate 341MBytes/s.	3
3 = fairness_timer	0 = None 1 = 20us 2 = Infinite	0
4 = arbitration_type	0 = Round Robin 1 = Priority	0
5 = timeout	Data to bus error time-out in microseconds 1us - 256us (default).	256

APPENDIXES

A

Information for VBT-325 users

This section is useful for users of CWCDS's older VBT-32x family of analyzers.

To support the new VME Renaissance protocol extensions and automatic system controller functionality, the Vanguard is configured somewhat differently than its predecessors in the VBT-32x family. There are more consequences of incorrect configuration and use when using the VG-VME than there was with the VBT-325, particularly with respect to daisy chain jumpers.

A-1 VG-VME Considerations

Here is a list of considerations and differences from the VBT-325 when using the VG-VME.

- The VME Daisy Chain jumpers must NEVER be installed in the VME slot where the Vanguard analyzer is installed. This will cause system crashes.
- Incorrect or incomplete Daisy Chain jumper settings will have negative effects on the system and the Vanguard. The VBT-325 was more "tolerant" in this regard due to the weak pull-up resistors on the BG signals.
- The VME arbitration statistics must only be run when the Vanguard is in slot 1 or 2, otherwise any data other than 0 is incorrect and not valid.
- The Vanguard can sit in slot 1 only when VG-VE (Exerciser option) is installed.
- The VME arbitration will only be enabled if BusView status says SysCon.
- If there is an open slot between the system controller and the Vanguard, and the backplane does NOT implement auto daisy chaining (Auto Jumpered), the system will fail if the open slot(s) do(es) not have jumpers.
- Always install jumpers in a non auto jumpered backplane in the slots between the system controller and the Vanguard.

B Signals

This section is a reference to all the signals used by the Vanguard. All the signals are monitored by the analyzer.

B-1 VME Signals

Address Lines

A01 - A31 (Address)

Address lines that are used to broadcast a short, standard, or extended address.

AM0 - AM5 (Address Modifier)

Used to broadcast information such as address size, cycle type, and/or MASTER identification.

DS0*, DS1* (Data Strobe)

Signals used in conjunction with LWORD* and A01 to indicate how many data bytes are being transferred.

LWORD* (Longword)

Signal used in conjunction with DS0*, DS1*, and AO1 to select which byte locations within the 4 byte group are accessed during the data transfer.

Data Lines

D00 - D31 (Data Bus)

Bidirectional data lines used to transfer data between Masters and Slaves.

Interface Control Lines

AS* (Address Strobe)

A signal that indicates when a valid address has been placed on the address bus.

DS0*, DS1* (Data Strobe)

On Write cycles, the first data strobe falling edge indicates when the Master has placed valid data on the data bus. On Read cycles, the first rising edge tells the Slave when it can remove valid data from the data bus.

In 2eSST write cycles, both edges of DS1 are used to qualify data.

BERR* (Bus Error)

This signal is generated by a Slave or Bus Timer. This signal indicates to the Master that the data transfer was not completed.

DTACK* (Data Transfer Acknowledge)

The Slave drives DTACK* low to indicate that it has successfully received the data on a write cycle. On a read cycle, the Slave drives DTACK* low to indicate that it has placed data on the data bus.

In 2eSST read cycles, both edges of DTACK are used to qualify data.

WRITE*

Used by the Master to indicate the direction of data transfer operations. When WRITE* is driven low, the data transfer direction is from the Master to the Slave. When WRITE* is driven high, the data transfer direction is from the Slave to the Master.

RETRY*/RESP*

Slaves can request a retry of the transfer by asserting Retry*/Resp*.

Other VME Connector/Pin descriptions

ACFAIL* (AC power failure)

Used to indicate that the AC input to the power supply is no longer being provided or that the required AC input voltage levels are not being met.

BBSY* (Bus Busy)

This signal is driven low by the current Master to indicate that it is using the bus.

BCLR* (Bus Clear)

Generated by an Arbiter to indicate when there is a higher priority request for the bus.

BR[3:0]* (Bus Request 0-3)

Generated by Requesters. A low level on one of these lines indicates that some Master needs to use the Data Transfer Bus.

IRQ1*-IRQ7* (Interrupt Request)

Generated by an Interrupter, which carry interrupt requests. When several lines are monitored by a single Interrupt Handler the highest number line is given the highest priority.

IRQ[7:1]*, the IRQ field can be negated. There are no predefined symbols, but special formatting applies. When an IRQ* signal is asserted (i.e. low value), it is displayed with its IRQ number, non-asserted, (i.e. high value) are displayed as a dot ".".

For example, IRQ5* asserted, and IRQ1* as don't care, and the rest non-asserted, will be displayed "..5...X".

SYSCLOCK (System Clock)

Provides a constant 16MHz clock signal that is independent of any other bus timing.

SYSFAIL* (System Failure)

Indicates that a failure has occurred in the system and can be generated by any board on the VMEbus.

IACK*

Interrupt Acknowledge.

IACKIO*

Interrupt Acknowledge Daisy Chain.

SERA

Serial Bus A.

SERB

Serial Bus B.

SYSRESET*

System Reset.

AM/XAM

AM[5:0], the Address Modifier bits.

TABLE B-1. AM/XAM codes[1]

Predef. Symbol		Predef. Symbol	
2eVME/2eSST	10000xxxxxxxxx	**A32sP**	001110xxxxxxxx
2eSST	10000x000100xx	**A32sBLT**	001111xxxxxxxx
2eSSTBC	10000x001000xx	**A64MBLT**	000000xxxxxxxx
A32_D64_2eSST	10000000010001	**A64D**	000001xxxxxxxx
A64_D64_2eSST	10000000010010	**A64BLT**	000011xxxxxxxx
A32_D64_2eSSTBC	10000000100001	**A64LCK**	000100xxxxxxxx
A64_D64_2eSSTBC	10000000100010	**CS/CSR**	101111xxxxxxxx
A32_D32_2eSST	10000100010001	**A24LCK**	110010xxxxxxxx
A40_D32_2eSST	10000100010010	**A24nMBLT**	111000xxxxxxxx
A32_D32_2eSSTBC	10000100100001	**A24nD**	111001xxxxxxxx
A40_D32_2eSSTBC	10000100100010	**A24nP**	111010xxxxxxxx
2eVME	10000x000000xx	**A24nBLT**	111011xxxxxxxx
A32_D64_2eVME	10000000000001	**A24sMBLT**	111100xxxxxxxx
A64_D64_2eVME	10000000000010	**A24sD**	111101xxxxxxxx
A32D32_2eVME	10000100000001	**A24sP**	111110xxxxxxxx
A40D32_2eVME	10000100000010	**A24sBLT**	111111xxxxxxxx
A32LCK	000101xxxxxxxx	**A16nD**	101001xxxxxxxx
A32nMBLT	001000xxxxxxxx	**A16LCK**	101100xxxxxxxx
A32nD	001001xxxxxxxx	**A16sD**	101101xxxxxxxx
A32nP	001010xxxxxxxx	**A40D**	110100xxxxxxxx
A32nBLT	001011xxxxxxxx	**A40LCK**	110101xxxxxxxx
A32sMBLT	001100xxxxxxxx	**A40BLT**	110111xxxxxxxx
A32sD	001101xxxxxxxx		

[1] nD - Non privileged data
sD - Supervisor data
nP - Non privileged program
sP - Supervisor program
MBLT - 64 Bit block transfers
BLT - up to 32 bit block transfers
LCK - Lock transfer

Vanguard Generated Signals

The selection of VME 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.

_D64

Set if 64 bit data transfer.

_D32

Set if 32bit data transfer.

2eSSTXfer

2eSST Transfer rate in MBytes/Sec.

TABLE B-2. **2eSSTXfer**

Predef. Symbol		
SST160	00000	160 MBytes/sec (Maximum)
SST267	00001	267 MBytes/sec
SST320	00010	320 MBytes/sec
3USST80	10000	80 MBytes/sec
3USST133	10001	133 MBytes/sec
3USST160	10010	160 MBytes/sec

2eSSTOdd

2eSST Odd bit. This is fetched from D[4] in address phase 2 during 2eVME/SST cycles.

2eCycle Count

Counts the number of data cycles in a 2eVME/SST transfer. A data cycle contains 2 data beats (rising/falling edge). The counter counts down from the loaded cycle count available in Address Phase 2. The maximum CycleCount loaded is 0x80 (128), and since there are 2 data beats per CycleCount, the counter is decremented each 2nd data beat.

AddrPh

In Timing Mode, these signals have the following meaning:

TABLE B-3. **AddrPh in Timing Mode**

Predef. Symbol		
APh	01	2eVME/SST Address Phase 1 / Regular Address Phase
APh2	10	2eVME/SST Address Phase 2
APh3	11	2eVME/SST Address Phase 3
-	00	Not Address Phase

In State Mode only bit 0 is used and has the following meaning:

TABLE B-4. **AddrPh in State Mode**

1	First Data Cycle in a Block/Burst Transfer. Will be set in cycles with only one data phase, and in both read and write phase of RMW. Will also be set when RETRY or BERR is active in the 2eVME/SST address phase.
0	Remaining Data Cycles in a Block/Burst Transfer.

ASToData

Number of sampling clocks from AS (Address Strobe) to the first data.

Block

Set when Block/Burst cycle.

BgL

BG[3:0]*, internally latched. This field contains bus grant information latched on the falling edge of BBSY*.

TABLE B-5. **BgL signal field**

Predef. Symbol	Comment
---0	BG0* active
--1-	BG1* active
-2--	BG2* active
3---	BG3* active
----	BG* hidden

Cycle

The Cycle field contains the bus signals that define the current cycle type.

TABLE B-6. **Cycle field**

Predef. Symbol		
Wr	x0	Write
Rd	x1	Read
RMW	1x	Read Modify Write

DataValid

Set when there is a data cycle on the bus.

EXERtrg

Exerciser cross trigger. The Exerciser activates this signal if the Compare and Test functions fail.

TABLE B-7. **EXERtrg Field**

Predef. Symbol		
Trig	1	The Exerciser cross trigger has triggered.
-	0	The Exerciser cross trigger has not triggered.
x	x	Disable the Exerciser cross trigger cross trigger

Fail

The Fail group contains the two VMEbus fail signals ACFAIL* and SYSFAIL*.

TABLE B-8. **Fail field**

Predef. Symbol		
AcSy	00	Both ACFAIL and SYSFAIL asserted
--Sy	01	SYSFAIL
Ac--	10	ACFAIL
----	11	None asserted

GA[4..0]

Geographical address for the Master is fetched from the following VME signal lines in Address Phase 2 during 2eVME/SST cycles:

GA[4..0], A[23..16]

The analyzer must be used in a system running 2eVME/SST cycles in order for the GA signals to be valid.

Iack (Interrupt Acknowledge)

Used by an Interrupt Handler to acknowledge an interrupt request. Iack is displayed as a numeric code on the signals A[3:1] and driving IACK* low.

TABLE B-9. **Iack field**

Predef. Symbol		
IACK1	0001	
IACK2	0010	
IACK3	0011	
IACK4	0100	
IACK5	0101	
IACK6	0110	
IACK7	0111	
Undef	0000	
IACKx	0xxx	Any IACK
-----	1xxx	No IACK

P2trg

Cross trigger signal from the P2 Analyzer.

TABLE B-10. **P2trg field**

Predef. Symbol		
Trig	1	The P2 analyzer has triggered.
-	0	The P2 analyzer has not triggered.
x	x	Disable the P2 analyzer cross trigger

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 B-11. **PCHKTrg Field**

Predef. Symbol		
Trig	1	The Protocol Checker has triggered.
-	0	The Protocol Checker has not triggered.
x	x	Disable the Protocol Checker cross trigger

RelTime

Relative Timetag. The time from the previous to the current sample.

RMW

Set when RMW cycle.

Size

TABLE B-12. Size field

Predef. Symbol		
UNAL3L	x000100	
UNAL3H	x000010	
WORD	x00x001	
UNAL2	x001000	
UBYTE	x00x011	
LBYTE	x00x101	
D32	xx1xxxx	
D64	x1xxxxx	

SubUnit

SubUnit number in Master. SubUnit is fetched from A31-A24 in address phase 2 during 2eVME/SST cycles.

Status

The bus cycle status field.

TABLE B-13. Status field

Predef. Symbol		
Data	1111	Data Transfer
BERR	0011	Bus error
Err	1011	Illegal combination of signals
. .	0111	Idle (not defined)
ARetry	010x	Address Retry for 6U
DRetry	000x	Data Retry for 6U
DInvalid	110x	Data Invalid for 6U
ARetry3U	01x0	Address Retry for 3U
DRetry3U	00x0	Data Retry for 3U
DInvalid3U	11x0	Data Invalid for 3U

Transfer

TABLE B-14. Transfer Field

Predef. Symbol	Start	
Start	11	Start of a Block Transfer
Block	01	Inside a Block
Single	10	Single Transfer
-	00	Idle

C Specifications

- Power Consumption
- General
- Analyzer
- Exerciser (Optional Module)
- Protocol Checker
- Statistics

C-1 Vanguard Specifications

Power Consumption

TABLE C-1. **Vanguard VME 5.0V**

Min(Idle)	**Max**	**Sampling Clock frequency**
1.52A Without Exerciser	1.68A State mode without Exerciser	-
1.52A Without Exerciser	2.78A Timing Mode without Exerciser	133 MHz
2.12A With Exerciser	4.0A Timing Mode with Exerciser	133 MHz

General

VME bus:	Up to 320 MB/s (2eSST320)
Interfaces:	USB port, 12 Mb/s. Ethernet port 10/100 Mb/s
Power supply requirements:	+5VDC +/-5% from VME connector or from ext. power supply via front panel inlet.
Dimensions:	160 x 233.4 mm (6U)
Compliant to:	ANSI/VITA 1-1994 VME64 Standard ANSI/VITA 1.1-1997 VME64 Extensions ANSI/VITA 1.5-2003 2eSST
Operating Temperature:	0-50 ºC / 32-122 ºF
Measurements:	Temperature: 0-120 °C/ 32-248°F Voltage: 3.3V, 5V, 12V

Analyzer

Trace Memory:	2M x 256 bits (64 MBytes total)
Input channels:	102 bus signals, plus 16 ext. inputs on pin headers.
Monitored signals:	A31-01, D31-00, DS1*, DS0*, AS*, LWORD*, DTACK*, BERR*, WRITE*, AM5-0, IRQ7-1*, IACK*, IACKIN/IACKOUT*, BBSY*, BR3-0*, BCLR*, RETRY*, ACFAIL*, SER_A, SER_B, RESP*, SYSFAIL*, SYSRES*, SYSCLK*. BG3-0IN/OUT*. Also clocked separately, encoded and stored are: XAM7-0, GA4-0, Subunit7-0, 2eCycleCount7-0, 2eSSTodd and 2eSSTxfer3-0. Internally generated are: Bus Level, RMW, Block w/VME64, D32, D64 and Datavalid. In addition, sampling of external inputs: EXT15-0, and cross triggers from other functional units.
Trigger:	8 word recognizers covering all 102 VMEbus signals and 16 Ext. inputs. 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
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:	8 TTL level inputs on pin header 8 TTL level inputs on front panel

Exerciser (Optional Module)

Master:	Supports normal, block and RMW cycles on any BR level, selectable RMW, ROR, ROC release options.
Target:	8MBytes SDRAM memory, with software controlled base address and window size.

Protocol Checker

VME Violations	60 Protocol Violations and 3 Protocol Warnings

Statistics

Event counters:	8 x 30-bits.
Real-Time Statistics Counters:	53 x 30-bits counters.
Time Tag:	Range: 7.5ns-1172min Resolution: 7.5ns

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. 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. Includes cPCI carrier, SAM and BusView GUI. (See separate data sheet)
VG-PMC	PMC State Analyzer and Statistics Module. (Single PMC for all functions) Includes PCI-X carrier, SAM and BusView GUI.

Individual boards and SAM modules may be purchased separately. Packages are also available. Please consult CWCDS.

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

Contact Curtiss-Wright Controls Defense Solutions
See the Technical Support section.

References

American National Standard for 2eSST, ANSI/VITA 1.5-2003

American National Standard for VME64 Extensions, ANSI/VITA 1.1-1997

American National Standard for VME64, ANSI/VITA 1-1994