Instruction Manual

Tektronix

VX4820
Digital Test Module

070-8319-01

Warning
The servicing instructions are for use by qualified personnel only. To avoid personal injury, do not perform any servicing unless you are qualified to do so. Refer to the Safety Summary prior to performing service.

Please check for change information at the rear of this manual.

First Printing: April 1991
Revised Printing: August 1991

Printed in the U.S.A.

Tektronix, Inc., P.O. Box 1000, Wilsonville, OR 97070–1000

Table of Contents

General Safety Summary

Preface

Purpose of this Document ix
Organization of this Manual ix
Notational Conventions x
Active Signal States x
Number Base x
Related Publications x

Section 1 General Information

Introduction 1-1
Features 1-1
Description 1-2
Pin I/O Functions 1-2
Test PASS/FAIL 1-3
Test Commands 1-4
Test Data 1-5
Test Execution 1-6
Test Results 1-6
Learn Mode 1-7
I/O Pins 1-7
Backplane Triggers and Clocks 1-8
Systems of 128 — 768 Pins 1-8
Diagnostics 1-9
Calibration 1-9
Standard Accessories 1-10
Optional Accessories 1-10
Specifications 1-11
Performance Conditions 1-11
Definition of Terms 1-11

Section 2 Preparation for Use

Configuration 2-1
Calculating System Power and Cooling 2-1
Auxilary Pod Power 2-1
Module Jumpers 2-2
Logical Address Switch 2-3
Configuring the Backplane Jumpers 2-4
GPIB Communications Terminator 2-4
Installation 2-5
Module Installation Using Ejector Handles 2-5
Module Removal Using Ejector Handles 2-5
Pod Cable Assembly 2-5
Group Calibration 2-6
Group Calibration Procedure 2-6
Internal Clock Calibration 2-7
Required Equipment 2-7
External Clock Calibration 2-9
Required Equipment 2-9

Section 3 Operation and Programming

Connectors and Indicators 3-1
Connectors 3-1
Indicators 3-1
Getting Started 3-2
Module Installation 3-2
Basic Communications 3-3
Exercising the Module I/O Pins 3-4
Writing a Simple Test 3-6
Retrieving Test Failure Data 3-6
Test Debug Support 3-7
Learn Mode 3-9
Branching 3-10
Using VXI STST Protocol 3-12
Multi-Module Systems 3-12
Multi-Module Systems Using an External Clock 3-14
Diagnostics and Calibration 3-14

Section 4 Rear Panel Interface

Introduction 4-1
P1 Pinout 4-2
P2 Pinout 4-3

Section 5 Command Notation and Formats

Introduction 5-1
Command Notation 5-1
Program Message 5-2
ASCII Code 5-3
Syntactic Delimiters 5-3
White Space 5-4
Special Characters 5-4
Program Data 5-4
- Numeric Program Data Value Range 5-4
- Decimal Numeric Program Data 5-4
- Non-Decimal Numeric Program Data 5-5
- String Program Data 5-5

Command Formats 5-6
- Simple Command Program Header 5-6
- Simple Query Program Header 5-6
- Compound Command Program Header 5-7
- Compound Query Program Header 5-7
- Common Command Program Header 5-8
- Common Query Program Header 5-8
- Multiple-Command Format With Same 1st Program Mnemonic . 5-8

Instrument Response Queuing 5-9
Case Sensitivity 5-9
Command Types 5-9
Command Dictionary Layout 5-10

Section 6 Instrument Control Command Dictionary

Introduction 6-1
Alphabetical Command Listing 6-1
Functional Command Listing 6-3
ARM 6-4
BLRANGE 6-5
CLKCAL 6-6
CONNECT 6-8
CONTINUE 6-15
DESE 6-16
DEVINFO? 6-17
EVENT? 6-18
EVMSG? 6-19
FAILDATA? 6-20
FAILPIN? 6-21
GRPMODE 6-22
HEADER 6-24
HELP? 6-25
IDPOD 6-26
INHIBIT 6-27
INIT 6-28
INTCLKRATE 6-29
MODE 6-30
NEW 6-32
NEWPASS 6-33

NVRECALL 6-34
NVSAVE 6-35
PASSWORD 6-36
PAUSE 6-37
PINTEST? 6-38
PODTYPE? 6-39
READPIN? 6-40
REPORT 6-41
SEQ 6-42
SINGLESTEP 6-56
START 6-57
STATE? 6-58
STERR? 6-59
STOP 6-60
TSTPAT 6-61
USER 6-62
WRITEPIN 6-63
WSLOAD 6-64

Section 7 System Command Dictionary

Introduction 7-1
Alphabetical Listing 7-2

Section 8 Digital Toolbox

Introduction 8-1
Features 8-1
Description 8-1
What is required 8-2
Installation 8-2
Using Digital Toolbox 8-3
    Generating Test data 8-3
    Modular Test Generation 8-4
    Downloading, Executing, and Debugging a Test 8-4
    On-line Help 8-4
EDIT 8-5
    Starting the Editor 8-5
    Selecting the Test Configuration 8-5
    Display Areas 8-6
Display Formatting 8-6
    Display Colors 8-6
    Grouping Pins 8-7
    Changing Group Names 8-7
    Hiding Pins or Groups 8-7
    Redisplaying Pins or Groups 8-8
    Pin Notes 8-8
    Sequence Notes 8-8
    Marks 8-9
    Group Radix 8-9
Editing Test Data 8-10
    Typing Data in Binary Radix 8-10
    Typing Data in Octal and Hex Radix 8-10

Viewing Test Data 8-11
Copy and Paste Data 8-11
Cut, Copy, and Paste Test Sequences 8-12
Fill Data Pattern 8-12
Swapping Pin Data 8-13
Find and Change 8-13
Selecting the Starting Sequence 8-14
Sequence Commands 8-15
Test Files 8-16
Saving and Retrieving Test Files 8-16
Printing Test Files 8-16
Macros 8-17
Using Macros 8-17
Defining Macros 8-19
Macro File Example 8-20
Import and Export of ASCII Files 8-21
VERIFY 8-22
Starting Verify 8-22
Checking Macro Files 8-22
MERGE 8-23
STARTING THE MERGE UTILITY 8-23
SELECTING INPUT FILES 8-23
Saving the Merge Configuration 8-23
DEBUG 8-24
Starting the Loader/Debugger 8-24
Configuring the Test 8-24
Downloading / Uploading Test Data 8-24
Executing the Test 8-25
The Debug Display 8-26
Saving the Debug Configuration 8-27
Trace Display 8-27
Sequence Line Editor 8-28
Talker / Listener 8-29

Section 9 Module Service

General Information 9-1
Standard Accessories 9-1
Service Safety Summary 9-2
Performance Check 9-3
Equipment List 9-3
Test Procedure Descriptions 9-3
Troubleshooting 9-9
Diagnostics 9-10
Replacing Pin Driver and Receiver ICs 9-11
Module Removal/Replacement Procedure 9-12
Removal 9-12
Replacement 9-12
Preparing the Module for Exchange 9-12
Switch and Jumper Settings 9-12
Replaceable Parts 9-13
Replaceable Parts List, Module 9-13
Replaceable Parts List, Pod 9-14

Appendix A ASCII/GPIB Code Chart

Appendix B VXIbus Glossary

Appendix C Fast Data Channel Reference

Appendix D VX4820 Diagnostic Routines

GENERAL SAFETY SUMMARY

Only qualified personnel should perform service procedures. This Service Safety Summary and the General Safety Summary should be read before performing service procedures.

The general safety information in this summary is for operating and servicing personnel. Specific warnings and cautions can be found throughout the manual where they apply, and may not appear in this summary.

TERMS IN THIS MANUAL

CAUTION statements identify conditions or practices that could result in damage to the equipment or other property.

WARNING statements identify conditions or practices that could result in personal injury or loss of life.

TERMS AS MARKED ON EQUIPMENT

CAUTION indicates a hazard to property, including the equipment itself, and could cause minor personal injury.

WARNING indicates a personal injury hazard not immediately accessible as you read the marking.

DANGER indicates a personal injury hazard immediately accessible as you read the marking.

SYMBOLS AS MARKED ON EQUIPMENT

DANGER-High voltage

Protective ground (earth) terminal.

ATTENTION-REFER TO MANUAL

GROUNDING THE PRODUCT

WARNING: This product is grounded through the grounding conductor of the power cord. To avoid electrical shock, plug the power cord into a properly wired receptacle. A protective-ground connection by way of the grounding conductor in the power cord is essential for safe operation. (I.E.C. Safety Class I)

DANGER ARISING FROM LOSS OF GROUND

Upon loss of the protective-ground connection, all accessible conductive parts (including knobs and controls that may appear to be insulated) can render an electric shock.

USE PROPER FUSE

To avoid fire hazard use only a fuse of the correct type, voltage rating, and current rating.

REMOVE LOOSE OBJECTS

During disassembly or installation procedures, screws or other small objects may fall to the bottom of the mainframe. To avoid shorting out the power supply, do not power-up the instrument until such objects have been removed.

DO NOT OPERATE WITHOUT COVERS

To avoid personal injury or damage to the product, do not operate this product with the covers or panels removed.

USE CARE WITH COVERS REMOVED

To avoid personal injury, remove jewelry such as rings, watches, and other metallic objects before removing the cover. Do not touch exposed connections and components within the product while the power cord is connected.

REMOVE FROM OPERATION

If you have reason to believe that the instrument has suffered a component failure, do not operate the instrument until the cause of the failure has been determined and corrected.

DO NOT OPERATE IN EXPLOSIVE ATMOSPHERES

To avoid explosion, do not operate this product in an explosive atmosphere unless it has been specially certified for such operation.

SAFETY CERTIFICATION OF MODULES

For modules which are safety certified by Underwriters Laboratories, UL Listing applies only when the module is installed in a UL Listed product.

Preface

Purpose of this Document

This manual provides the information necessary to install, configure, and operate the VX4820 Digital Test Module.

Organization of this Manual

This manual consists of the following sections and appendices:

Section 1. General Information — Provides a brief overview of the VX4820 and its standard and optional accessories. This section also includes the VX4820 specifications.

Section 2. Preparation for Use — Provides information about how to configure and install the VX4820.

Section 3. Operation and Programming — Describes the connectors, the power-up sequence and module programming.

Section 4. Rear-Panel Interface — Describes the connectors on the VX4820 rear panel.

Section 5. Command Notation and Formats — Contains introductory information about the command dictionary.

Section 6. Instrument Control Command Dictionary — Contains an alphabetical and functional listing of the instrument control commands and a description of each command.

Section 7. System Command Dictionary — Contains information about the IEEE 488 commands.

Section 8. Digital Toolbox — Describes test development software.

Section 9. Module Service — Provide test and performance verification procedures.

Appendix A. ASCII and GPIB Code Chart

Appendix B. VXIbus Glossary

Appendix C. Fast Data Channel Reference

Appendix D. Diagnostic Tests

Notational Conventions

Active Signal States

An asterisk (*) following a signal mnemonic (as in ACFAIL*) denotes the signal is active when in its low state.

A signal mnemonic without an asterisk (*) denotes the signal is active when in its high state.

Number Base

Unless otherwise noted, all numbers are assumed to be decimal (base 10).

Binary numbers (base 2) are followed by the character "b" (for example: 1101b).

Hexadecimal numbers (base 16) are followed by the character "h" (for example: 97FFh).

Octal numbers (base 8) are followed by the character "o" (for example: 307o).

Related Publications

The following documents on related subjects may be useful for efficient use of the VX4820:

- *VXIbus System Specification, Version 1.3, July 14, 1989*
- *VMEbus Specification Manual, Revision C.1, October, 1985*
- *ANSI/IEEE Standard 1014-1987, IEEE Standard for a Versatile Backplane Bus: VMEbus*

Section 1
General Information

Introduction

The VX4820 Digital Test Module provides 32 or 64 pins of TTL digital I/O for the test of digital modules and systems. Testing of bidirectional bus systems, random and sequential logic may be accomplished by programming a sequence of test vectors. During test execution these test vectors control the function of each I/O pin. Data can be driven onto the pin by the VX4820 providing force data to the unit under test (UUT). The pin can be inhibited (set to a high impedance) so that the UUT can drive the pin and the UUT drive data compared against an expected value. Real-time data comparison ensures fast test execution by eliminating the post processing of test data. Advanced features such as branching and test execution pause enhance the ability to synchronize the test system to the UUT activity.

Features

The VX4820 is a single-wide, C-size, VXIbus module with the following features:

- 32/64 independent I/O pins with DRIVE, INHIBIT, COMPARE, and DRIVE AND COMPARE functionality
- 16,351 programmable test sequences
- Data rates up to 20 MHz
- Bidirectional dynamic pin control
- Real-time comparison requires no post processing of data
- Conditional and unconditional branching
- Test debug support with PAUSE and SINGLE STEP
- Test development support with LEARN MODE execution
- Digital Toolbox provides Windows$_{TM}$ 3.0-based pattern editor and loader/debugger
- Module groups provide test systems of up to 768 pins

Description

The Tektronix VX4820 Digital Test Module is a single-wide, C-size instrument module designed for use in a VXIbus system. It provides 32 or 64 pins of digital I/O which can be uniquely programmed to DRIVE, COMPARE with expected value, DRIVE AND COMPARE, or INHIBIT. Multiple VX4820 modules may be combined within a VXI mainframe to form a digital test system with up to 768 I/O pins.

Test sequences are stored in pattern memory and control the pin functions. The pattern memory can contain up to 16,351 test sequences. A pin can be programmed to any of the I/O functions on each test sequence. Results of pin input comparisons with expected data are combined to provide a Test State of PASS or FAIL.

Each test sequence may additionally contain programmed commands. These commands are executed in conjunction with the executing test sequence and control actions which may alter the test sequence or progress. These commands include SEQ:PAUSE, SEQ:BRANCH, SEQ:CLEAR, and SEQ:TRIG.

Tests are executed sequentially. The test progresses from the current sequence by incrementing to the next sequential step after the programmed clock interval has elapsed. An internal clock generator provides clock durations between 50 — 3,276,700 ns. Synchronous operation is provided by external clock inputs on the pod and front panel. Clock control commands include INTCLKRATE and CONNECT:CLOCK.

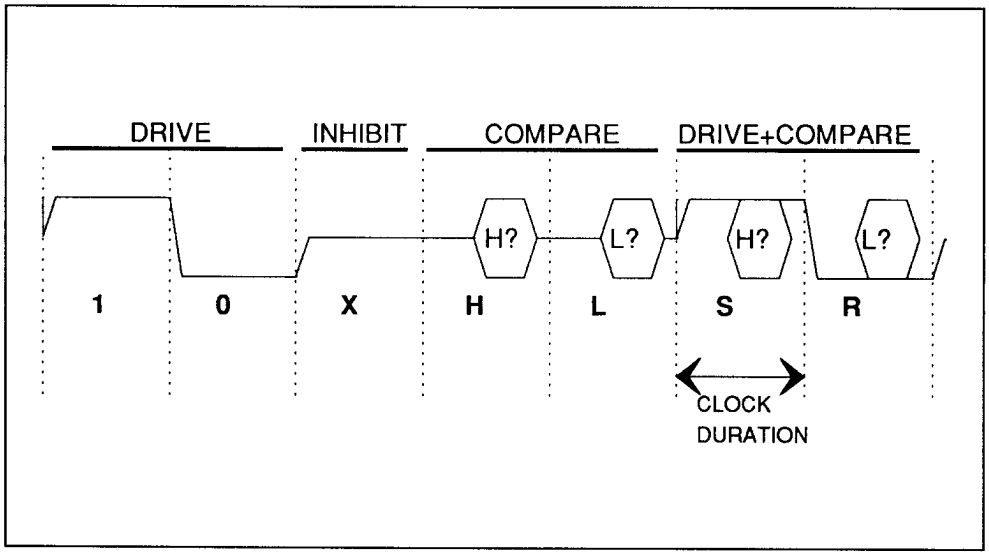
Pin I/O Functions

I/O pins can be individually programmed on each sequence of the test to one of seven functions. When the DRIVE HI and LO functions are programmed, test data is forced onto the I/O pin for the duration of the test sequence.

The INHIBIT function does not drive the I/O pin. It presents a high impedance to the UUT and no comparisons are done. Inhibit is used for the COMPARE MASK function.

The COMPARE function presents a high impedance to the UUT. The data present on the pin during the sequence is sampled and compared with the expected test data. If the expected data matches the sampled data then the comparison passes. If the data does not match the comparison fails. A comparison failure causes the Test State to change to FAIL.

The DRIVE AND COMPARE function both drives the I/O pin and compares the data present on the pin with the driven value. The result of the comparison affects the Test State (PASS/FAIL). This function is useful for finding faults where the UUT has a component that will not change state.



Pin I/O Functions

Test PASS/FAIL

The VX4820 executes a sequential test of the UUT through interaction with the I/O pins. On each test sequence, values driven by the UUT are compared with the COMPARE data. The result of the comparisons are combined into a single value Test State, PASS/FAIL. All comparisons are logically AND'ed. If all comparisons are PASSED, input data equals the expected data, and the combined Test State is PASSED. If, however, any I/O pin or group of pins returns a FAIL comparison, the combined Test State is FAIL.

The Test State is initialized to PASS automatically before a test starts. The Test State is updated as each test sequence executes. The test state remains PASS until an I/O comparison returns FAIL. The FAIL comparison will cause the Test State to transition from PASS to FAIL. The Test State will remain FAIL for the remainder of the test unless it is explicitly set back to PASS as the result of executing a test sequence in which a SEQ:CLEAR command has been placed.

When a comparison FAILs, information about the failed test sequence is captured. The binary state of all I/O pins is saved, including the I/O pins being driven. The sequence number of the failed comparison is also saved. This failure information can be retrieved after the termination of the test with the FAILDATA? query.

Only one test sequence can be saved as the result of a failed comparison. If a SEQ:CLEAR command is executed, the Test State will return to PASS and any captured failure information is lost. New failure information will be captured when the Test State again transitions to FAIL.

Test Commands

Test commands control actions which are taken during the execution of a test. These actions include generation of triggers, modification of the test sequence, and control of test progress. These commands support test debugging, test flow control, and system synchronization.

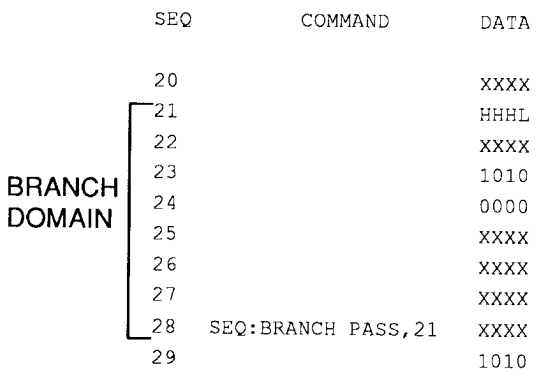
Branch commands allow the test sequence to be changed as a result of information captured during the test. Conditional and unconditional branch capability is provided. Conditional branches are based on the current Test State which may be either PASS or FAIL.

The SEQ:BRANCH ALWAYS command results in an unconditional branch from the current test sequence to a new sequence within the test memory. The branch will be taken without regard to the Test State. It may be used to build IF-THEN-ELSE or WHILE-DO control structures, or to link test sequences which are in noncontiguous test modules.

The SEQ:BRANCH PASS and SEQ:BRANCH FAIL commands conditionally branch to a new test sequence. The SEQ:BRANCH PASS command will jump to the new sequence if the current Test State is PASS. If the Test State is FAIL, the test will continue at the test sequence following the branch. The SEQ:BRANCH FAIL command will jump to the test sequence if the current Test State is FAIL. If the Test State is PASS, the test will continue at the test sequence following the branch.

Each branch command requires a minimum of eight steps in a sequence to execute. The seven sequences prior to a branch should be executed sequentially to ensure correct branch operation (the test should not jump to any of these eight sequences). Checks are made to ensure correct branch placement and target sequences. Any comparison which is to affect the outcome of a branch command should precede the branch command by at least 8 sequences.

Example:



In the above example, an SEQ: BRANCH PASS command has been placed on Sequence 28. The prior seven sequences must be executed sequentially to ensure correct branch operation. It is not legal to branch to sequences 22 — 28. Any pin function may be programmed on sequences 21 — 28.

Executing a test sequence that contains the SEQ:CLEAR command forces the Test State to return to PASS. If a conditional loop is formed using the branch commands, the loop may have exited in the FAIL state. If comparisons of data following this loop are to affect the test, the Test State must be returned to PASS. Loops of this nature often provide synchronization of the test with external events.

Executing a test sequence that contains the SEQ:PAUSE command causes the current test sequence to hold indefinitely. The test will not proceed until the CONTINUE command is received from the system controller. While paused, the state of the I/O pins remains constant. Comparison samples are taken just prior to the execution of the following sequence.

Executing a test sequence that contains the SEQ:TRIG command causes the selected VXI backplane trigger to be asserted for the duration of that sequence. The trigger output can be directed to a TTLTRIG or ECLTRIG line with the CONNECT:TRIGGEROUT command. If a TTLTRIG line is selected, the line is asserted low. If an ECLTRIG is selected the line is asserted high.

Test Data

The test data controls the function of each I/O pin during test execution. Each I/O pin may be selected to generate one of seven functions during each test sequence: DRIVE HI, DRIVE LOW, INHIBIT, COMPARE HI, COMPARE LO, DRIVE HI AND COMPARE HI, and DRIVE LOW AND COMPARE LOW. Each of these functions are represented by a single character mnemonic (1, 0, X, H, L, S and R respectively, as shown in the following chart). Compare values should not be programmed on the last test vector in the test (defined by SEQ:END). It is recommended that tests terminate with all pins inhibited.

The following chart shows the Test Data Representation:

Character	Driver	Comparator
1	FORCE HI	MASK
0	FORCE LO	MASK
X	HIGH IMPEDANCE	MASK
H	HIGH IMPEDANCE	COMPARE HI
L	HIGH IMPEDANCE	COMPARE LO
S	FORCE HI	COMPARE HI
R	FORCE LO	COMPARE LO

Test data may be loaded into the VX4820 either via VXI Word Serial Protocol commands which utilize IEEE 488.2 syntax or through a binary download format, Fast Data Channel (FDC). Four commands support the programming of test data using Word Serial Protocol: SEQ:ADDRESS, SEQ:VECTOR, SEQ:VECTORINC, and WSLOAD. Test data files generated by Digital Toolbox include the WSLOAD command in the header and can be sent directly to the VX4820 using the Word Serial Protocol.

Fast Data Channel protocol significantly reduces the time required to load test data from the system controller. FDC transfers test data in binary format to/from the VX4820. The FDC file format is described in *Appendix C, Fast Data Channel Reference*, in this manual. The use of FDC protocol requires that the appropriate driver software exist on the system controller. Four commands support the transfer of test data by FDC protocol: FDCBASE?, FDCSIZE?, FDCLOAD and FDCLOAD?.

Test Execution

Executing a test requires three simple steps. First the test data must be programmed. This can be accomplished either by VXI Word Serial commands or downloading a binary test file via FDC protocol. Other test parameters such as clock rate may also be programmed at this time. Next, the test limits must be defined. The SEQ:START command defines the beginning sequence of the test and the SEQ:END command defines the last sequence of the test. Finally, the ARM and START commands begin the test execution.

Multiple tests can be loaded into the VX4820 as described above. Any of the tests can be individually executed by programming the start and end of test sequence. Once the test data and parameters have been programmed, the test can be executed as many times as necessary by sending the ARM and START commands.

Test Results

The current Test State of the VX4820 may be determined at any time by sending the STATE? command. This command provides information about test execution status, the number of sequence steps executed since the start of the test, and the current Test State. Once a failed test has stopped, test failure data can be retrieved with the FAILDATA? command.

Learn Mode

The VX4820 may be programmed to learn the data presented by the UUT during test development. In Learn mode, the COMPARE function is replaced with a LEARN function. The LEARN function captures the digital value of all pins that have been programmed to inhibit (X). After test completion, the captured values may be uploaded to the system controller for evaluation or test generation.

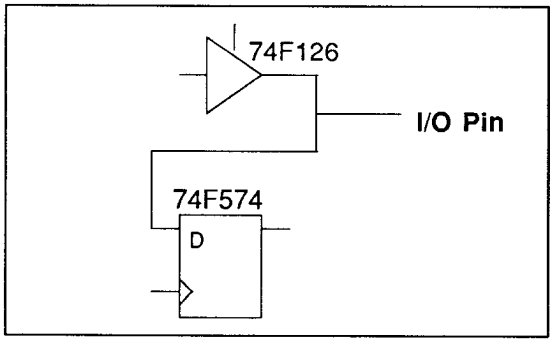
Executing a Learn mode test involves four steps. First, the test data must be programmed. Pins that are to drive the UUT should be set to 1 or 0. Pins that are to be learned should be set to X. The module is then set to Learn mode with the MODE command, and the test is executed in a normal manner. When the test completes, the test data is uploaded to the system controller using the SEQ:VECTOR? or FDCLOAD? commands. The test data will have been modified by the Learn mode execution. Pins which were set to 1 or 0 will still be set to the same values. However, pins which were set to X (or H or L) will have been modified to the learned H or L values.

The test state remains PASSED throughout the test. Branching should not be used when in Learn mode. The last test sequence (defined by SEQ:END) will not be learned.

I/O Pins

I/O pins are driven and received by an extended I/O pod. The pod supports short interconnection to the UUT, improving signal fidelity and simplifying tester configuration.

The pin driver and receiver are implemented using 74F126 and 74F574 TTL ICs. If a UUT fault exposes the pin to excessive potentials, damage is usually limited to the driver/receiver pair. These are socketed for easy replacement. The PINTEST? command can verify the function of the driver/receiver pairs. However, this command causes a test pattern to be driven onto all of the I/O pins and care should be used to ensure that the UUT is isolated from the pins.



TTL I/O Pin

To repair a damaged I/O pod, the pod must be removed from the test system and delivered to an appropriate repair station. The pod cable is removable so that the pod can be easily withdrawn from the system. Refer to *Section 9, Module Service*, for pod repair procedure.

CAUTION:

The product warranty does not cover damage to the pod as a result of exposing the pins to excessive potentials.

Backplane Triggers and Clocks

The VX4820 can route system and internally generated signals to and from VXIbus backplane trigger lines. Both the TTLTRG and ECLTRG backplane trigger busses are supported. The VX4820 utilizes the VXI local bus when multiple VX4820 modules are combined into a group. Group operation requires that the VX4820 modules be installed in adjacent slots for correct operation. Configuration jumpers allow the VX4820 to be configured as a VXI Anchor or Expander module.

Backplane triggers can be generated by programming the SEQ:TRIG ON command on selected test sequences. Triggers can be output on either TTLTRG 0 — 7, or ECLTRG 0 and ECLTRG 1, as selected by the CONNECT:TRIGGEROUT command. The Test State may also be routed to a VXI backplane trigger bus by the CONNECT:TSTSTATEOUT command. The Test State can be output on ECLTRG 0 or ECLTRG 1.

The VX4820 system clock can be driven onto, or received from, ECLTRG 0 or ECLTRG 1, as selected by the CONNECT:CLOCK command. This distribution path is used when an external clock source is selected for multi-module systems. It can also be used to synchronize other VXI modules to the VX4820. External clocks are received at the pod clock inputs or the front panel BNC.

The VX4820 utilizes the VXI Local Bus to distribute internal signals between modules that are operated in a group. The VX4820 module is keyed to avoid Local Bus conflicts with other VXI modules which utilize Local Bus lines. If the VXI Anchor/Expander protocol is followed there should be no Local Bus conflicts.

Systems of 128 — 768 Pins

Multiple VX4820 modules may be combined into a group. The module group operates in tandem and shares PASS/FAIL and START/STOP information. System pin skew specifications are guaranteed across all modules in the group, if the group has been correctly calibrated and configured.

All modules in a group are programmed to execute the same test sequence path. Commands such as SEQ:BRANCH and SEQ:PAUSE should be identically programmed on all group members. This assures that PASS/FAIL information shared by all group members is coherent. The test sequence trigger output operates independently and can be uniquely programmed on each module.

The module group utilizes Start/Stop protocol to synchronize the starting and stopping of each module. The GRPMODE and CONNECT:STST commands must be sent to each group module to ensure correct connection to the protocol. One of the group modules must be designated the group commander. The group commander is sent the START, STOP, PAUSE, and CONTINUE commands which control group test execution. The group commander is the only module in a group that sends signals to the system controller when test execution completes or pauses.

When using the STST Protocol, after a PAUSE sequence has executed (the SEQ:PAUSE must be set on all the modules), the test execution will pause. To continue the test, the Resource Manager should de-assert, then reassert the TTL STST trigger line. Reasserting the trigger line will resume the test execution on all the modules.

The maximum number of VX4820 modules that may be grouped into a system is 12, thus supporting systems of up to 768 pins. All modules must be installed into adjacent slots of the same contiguous VXI backplane.

NOTE

COMPARES should not be programmed on the last five test sequences in multi-module systems.

The mainframe must be capable of supplying adequate power for the modules installed. On large systems, the power required from the +24V and -24V can exceed the mainframe capacity. An auxiliary input has been provided on the front panel of the VX4820 module for these supplies.

Diagnostics

The VX4820 provides 2 front panel LEDs that indicate the operation level of the module. At system power-up, the module executes internal diagnostics to determine its fitness for operation. During execution of the diagnostics, the green READY LED is not illuminated; if the module passes its internal diagnostics the READY LED will illuminate within 5 seconds of system power-up. If the READY LED fails to illuminate, a module failure is indicated. This failure can be the result of a module fault discovered during power up diagnostics, or a mainframe fault in which the correct power or signals are not provided. After power-up, a complete set of module and pod diagnostics can be executed using the TST? command.

During normal operation, a runtime diagnostic is provided that tests the driver/receiver pairs in the I/O pods. The PINTEST? command causes the I/O channels to be driven both HIGH and LOW in a specific test pattern. This diagnostic is intended to be used whenever the integrity of the I/O pin driver/receiver pairs needs to be checked, such as after a failed UUT has been found that could cause damage to the I/O drivers.

Calibration

When multiple VX4820 modules are configured in a mainframe, a calibration that eliminates the skew between modules must be performed to guarantee correct performance of the group, and to ensure pin skew specifications are met. This calibration need only be repeated if the configuration of the group is modified, such as by module replacement or rearrangement of the VX4820 modules in the mainframe.

A CAL OUT BNC is provided on the VX4820 front panel to support system calibration. This output is TTL compatible and is capable of driving 50 Ω terminated coaxial cables. Two calibration constants are stored on each module in nonvolatile memory. The first constant is for calibrating internally generated clocks and the second constant is for calibrating externally supplied clocks. During the calibration procedure, the output edges of the CAL OUT BNC are aligned by modifying the calibration constants. For a detailed description of the calibration procedure refer to *Section 2, Preparation for Use.*

Standard Accessories

The following standard accessories are provided with the VX4820:

- *VX4820 Digital Test Module Users Manual*
- Digital Toolbox Software
- Virginia Panel 64-pin Connector Housing — Vendor Part # 510-108-101
- Virginia Panel Crimp Male Pins* — Vendor Part # 610-110-108

Options

The VX4820 is available with an Option 1 configuration that includes a second pod, giving a total of 64 pins.

Optional Accessories

The following optional accessories are available for the VX4820:

- Auxiliary pod power connector — 1 x 3 connector body. Vendor Part # MOLEX 09-50-7031, Tektronix Part # 204-0671-00
- Auxiliary pod power pins — Vendor Part # MOLEX-108-0105, Tektronix Part # 131-1790-00
- Virginia Panel 64-pin Connector Housing — Vendor Part # 510-108-101
- Virginia Panel Wire Wrap Male Pins — Vendor Part # 610-110-113
- Virginia Panel Crimp Male Pins* — Vendor Part # 610-110-108

*Refer to the Virginia Panel catalog for crimper and removal tool.

Specifications

Performance Conditions

The performance characteristics listed in the specifications tables are valid within the following limits:

- The VX4820 must have warmed up for 20 minutes or more.
- The VX4820 must be operating in an environment within the limits described in the Environmental Specifications.
- The VX4820 has been calibrated at the specified operating temperature.

Any condition unique to a particular characteristic but not listed above is stated as part of that characteristic.

The electrical and environmental performance limits, together with the related validation procedure, comprise a complete statement of the electrical and environmental performance of a calibrated Digital Test Module.

Definition of Terms

- **UUT**: Unit Under Test
- **POD**: A device connected to the VX4820 module via a ribbon cable providing active drive to/from the UUT.
- **GROUP**: VX4820 modules installed into adjacent slots within a VXI mainframe. These modules share information and operate in tandem, as if they were a single module.
- **TEST STATE**: A global signal which reflects the current test results. The Test State can be PASSED or FAILED.
- **NRZ**: Non-Return to Zero. The output data remains until the end of the cycle.
- V_{OH} , V_{OL} , V_{IH} , V_{IL}: Voltage of the output or input at a High / Low drive level.
- I_{OL} , I_{OH}: Current with the output Low / High.

Table 1-1. Environmental Specifications

Characteristic	Description
Temperature:	
Operating	0 — 50° C
Non-operating	-55 — +75° C
Humidity:	
0 — 30° C	95% relative humidity
30 — 40° C:	75% relative humidity
40 — 50° C	45% relative humidity
Altitude:	
Operating	3.05 km (10,000')
Non-operating	12 km (40,000')
Vibration:	
Non-operating	Withstands 0.38 mm (0.015") p-p; 10 — 55 Hz sine wave, 15 minutes each axis; and 10 minutes each axis at resonance or 55 Hz
Shock:	
Non-operating	Withstands 30 g, half sine, 11 ms, 18 shocks
Electromagnetic Compatibility	Within limits of FCC Regulations, Part 15, Subpart J, Class A (Module without pods).
Electrostatic Immunity while Operating:	(POD Cables must be installed, does not include pin electronics)
Non-interruptive	15 kV, 100Ω in series with 500 pF
Non-destructive	20 kV, 100Ω in series with 500 pF
Safety	Listed to UL 1244 standard as an accessory for test and measurement equipment. UL listing applies only when installed in a UL listed product.

NOTE

The VX4820 will meet selected categories of MIL-T-28800D, Class 5 environmental requirements. Testing is accomplished using procedures contained in Tektronix Standards for Class 5, which, in some instances, are stricter than MIL-T-28800D, Class 5.

Table 1-2. Electrical Specifications

Characteristic	Performance Requirements	Supplemental Information
VXI Revision Level		VXI System Specification, Revision 1.3
Interface Type		Message Based (I4)
Protocols		Word Serial (WSP) Fast Data Channel (FDC)
Execution Times		Run time commands are executed in <0.25s. FDC of 16,350 vectors to a single VX4820 module from RAM on the VX4530 in <5 s
Interfaces		ECLTRG 0, 1: Test State output, clock input and output, trigger output,TTLTRG 0 — 7: Trigger output, STST input and output, LOCAL BUS A00 — A01, C00 — C01: ECL class
Protocol		Start/Stop (STST)
Number of pins		32/64 (1 or 2 POD's, 32 pins each)
Number of sequence steps		16,351
Pin Functions		FORCE INHIBIT COMPARE w/MASK DRIVE & COMPARE w/MASK LEARN (Learn mode only)
Drive	High Level ≥ 2 V @ I_{OH} ≤ 15 mA Low Level ≤ 0.8 V @ I_{OL} ≤ 64 mA (sink)	High Level ≥ 2.4 V when I_{OH} ≤ 3 mA (source) Low Level ≤ 0.55 V when I_{OL} ≤ 48 mA (sink)48 mA
Short Circuit Current		-225 mA maximum
Format		Fixed NRZ
Skew between any two pins	≤ 10 ns	
Inhibit		Presents a single FTTL load -0.6 mA max.
Compare	Low ≤ 0.8 V High ≥ 2 V	Intermediate band > 0.8 V — < 2 V Setup time 25 ns prior to end of cycle. Hold time 0 ns prior to end of cycle (first output transition)

Table 1-2. Electrical Specifications (continued)

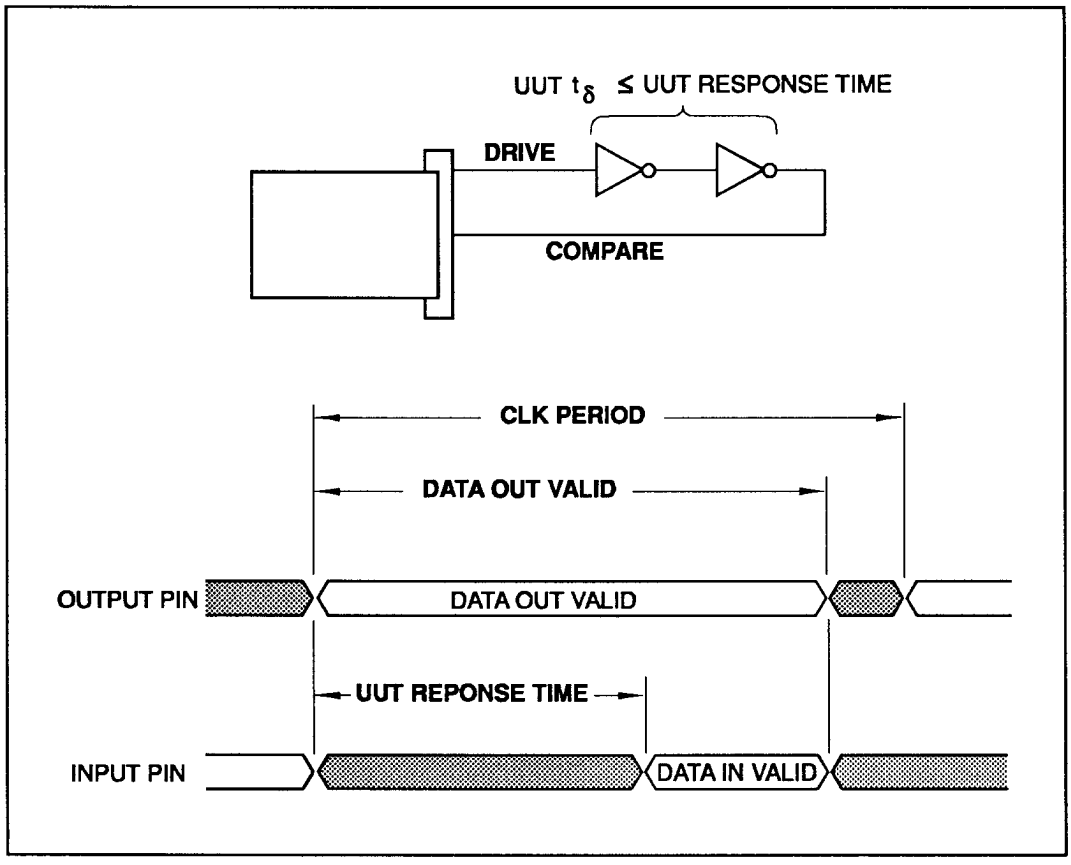
Characteristic	Performance Requirement	Supplemental Information
Pin termination		Diode clamp to +5V and Gnd, Optional AC termination to Gnd (100pF in series with 100 Ω).
Pin Capacitance		50 pF in parallel with the optional AC termination
Non-destructive I/O Voltage		-0.5 — +7 V Drivers and receivers are socketed
Internal Clock Rate	20 MHz max	50 — 3,276,700 ns, programmable, Accuracy ± 100 ppm (same as CLK10)
External Clock Rate	20 MHz max	CLK BNC connector or pod CLK input, positive polarity. ECLTRG with programmable polarity
CLK BNC Input	25 ns min Clk low 25 ns min Clk high $V_{IL} \leq$ 0.8 V, $V_{IH} \geq$ 2 V	Safe Range ± 10 V Impedance ≥ 10 KΩ Typical delay BNC to STB: 68 ns
POD CLK input	25 ns min Clk low 25 ns min Clk high $V_{IL} \leq$ 0.8 V, $V_{IH} \geq$ 2 V	Safe Range ± 10 V Impedance: 1 FTTL load (-0.6 mA max) Typical delay CLK to STB: 84 ns
POD STB output		PIN data setup time prior to STB ≥ 25 ns, PIN data hold time after STB ≥ 7 ns. Compare input setup time prior to STB ≥ 10 ns. Compare input hold time after STB ≥ 1 ns
UUT Response Time		Data Out Valid ≥ CLK period – 10 ns. UUT Response Time ≤ CLK period – 25 ns (for same cycle compare). See the UUT response time figure for details.
Fail Data Register		Stores sequence and pin data sampled on all module pins when the Test State transitions from PASSED to FAILED. Fail Data persists while the Test State remains FAILED.
Test State Outputs		The Test State can be driven onto ECLTRG0 or 1 FAILED: ECLTRG = HIGH PASSED: ECLTRG = LOW
Trigger Output		Triggers can be driven onto TTLTRG0..7 or ECLTRG0..1 ON: TTLTRG = LOW, ECLTRG = HIGH OFF: TTLTRG = HIGH, ECLTRG = LOW

Table 1-2. Electrical Specifications (continued)

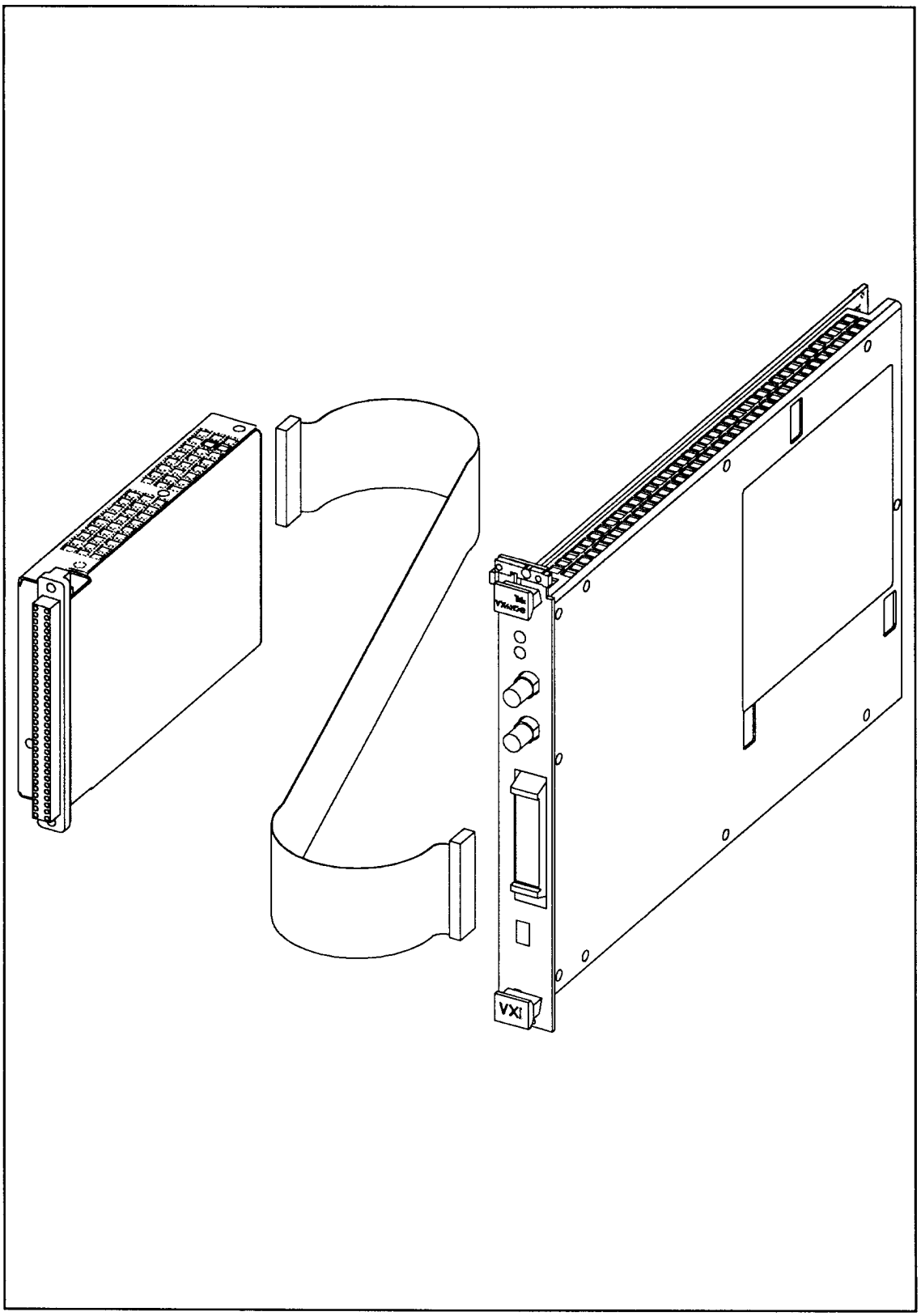
Characteristic	Performance Requirement	Supplemental Information
LEARN Mode	Max clock rate 20 MHz 25 ns min Clk low 25 ns min Clk high	Channels can be driven or inhibited. Non-driving pins are stored on each test step and can be recalled after termination of the test.
Start		Begin test
Stop		Force termination of test
Pause		Halt on programmed test sequence(s)
Single Step		Halt on every test sequence, Continue to next sequence
Test State		All pin comparisons are logically combined to a single Test State. Any failed comparison transitions the Test State to FAILED. Sourced and sensed by all modules in a group. Driven by all test comparisons on modules in a group
Branch		Settable branch on any sequence. Conditional BRANCH on Test State PASSED or FAILED. Unconditional BRANCH ALWAYS. Eight sequences are required for each branch.
Vector Counter		A 32 bit counter provides the number of sequences executed since the start of the test.
Supplies		All supplies must meet *VXI Specification, Revision 1.3* for tolerance and noise
Voltages and Currents		+24 Vdc @ 0.90 A I_{PM} +12 Vdc @ 0.00 A I_{PM} +5 Vdc @ 3.20 A I_{PM} -2 Vdc @ 0.14 A I_{PM} -5.2 Vdc @ 0.44 A I_{PM} -12 Vdc @ 0.30 A I_{PM} -24 Vdc @ 0.78 A I_{PM}

Table 1-3. Mechanical Specifications

Characteristic	Description
Overall Module Dimensions	
Height	261.85 mm (10.309")
Width	30.18 mm (1.188")
Depth	366.42 mm (14.426")
Overall POD Dimensions	
Height	30.48 mm (1.2")
Width	144.2 mm (5.6")
Depth	226.1 mm (8.9")
Weight	
Nominal (Module and Pods)	3.1 kg (6.8 lbs)
Shipping	6.8 kg (15 lbs)
Cooling Requirements	2 L/s at 0.75 mm H_2O



UUT Response Time Specification



VX4820 Digital Test Module

Section 2
Preparation for Use

Configuration

Calculating System Power and Cooling

When a VXI system is integrated the power delivery and cooling capacity of the mainframe should meet or exceed the power and cooling requirements of the installed modules. The power requirements can be calculated by simply adding the individual module requirements and comparing the sum with the power delivered by the mainframe. The power requirements for the VX4820 are listed in the following chart:

Power Supply	IPM (Peak module)
+24 V	0.90 A*
+12 V	0.00 A
+ 5 V	3.20 A
- 2 V	0.14 A
- 5.2 V	0.44 A
- 12 V	0.30 A
- 24 V	0.78 A*

* See *Auxiliary Pod Power* topic, following.

The mainframe should provide adequate cooling to ensure reliable operation of the module. The module cooling requirement is:

2 L/s with 0.75 mm H_2O backpressure

Auxiliary Pod Power

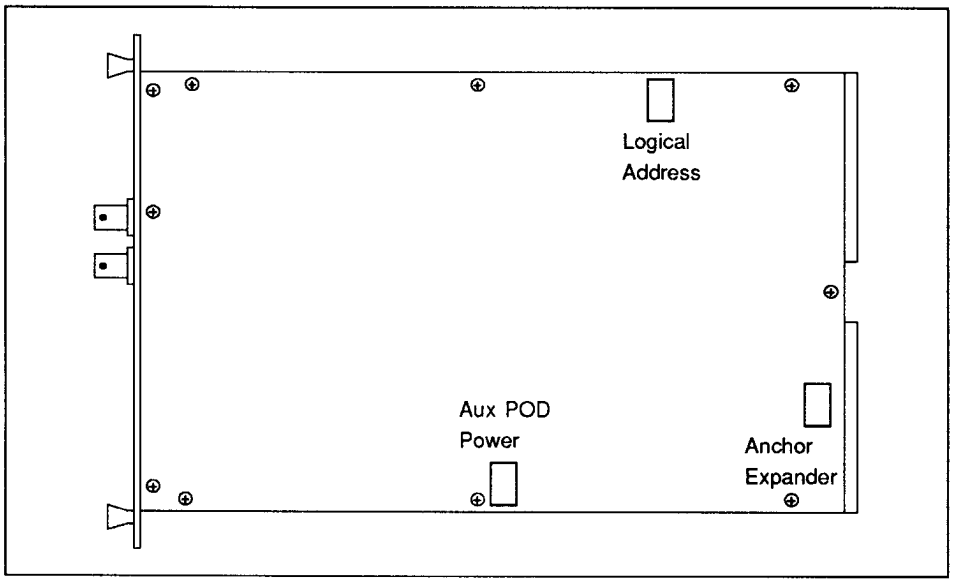
The VX4820 has the capability to utilize an external power source for the +24V and -24V supplies. This may be necessary when using more than 5 VX4820 modules in a typical VXI mainframe. If the mainframe delivers less power than the power required by the VX4820 modules on the 24V supplies, auxiliary power can be used to eliminate the conflict. To utilize the auxiliary power, J5900 and J5902 must be moved from the **INT** position to the **EXT** position. The front panel connector, AUX POD PWR IN is clearly marked +24DCV, GND, -24DCV.

CAUTION

Care should be taken to ensure correct polarity, as damage to the VX4820 module will occur if the polarity is reversed. The external power must provide +24V ± 5%, -24V ± 5% with ≤ 100mV ripple.

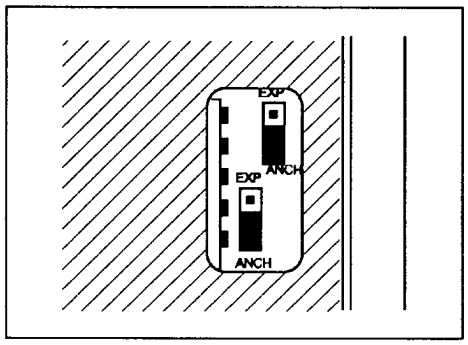
Module Jumpers

The VX4820 contains four user-configurable jumpers. Openings in the module shield allow easy access to the jumpers. To change the jumper settings, the module must be removed from the mainframe.



Logical Address and Jumper Locations

Two jumpers, J9700 and J9800, disconnect the VXI Local Bus A00 and A01, allowing the module to be configured according to VXI Anchor/Expander protocol. Anchor/Expander protocol need only be observed if there is a local bus conflict between the VX4820 module and other adjacent modules. When Anchor/Expander protocol is observed, the module in the lowest number slot is designated the system anchor and other VX4820 modules are designated expanders.

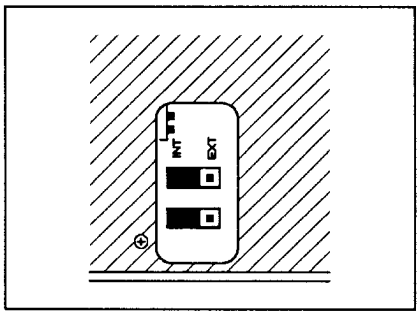


Anchor/Expander Jumpers

If the module is to be configured as an anchor card, J9700 and J9800 should be configured to the **ANCH** side. If the module is an expander (factory default), then J9700 and J9800 should be configured to the **EXP** side. This will disconnect the module from the left side (A) local bus. When a module is configured as an anchor, it must be the left-most module of a VX4820 group.

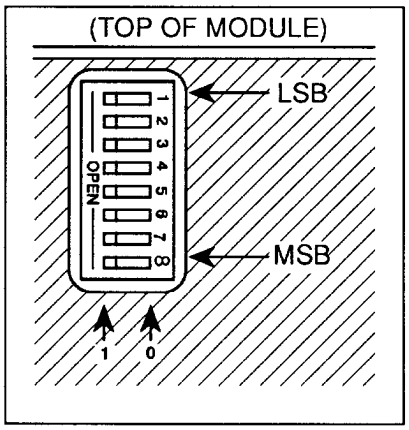
Normally, a VX4820 should be configured as a VXI Anchor module (see previous illustration). If two VX4820 modules are to be operated independently, then they should either be placed in non-adjacent slots within the Mainframe, or they should be configured as VXI Expanders.

The POD power jumpers determine the source of the +24V and -24V power used to supply the pods. For a description of the use of these jumpers, refer to *Auxiliary Pod Power* (at the beginning of this section).



Aux Pod Power Jumpers

Each VXI module in a system must have a unique logical address. An 8 bit switch is provided to select the VX4820 logical address. The LSB of the switch is toward the top of the module, and the MSB is toward the bottom. When the switch is in the open position, a 1 is selected for that bit position. When the switch is in the closed position, a 0 is selected for that bit position.



Logical Address Switch

The VX4820 supports VXI Dynamic Auto Configuration (DAC). The logical address of a DAC device is automatically assigned at system power-up. To select Dynamic Auto Configuration all switch positions should be set to open (1). When a VXI Resource Manager that supports DAC devices configures the system, it will automatically assign a logical address to the module.

Configuring the Backplane Jumpers

VXIbus mainframes contain daisy-chain jumpers for the Bus Grant (BG0 — BG3) and Interrupt Acknowledge (IACK) signals. When a VX4820 module is installed into a mainframe slot, the IACK jumper must be configured so that the slot is not bypassed (jumper removed).

WARNING

The mainframe compartment may contain high current. Do not configure jumpers unless you are qualified to do so.

GPIB Communications Terminator

The IEEE-488.1 GPIB Standard allows the message terminator to be either an EOI (End-or-Identify) terminator or a LF (Line-Feed) character. If the slot 0 module uses only the LF terminator, the Word Serial NEWLINE command should be sent to the VX4820 to enable or disable the LF character as the termination character.

NEWLINE Termination Character

The Word Serial NEWLINE command can be enabled or disabled by using the NEWLINE Enable or NEWLINE Disable commands.

The NEWLINE Enable command (0x0100) is used to enable the NEWLINE (0x0A) character as the Word Serial Protocol read operation termination character.

The NEWLINE Disable command (0x0200) is used to disable the NEWLINE (0x0A) character as the Word Serial Protocol read operation termination character.

Installation

Module Installation Using Ejector Handles

Use the following procedure to install the module into a Tektronix Mainframe. If installing the module in another mainframe the procedure may need to be modified.

1. Ensure that both the rear panel power ON/OFF switches and front panel ON/STANDBY switch on the Mainframe are turned to OFF and STANDBY, respectively.

2. Insert the module into the Mainframe top and bottom module guides for the slot(s) where the module is to be installed; push the module into the Mainframe as far as it will go without forcing it, seating the P1 and P2 connectors.

   CAUTION

   A C-size module can be damaged if installed wrong. Use care when installing modules in a VX1500 D-size mainframe. We recommend using a C-size to D-size adapter (Tektronix part number 014-0070-00) which should be installed in the mainframe P3 (bottom) connector. When installing the module, insert the module top edge into the mainframe top guide for the slot(s) where the module is to be installed; insert the module bottom edge into the adapter card guide. Make sure the module is fully seated into the mainframe P1 (top) and P2 (middle) connectors.

3. Ensure the module is fully seated into its VXI connectors and the front panel is flush with the front of the Mainframe chassis. Then tighten the top and bottom module retainer screws to secure the module in the Mainframe.

Module Removal Using Ejector Handles

Use the following procedure to remove the module from a Tektronix Mainframe. If removing the module from another mainframe, the procedure may need to be modified.

1. Ensure that both the rear panel power ON/OFF switches and front panel ON/STANDBY switch on the Mainframe are turned to OFF and STANDBY, respectively, and the top and bottom module retainer screws are disengaged.

2. Simultaneously push the upper Ejector Handle fully up and the lower handle fully down to EJECT the module. Once the module is ejected, it can be removed from the Mainframe.

Pod Cable Assembly

The VX4820 Digital Test Module includes two remotely-mountable pods. These pods connect to the module via 6' ribbon cables. Both the connectors on the POD and module are keyed to ensure correct orientation. The keys are two triangular shaped grooves on the side of the connector.

Either end of the cable can be inserted into the module or the POD; the cable is electrically symmetrical. However, the cables are not physically symmetrical. The

exit path of the ribbon cable from the front of the VX4820 module will be different depending on which end of the cable is installed in the module.

The module and POD connectors provide a positive locking mechanism to ensure a reliable connection. When installing the connector, the locking ears should be rotated out away from the connector. As the connector is inserted, the locking ears will move into position around the connector body, providing a strain relief for the connector. To remove the connector, rotate both the bottom and top ears away from the connector and then pull the connector free.

The POD has been designed to be easily integrated into a Virginia Panel™ test head adapter. The POD occupies two test head slots and provides a standard high quality Virginia Panel connector. The POD can be mounted in the Virginia Panel interface directly, using two standard screws to mount the Virginia Panel connector.

The pods can dissipate up to 10 W of heat when operated at a 100 % duty cycle at 20 MHz. Care should be taken to ensure adequate airflow is available in the test head. In most manufacturing test environments the actual test execution duty cycle is much less than 50 % and there are no special cooling requirements.

Group Calibration

Multiple VX4820 modules can be combined in a group to form a digital test system of up to 768 pins. When such a group is configured or the group configuration changed, due to module exchange or rearrangement, the modules forming the group should be calibrated. Group calibration ensures correct group operation and is required to meet pin skew specifications across all pins in the system.

Each VX4820 module contains two calibration values that are stored in nonvolatile memory. The first value is for internally generated clocks, and the second for externally sourced clocks. If one of the clock sources is not used, the adjustment of the respective calibration constant is not required.

A front panel CAL OUT BNC provides a calibration signal for each VX4820 module. The CAL OUT signal is TTL level and can drive a 50 Ω terminated coax cable.

Calibration of a VX4820 group is required to eliminate clock skew generated by both the VXI backplane and internal module circuits in multi-module systems. When a test executes, the VX4820 system clock is output on the CAL OUT BNC. Calibration involves adjusting the calibration constants of each module so that the system clocks presented at the CAL OUT BNC are aligned within 2 ns

Group Calibration Procedure

When multiple VX4820 modules are grouped, each module should be calibrated to eliminate clock skew. This calibration is done once when the modules are configured in the mainframe. If an external clock is used, the module that accepts the external clock is considered part of the configuration (changing which module accepts the external clock changes the configuration). VX4820 group calibration ensures that skew between I/O pins is held to a minimum.

The VX4820 CAL OUT front panel BNC provides visibility of clock timing relationships between modules. Adjustable clock delay circuits in the VX4820 module allow these timing relationships to be adjusted. The goal of the calibration procedure is to minimize clock skew between modules, as seen at the CAL OUT BNC.

The calibration procedure contains two major steps. First, the clock delay on all modules is set to 0. The VX4820 group is then programmed to execute an infinite loop, forcing clocks onto the CAL OUT BNC's. Using the oscilloscope, the timing relationship between each of the CAL OUT BNC's is measured to determine which module has the greatest clock delay (the clock transition latest in time). This module is designated as the reference. The calibration delay of the reference will remain 0. Next, the clock delay of each of the remaining modules is increased until its CAL OUT signal is aligned with the reference module's CAL OUT. Once the clock delay is correctly adjusted, the calibration constant should be saved in the modules nonvolatile memory.

Internal Clock Calibration

Required Equipment

250 MHz dual trace oscilloscope, i.e. Tektronix 2465

VXI System, i.e. Tektronix VX1405, VX4530

Perform Steps 1 and 2 on each module to be calibrated.

1. Set the module's internal clock calibration constant to zero.

   NOTE

   The CLKCAL command programs both the internal and external clock calibration values. The CLKCAL? command should be sent to retrieve the current calibration values. When the internal clock is being calibrated, the external clock calibration constant returned by the CLKCAL? command should be saved. When the CLKCAL command is sent to adjust the delay for the internal clock value, the external clock calibration should be set to its prior saved value, leaving it undisturbed.

   Send the following commands to each module to be calibrated:

   ```
   INIT
   PASSWORD "DT"
   CLKCAL?
   CLKCAL 0,<saved external value>
   ```

2. Program the module with an infinite loop. Send the following commands to each module to be calibrated:

   ```
   NEW
   INTCLKRATE 1000
   SEQ:START 0
   SEQ:BRANCH ALWAYS, 0, 15
   SEQ:END 30
   ARM
   START
   ```

3. Measure the relative delay between the rising edges supplied by the CAL OUT BNC of the modules in the group utilizing a dual-channel oscilloscope. Determine which module provides the most delayed (latest) output clock edge. This module will be the reference module, and its internal clock calibration value

will remain zero. All other modules in the group will be referred to as test modules.

Perform Steps 4 through 8 on each test module.

4. Compare the CAL OUT rising edge of the reference module with the CAL OUT rising edge of the test module. Determine which rising edge occurs later in time. If the reference module edge occurs later than the test module, the test module's calibration value should be incremented by 1 in Step 6. In either case, continue on to Step 5.

5. Stop the current test module by sending the command:

   STOP

6. Adjust the clock calibration constant as determined in Step 4. If it was determined that the internal clock calibration value of this test module is correct, go to Step 8; otherwise, increment the value. Send the following commands to the test module:

   CLKCAL?
   CLKCAL <internal value + 1>,<saved external value>

7. Restart the test module by sending the following commands:

   ARM
   START

Go to Step 4.

8. When the internal clock calibration value of the current test module is correct, the calibration value should be saved into nonvolatile memory. Send the following command to the test module:

   NVSAVE

Repeat Steps 4 through 8 until all the test modules have had their internal clock calibration constants saved into nonvolatile memory.

9. The reference module's internal clock calibration value also needs to be saved into nonvolatile memory. Send the following commands to the reference module:

   STOP
   NVSAVE

Internal clock calibration of the group is now complete.

External Clock Calibration

Required Equipment

250 MHz dual trace oscilloscope, i.e. Tektronix 2465

VXI System, i.e. Tektronix VX1405, VX4530

External Clock Source, i.e. Tektronix PG502

For external clock calibration, the VX4820 modules share a common external clock connected to the front panel CLOCK IN BNC on one of the modules. **The module selected to receieve the external clock source must be the designated group commander used during normal operation.** The group commander can be any of the modules, although, usually the left-most module in the mainframe is selected.

NOTE

The method for starting and stopping described here should no be used except for external clock calibration. Modules will not necessarily execute the same sequence at the same time, as required during normal operation as a group.

Select a 1 MHz square wave, and adjust the drive level of the external clock source to TTL levels (0 V LO and 2.4 V HI) when terminated into 50 ohms.

Connect the external clock source to the CLOCK IN BNC on the group commander selected above using a 50-ohm coax cable which is terminated into 50 ohms at the VX4820 end.

1. Set each module's external clock calibration constant to zero.

   NOTE

   The CLKCAL command programs both the internal and external clock calibration values. The CLKCAL? command should be sent to retrieve the current calibration values. When the external clock is being calibrated, the internal clock calibration constant returned by the CLKCAL? command should be saved. When the CLKCAL command is sent to adjust the delay for the external clock value, the internal clock calibration must be set to its prior saved value, leaving it undisturbed.

   Send the following commands to each module to be calibrated:

   ```
   INIT
   PASSWORD "DT"
   CLKCAL?
   CLKCAL <saved internal value>,0
   ```

2. Program all modules except the group commander with an infinite loop and to receive the external clock from the backplane. Send the following commands to all modules, except the group commander:

   ```
   NEW
   GRPMODE GROUP
   CONNECT:CLOCK ECLTRG0
   ```

```
CONNECT:STST TTLTRG0
SEQ:START 0
SEQ:BRANCH ALWAYS,0,15
SEQ:END 30
ARM
```

3. Program the group commander with an infinite loop and to source the external clock onto the backplane. Send the following commands to the group commander only:

```
NEW
GRPMODE COMMANDER
CONNECT:CLOCK CLKBNCECL0
CONNECT:STST TTLTRG0
SEQ:START 0
SEQ:BRANCH ALWAYS, 0, 15
SEQ:END 30
ARM
START
```

4. Measure the relative delay between the rising edges supplied by the CAL OUT BNC of the modules in the group utilizing a dual-channel oscilloscope. Determine which module provides the most delayed (latest) output clock edge. This module will be the reference module and its external clock calibration value will remain zero. All other modules in the group will be referred to as test modules.

Perform Steps 5 through 9 on each test module.

5. Compare the CAL OUT rising edge of the reference module with the CAL OUT rising edge of the test module. Determine which rising edge occurs later in time. If the reference module edge occurs later than the test module, the test module's calibration value should be incremented by 1 in Step 7. Otherwise, the current external clock calibration value is correct. In either case, continue on to Step 6.

6. Stop the current test module by sending the command:

```
STOP
```

7. Adjust the clock calibration constant as determined in Step 5. If it was determined that the external clock calibration value of this test module is correct, go to Step 9; otherwise, increment the value. Send the following commands to the test module:

```
CLKCAL?
CLKCAL <saved internal value>,<external value + 1>
```

8. Restart the test module. If the current test module is the group commander, send the following commands:

```
ARM
START
```

Otherwise, send the following command to the current test module:

ARM

Go to Step 5.

9. When the external clock calibration value of the current test module is correct, the calibration value should be saved into nonvolatile memory. Send the following command to the test module:

   NVSAVE

Repeat Steps 5 through 9 until all the test modules have had their external clock calibration constants saved into nonvolatile memory.

10. The reference module's external clock calibration value also needs to be saved into nonvolatile memory. Send the following commands to the reference module:

    STOP
    NVSAVE

External clock calibration of the group is now complete.

Section 3
Operation and Programming

Connectors and Indicators

The following paragraphs describe the VX4820 connectors and indicators.

Connectors

There are four connectors on the front panel that allow connection of external clock, calibration output, and two Pod Interface cables.

- CLK IN BNC Connector — A TTL level input that allows the user to provide an external system clock.
- CAL OUT BNC Connector — A TTL level output that provides a calibration signal for multi-module groups
- POD INTERFACE Connectors (2) — Each of these connectors supports one Input/Output POD.

Indicators

There are two LED indicators on the VX4820 front panel: READY and ACCESSED.

- READY — A green indicator that illuminates when the VX4820 is operating properly. Failure of this indicator to illuminate within 5 seconds of powering up the system indicates a module failure condition.
- ACCESSED — An amber indicator that illuminates when the MODID line is asserted by the Slot 0 device, or flickers when the VX4820 is engaged in communication via the VXIbus.

Getting Started

This section will help you become familiar with the VX4820 module. The initial portions are intended to help establish communications with the module and build confidence in your ability to program and execute simple tests. The later portions cover more advanced topics. Examples are given throughout that you can use to verify system operation.

Module Installation

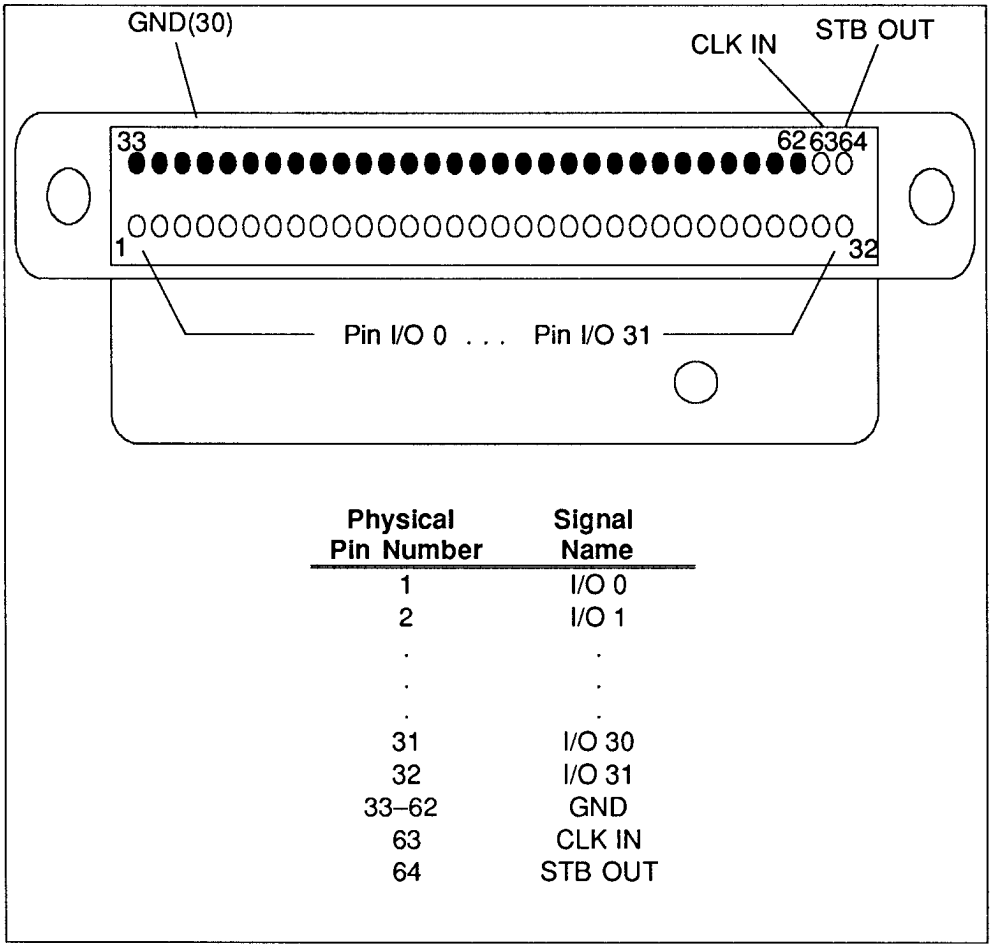
When the VX4820 is unpacked from its shipping carton, the contents should include the following:

- (1) VX4820 module
- (1 or 2) VX4820 TTL pods, 32/64 pins
- (1 or 2) 1.8-meter (6 ft) POD cables
- (1) Users manual
- (1) 5 1/4" (1.2MB) Digital Toolbox floppy disk
- (1) 3 1/2" (1.4MB) Digital Toolbox floppy disk

All components should be inspected for any damage due to shipping. The software registration forms should be filled out and mailed.

The VX4820 module may then be installed into any appropriate C- or D-size VXI mainframe in slots 1 — 12. The mainframe should be checked to ensure it is capable of providing adequate power and cooling for the VX4820. The power and cooling requirements are clearly labeled on the VX4820 side cover. For further details refer to *Section 2, Preparation for Use.*

The pods are connected to the VX4820 module using the 1.8-meter POD cables. Connect one end of the POD cable to one of the pods. The connector is keyed to ensure correct orientation and either end of the cable may be connected to the POD. Next insert the remaining end of the POD cable into the POD 0 connector on the module front panel (this connector is also keyed). Repeat the process for the second pod connecting to the module POD 1 connector (if purchased).



Physical Pin Number	Signal Name
1	I/O 0
2	I/O 1
.	.
.	.
.	.
31	I/O 30
32	I/O 31
33–62	GND
63	CLK IN
64	STB OUT

VX4820 Pod Pin Assignment

Once the module and pods have been installed, the mainframe may be powered up. The VX4820 will initially execute its internal diagnostics, which should complete in less than 5 seconds. When the module has successfully completed diagnostics the green READY LED will illuminate.

Basic Communications

Communications with the VX4820 involves sending ASCII command strings and retrieving response strings. The method used to address the module is specific to the system controller and interface device. The following examples assume that a Tektronix VX4530 embedded controller is being used. However, the command strings may be sent by any VXI interface device which is capable of communicating via Word Serial Protocol.

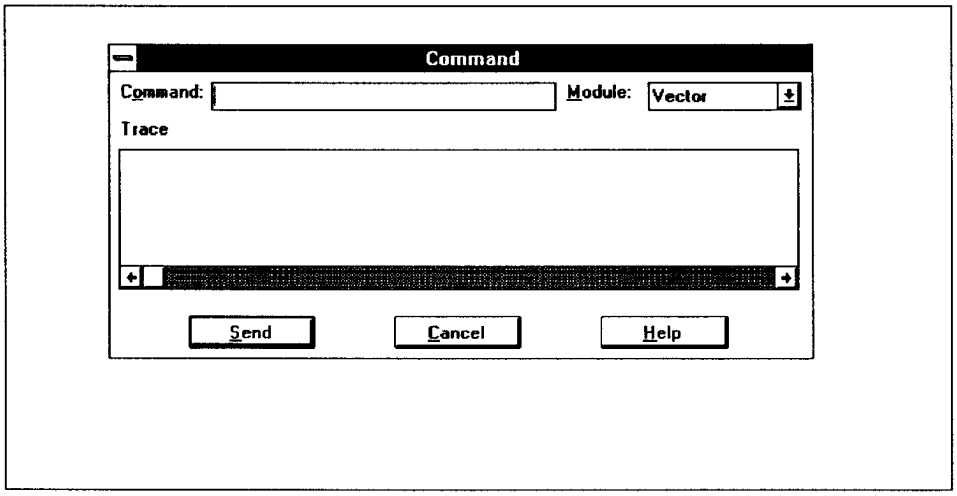
The VX4820 product includes the Digital Toolbox software package. Included in this software is an application called *Debug*. Debug can be used to send command strings to the VX4820 module, and can be used to send commands to the VX4820 to try out the examples shown.

Install the VX4820 with pods in an appropriate VXI mainframe with VX4530 controller. Start up Windows and double-click the Debug icon. Pull down the **File** menu

and select **New**. In the Configure dialog box, select your module from the device list and click **Add** and **Ok**. This configures Debug to talk to the VX4820 module.

Click on the **Send Command...** menu to display the Command dialog box. VX4820 commands may be directly typed into the **Command** entry field. If the command terminates in "?" , the response is automatically retrieved. Click on the **Command** entry field and type **ID?**. Click **Send** to send the command. The ACCESSED LED on the VX4820 should flash as the command is received. The response from ID? will be retrieved automatically and displayed in the Trace box. Commands are displayed in blue and responses in red.

Any command can be sent in this manner using Debug. If the VX4820 detects an error with the command or its parameters, it will place a message in its event queue. The message can be retrieved by sending the **EVMSG?** command. The queue will hold a maximum of five messages.



Debug Send Command Dialog Box

Exercising the Module I/O Pins

The simplest way to generate viewable signals from the VX4820 is to write a test that executes repeatedly. This allows the output signals to be viewed directly with an oscilloscope.

Following is a series of commands that can be sent to the VX4820 which generate a continuous binary count pattern output on Pins 1 — 3 of POD 0:

Example 1

```
INIT
NEW
SEQ:VECTOR "0000",0
SEQ:VECTOR "0001",1
SEQ:VECTOR "0010",2
SEQ:VECTOR "0011",3
SEQ:VECTOR "0100",4
```

```
SEQ:VECTOR "0101",5
SEQ:VECTOR "0110",6
SEQ:VECTOR "0111",7
SEQ:BRANCH ALWAYS,0,7
SEQ:START 0
SEQ:END 30
INTCLKRATE 1000
ARM
START
```

To check the test execution status from the system controller send the following command:

```
STATE?
```

The response from this command should be similar to the following:

```
STATE RUNNING,4,8781824,PASS,PASS;
```

This response informs you that the test is running, the current sequence, number of executed test sequences, and that the Test State is PASSED. Since the test is running, the current sequence and number of executed test sequences will be different each time you send STATE?

To stop the execution send the following:

```
STOP
```

If the test fails to execute as expected, try sending the following:

```
EVMSG?
```

This will return information about any errors that may have occurred while typing in the test data and commands. An example response (when there are no messages) is:

```
EVMSG 0,"No events to report - queue empty";
```

Writing a Simple Test

In example 1, the test executed exercised only one of the four possible pin functions, DRIVE. The other functions, INHIBIT, COMPARE, and DRIVE AND COMPARE, will be used in the next example. In order to reproduce this example Pins 0, 1, and 2 of POD 0 need to be connected together. Each of the pins will take turns driving the connection while the remaining pins compare or ignore the value that is driven.

Example 2

```
INIT
NEW
SEQ:VECTOR "XXX",0
SEQ:VECTOR "XH1",1
SEQ:VECTOR "HX1",2
SEQ:VECTOR "HHS",3
SEQ:VECTOR "X0L",4
SEQ:VECTOR "L0X",5
SEQ:VECTOR "LRL",6
SEQ:VECTOR "1XH",7
SEQ:VECTOR "1HX",8
SEQ:VECTOR "SHH",9
SEQ:START 0
SEQ:END 20
ARM
START
```

In sequence 1, Pin 0 drives a 1, Pin 1 compares with a HI (1), and Pin 2 is inhibited. In sequence 3, Pin 0 drives a 1 and compares for a HI, and Pins 1 and 2 compare HI.

Retrieving Test Failure Data

In example 2, the test passed if the pins where correctly connected. After the test completes it can be determined that it passed by sending the STATE? command. To simulate a test failure we will modify sequence 3:

```
SEQ:VECTOR "LHS", 3
ARM
START
```

These commands cause the comparison on Pin 3 to fail on sequence 3. When the STATE? command is sent, the response confirms the failure. The sequence of the failure and the state of all pins on that sequence can be retrieved from the fail latch by sending the following command:

```
STATE?
```

The response will be:

```
STATE  STOPPED,20,21,FAIL,FAIL;
```

Then send the following command:

```
FAILDATA?
```

The response will be:

```
FAILDATA 3,"xxxx...x1111...111"; (32 x's and 32 1's)
```

This response contains the failed sequence number and the state of the pins which was sampled on that sequence.

Test Debug Support

The modified version of example 2 fails because of a known error in the test. One way to find this error and to correct it is to run the test on a known good UUT (in this case, the 3 wire connection). Failure data can be retrieved and the test data inspected to determine the reason for the failure. This method works for combinatorial logic but may be inadequate for sequential logic. With sequential logic the test data error may exist many sequences prior to the actual failed comparison.

When sequential failures exist, single-stepping the test can help find the error. The *SINGLESTEP* command controls the single step function. Single Step mode can only be entered when the VX4820 is stopped or paused. Example 3 shows the use of Single Step mode. **Be sure to remove the three-wire connection used in example 2.**

Example 3

```
INIT
NEW
SEQ:VECTOR "0000",0
SEQ:VECTOR "0001",1
SEQ:VECTOR "0010",2
SEQ:VECTOR "0011",3
SEQ:VECTOR "0100",4
SEQ:VECTOR "0101",5
SEQ:VECTOR "0110",6
SEQ:VECTOR "0111",7
SEQ:START 0
SEQ:END 7
INTCLKRATE 1000
SINGLESTEP ON
ARM
START
```

When the START command is entered, sequence 0 is executed, but the test does not progress to sequence 1. Pins 1 — 3 drive lows onto their respective outputs. To progress to sequence 2, send the CONTINUE command.

```
CONTINUE
```

Now Pin 1 should be HI, and Pins 2 and 3 low. The VX4820 executes a pause command on every sequence while in Single Step mode. Single Step mode can be exited while the VX4820 is paused.

Send the SINGLESTEP command again to exit Single Step mode.

```
SINGLESTEP OFF
CONTINUE
```

The test will execute at the programmed clock rate as soon as the CONTINUE command is received. Single Step mode can be switched on and off as many times as is required to debug the test.

If the portion of the test which is to be observed is not near the beginning of the test, single-stepping to the desired sequence can be tedious. To single step a portion of the test not at the beginning, the test should be executed with single step off. A *SEQ:PAUSE* command should be set on the first sequence to be observed. Once the test is paused, Single Step mode can be entered. Example 4 uses the PAUSE command to stop the test and allow Single Step mode to be entered.

Example 4

```
INIT
NEW
SEQ:VECTOR "0000",0
SEQ:VECTOR "0001",1
SEQ:VECTOR "0010",2
SEQ:VECTOR "0011",3
SEQ:VECTOR "0100",4
SEQ:VECTOR "0101",5
SEQ:VECTOR "0110",6
SEQ:VECTOR "0111",7
SEQ:START 0
SEQ:END 7
INTCLKRATE 1000
SEQ:PAUSE ON,3
ARM
START
```

When the START command is sent, the test executes at the programmed clock rate. When sequence 3 executes, the PAUSE command will cause sequence 3 to be held indefinitely. At this point, the SINGLESTEP ON command may be sent. Once in Single Step mode, CONTINUE commands will cause the next sequence to be executed. Single Step mode may be exited at any time, and full speed testing resumed, by sending SINGLESTEP OFF and then CONTINUE.

Learn Mode

Learn mode allows the user to capture the signals provided by the UUT. When in Learn mode, the pin functions are reduced to DRIVE and LEARN. The DRIVE HI (1) and DRIVE LOW (0) functions operate in the same manner as in test mode. DRIVE AND COMPARE HI (S) and DRIVE AND COMPARE LOW (R) are reduced to DRIVE HI and DRIVE LOW respectively. INHIBIT (X), COMPARE HI (H) and COMPARE LOW (L) are converted to the new LEARN compare values.

The LEARN function stores the data sampled on each sequence as COMPARE HI (H) and COMPARE LOW (L) symbols. The Learn mode test is executed in the same manner as a Test mode test. Test data is loaded into the VX4820 and the test is executed.

Test execution will modify the test data to reflect the data sampled on the pin as shown in the following chart.

Programmed Value	Data Sampled on Pin	Returned Value
1, S	Don't Care	1
0, R	Don't Care	0
X,H,L	1	H
X,H,L	0	L

In the Learn mode example following, Pins 0, 1, and 2 of POD 0 should be connected together to reproduce the results.

Example 5

```
INIT
NEW
SEQ:VECTOR "XX0",0
SEQ:VECTOR "XX1",1
SEQ:VECTOR "HH0",2
SEQ:VECTOR "LL1",3
SEQ:VECTOR "XXS",4
SEQ:VECTOR "XXR",5
SEQ:START 0
SEQ:END 6
INTCLKRATE 1000
MODE LEARN
ARM
START
```

After executing example 5, the captured data can be retrieved. After each query, the response string must be retrieved.

```
SEQ:VECTOR? 0
SEQ:VECTOR "xxx...xHHH...HLL0";
SEQ:VECTOR "xxx...xHHH...HHH1";
SEQ:VECTOR "xxx...xHHH...HLL0";
SEQ:VECTOR? 1
SEQ:VECTOR? 2
```

The returned data should reflect the learned values on Pins 1 and 2. The S and R functions are converted to 1 and 0 respectively.

After completion of Learn mode, the mode should be returned to Test by sending the MODE TEST command.

```
MODE TEST
```

Branching

Example 1 utilized branching to form an infinite loop, so that test execution could be observed with an oscilloscope. The BRANCH ALWAYS command caused test execution to jump back to sequence 0. Conditional branching can also be used to provide test synchronization to external events or to build conditional structures such as IF-THEN-ELSE and DO-WHILE. Example 6 will show a DO-WHILE structure. To replicate this example, Pins 0, 1, and 2 of POD 0 must be connected together.

Example 6

```
INIT
NEW
SEQ:VECTOR "0000HH1",0
SEQ:VECTOR "0001XX1",1
SEQ:VECTOR "0010XX1",2
SEQ:VECTOR "0011XX1",3
SEQ:VECTOR "0100XX1",4
SEQ:VECTOR "0101XX1",5
SEQ:VECTOR "0110XX1",6
SEQ:VECTOR "0111XX1",7
SEQ:VECTOR "1000XX1",8
SEQ:VECTOR "1001XX1",9
SEQ:VECTOR "1010XX1",10
SEQ:START 0
SEQ:END 10
SEQ:BRANCH PASS,0,8
INTCLKRATE 1000
ARM
START
```

In example 6, test execution will remain trapped in a DO-WHILE loop as long as the COMPARE done on sequence 0 returns PASS (expected data = sampled data). This test has no external stimulus to break the loop so we will modify the comparison to force a failure to occur.

```
STOP
SEQ:VECTOR "0000HL1",0
ARM
START
```

With the modification, the comparison on sequence 0 will fail. Test execution will fall through the BRANCH command on sequence 8 and execute sequences 9 and 10.

There are some restrictions placed on BRANCH commands. The BRANCH command has a domain which includes the seven prior sequences. In example 6, the branch domain contains sequences 1 — 8. The branch domain allows the VX4820 to prepare for the BRANCH.

All of the pin functions can be utilized within the branch domain. However, comparisons executed within the branch domain or SEQ:CLEAR commands executed within the branch domain will not affect the outcome of this conditional branch instruction (it will affect the Test State). Branch domains may not overlap. If programming a branch would cause overlapping domains, an error event is generated. The NEW command may be sent to clear any preexisting branches before new test data is loaded. Branch domains must be executed linearly. Jumping into a branch domain from another part of the test will cause unpredictable results.

Using VXI STST Protocol

The VX4820 can receive the VXI Start/Stop protocol provided by the Slot 0 module. This allows it to start in tandem with other modules within the mainframe (i.e. Arbitrary Waveform Generator, Digitizer, etc.). All modules that are started from STST protocol commence their operation on a particular edge of the CLK10 signal. This edge is selected by the Slot 0 module and transmitted to STST receiving modules.

In the example below, the VX4820 is programmed to execute a test when STST protocol is received.

Example 7

```
INIT
NEW
SEQ:VECTOR "XXX",0
SEQ:VECTOR "00L",1
SEQ:VECTOR "01H",2
SEQ:VECTOR "10H",3
SEQ:VECTOR "11H",4
SEQ:VECTOR "XXX",5
SEQ:START 0
SEQ:END 20
GRPMODE STST
CONNECT:STST TTLTRG 0   must be same as selected on slot 0
ARM
```

Note that the START command was not sent. The START function is entirely controlled by the STST protocol. Executing sequence 20 will automatically generate an internal STOP command. If the test is written as an infinite loop, and STST is used to stop the test, the STOP command must be sent before test results can be retrieved.

Multi-Module Systems

Multiple VX4820 modules may be grouped together to form a digital test system of up to 768 pins. The grouped modules operate in tandem and can logically be considered to be a single entity. All of the features that are available on a single module are also available in a multi-module group. The Test State is shared within the group. A failed comparison on any pin, on any module in the group, will cause the test state for the group to transition to FAILED.

VX4820 groups must share information to ensure tandem operation. Group modules must be installed in adjacent mainframe slots. Each module should be programmed to use the same clock. The commands SEQ:BRANCH and SEQ:PAUSE should be programmed identically on all modules in the group. If Single Step mode is used, the SINGLESTEP command must be sent to all modules in the group. These rules ensure the modules in the group are always executing the same test sequence (the test remains coherent).

The modules should be programmed for their participation in the group. One of the modules must be assigned to be the *group commander*. The group commander controls the starting and stopping of the modules. It also sends signals to the system controller when a pause or test completion occurs. The group commander can be any of the modules, although, usually the left-most module is selected. The START, STOP, PAUSE, and CONTINUE commands are sent to the group commander only. Other modules in the group must be sent the command GRPMODE GROUP. After sending the ARM command, the STATE? query is sent to ensure that the module is armed and ready to start test execution.

In the example following, two VX4820 modules, module1 and module2, will be programmed to execute a short test. The modules must be placed in adjacent slots within the VXI mainframe. Module1 has been designated the COMMANDER and module2 a member of the group.

Example 8

MODULE1

```
INIT
NEW
SEQ:VECTOR "XXX",0
SEQ:VECTOR "00L",1
SEQ:VECTOR "10L",2
SEQ:VECTOR "01L",3
SEQ:VECTOR "11H",4
SEQ:VECTOR "XXX",5
SEQ:START 0
SEQ:END 20
INTCLKRATE 150
```

```
GRPMODE COMMANDER
CONNECT:STST TTLTRG0
```

```
ARM
STATE?
```

```
START
```

MODULE2

```
INIT
NEW
SEQ:VECTOR "XXX",0
SEQ:VECTOR "HLH",1
SEQ:VECTOR "HHH",2
SEQ:VECTOR "LHX",3
SEQ:VECTOR "LLL",4
SEQ:VECTOR "XXX",5
SEQ:START 0
SEQ:END 20
INTCLKRATE 150
```

```
GRPMODE GROUP
CONNECT:STST TTLTRG0
```

```
ARM
STATE?
```

Multi-Module Systems Using an External Clock

A multi-module system may use internal or external clocks. When internal clocks are used, each module should be programmed to the same clock rate. If an external clock is supplied, it must be received by the group commander and distributed to all group members.

External clocks must be input to a multi-module system on the group commander's CLOCK IN BNC. The clock is then distributed to the remaining modules via the VXI ECL trigger lines. Once the clock input and distribution is selected, the clock path must be calibrated to eliminate the distribution skew (see multi-module calibration in this section). Example 9 shows the programming of clocks for a two module system using an external clock. MODULE1 is the group commander and has the external clock connected to its CLOCK IN BNC. MODULE2 is the group member who receives the external clock from the group commander on ECLTRG0.

Example 9

MODULE1	*MODULE2*
CONNECT:CLOCK CLKBNCECL0	
	CONNECT:CLOCK ECLTRG0

The CLKBNCECL0 parameter selects the BNC input as the clock source for module1. It also instructs module1 to export the clock on ECLTRG0. The parameter sent to module2, ECLTRG0, programs the module to receive its clock on VXI ECL trigger line 0.

Diagnostics and Calibration

The diagnostic routines provide information regarding VX4820 functionality. These routines isolate to the functional level only and thus do not provide direct information about failed components. The objective of these diagnostics is to decrease the time to isolate to a functional module failure.

Calibration is required when the configuration of a VX4820 group is changed (a module is added, moved, or replaced). Single module systems require no calibration. For further information on group calibration, or which module accepts the external clock on its CLOCK IN BNC, refer to *Section 3, Group Calibration.*

Section 4
Rear Panel Interface

Introduction

The VX4820 rear panel interface consists of VXIbus connectors P1 and P2. Those pins used by the VX4820 on P1 and the inner row (Row B) on P2 are configured as defined in the *VMEbus Specification, IEEE Standard 1014.* Those pins used by the VX4820 on the outer rows of P2 (Rows A and C) are configured as defined in the *VXIbus System Specification, Revision 1.3, dated July 14, 1988.* The pin assignments used by the VX4820 on these two connectors are listed in Tables 4-1 and 4-2. For detailed information regarding the signals on the pins, refer to the *VMEbus Standard* and the *VXIbus Specification.*

The VX4820 utilizes four VXI Local Bus lines on the P2 connector: LB00A, LB00C, LB01A, and LB01C. The signals on these lines are of the ECL class. The module is keyed for ECL class to prevent most Local Bus conflicts. Jumpers are also provided on the module to disconnect the Local Bus signals from the module logic.

P1 Pinout (Slots 1 — 12)

(nc = no connection)

Pin Number	Row A Signal Mnemonic	Row B Signal Mnemonic	Row C Signal Mnemonic
1	D00	nc	D08
2	D01	nc	D09
3	D02	ACFAIL*	D10
4	D03	BG0IN*	D11
5	D04	BG0OUT*	D12
6	D05	BG1IN*	D13
7	D06	BG1OUT*	D14
8	D07	BG2IN*	D15
9	GND	BG2OUT*	GND
10	SYSCLK	BG3IN*	SYSFAIL*
11	GND	BG3OUT*	BERR*
12	DS1*	BR0*	SYSRESET*
13	DS0*	BR1*	LWORD*
14	WRITE*	BR2*	AM5
15	GND	BR3*	A23
16	DTACK*	AM0	A22
17	GND	AM1	A21
18	AS*	AM2	A20
19	GND	AM3	A19
20	IACK*	GND	A18
21	IACKIN*	SERCLK	A17
22	IACKOUT*	SERDATA	A16
23	AM4	GND	A15
24	A07	IRQ7*	A14
25	A06	IRQ6*	A13
26	A05	IRQ5*	A12
27	A04	IRQ4*	A11
28	A03	IRQ3*	A10
29	A02	IRQ2*	A09
30	A01	IRQ1*	A08
31	-12V	+5V STDBY	+12V
32	+5V	+5V	+5V

P2 Pinout (Slots 1 — 12)

(nc = no connection)
(LB = Local Bus)

Pin Number	Row A Signal Mnemonic	Row B Signal Mnemonic	Row C Signal Mnemonic
1	ECLTRG0	+5V	CLK10+
2	-2V	GND	CLK10-
3	ECLTRG1	Reserved	GND
4	GND	A24	-5.2V
5	FAIL1 IN/LB	A25	FAIL1 OUT/LB
6	FAIL2 OUT/LB	A26	FAIL2 IN/LB
7	-5.2V	A27	GND
8	LB	A28	LB
9	LB	A29	LB
10	GND	A30	GND
11	LB	A31	LB
12	LB	GND	LB
13	-5.2V	+5V	-2V
14	LB	D16	LB
15	LB	D17	LB
16	GND	D18	GND
17	LB	D19	LB
18	LB	D20	LB
19	-5.2V	D21	-5.2V
20	LB	D22	LB
21	LB	D23	LB
22	GND	GND	GND
23	TTLTRG0*	D24	TTLTRG1*
24	TTLTRG2*	D25	TTLTRG3*
25	+5V	D26	GND
26	TTLTRG4*	D27	TTLTRG5*
27	TTLTRG6*	D28	TTLTRG7*
28	GND	D29	GND
29	RSV2	D30	RSV3
30	MODID	D31	GND
31	GND	GND	+24
32	SUMBUS	+5V	-24

Section 5
Command Notation and Formats

Introduction

This section contains introductory information about the programming commands for the VX4820. The actual command dictionaries are located in *Section 6, Instrument Control Command Dictionary*; and *Section 7, System Command Dictionary*. Commands follow the formats and conventions established by the *ANSI/IEEE Std 488.2 — 1987, IEEE Standard Codes, Formats, Protocols, and Common Commands.*

Command Notation

Command format descriptions and descriptive tables use the following standard notation:

< >	Delimits a required, defined element to be provided by the user.
[ ]	Delimits an optional element.
[ ]...	Delimits an optional element that can be omitted, or repeated one or more times .
{ }	Delimits a group of elements from which one must be selected.
|	Delimits EXCLUSIVE OR selectable elements.
::=	Means "is defined as".
: (colon)	Compound Command Program Header Separator used to separate "device names" from Command/Query Program Headers, and to separate Compound Command Headers.
; (semicolon)	Program Message Unit Separator used to separate functional parts of compound commands.
, (comma)	Program Data Separator used to separate compound Program Data elements.

Program Message

A VXIbus Program Message is a sequence of zero or more Program Message Units separated by a Program Message Unit separator, the semicolon (;). A Program Message Unit represents a single command message, programming data, or a query message intended for a device such as the VX4820. A Command Message represents a single command or programming data sent from the controller through an interface device to the VX4820. A Query Message represents a single query sent over this same path to the VX4820.

Each Program Message Unit contains a series of syntactic elements, which may include a Device Name, a Command/Query Program Header (the command) and the associated Program Data elements (the command parameters), or multiple Program Message Units separated by semicolons (;). For a more detailed explanation about Program Messages, refer to the *ANSI/IEEE Std 488.2-1987*.

After the automatic execution of its initial power-up and configuration sequence, the VX4820 relies on Program Messages received over the VXIbus to control its ongoing operations. These messages may originate from an external controller, then be transferred through a serial (RS232) or IEEE 488 (GPIB) interface to a VXIbus Interface Device which finally transfers the messages to the VX4820 over the VXIbus. Messages also may originate from an embedded controller (installed in the VXIbus Mainframe), which would transfer all messages to the VX4820 directly over the VXIbus. Each message is parsed by the Interface Device to determine its destination, or action to be taken.

Messages sent to the VX4820 through its commander are of the form:

```
:<device name>:<command>
```

such as:

```
:VDEV0:SEQ:ADDRESS 0
```

where:

:VDEV0 is the *< device name>* ; the rest of the message is the command and its parameters.

ASCII Code

The 7-bit ASCII code is used for the majority of syntactic elements and semantic definitions. However, 8-bit ASCII code is allowed in special block messages, such as Arbitrary Block Program Data.

Syntactic Delimiters

Syntactic elements in a Program Message Unit are delimited with colons, white space, commas, or semicolons.

Colons (:) typically delimit "<device names>" and device-dependent Command/Query Program Headers, such as in the following message:

```
:VDEV0:SEQ:END 12
```

where *:VDEV0* is the *<device name>*; *:SEQ:END is the Command Program Header.*

White Space (defined below under the White Space topic) typically delimits device-dependent Command/Query Program Headers from Program Data, such as reserved word command names in a command set from device-dependent command arguments. In the example above, *:SEQ:END* is the Command Program Header and *12* is the Program Data.

Commas (,) typically delimit multiple Program Data elements (arguments).

Semicolons (;) typically delimit multiple Program Message Units and compound Command Program Header tree structures in commands suited for compound program headers.

White Space

White space is essential to a program, but is not a part of the program. Typically, it is used to separate certain syntactic elements in a command. In the VX4820, white space is defined the same as in the *ANSI/IEEE Std 488.2-1987*, which is as a single ASCII-encoded byte in the range ASCII 0-32 (decimal), except ASCII 10, which is the Line Feed (LF) or Newline (NL) character.

Special Characters

The VX4820 firmware defines Special Characters as the Line Feed (LF) or Newline (NL) character (ASCII 10), and all characters in the range ASCII 127 — 255. These characters may be used in Arbitrary Block Program Data without consequence; however, when used in any other context, the outcome is unpredictable.

Program Data

The types of Program Data recognized by the VX4820 are:

- Decimal Numeric
- Non-Decimal Numeric
- String

Numeric Program Data Value Range

The range of acceptable Program Data numeric values is limited to that allowed by a 32-bit signed binary number. If out-of-range values are entered, an error message is generated, and execution of the command, or the multiple message unit containing the error, is terminated.

Decimal Numeric Program Data

Decimal Numeric Program Data is defined the same as in *ANSI/IEEE Std 488.2-1987*, which is as three numeric representations NR1, NR2, and NR3. When any one of the three is acceptable, the flexible representation NRf is specified. The general description of NRf data is shown below.

Type	Format	Examples
NR1	Implicit-point (Integer)	1, +3, -2, +10, -20
NR2	Explicit-point-unscaled (Fixed Point)	1.2, +23.5, -0.15
NR3	Explicit-point-scaled (Floating Point)	1E+2, +3.36E-2, -1.02E+3

If NR1 is specified as the input value format, NRf is acceptable. Query response is NR1 if the input is NR1, otherwise it is NR2.

Non-Decimal Numeric Program Data

Non-Decimal Numeric Program Data is defined the same as in *ANSI/IEEE Std 488.2-1987*, which is as the three alternate radix representations: hexadecimal, octal, and binary. These are represented as follows:

Hexadecimal

#Hd..d

where:	H	is H or h.
	d	is 0..9, A..F, or a..f

Octal

#Qd..d

where:	Q	is Q or q.
	d	is 0..7.

Binary

#Bd..d

where:	B	is B or b.
	d	is 0 or 1

String Program Data

String Program Data, sometimes referred to as "string literals", "literals", or just "strings", is defined as:

```
"[<non-double quote character>]..."
```

such as:

"This is a string constant."

or

```
"0..127"
```

To include a double quote character in the string, insert an additional double quote character ahead of the double quote character in the string. For example, the string:

```
SN"010000"
```

would be sent as:

```
"SN""010000"""
```

String constants may be of any length, up to the memory limits of the VX4820.

Command Formats

All commands follow the Functional Element Syntax formats for Command Message Units and Query Message Units as defined by *ANSI/IEEE Std 488.2-1987*. These formats are defined to have Simple Command Program Headers, Simple Query Program Headers, Compound Command Program Headers, Compound Query Program Headers, Common Command Program Headers, and Common Query Program Headers, which are further described with examples in the paragraphs that follow.

Simple Command Program Header

A command with a Simple Command Program Header is one containing only one Program Mnemonic, or only one Program Mnemonic plus Program Data. Its message format is:

```
<Program Mnemonic> [<Program Data>]
```

such as:

```
START
```

or:

```
GRPMODE SINGLE
```

Simple Query Program Header

A command with a Simple Query Program Header is one that contains only one Program Mnemonic followed by a question mark (?). Its message format is:

```
<1st Program Mnemonic>? [<Program Data>]
```

such as:

```
INTCLKRATE?
```

or:

```
PODTYPE? 1
```

Compound Command Program Header

A command with a Compound Command Program Header is one that contains multiple Program Mnemonics plus Program Data. Its message format is:

```
<1st Program Mnemonic>[:<2nd Program Mnemonic>]
```

or:

```
<1st Program Mnemonic>[:<2nd Program Mnemonic>] <Pro-
gram Data>
```

such as:

```
CONNECT:STST TTLTRG0
```

or:

```
SEQ:VECTOR "00010X"
```

The Compound Command Program Header is in the same format as embedded addresses. An example is:

```
:VDEV0:SEQ:VECTOR "00010X"
```

where *VDEV0* is the device name assigned to the VX4820 by the VX4530 during power-up configuration. When this message is received by the VX4530, it strips off the destination address (*:VDEV0*) and sends the remaining message to the VX4820.

Compound Query Program Header

A command with a Compound Query Program Header is one containing multiple Program Mnemonics followed by a question mark (?). Its message format is:

```
<1st Program Mnemonic>:<2nd Program Mnemonic>? [<Pro-
gram Data>]
```

such as:

```
SEQ:ADDRESS?
```

or:

```
CONNECT:STST?
```

Common Command Program Header

A command with a Common Command Program Header is one that precedes its Program Mnemonic with an asterisk (*). Its message format is:

```
<1st Program Mnemonic> [<Program Data>]
```

such as

```
*RST
```

Common Query Program Header

A command with a Common Query Program Header is one that precedes its Program Mnemonic with an asterisk (*), and follows it with a question mark (?). Its message format is:

```
<1st Program Mnemonic>? [<Program Data>]
```

such as:

```
*IDN?
```

Multiple-Command Format With Same 1st Program Mnemonic

The multiple-command message format for commands having the same 1st Program Mnemonic is:

```
<1st Program Mnemonic>:<2nd Program Mnemonic> [<Pro-
gram Data>] [;<2nd Program Mnemonic> [<Program
Data>]]...
```

such as:

```
SEQ:SETPAUSE OFF,10;SETPAUSE ON,20
```

Instrument Response Queuing

Queries sent to the VX4820 will generate a response that must be read prior to sending further commands or queries to that instrument. The proper procedure is to send a query followed by a read of the response to that query. If a command or query is sent before reading the previous response, the instrument will not accept that command and the controller will time out. In this case, the previously ignored response should then be read. Commands that do not produce a response can be sent without performing a read.

NOTE

The size of the buffer needed for the responses from the queries depends on the query sent. The average response will need approximately 100 bytes. However, some queries will generate a response that can require 1.5K — 390K bytes. Refer to the individual query descriptions in Section 6 for further information.

Case Sensitivity

The command set parser is insensitive to case; therefore, command mnemonics or Program Data may be entered in either uppercase or lowercase characters. Responses are returned in all uppercase, except for string data, which is returned in the same case as entered. For ease of recognition, command mnemonics, predefined parameter words (such as : ON and OFF), and decimal number formats (such as: NR1, NRf, etc.) are capitalized in the command set. Where the user must provide a parameter, the parameter name is set off with left and right angle symbols (<>), and have lower-case parameter names such as : <length>. String Program Data is case sensitive.

Command Types

There are two command types described in the command dictionary sections: VX4820 Commands and System Commands.

Command Dictionary Layout

The Command Dictionary for the VX4820 firmware has a layout similar to many command dictionaries, but with some differences to accommodate the device specific capabilities of the VX4820. The following describes the elements included in each Command Dictionary entry.

- Each command is listed alphabetically at the top of a new page.
- *Purpose* — Provides a brief description of what the command does.
- *Syntax* — Shows the command format as it must appear at the VXIbus input to the VX4820. The command name, and some common Program Data elements (e.g., ON, OFF, NONE, etc.), are shown in all capital letters for readability only, as the firmware treats uppercase and lowercase letters the same. Mandatory user-defined entries are listed in lowercase letters and are enclosed in angle brackets (<>). Where the command descriptions include both command and query forms, there may be a separate Command and Query header titles.
- *Parameters* — Describes the parameters values, including their allowable ranges, and any required delimiters, such as commas.
- *Description* — Provides additional details about the command.
- *Examples* — Provides one or more examples of actual command use including input parameters and returned values.
- *Related Commands* — Lists other commands that may perform either parallel functions on different data, or different functions on the same data.

Section 6
Instrument Control Command Dictionary

Introduction

Instrument Control commands control the primary VX4820 functions. They initialize and change its function settings, query for its status and control test execution.

Alphabetical Command Listing

ARM	Prepares the VX4820 to start a test
BLRANGE	Select FDCLOAD? range
CLKCAL	Adjust clock calibration value
CONNECT	Route clock and trigger signals
CONTINUE	Advance test when paused
DESE	Controls Device Event Status reporting
DEVINFO?	Returns module configuration information
EVENT?	Returns Event code
EVMSG?	Returns Event code and message
FAILDATA?	Returns test failure data
FAILPIN?	Returns fail sequences and failure pins
GRPMODE	Multi-module system control
HEADER	Enables command response headers
HELP?	Returns a list of the VX4820 commands
IDPOD	Controls the selected POD ID LED
INHIBIT	Force I/O channels to high impedance state immediately
INIT	Returns all programmable settings to power-up default
INTCLKRATE	Sets internal clock rate

MODE	Selects between TEST and LEARN modes
NEW	Clears pattern memory to the default state
NEWPASS	Changes the password
NVRECALL	Restores calibration and user-defined nonvolatile data
NVSAVE	Saves current calibration and user-defined data
PASSWORD	Enables restricted commands
PAUSE	Pause a running test
PINTEST?	Verifies pin driver/receiver function
PODTYPE?	Returns POD ID code
READPIN?	Reads current pod data pattern at the pin
REPORT	Enables error, event, and warning report functions
SEQ	Sets sequence, pattern data, and commands
SINGLESTEP	Enables Single Step mode
START	Initiates test execution
STATE?	Returns status of test
STERR?	Returns error codes from nonvolatile memory
STOP	Terminates a test
TSTPAT	Loads a test pattern into memory
USER	Allows user to store information in nonvolatile memory
WRITEPIN	Drives the pod output
WSLOAD	Initiate binary load over Word Serial Protocol

Functional Command Listing

Setup Test Data	Operational Setup	Runtime	Misc
BLRANGE(?)	ARM	CONTINUE	DESE
NEW	CLKCAL(?)	STATE?	DEVINFO?
SEQ:ADDRESS(?)	CONNECT:CLOCK(?)	STOP	EVENT?
SEQ:BRANCH(?)	CONNECT:TSTSTATEOUT(?)	PAUSE	EVMSG?
SEQ:CLEAR(?)	CONNECT:TRIGGEROUT(?)	INHIBIT	FAILDATA?
SEQ:DESC?	CONNECT:STST(?)	SINGLESTEP(?)	FAILPIN?
SEQ:END(?)	GRPMODE(?)		HEADER(?)
SEQ:PAUSE(?)	INTCLKRATE(?)		HELP?
SEQ:SEQINC	MODE(?)		IDPOD
SEQ:START(?)	START		INIT
SEQ:TRIG(?)			NEWPASS
SEQ:VECTOR(?)			NVRECALL
SEQ:VECTORINC			NVSAVE
WSLOAD(?)			PASSWORD
			PINTEST?
			PODTYPE?
			READPIN?
			REPORT(?)
			STERR?
			TSTPAT
			USER(?)
			WRITEPIN

ARM **ARM**

SETUP

Purpose: *ARM* prepares the module to start

Command Syntax: `ARM`

Command Parameters: None

Description: ARM prepares the module to start a test. SEQ:START and SEQ:END must have been set prior to sending the ARM command. The START command or STST protocol initiates test execution. Multi-module systems and modules using an external clock source should send a STATE? query to ensure that the module is armed and ready to start test execution.

Example: In this example, a test will be executed from sequence 0 to sequence 100.

```
SEQ:START 0
SEQ:END 100
ARM
START
```

Related Commands START, STOP

BLRANGE, BLRANGE? **BLRANGE, BLRANGE?**

SETUP

Purpose: *BLRANGE* specifies the range of test steps transferred during binary upload. *BLRANGE?* returns the current selected range.

Command Syntax:

```
BLRANGE <first>,<last>
```

Command Parameters:

`first`

A numeric value that specifies the first test step to be transferred.

`last`

A numeric value that specifies the last test step to be transferred.

Query Syntax:

```
BLRANGE?
```

Query Parameters None

Query Response

```
BLRANGE <start>,<end>;
```

`start`: numeric value 0 .. 16350 (default: 0)

`end`: numeric value 0 .. 16350 (default: 16350)

Description: BLRANGE selects the range of test vectors uploaded during FDCLOAD? or WSLOAD? execution. *BLRANGE?* returns the selected beginning and ending test sequence.

Example: In this example, test sequence 100 through 999 will be specified as the upload range for subsequent FDCLOAD? commands.

```
BLRANGE 100,999
```

In this example, the FDCLOAD? range is the default values.

```
BLRANGE?
BLRANGE 0,16350;
```

Related Commands FDCLOAD?, WSLOAD?

CLKCAL, CLKCAL? **CLKCAL, CLKCAL?**

SETUP

Purpose: *CLKCAL* sets the calibration values for the internal and external clock. *CLKCAL?* returns the current internal and external clock calibration values.

Command Syntax:

```
CLKCAL <internal clock delay>, <external clock delay>
```

Command Parameters:

`internal clock delay`

A numeric value in the range 0 — 30 that specifies the calibration delay for the internal clock.

`external clock delay`

A numeric value in the range 0 — 30 that specifies the calibration delay for the external clock.

Query Syntax:

```
CLKCAL?
```

Query Parameters: None

Query Response

```
CLKCAL <internal clock delay>,<external clock delay>;
```

`internal clock delay`: numeric value 0 .. 30
`external clock delay`: numeric value 0 .. 30

Description: CLKCAL is used during system calibration. In a system with a group of VX4820 modules, the clock delay of each VX4820 can be adjusted with this command such that each module receives the clock edge at the same time (see *Section 2, Group Calibration Procedure*). These delay constants can be saved to the nonvolatile memory by the NVSAVE command and recalled back by the NVRECALL command. On power-up, NVRECALL is executed automatically.

CLKCAL? returns the current internal and external clock delay setting. CLKCAL requires a valid password to disable the access restriction. The password is entered through the PASSWORD command.

Example: In this example, the internal clock delay will be set to 12 and the external clock delay is set to 30.

```
PASSWORD "DT"
CLKCAL 12,30
```

In this example, the current internal clock delay is 10 and the external clock delay is 20.

```
CLKCAL?
CLKCAL 10,20;
```

Related Commands: NVSAVE, NVRECALL, PASSWORD

CONNECT:CLOCK, CLOCK? **CONNECT:CLOCK, CLOCK?**

SETUP

Purpose: *CONNECT:CLOCK* selects the source of the system clock.
CONNECT:CLOCK? returns the current source of the system clock.

Command Syntax:

```
CONNECT:CLOCK {INTCLK | CLKBNC | CLKBNCECL0 |
               CLKBNCECL1 | ECLTRG0 | ECLTRG1 |
               ECLTRG0N | ECLTRG1N | EXTPOD0 |
               EXTPOD0ECL0 | EXTPOD0ECL1 | EXTPOD1 |
               EXTPOD1ECL0 | EXTPOD1ECL1}
```

Command Parameters:

INTCLK

The module will be clocked by the internally generated clock.

CLKBNC

The module will be clocked by the rising edge of the signal on the front panel CLK IN BNC.

CLKBNCECL0

The module will be clocked by the rising edge of the signal on the front panel CLK IN BNC. This clock will also be driven onto the VXI ECLTRG line 0.

CLKBNCECL1

The module will be clocked by the rising edge of the signal on the front panel CLK IN BNC. This clock will also be driven onto the VXI ECLTRG line 1.

ECLTRG0

The module will be clocked by the rising edge of the signal on VXI ECLTRG line 0.

ECLTRG1

The module will be clocked by the rising edge of the signal on VXI ECLTRG line 1.

ECLTRG0N

The module will be clocked by the falling edge of the signal on VXI ECLTRG line 0.

ECLTRG1N

The module will be clocked by the falling edge of the signal on VXI ECLTRG line 1.

EXTPOD0

The module will be clocked by the rising edge of the signal on the POD 0 CLK IN pin.

EXTPOD0ECL0

The module will be clocked by the rising edge of the signal on the POD 0 CLK IN pin. This clock will also be driven onto the VXI ECLTRG line 0.

EXTPOD0ECL1

The module will be clocked by the rising edge of the signal on the POD 0 CLK IN pin. This clock will also be driven onto the VXI ECLTRG line 1.

EXTPOD1

The module will be clocked by the rising edge of the signal on the POD 1 CLK IN pin.

EXTPOD1ECL0

The module will be clocked by the rising edge of the signal on the POD 1 CLK IN pin. This clock will also be driven onto the VXI ECLTRG line 0.

EXTPOD1ECL1

The module will be clocked by the rising edge of the signal on the POD 1 CLK IN pin. This clock will also be driven onto the VXI ECLTRG line 0.

Query Syntax

```
CONNECT:CLOCK?
```

Query Parameters

None

Query Response

```
CONNECT:CLOCK {INTCLK | CLKBNC | CLKBNCECL0 |
               CLKBNCECL1 | ECLTRG0 | ECLTRG1 |
               ECLTRG0N | ECLTRG1N | EXTPOD0 |
               EXTPOD0ECL0 | EXTPOD0ECL1 | EXTPOD1 |
               EXTPOD1ECL0 | EXTPOD1ECL1};
```

Description: CONNECT:CLOCK selects the source of the module clock. The module clock signal can be internally generated or can be sourced from the pods, a front panel BNC, or ECL trigger lines 0 or 1. External clocks may be optionally driven onto the VXI backplane for distribution to other VX4820 modules or VXI instruments.

NOTE

When using a slow external clock, use the STATE? query to check that the state has changed for the START, STOP, PAUSE, and CONTINUE commands. Several clock periods are required for the state changes to occur.

CONNECT:CLOCK? returns the current module clock source.

Examples: In this example, the front panel CLK IN BNC is the module clock source. This clock is also driven onto the VXI backplane on ECLTRG line 0.

```
CONNECT:CLOCK CLKBNCECL0
```

In this example, the internally generated clock is selected and the clock rate is set to 150 ns.

```
CONNECT:CLOCK INTCLK
INTCLKRATE 150
```

In this example, the selected module clock is POD 0 CLK IN.

```
CONNECT:CLOCK?
CONNECT:CLOCK EXTPOD0;
```

Related Commands: INTCLKRATE

CONNECT:STST, STST? **CONNECT:STST, STST?**

SETUP

Purpose: *CONNECT:STST* assigns a TTL trigger line to START/STOP protocol. *CONNECT:STST?* returns the current assigned TTL trigger line for the START/STOP protocol.

Command Syntax:

```
CONNECT:STST {TTLTRG0..7 | NONE}
```

Command Parameters:

`TTLTRG0..7`

Assign TTL trigger line 0, 1, 2, 3, 4, 5, 6, or 7 for execution control.

`NONE`

Remove TTL trigger assignments for execution control.

Query Syntax:

```
CONNECT:STST?
```

Query Parameters: None

Query Response

```
CONNECT:STST {TTLTRG0 ... TTLTRG7 | NONE};
```

Description: CONNECT:STST assigns one of the VXI TTL trigger lines for test execution control. The VX4820 utilizes START/STOP protocol to control test execution of multi-module systems. Modules in this configuration must have their GRPMODE set to either GROUP, COMMANDER, or STST. If GRPMODE SINGLE is specified, the CONNECT:STST command need not be sent.

When multiple VX4820 modules are configured as a group, they execute their test as a single super instrument. One of the modules is designated group commander and the rest of the modules are designated group members. This is accomplished utilizing the GRPMODE command. To control test execution, the group commander drives a START/STOP signal onto a selected TTLTRG line, and group members receive the this signal from the selected TTLTRG line. The CONNECT:STST command facilitates connection to a particular TTLTRG line.

When the VX4820 execution is to be controlled by the VXI STST or ESTST protocol, the START/STOP signal is sourced by the Slot 0 module. This allows different instrument module types, within a single chassis, to operate synchronously. The VX4820 receives VXI STST and ESTST protocol when the GRPMODE is set to STST. The selected TTLTRG line should be the same as that driven by the Slot 0 module. The VX4820 must receive the ARM command before STST protocol can control its test execution. VXI STST and ESTST protocol require that participating modules operate synchronously to VXI CLK10. As a result, the GRPMODE STST may only be used with the internal clock selected.

Example: In this example, TTL trigger line 3 is assigned for START/STOP protocol.

```
GRPMODE  STST
CONNECT:STST TTLTRG3
```

In this example, TTL trigger line 2 is currently assigned to START/STOP protocol.

```
CONNECT:STST?
CONNECT:STST TTLTRG2;
```

Related Commands: GRPMODE, ARM, STOP

CONNECT:TRIGGEROUT, TRIGGEROUT? **CONNECT:TRIGGEROUT, TRIGGEROUT?**

SETUP

Purpose: *CONNECT:TRIGGEROUT* connects trigger outputs to a particular TTL or ECL trigger line.
CONNECT:TRIGGEROUT? returns the TTL or ECL trigger line assigned to the trigger function.

Command Syntax:

```
CONNECT:TRIGGEROUT {TTLTRG0..7 | ECLTRG0..1 | NONE}
```

Command Parameters:

`TTLTRG0..7`

Output triggers are driven onto TTL trigger line 0 — 7

`ECLTRG0..1`

Output triggers are driven onto ECL trigger line 0 or 1

`NONE`

Output triggers are disconnected from TTL and ECL triggers

Query Syntax:

```
CONNECT:TRIGGEROUT?
```

Query Parameters: None

Query Response:

```
CONNECT:TRIGGEROUT {TTLTRG0 .. TTLTRG7 | ECLTRG0 |
                    ECLTRG1 | NONE};
```

Description: CONNECT:TRIGGEROUT selects which TTL or ECL trigger line will be driven by the trigger output function during test execution. Triggers are generated on test sequences which have been programmed with the SEQ:TRIGGER ON command. When that test sequence is executed, the trigger line will be asserted for the duration of the test step. If the trigger line has already been assigned, an error message will be generated.

CONNECT:TRIGGEROUT? returns the trigger line currently assigned to the sequence trigger function.

Example: Connects the trigger function to TTLTRG5:

```
CONNECT:TRIGGEROUT TTLTRG5
```

Related Commands: SEQ:TRIGGEROUT

CONNECT:TSTSTATEOUT,TSTSTATEOUT? CONNECT:TSTSTATEOUT,TSTSTATEOUT?

SETUP

Purpose: *CONNECT:TSTSTATEOUT* connects the Test State function to an ECL trigger line.
CONNECT:TSTSTATEOUT? returns the ECL trigger line connected to the Test State function.

Command Syntax:

```
CONNECT:TSTSTATEOUT {ECLTRG0 | ECLTRG1 | NONE}
```

Command Parameters:

```
ECLTRG0, ECLTRG1
```

Test State is driven onto ECL trigger line 0 or 1

```
NONE
```

Test State is disconnected from ECL triggers

Query Syntax:

```
CONNECT:TSTSTATEOUT?
```

Query Parameters: None

Query Response:

```
CONNECT:TSTSTATE {ECLTRG0 | ECLTRG1 | NONE};
```

Description: CONNECT:TSTSTATEOUT assigns an ECL trigger line to the Test State function. The Test State function reflects the PASS/ FAIL state of the executing test. If the test state is failed, the selected ECL trigger line will be asserted (HIGH). If the ECL trigger line has already been assigned, an error message will be generated.

CONNECT:TSTSTATEOUT? returns the ECL trigger line currently assigned to the test state function.

Example: In this example, ECL trigger line 0 is assigned to the Test State function.

```
CONNECT:TSTSTATEOUT ECLTRG0
```

In this example, ECL trigger line 1 is currently assigned to the Test State function.

```
CONNECT:TSTSTATEOUT?
CONNECT:TSTSTATE ECLTRG1;
```

Related Commands: STATE?, FAILDATA?

CONTINUE **CONTINUE**

RUNTIME

Purpose: *CONTINUE* advances test to the next sequence when in PAUSED or SINGLE STEP state.

Command Syntax: `CONTINUE`

Command Parameters: None

Description: CONTINUE advances the test to the next sequence when in PAUSED or SINGLESTEP state. When the VX4820 is used in a group, this command must be sent to the group commander only. If the new test sequence contains a SEQ:PAUSE command or SINGLESTEP is selected, the test will pause. Otherwise, it will continue executing the test at the programmed clock rate.

When the VX4820 test execution is controlled by VXI STST or ESTST protocol, the CONTINUE command can not be used to advance a paused test. To advance a test when STST protocol is used, the STST Protocol must be set to stop and then start.

Related Commands: SEQ:PAUSE, START, SINGLESTEP, PAUSE

DESE, DESE? **DESE, DESE?**

Purpose: *DESE* controls the setting and reading of the Device Event Status Enable Register.
DESE? reads the Device Event Status Enable Register

Command Syntax:
```
DESE <register data>
```

Command Parameters:
`register data`

An integer in the range of 0 — 255. When expressed in binary, this integer represents the bit values of the Device Event Status Enable Register

Query Syntax:
```
DESE?
```

Query Parameters: None

Query Response:
```
DESE <register data>;
```

Description: The Device Event Status Enable Register is an 8-bit register with bit definitions corresponding to the Standard Event Status Register. Bits in the Device Event Status Enable Register allow reporting through the Standard Event Status Register. Bits not set in the Device Event Status Enable Register disable reporting of that event (see *ANSI/IEEE Std 488.2-1987*). See the Register Bit Definition figure at the end of Section 7, System Command Dictionary for details.

Related Commands: None

DEVINFO? **DEVINFO?**

SETUP
RUNTIME

Purpose: *DEVINFO?* returns the configuration information for this VX4820.

Query Syntax:
```
DEVINFO?
```

Query Parameters: None

Query Response:
```
DEVINFO <max seq. steps module>,<max seq. steps Pod 0>,
        <max seq. steps Pod 1>,<max pins>,
        <max internal clock rate>,
        <min internal clock rate>,
        <internal clock resolution>,
        <min internal clock rate>,<max pods>;
```

`max seq. steps module`: numeric value (16351)
`max seq. steps Pod 0`: numeric value (16351)
`max seq. steps Pod 1`: numeric value (16351)
`max pins`: numeric value (64)
`max internal clock rate`: numeric value in ns (50)
`min internal clock rate`: numeric value in ns (3276700)
`internal clock resolution`: numeric value in ns (50)
`max pods`: numeric value (2)

Description: DEVINFO? returns the hardware configuration of this instrument. The valid sequence step range is the smallest number of sequence steps in the module or either pod.

Example: In this example, the configuration of the VX4820 is:16351 sequences supported by the module, 16351 sequences supported by pod 0, 16351 sequences supported by pod 1, a maximum of 64 I/O pins, maximum clock rate is 50 ns, minimum clock rate is 3276700 ns, clock resolution is 50 ns, 2 pods can be attached.

```
DEVINFO?

16351,16351,16351,64,50,3276700,50,2
```

Related Commands: PODTYPE

EVENT? **EVENT?**

SETUP
RUNTIME

Purpose: *EVENT?* recalls events from the event queue.

Query Syntax: `EVENT?`

Query Parameters: None

Query Response: `EVENT <code>;`

Description: EVENT? returns the event code of the most recent event. Once reported, the event is removed from the event queue.

Related Commands: EVMSG?

EVMSG? **EVMSG?**

SETUP
RUNTIME

Purpose: *EVMSG?* recalls an event message

Query Syntax: `EVMSG?`

Query Parameters: None

Query Response: `EVMSG <code>,"<message>";`

Description: EVMSG? returns the event code and a descriptive message string from the most recent event. Once reported, the event is removed from the event queue.

Related Commands: EVENT?

FAILDATA? **FAILDATA?**

SETUP

Purpose: *FAILDATA?* provides the failure sequence and captured pin data from a test.

Query Syntax: `FAILDATA?`

Query Parameters: None

Query Response `FAILDATA <sequence>,<data>;`

`sequence`: numeric value -1 .. 16350

-1 if not FAILED

`data`: string of char (64)

Only valid if test state is FAILED

Description: FAILDATA? returns information captured during test execution. A Fail register captures the binary state of all pins and the test sequence of the first test failure. This data is returned by FAILDATA?. If a test terminates in the PASSED state, FAILDATA? will return a sequence of -1 and the data string is invalid.

Example: In this example, the event occurred at sequence address 10.

FAILDATA?

FAILDATA 10,"00000000001111111111000000000011111111111000000000011111111110000"

Related Commands: FAILPIN?, STATE?

FAILPIN? **FAILPIN?**

SETUP

Purpose: *FAILPIN?* provides the failure sequence and failure pins from a test.

Query Syntax: `FAILPIN?`

Query Parameters: None

Query Response: `FAILPIN? <sequence>,<data>;`

`sequence`: numeric value – 1...16350

– 1 if not FAILED

`data`: string of char (64)

Only valid if test state is FAILED. Each character in the string represents a pin. If the character is a "0", then the pin passed. If the character is a "1", then the pin is failed. The least significant pin is the right-most character.

Description: FAILPIN? returns information on a failed test. A Fail register captures the failed pin and the test sequence of the first test failure. This data is returned by FAILPIN?. If a test terminates in the PASSED state, FAILPIN? returns a sequence of – 1 and the data string is valid.

Example: In this example, the event occurred at sequence address 100. Pins 20 to 29, 40 to 49, and 60 to 63 are failed.

```
FAILPIN?
FAILPIN 100,"1111000000000011111111110000000000111111111100000000000000000000";
```

Related Commands: FAILDATA?, STATE?

GRPMODE, GRPMODE? **GRPMODE, GRPMODE?**

SETUP

Purpose: *GRPMODE* controls the start/stop function
GRPMODE? returns the current GRPMODE setting

Command Syntax:

```
GRPMODE {SINGLE | COMMANDER | GROUP | STST}
```

Command Parameters:

`SINGLE`

The instrument operates as a single independent module.

`COMMANDER`

The module is designated as a group commander. It will send pseudo STST protocol to control other modules in the group.

`GROUP`

The module is designated as a group member. It expects to receive pseudo STST protocol from the group commander.

`STST`

The instrument will receive VXI STST protocol, starting and stopping test execution according to the STST state.

Query Syntax:

```
GRPMODE?
```

Query Parameters: None

Query Response:

```
GRPMODE {SINGLE | COMMANDER | GROUP | STST}
```

Description: Each VX4820 can operate as a single module or in a group with other VX4820s executing in tandem to form a monolithic instrument. The GRPMODE command is used to select between operating a single as a single module (GRPMODE SINGLE), or in tandem. If the module is to operate in tandem with other modules, GRPMODE determines what its role will be; either: a group commander (GRPMODE COMMANDER), a group member (GRPMODE GROUP), or supporting the VXI STST and ESTST protocols (GRPMODE STST).

When modules are to execute as a group, one of the modules must be designated as the group commander. This module must be the same one that was used during clock calibration. As mentioned above, the group commander is programmed with GRPMODE COMMANDER and all other group members are programmed with GRPMODE GROUP. Setup commands, such as INTCLKRATE, SEQ:START, and ARM, are sent to all modules in the group. Execution control commands, such as START, STOP, PAUSE, AND CONTINUE, are sent only to the group commander. In addition to sending PAUSE to the group commander, pausing test execution can be done by using the SEQ:PAUSE command at a selected test sequence (SEQ:PAUSE must be set on all modules in the group at the same test sequence). To restart test execution, a CONTINUE command is sent to only the group commander.

For operation according to the VXI STST and ESTST protocols, all modules are programmed with the GRPMODE STST command. Modules programmed with the GRPMODE STST command will not respond to STST or ESTST protocols until the ARM command is sent. Once test execution has started, asserting and deasserting the TTL STST trigger will move the state of the modules between RUNNING and PAUSED, respectively, until the SEQ:END test sequence is reached. To pause on a specific test sequence, SEQ:PAUSE must be set on all modules at the same test sequence. After pausing, the Resource Manager must deassert the selected TTL STST trigger prior to continuing the test. Asserting the TTL STST trigger will resume test execution. If you wish to terminate the test before the SEQ:END test sequence is reached, the TTL STST trigger should be deasserted. Then each module must be moved from the PAUSED state to the STOPPED state by sending the STOP command.

Modules operating in tandem must be in adjacent slots with their respective Anchor/Expander jumpers configured to ensure local bus communication (see Section 2, Module Jumpers for details). They must also share a common TTLTRG line, selected with the CONNECT:STST command. If multiple VX4820 modules are to operate independently, they should not be in adjacent slots, or their Anchor/Expander jumpers must be set to Anchor.

Example: In this example, the module is set to group commander mode:

```
GRPMODE COMMANDER
```

In this example the module group mode has been previously set to GROUP:

```
GRPMODE?
GRPMODE GROUP;
```

Related Commands: CONNECT:STST

HEADER, HEADER? **HEADER, HEADER?**

SETUP
RUNTIME

Purpose: *HEADER* turns the command response headers on/off.
HEADER? queries whether the command response headers are turned on or off.

Command Syntax:

```
HEADER {ON | OFF}
```

Command Parameters:

ON

Turns headers on.

OFF

Turns headers off.

Query Syntax:

```
HEADER?
```

Query Parameters: None

Query Response:

```
HEADER {1 | 0};
```

Description: The HEADER command turns the command headers in a response message on or off. When the command headers are turned OFF, only the parameters are returned.

Example: In this example, the CLKRATE command is used to show the different responses with headers set to on and off.

```
HEADER ON

INTCLKRATE?
INTCLKRATE 100;

HEADER OFF

INTCLKRATE?
100;
```

Related Commands: None

HELP? **HELP?**

SETUP
RUNTIME

Purpose: *HELP?* provides a list of VX4820 commands.

Query Syntax: `HELP?`

Query Parameters: None

Query Response: `HELP cmd1,cmd2,cmd3, ...,cmdN;`

Description: HELP? returns a list of all commands provided by the VX4820.

NOTE

The returned string includes all command headers and is very large. The size of the string is approximately 1130 characters.

Related Commands: None

IDPOD **IDPOD**

SETUP

Purpose: *IDPOD* controls the selected POD ID LED.

Command Syntax: `IDPOD <pod number>,{ON | OFF}`

Command Parameters: `pod number`

A numeric specifies which pod to identify (0 or 1).

`ON, OFF`

ON specifies turn on the POD ID LED.
OFF specifies turn off the POD ID LED.

Description: IDPOD is used to physically identify a particular pod in a system by turning the POD ID LED on/off .

Example: Identifies pod 1.

IDPOD 1,ON

Related Commands: None

INHIBIT **INHIBIT**

SETUP
RUNTIME

Purpose: *INHIBIT* sets all pins immediately to the high impedance state and terminates executing tests.

Syntax: `INHIBIT`

Parameters: None

Description: INHIBIT immediately terminates any test execution and inhibits (disables the pin output drivers) all pins. Test results will not be valid if a test is terminated with the INHIBIT command.

Related Commands: STOP

INIT **INIT**

SETUP

Purpose: Initialize all current settings to power-up default condition.

Command Syntax: `INIT`

Command Parameters: None

Description: INIT returns the module to the power up default settings. It does not change the test data.

Related Commands: *RST, NEW

INTCLKRATE, INTCLKRATE? **INTCLKRATE, INTCLKRATE?**

SETUP

Purpose: *INTCLKRATE* sets the internal clock rate.
INTCLKRATE? returns the current internal clock rate.

Command Syntax:

```
INTCLKRATE <Value>
```

Command Parameters:

`Value`

A numeric value between 50 and 3276700 which must be an integer multiple of 50. This value sets the internal clock rate in ns.

Query Syntax:

```
INTCLKRATE?
```

Query Parameters: None

Query Response:

```
INTCLKRATE <value>;
```

Description: INTCLKRATE sets the internal clock rate. If the parameter *Value* is not an integer multiple of 50, the VX4820 will round the clock rate to the closest multiple of 50 and generate a warning.

INTCLKRATE? returns the current internal clock rate. It returns a numeric value between 50 and 3276700 indicates the current internal clock cycle rate in ns.

Example: Set the internal clock rate to 150ns.

```
INTCLKRATE 150
```

In the following example, the internal clock rate is currently set to 250ns.

```
INTCLKRATE?
250
```

Related Commands: CONNECT:CLOCK

MODE, MODE? **MODE, MODE?**

SETUP

Purpose: *MODE* control operating mode.
MODE? return operating mode

Command Syntax:

```
MODE {TEST | LEARN}
```

Command Parameters:

`TEST`

In test mode, seven pin functions are available. These are DRIVE HI (1), DRIVE LO (0), INHIBIT (X), COMPARE HI (H), COMPARE LO (L), DRIVE HI and COMPARE HI (S), and DRIVE LO and COMPARE LO (R).

`LEARN`

In LEARN mode, three pin functions are available. These are DRIVE HI (1), DRIVE LO (0), and LEARN (X). The DRIVE functions are the same as in TEST mode. The LEARN function (X) causes the sampled state of the pin to be captured during test execution. The learned pin data (X) will be changed to COMPARE HI (H) and COMPARE LO (L) values as the result of test execution.

Query Syntax:

```
MODE?
```

Query Parameters: None

Query Response:

```
MODE {TEST | LEARN};
```

Description: MODE sets the current module operating mode. When the operating mode is set to TEST, the feature set of the instrument contains the ability to force data, compare data, force and compare data, and inhibit. When the instrument operating mode is set to LEARN, the instrument feature set changes; the abilities now include: force and learn (1, 0, X). The data present on the pins are sampled and stored in memory for each test sequence executed. This data can be later retrieved via SEQ:VECTOR?, FDCLOAD? or WSLOAD? commands.

MODE? returns the current operating mode.

NOTE

When LEARN mode is executed, the pin data is altered by the test to reflect the sampled input. Branch commands should not be used.

Example: Set the operating mode to LEARN:

```
MODE LEARN
```

In this example, it is currently set to Test mode.

```
MODE?
MODE TEST;
```

Related Commands: None

NEW **NEW**

SETUP

Purpose: *NEW* sets the test data to the default state.

Command Syntax: `NEW`

Command Parameters: None

Description: NEW sets all pins on all test steps to INHIBIT (X) and remove all TRIGGER, PAUSE, CLEAR, and BRANCH commands. Also, SEQ:START is set to 0, and SEQ:END is made invalid (set to -1).

Related Commands: INIT

NEWPASS **NEWPASS**

SETUP
RUNTIME

Purpose: *NEWPASS* changes the password.

Command Syntax: `NEWPASS <new password string>`

Command Parameters: new password string

Contains the new valid password string to be used with the PASSWORD command. There can be at most 2 characters in the string.

Description: NEWPASS changes the password string for the PASSWORD command. This command is operational only if a valid password string has been previously sent to the device with the PASSWORD command. The new password can be saved in the NVRAM using the NVSAVE command.

Example: In this example, the old password "DT" is changed to the new password "TK".

```
PASSWORD "DT"
NEWPASS "TK"
```

Related Commands: PASSWORD, NVSAVE

NVRECALL **NVRECALL**

SETUP

Purpose: *NVRECALL* sets calibration values and user-defined data to the values stored in nonvolatile memory.

Command Syntax: `NVRECALL`

Command Parameters: None

Description: NVRECALL restores calibration and user-defined data values from the nonvolatile memory. Any settings which have been modified since the last execution of the NVSAVE command are lost. When the module is powered up NVRECALL is automatically executed, initializing these values.

Related Commands: NVSAVE, CLKCAL, USER, PASSWORD, NEWPASS

NVSAVE **NVSAVE**

SETUP

Purpose: *NVSAVE* stores the current clock calibration values and user-defined data to the nonvolatile memory.

Command Syntax: NVSAVE

Command Parameters: None

Description: NVSAVE saves the current clock calibration and user-defined data in nonvolatile memory. On power up, the data will be restored from the nonvolatile memory automatically.

Related Commands: NVRECALL, CLKCAL, USER, PASSWORD, NEWPASS

PASSWORD **PASSWORD**

SETUP
RUNTIME

Purpose: *PASSWORD* enables some commands or functions that must be user restricted.

Command Syntax: `PASSWORD <password string>`

Command Parameters: password string

The character string password being sent to the device. If this parameter is omitted, it will enable the restriction of commands or functions.

Description: PASSWORD, with a valid password string, enables the restriction of commands or functions. A valid password is created with the NEWPASS command. Currently, the CLKCAL and USER commands are restricted commands. On reset and the execution of INIT and *RST commands, the access restriction will be enabled. The default password is "DT".

Example: In this example, the password "DT" is sent to disable the restriction of commands.

```
PASSWORD "DT"
```

Related Commands: NEWPASS, CLKCAL, USER, NVSAVE, NVRECALL

PAUSE **PAUSE**

RUNTIME

Purpose: *PAUSE* halts test progression until a CONTINUE or STOP command is received.

Command Syntax: PAUSE

Command Parameters: None

Description: PAUSE halts an executing test. To proceed with the test, a CONTINUE command must be sent. To terminate the test, a STOP command must be sent. If used with a module group, the PAUSE command must be sent to the group commander only.

Related Commands: CONTINUE, STOP

PINTEST? **PINTEST?**

SETUP

Purpose: *PINTEST?* tests the I/O pin hardware

Query Syntax: `PINTEST? <pod_number>`

Query Parameters: `pod_number`

A numeric value (0 or 1) selecting which pod to test.

Query Response: `PINTEST {PASS | FAIL},expect pin data,actual pin data;`

`expect pin data`: hexadecimal string
`actual pin data`: hexadecimal string

Description: PINTEST? tests the functionality of the pod I/O pins. The test includes DRIVE HI, DRIVE LO, COMPARE HI, and COMPARE LO for all data pins on the selected pod. Care should be taken to ensure the test adapter does not encumber the ability to drive the I/O pins during the execution of PINTEST?

CAUTION

This command causes a test pattern to be driven onto all of the I/O pins. Care should be used to ensure that the UUT is isolated from the pins during this test.

Example: In this test, Pin 31 of pod 1 has been damaged.

PINTEST? 1

PINTEST FAIL,#HFFFFFFFF,H7FFFFFF;

Related Commands: TST?

PODTYPE? **PODTYPE?**

SETUP

Purpose: *PODTYPE?* returns the pod description.

Query Syntax:
```
PODTYPE? <pod_number>
```

Query Parameters:
`pod_number`

A numeric value (0 or 1) selecting which pod to query.

Query Response
```
PODTYPE pod_identifier;
```

Description: Each active pod type has a unique identifier. This allows the user to determine the current configuration. If a pod is not attached it will return -1.

POD Identifier List

POD ID	POD DESCRIPTION
-1	No Pod Present
289	32 Pins of TTL I/O, 20 MHz, 16,351 sequence steps

Example: In this example, the pod type of unit 0 is 289.

```
PODTYPE? 0
PODTYPE 289;
```

In this example, pod 1 is not present.

```
PODTYPE? 1
PODTYPE -1;
```

Related Commands: DEVINFO?, IDPOD

READPIN? **READPIN?**

SETUP

Purpose: *READPIN?* reads the current data pattern at the pin.

Query Syntax: `READPIN?`

Query Parameters: `None`

Query Response `READPIN <data>;`

data: string of char (64)

Description: READPIN? reads the current data pattern at the pin. Before executing this command, the pins that are currently set up as compare or inhibit read the actual data at the pins. The pins that are set up as drive HIGH or LOW should read back the drive value if there isn't any other signal source at the pin.

Example: In this example, the current data pattern at the pin is as follows:

"1111111111111111111111111111111111111111111111111100001111
01101001"

Thus,

READPIN?

returns,

READPIN "1111111111111111111111111111111111111111111111111111
0000111101101001"

Related Commands: WRITEPIN

REPORT, REPORT? **REPORT, REPORT?**

SETUP
RUNTIME

Purpose: *REPORT* enables or disables various error, event, and warning report functions. *REPORT?* or *REPORT:< arg> ?* requests report settings.

Command Syntax:

```
REPORT:{CMDERR|CMDEVENT|CMDWARN|DEVERR|DEVEVENT|DEVWARN|
          EXEERR|EXEEVENT|EXEWARN} {1|0}
```

Command Parameters:

CMDERR: Command Errors
CMDEVENT: Command Events
CMDWARN: Command Warnings
DEVERR: Device Errors
DEVEVENT: Device Events
DEVWARN: Device Warnings
EXEERR: Execution Errors
EXEEVENT: Execution Events
EXEWARN: Execution Warnings

Query Syntax:

```
REPORT:{CMDERR|CMDEVENT|CMDWARD|DEVERR|DEVEVENT|DEVWARN|
          EXEERR|EXEEVENT|EXEWARN}?
or REPORT?
```

Query Parameters: < arg> is a variable representing one of the command parameters.

Query Response:

```
REPORT:{CMDERR|CMDEVENT|CMDWARN|DEVERR|DEVEVENT|DEVWARN|
          EXEERR|EXEEVENT|EXEWARN} {1|0};

or

REPORT:{CMDERR {1|0};|CMDEVENT {1|0};|CMDWARN {1|0};|
          DEVERR {1|0};|DEVEVENT {1|0};|DEVWARN {1|0};|
          EXEERR {1|0};|EXEEVENT {1|0};|EXEWARN {1|0};}
```

Description: REPORT sets the reporting mode for each class of event. When the reporting mode is set to off (0), no response is generated. When it is set to on (1), the results are as shown in the parameter definitions.

When REPORT? is used without parameters, it requests all nine of the report settings and produces a response to each.

Example: In this example, execution events will be disabled eliminating the reporting of sequence counter overflow, execution paused, and execution stopped.

```
REPORT:EXEEVENT 0
```

Related Commands: EVMSG?, EVENT?

SEQ:ADDRESS, SEQ:ADDRESS? **SEQ:ADDRESS, SEQ:ADDRESS?**

SETUP

Purpose: *SEQ:ADDRESS* sets the default sequence to a specific value. *SEQ:ADDRESS?* returns the current default sequence value.

Command Syntax:

```
SEQ:ADDRESS <address>
```

Command Parameters:

`address`

A numeric value specifies sequence address.

Query Syntax:

```
SEQ:ADDRESS?
```

Query Parameters: None

Query Response:

```
SEQ:ADDRESS address;
```

Description: SEQ:ADDRESS sets the sequence to a specific sequence address such that the test sequence can be edited by other commands.

SEQ:ADDRESS? returns a numeric value indicates the current sequence address.

Example: In this example, the current default sequence is set to 65. The following PAUSE? command references the default sequence set to 65.

```
SEQ:ADDRESS 65
SEQ:PAUSE?
SEQ:PAUSE OFF;
```

In this example, it indicates the current sequence address is set to 34.

```
SEQ:ADDRESS?
34
```

Related Commands: SEQ:DESC?, SEQ:VECTOR, SEQ:END, SEQ:PAUSE, SEQ:TRIG, SEQ:BRANCH

SEQ:BRANCH, SEQ:BRANCH? **SEQ:BRANCH, SEQ:BRANCH?**

SETUP

Purpose: *SEQ:BRANCH* sets a branch command on a test sequence.
SEQ:BRANCH? returns the branch information of a test sequence.

Command Syntax:
```
SEQ:BRANCH {PASS | FAIL | ALWAYS | NONE},<destination>
          [,sequence]
```

Command Parameters:

`PASS`

the branch will be taken if the Test State is PASSED.

`FAIL`

the branch will be taken if the Test State is FAILED.

`ALWAYS`

the branch will always be taken.

`NONE`

removes any previously placed branch command.

`destination`

A numeric value that specifies the next test step to be executed when the branch is taken.

`sequence`

A numeric value that specifies on which test sequence the branch instruction should be placed. If this parameter is omitted, the current default test sequence will be used.

Query Syntax:
```
SEQ:BRANCH? [sequence]
```

Query Parameters:

`sequence`

An optional numeric parameter that specifies the test sequence to query. If this parameter is omitted, the current default test sequence will be used.

Query Response:
```
SEQ:BRANCH {PASS | FAIL | ALWAYS | NONE},sequence;
```

Description: SEQ:BRANCH sets a branch instruction on a particular test sequence. The branch commands are placed on particular sequences and are executed when the sequence executes, at the programmed clock rate. If the branch is taken, the next test sequence executed following the branch test sequence will be the destination sequence. If the branch is not taken the next test sequence executed will be the branch sequence plus one.

Conditional branch command execution is dependent on the current Test State, passed or failed. An executing test begins in the PASS state. If, on execution of a test sequence, an input comparison fails (expected data not equal to actual), the Test State will transition to FAIL. The Test State will remain failed for the remainder of the test or until a SEQ:CLEAR command is executed. Any comparison which affects the outcome of a branch command should precede the BRANCH command by at least 8 sequences.

There are restrictions on branch placement and the branch destination sequence. Each branch command has a domain which includes the previous 7 sequences. Branch domains must exist and may not overlap each other (branches must be separated by at least 8 sequences). The branch destination sequence may not be a location within the domain of another branch. A branch command may not be placed on the test sequence 0 — 6.

Example: In this example, at the current sequence, set a branch to destination sequence 512 on test state = PASS.

```
SEQ:BRANCH PASS,512
```

In this example, at sequence 600, always branch to test sequence 18.

```
SEQ:BRANCH ALWAYS,18,600
```

In this example, the test sequence requested is 40, its branch type is PASS, and branch destination is sequence is 67.

```
SEQ:BRANCH? 40
SEQ:BRANCH PASS,67
```

In this example, the branch instruction on test sequence 40 is cleared.

```
SEQ:BRANCH NONE,0,40
```

Related Commands: SEQ:ADDRESS, SEQ:CLEAR, SEQ:DESC?

SEQ:CLEAR, SEQ:CLEAR? **SEQ:CLEAR, SEQ:CLEAR?**

SETUP

Purpose: *SEQ:CLEAR* sets the CLEAR command on a test sequence
SEQ:CLEAR? returns the current CLEAR command of a test sequence.

Command Syntax:

```
SEQ:CLEAR {ON | OFF}[,sequence]
```

Command Parameters:

```
{ON | OFF}
```

ON places a CLEAR command on the test sequence.

OFF removes a CLEAR command from the test sequence.

```
[,sequence]
```

A numeric value specifies the sequence. If this parameter is omitted, the current default sequence will be used.

Query Syntax:

```
SEQ:CLEAR? [sequence]
```

Query Parameters:

```
[sequence]
```

A numeric value specifies the test sequence. If this parameter is omitted, the current test sequence address will be used.

Query Response

```
SEQ:CLEAR {ON | OFF};
```

Description: When a test sequence is executed that contains a SEQ:CLEAR command, the Test State is set to PASS and any stored failure data is lost. The SEQ:CLEAR command should not be set within BRANCH domains (see SEQ:BRANCH).

Example: Set the CLEAR condition of sequence 45:

```
SEQ:CLEAR ON,45
```

In this example, the return value of the query indicates that a SEQ:CLEAR command is set on test sequence 91

```
SEQ:CLEAR? 91
ON
```

Related Commands: SEQ:ADDRESS, SEQ:DESC?

SEQ:DESC? **SEQ:DESC?**

SETUP

Purpose: *SEQ:DESC?* returns the pin data of a sequence.

Query Syntax:
```
SEQ:DESC? [sequence]
```

Query Parameters:
```
[sequence]
```

A numeric value specifies the test sequence. If this parameter is omitted, the current default test sequence will be used.

Query Response:
```
data,pause,clear,trigger,branch type,branch destination
```

```
data
```

reference SEQ:VECTOR

```
pause ::= {ON | OFF}
```

reference SEQ:PAUSE

```
clear ::= {ON | OFF}
```

reference SEQ:CLEAR

```
trigger ::= {ON | OFF}
```

reference SEQ:TRIG

```
branch type ::= {PASS | FAIL | ALWAYS | NONE}
```

reference SEQ:BRANCH

```
branch destination
```

reference SEQ:BRANCH

Description: SEQ:DESC? returns all information associated with a particular test sequence. For additional information regarding each returned field, refer to the referenced command.

Example: In this example, the PAUSE and CLEAR condition of this sequence is cleared, and the TRIGGER condition is set. The branch type is branch on pass and the branch address is 90.

```
SEQ:DESC?

SEQ:DESC "010101HLSC00011000000000011111111111HHHHHHH
HHHLLLLLLLLLLXXXXXXXXXX000000000",OFF,OFF,ON,PASS,90;
```

Related Commands: SEQ:BRANCH?, SEQ:CLEAR?, SEQ:PAUSE?, SEQ:TRIGGER?

SEQ:END, SEQ:END? **SEQ:END, SEQ:END?**

SETUP

Purpose: *SEQ:END* marks the last sequence of a test.
SEQ:END? returns the last sequence of a test.

Command Syntax:

```
SEQ:END <sequence>
```

Command Parameters:

sequence

A numeric value which specifies the last sequence of a test.

Query Syntax:

```
SEQ:END?
```

Query Parameters: None

Query Response:

```
SEQ:END sequence;
```

sequence

numeric value -1 .. 16350, -1 if sequence not valid

Description: SEQ:END specifies the last sequence to be executed within a test. When the SEQ:END is executed the test will STOP. The values driven on the I/O pins will freeze at the values programmed on that sequence.

There are restrictions around the SEQ:END command. In MODE TEST, functions which affect the Test State should not be programmed on the last 5 sequences within a test. This includes input comparisons (H, L, S, R) and the SEQ:CLEAR command. In MODE LEARN, the SEQ:END test sequence is not learned; it remains unaltered.

Example: In this example, the end of the test will be set to sequence 100.

```
SEQ:END 100
```

In this example, the last sequence is currently set to 145.

```
SEQ:END?
SEQ:END 145;
```

Related Commands: SEQ:START

SEQ:PAUSE, SEQ:PAUSE? **SEQ:PAUSE, SEQ:PAUSE?**

SETUP

Purpose: *SEQ:PAUSE* sets the PAUSE command on a test sequence
SEQ:PAUSE? returns the current PAUSE condition of a test sequence.

Command Syntax:

```
SEQ:PAUSE {ON | OFF}[,sequence]
```

Command Parameters:

```
{ON | OFF}
```

ON places a PAUSE command on the test sequence.

OFF removes a PAUSE command from the test sequence.

```
[,sequence]
```

A numeric value specifies the test sequence. If this parameter is omitted, the current default sequence address will be used.

Query Syntax:

```
SEQ:PAUSE? [sequence]
```

Query Parameters:

```
[sequence]
```

A numeric value specifies the sequence address. If this parameter is omitted, the current sequence address will be used instead.

Query Response

```
SEQ:PAUSE {ON | OFF};
```

Description: If a test sequence is set to pause, the progression of test sequences will stop when the PAUSE is executed. This condition will remain until a CONTINUE command is received.

Example: Clear the PAUSE condition on sequence 660:

```
SEQ:PAUSE OFF,660
```

In this example, the return value of the query indicates that the PAUSE condition of the current sequence address is set.

```
SEQ:PAUSE?
SEQ:PAUSE ON;
```

Related Commands: CONTINUE, PAUSE, SINGLESTEP

SEQ:SEQINC **SEQ:SEQINC**

SETUP

Purpose: SEQ:SEQINC increments the current default sequence value.

Command Syntax: `SEQ:SEQINC [offset]`

Command Parameters: `[offset]`

A numeric value specifies an offset to be added to the current sequence. If this parameter is omitted, the default sequence value will be increment by 1.

Description: SEQ:SEQINC allows the user to set the default sequence value relative to its current value.

Example: In this example, the sequence is changed to the current default sequence minus 4.

```
SEQ:SEQINC -4
```

Related Commands: SEQ:ADDRESS

SEQ:START, SEQ:START? **SEQ:START, SEQ:START?**

SETUP

Purpose: *SEQ:START* sets the starting sequence of a test.
SEQ:START? returns the starting sequence of a test.

Command Syntax:

```
SEQ:START <sequence>
```

Command Parameters:

```
<sequence>
```

A numeric value specifies the first test sequence executed when the START command is received.

Query Syntax:

```
SEQ:START?
```

Query Parameters: None

Query Response:

```
SEQ:START sequence;
```

Description: SEQ:START specifies the sequence at which test execution will begin. The power-up default SEQ:START sequence is 0.

Example: In this example, the starting sequence of the test will be set to 1000.

```
SEQ:START 1000
```

In this example, the starting sequence step has been previously set to 2000;

```
SEQ:START?
SEQ:START 2000;
```

Related Commands: ARM, SEQ:END, START

SEQ:TRIG, SEQ:TRIG? **SEQ:TRIG, SEQ:TRIG?**

SETUP

Purpose: *SEQ:TRIG* sets the TRIGGER command on a test sequence.
SEQ:TRIG? returns the current TRIGGER condition of a test sequence.

Command Syntax:

```
SEQ:TRIG {ON | OFF}[,sequence]
```

Command Parameters:

```
{ON | OFF}
```

ON places a TRIGGER command on the test sequence.
OFF removes a TRIGGER command from the test sequence.

```
[,sequence]
```

A numeric value specifies the test sequence. If this parameter is omitted, the current default test sequence will be used.

Query Syntax:

```
SEQ:TRIG? [sequence]
```

Query Parameters:

```
[sequence]
```

A numeric value specifies the test sequence. If this parameter is omitted, the current default test sequence will be used.

Query Response:

```
SEQ:TRIG {ON | OFF};
```

Description: TRIGGER commands may be placed on any test sequence. When the test sequence is executed, the selected trigger line will be asserted for the duration of the sequence. Triggers may be output on the VXI TTLTRG or ECLTRG lines. TTLTRG lines are asserted LOW true and ECLTRG lines are asserted HIGH true. The trigger line is selected by the CONNECT:TRIGGEROUT command.

Example: Clear the TRIGGER condition on current sequence address:

```
SEQ:TRIG OFF
```

In this example, the return value of the query indicates that the TRIGGER condition of the sequence 455 is set.

```
SEQ:TRIG? 455
SEQ:TRIG ON;
```

Related Commands: CONNECT:TRIGGEROUT

SEQ:VECTOR, SEQ:VECTOR? **SEQ:VECTOR, SEQ:VECTOR?**

SETUP

Purpose: *SEQ:VECTOR* sets pin data for a sequence.
SEQ:VECTOR? returns the pin data of a sequence.

Command Syntax:

```
SEQ:VECTOR <data> [,sequence]
```

Command Parameters:

`data`

An string of ASCII data represents test vector. Valid characters are "0", "1", "X", "L", "H", "R", "S", spaces and tabs.

"0" - Drive pin LOW
"1" - Drive pin HIGH
"X" - Inhibit pin driver
"L" - Compare pin with LOW
"H" - Compare pin with HIGH
"R" - Drive pin LOW and compare with LOW
"S" - Drive pin HIGH and compare with HIGH
* spaces and tabs are ignored

`[,sequence]`

A numeric value specifies the test sequence. If this parameter is omitted, the current default test sequence will be used.

Query Syntax:

```
SEQ:VECTOR? [sequence]
```

Query Parameters:

`[sequence]`

A numeric value specifies the test sequence. If this parameter is omitted the current default test sequence will be used.

Query Response:

```
SEQ:VECTOR data;
```

Description: SEQ:VECTOR allows the user to define pin data with an ASCII character string. The least significant pin is the right-most character, and data is right justified. High order pins which are not specified default to "X" (inhibit).

Example: In this example, pin data is defined for the current sequence address. Pins 1, 2 and 8 are set to drive LOW. Pins 0 and 7 are set to drive HIGH. Pin 3 is set to compare HIGH. Pin 4 is set to compare LOW. Pin 5 is drive output LOW and compare LOW. Pin 6 is drive output HIGH and compare HIGH. Pins 9 to 63 default to inhibit output driver ("X").

```
SEQ:VECTOR "01SRLH001"
```

In this example, the sequence data is defined for the sequence 55. Pins 0 to 9 are set to inhibit output driver. Pins 10 to 19, 40 to 59, 61 and 62 are set to drive LOW. Pins 20 to 39, 60 and 63 drive HIGH.

```
SEQ:VECTOR "1001 00000000000000000000 1111111111111111
1111 0000000000 XXXXXXXXXX",55
```

In this example, the pin data of sequence 989 is returned.

```
SEQ:VECTOR? 989
SEQ:VECTOR "1001000000000000000000001111111111111111
1111110000000000 XXXXXXXXXX"
```

Related Commands: SEQ:DESC?

SEQ:VECTORINC **SEQ:VECTORINC**

SETUP

Purpose: *SEQ:VECTORINC* sets the pin data of a sequence and increment the default test sequence by one

Command Syntax: `SEQ:VECTORINC <data> [,sequence]`

Command Parameters: `data`

See SEQ:VECTOR command for definition of sequence data.

`[,sequence]`

A numeric value that specifies a test sequence. If this parameter is omitted, the current default test sequence will be used.

Description: This command is similar to the SEQ:VECTOR command, except the sequence will be incremented after the pin data is set.

Example: In this example, after the data is written to sequence 10, the sequence will be incremented to 11

```
SEQ:ADDRESS?
SEQ:ADDRESS  10;
SEQ:VECTORINC "1010 XXXXHLHL"
SEQ:ADDRESS?
SEQ:ADDRESS 11;
```

Related Commands: SEQ:VECTOR

SINGLESTEP, SINGLESTEP? **SINGLESTEP, SINGLESTEP?**

SETUP
RUNTIME

Purpose: *SINGLESTEP* enables/disables Single Step mode.
SINGLESTEP? returns the Single Step status.

Command Syntax:

```
SINGLESTEP {ON | OFF}
```

Command Parameters:

```
{ON | OFF}
```

ON selects Single Step mode.
OFF disables Single Step mode.

Query Syntax:

```
SINGLESTEP?
```

Query Parameters: None

Query Response:

```
SINGLESTEP {ON | OFF};
```

Description: In Single Step mode, an executing test will pause on each test sequence. To advance to the next sequence in the test, a CONTINUE command must be sent. If the SINGLESTEP command is received before the ARM command is received, Single Step mode will commence with the first executed test sequence. Single Step mode may also be entered during a test if the test is paused.

CAUTION

Entering Single Step mode while a test is not paused or stopped may produce unpredictable results.

Example: Start the test using Single Step mode:

```
SINGLESTEP ON
ARM
START
```

In this example, the current execution mode is Single Step:

```
SINGLESTEP?
SINGLESTEP ON;
```

Related Commands: CONTINUE

START **START**

SETUP

Purpose: *START* begins the execution of a test

Command Syntax: `START`

Command Parameters: None

Description: START causes the VX4820 to begin executing a test. The starting address of the test is specified by the SEQ:START command. The ARM command must be sent to the module prior to the START command. In a multi-module group, this command should only be sent to the group commander.

Example: In this example, the test is started at sequence 55.

```
SEQ:START 55
SEQ:END  200
ARM
START
```

Related Commands: INHIBIT, ARM, STOP, SEQ:START

STATE? **STATE?**

SETUP
RUNTIME

Purpose: *STATE?* returns the status of test execution

Query Syntax:
```
STATE?
```

Query Parameters: None

Query Response:
```
STATE? {RUNNING | PAUSED | SINGLESTEP | STOPPED |
        ARMED}, sequence, sequence count, test state,
        module state;
```

`sequence`

numeric value -1 .. 16350
-1 if sequence invalid: module ARMED but not RUNNING or illegal branch taken.

`steps executed`

numeric value 0 .. 4294967295 (2^{32})

`test state`

PASS | FAIL

`module state`

PASS | FAIL

Description: STATE? returns the test execution status. The test state reflects the status of all modules in the test system. The module state reflects only the contribution to the Test State of that module. If the test is running, only the upper 16 bits of the sequence count are valid.

During runtime the test state has a 6 step latency: the displayed state is the state value which existed on the current sequence step - 6. The sequence value has a 2 step latency. When the test is stopped, all information returned in the STATE? command is current.

Example: In this example, the VX4820 is stopped at sequence 120, 23 sequences have executed, and the test is passed:

```
STATE?
STATE STOPPED,120,23,PASS,PASS;
```

Related Commands: SEQ:DESC?

STERR? **STERR?**

SETUP
RUNTIME

Purpose: *STERR?* returns any error codes and messages generated by self test and optionally clears the error buffer.

Query Syntax:

```
STERR? [CLEAR]
```

Query Parameters:

```
[CLEAR]
```

An optional parameter that clears all error codes and messages.

Query Response:

```
STERR number of errors, error code, error message, error
code, error message, error code, error message;
```

```
number of errors
```

number of errors in this list

```
error code
```

numeric error code, 0 is not an error

```
error message
```

descriptive error string

Description: STERR? allows the user to query the error information. Error information is generated by power-up diagnostics and the first three errors found are stored in nonvolatile memory. When the CLEAR parameter is added to the command, all the error information in the nonvolatile RAM will be cleared.

Example: This example queries for diagnostic errors. Only one error occurred in the last self test.

```
STERR?
STERR 1,13,"POD RAM error",0,"NONE",0,"NONE";
```

This example clears all the error information in the nonvolatile memory.

```
STERR? CLEAR
STERR 0,0,"NONE",0,"NONE",0,"NONE";
```

Related Commands: TST?

STOP **STOP**

SETUP
RUNTIME

Purpose: *STOP* terminates a test.

Command Syntax: `STOP`

Command Parameters: None

Description: Normally, a test terminates as the result of executing the test sequence specified by the SEQ:END command. The STOP command may be sent to terminate a running or paused test. It can be sent at any time to ensure the VX4820 is stopped. The STOP command performs the required hardware shut-down procedures to ensure test data is not lost. When the SEQ:END is executed, a STOP is automatically executed. In a multi-module GROUP, this command must be sent to the group COMMANDER first and then to the modules in the group.

Related Commands: INHIBIT

TSTPAT **TSTPAT**

SETUP

Purpose: *TSTPAT* loads the pod RAM memory with test patterns.

Command Syntax: `TSTPAT`

Command Parameters: None

Description: TSTPAT loads several test patterns into the pod RAM for diagnostic purposes. The patterns loaded are shown in the following chart:

Test #	Address	Function
1	0 — 7 (loops)	Walking 1s on 8-bit boundaries (0000 0001 0000 0001 ...)
2	10 — 17 (loops)	Alternate 1s and 0s
3	20 (set End at 28)	All LOW
4	30 (set End at 38)	All HIGH

Send the SEQ:START and SEQ:END commands to select which pattern to execute.

Related Commands: None

SETUP
RUNTIME

Purpose: *USER* provides a nonvolatile storage area for the user.

USER? returns the contents of the user nonvolatile storage area.

Command Syntax:

```
USER "<data>"
```

Command Parameters:

`data`

A string of data less than or equal to 20 characters.

Query Syntax:

```
USER?
```

Query Parameters: None

Query Response:

```
USER "data";
```

`data`: string 0 .. 20 characters

Description: *USER* defines a string of data that can be written or read by the user. This string is made nonvolatile when the NVSAVE command is issued. The length of the string cannot be longer than 20 bytes. USER command requires a valid password to disable the access restriction. The password is entered through the PASSWORD command.

Example: In this example, the user defined data is set and made nonvolatile:

```
PASSWORD "DT"
USER "6/22/1990"
NVSAVE
```

In this example, the user defined data is "TEKTRONIX"

```
USER?
USER "TEKTRONIX";
```

Related Commands: NVSAVE, PASSWORD

WRITEPIN **WRITEPIN**

SETUP

Purpose: *WRITEPIN* drives the pod output pin with a data pattern.

Command Syntax: `WRITEPIN <data>`

Command Parameters: data

A string of ASCII data represents test vectors. Valid characters are"0", "1", "X", spaces, and tabs.

"0" — Drive pin Low
"1' — Drive pin High
"X" — Inhibit pin driver
Spaces and Tabs are ignored

Description: WRITEPIN allows the user to define the current output pin data with an ASCII character string. The least significant pin is the right-most character, and data is right justified. High order pins which are not specified default to "X" (inhibit).

Example: In this example, the output pin data is driven as: pins 0, 1, 2, and 10 are set to drive HIGH; pins 3, 4, and 8 are set to inhibit; and pins 5, 6, 7, and 9 are set to drive LOW. Pins 11 and 63 default to inhibit output drive ("X").

```
WRITEPIN "10X000XX111"
```

Related Commands: READPIN?

WSLOAD,WSLOAD? **WSLOAD, WSLOAD?**

SETUP

Purpose: *WSLOAD* downloads binary test pattern data to the VX4820 utilizing IEEE 488.2 definite length arbitrary block program data.
WSLOAD? uploads binary pattern data from the VX4820 utilizing IEEE 488.2 definite length arbitrary block response data

Command Syntax:

```
WSLOAD <arbitrary block program data>
```

Command Parameters:

```
arbitrary block program data
```

This format contains a header section which describes how may bytes will be transferred, followed by the data. The indefinite format (#0) is not supported. An example of this format is shown below.

#< 1> < 5> ABCDE

In this example the "#" starts the header. The character (8 bit) following the "#" is interpreted as a binary number which defines how many decimal digits follow it. The decimal digits describe how many data bytes are to be transferred. Finally the "ABCDE", the actual data transferred, follows the header.

Query Syntax:

```
WSLOAD?
```

Query Response:

```
definite length arbitrary block response data
```

same as arbitrary block program data above

Description: WSLOAD supports an alternate method for uploading or downloading test pattern data. The pattern data is transferred in the same low level binary format as is used for Fast Data Channel data buffer format. For a more detailed description of the data buffer format see *Appendix C*.

Related Commands: FDCLOAD

Section 7
System Command Dictionary

Introduction

Since the VX4820 is a VXI instrument rather than a GPIB instrument, it does not directly respond to GPIB commands except through the 488-VXIbus Interface Device. If the VX4820 is operated in an environment where the Host-to-VXI Subsystem interface is an IEEE 488.1 (GPIB) Interface, the 488-VXIbus Interface device must be prepared to intercept and implement any IEEE 488.2 commands sent to the VX4820.

This section describes the IEEE 488.2 Common Commands and Low-Level VXIbus commands implemented by the VX4820. Commands prefixed with an asterisk (*) are IEEE 488.2 Common Commands. All others are classified as VXI Low-Level commands. These commands need to be read very closely, as an IEEE 488.2 Common Command received by the 488-VXIbus Interface Device may be implemented by sending an IEEE 488.2 Common Command.

When a VXIbus Interface Device intercepts and implements the mandatory IEEE 488.2 Common Commands, it does so in one of the following ways:

- Accepts the command and rebroadcasts it to the VX4820.
- Accepts the command and processes it without sending it to the VX4820.
- Accepts the command and issues another command (may be a Word Serial Protocol, IEEE 488, Tektronix Instrument, or some other command) to the VX4820.

To support embedded addressing, each of the IEEE 488 Common Commands implemented by the VX4820 also has a version not prefixed with an asterisk (*). This non-prefixed version can be addressed directly to the VX4820 at any time. The action by the VX4820 is identical for both versions.

Alphabetical Listing

Following is an alphabetical listing and a purpose statement for each VX4820 IEEE 488.2.

***CLS, CLS** — Clears instrument event registers and queues, except the output queue, and places a "1" in the output queue to indicate the **OPC?* idle state.

***ESE, ESE** — Sets the Event Status Enable Register bits.

***ESE?, ESE?** — Queries the setting of the Event Status Enable Register bits.

***ESR?, ESR?** — Reads the contents of the Event Status Register.

ID? — An older Tektronix GPIB command that returns unique identification for the instrument.

***IDN?, IDN?** — Query only. Returns the unique identification for the instrument.

***OPC, OPC** — Causes the VX4820 to set the Operation Complete bit in the Standard Events Status Register when all operations are completed.

***OPC?, OPC?** — Determines if all pending operations have been completed.

***RST, RST** — Initializes the instrument to its power-up state (settings made with the *INST* command only), but doesn't alter the VXIbus interface data or change calibration constants.

***SRE, SRE** — Sets the bits in the Service Request Enable Register.

***SRE?, SRE?** — Queries the bit settings of the Service Request Enable Register.

***STB?, STB?** — Reads the Status Byte Register.

***TST?, TST?** — Causes the instrument to execute its power-up self-test routines, except the kernel verification tests, and reports any failures that occur.

***WAI, WAI** — Prevents the VX4820 from executing further commands or queries until the No-Operation-Pending flag is set.

***CLS, CLS** ***CLS, CLS**

Purpose: Clears all instrument Event Status registers, clears all queues, except the output queue, and leaves the instrument in the *OPC?* idle state.

Syntax:
```
*CLS
 CLS
```

Parameters: None

Description: The 488-VXIbus Interface Device issues this command whenever it receives a software *CLEAR* command.

Related Commands: None

***ESE, ESE, *ESE?, ESE?** ***ESE, ESE, *ESE?, ESE?**

Purpose: **ESE* sets the Event Status Enable Register bits.
**ESE?* queries the setting of the Event Status Enable Register bits.

Command Syntax:

```
*ESE <integer>
 ESE <integer>
```

Query Syntax:

```
*ESE?
 ESE?
```

Command Parameters:

`<integer>`

An integer in the range of 0 — 255 that represents the desired setting of the Event Status Enable Register bits.

Query Parameters: None

Description: If an out-of-range integer is entered for the value of *<integer>* , a command error is generated.

The **ESE?* query generates an NR1 integer response (in the range of 0 — 255) that corresponds to the bits set in the Event Status Enable Register.

Example:

In the following example, the **ESE*command sets bits 1 and 2:

```
*ESE 3
```

In the following example, the **ESE?*query determines that bits 1, 2, and 6 are set:

```
*ESE?
9
```

Related Commands: None

***ESR?, ESR?** ***ESR?, ESR?**

Purpose: Reads the contents of the Event Status Register.

Syntax:
```
*ESR?
 ESR?
```

Parameters: None

Description: The **ESR?* query clears the Event Status Register after reading. The **ESR?* query generates an NR1 integer response (in the range of 0 — 255) corresponding to the bits set in the Error Status Register.

Example: In the following example, the **ESR?*query determines that bits 1, 2, and 6 are set:

```
*ESR?
9
```

Related Commands: None

ID? **ID?**

Purpose: Returns the unique identification of the instrument.

Syntax: `ID?`

Parameters: None

Description: This is an older Tektronix GPIB command used in previous software programs to query for the unique identification of an instrument. In newer instruments, it has been replaced with the IEEE 488.2 **IDN?* command. This command is implemented for compatibility with older Tektronix software.

The returned value is:

```
ID TEK/VX4820,CF:89.1CN,RM:x.y
```

where *x.y* is the firmware version number.

Related Commands: None

***IDN?, IDN?** ***IDN?, IDN?**

Purpose: Query only. Returns the unique identification of the instrument.

Syntax:
```
*IDN?
 IDN?
```

Parameters: None

Query Response: `TEKTRONIX,VX4820,0,CF:89.1CN RM:<firmware revision>`

Description: This is the identify device query per the Tektronix Codes and Formats Standard.

This information is drawn from the VX4820 ID-ROM. Note that this is a query-only command; there is no associated statement-type command. For more detailed information on this command, refer to IEEE Standard 488.2.

Related Commands: None

***OPC, OPC, *OPC?, OPC?** ***OPC, OPC, *OPC?, OPC?**

Purpose: **OPC* causes the VX4820 to set the Operation Complete bit in the Standard Event Status Register when all operations are completed.

**OPC?* determines if all pending operations have been completed.

Command Syntax:
```
*OPC
 OPC
```

Command Parameters: None

Query Syntax:
```
*OPC?
 OPC?
```

Query Parameters: None

Description: When the **OPC?* query is issued, it remains in a waiting state until all operations are complete, then sets the Operation Complete bit in the Standard Event Status Register.

NOTE

*When all operations are complete, the *OPC command places an operation complete message in the event queue.*

Related Commands: `*WAI`

***RST, RST** ***RST, RST**

Purpose: Resets the VX4820.

Syntax:
```
*RST
 RST
```

Parameters: None

Description: This command resets the VX4820 to the power up state and is equivalent to sending the INIT command.

Related Commands: `None`

***SRE, SRE, *SRE?, SRE?** ***SRE, SRE, *SRE?, SRE?**

Purpose: **SRE* sets the bits in the Service Request Enable Register.

**SRE?* queries the bit settings of the Service Request Enable Register.

Command Syntax:
```
*SRE <number>
 SRE <number>
```

Command Parameters:

`<number>`

Decimal numeric program data in the range of 0 — 255. This number, when rounded to an integer value and expressed in binary, represents the desired bit values of the Service Request Enable Register.

Query Syntax:
```
*SRE?
 SRE?
```

Query Parameters: None

Description: When using the **SRE* command, the value specified for Bit 6 will be ignored. (Bit 6 is always 0.)

The response to the **SRE?* query is an NR1 in the ranges of 0 — 63 or 128 — 191. When converted to binary, this number represents the current bit values of the Service Request Enable Register. Bit 6 of the Service Request Enable Register bit values will always be 0.

Related Commands: None

***STB?, STB?** ***STB?, STB?**

Purpose: Reads the status byte from the Status Byte Register.

Syntax:
```
*STB?
 STB?
```

Parameters: None

Description: The response to the **STB?* query is an NR1 that corresponds to the bits set in the Status Byte Register.

Example: In the following example, the **STB?* query indicates that bits 1 and 2 of the Status Byte Register are set:

```
*STB?
3
```

Related Commands: None

***TST?, TST?** ***TST?, TST?**

Purpose: Executes instrument diagnostic routines.

Syntax:
```
*TST?
 TST?
```

Parameters: None

Query Response
```
number of errors;
```

Description: This command causes the VX4820 to execute the internal diagnostic routines. In multi-module systems, each module must be reset (INIT or RST commands) prior to executing the internal diagnostic routines or incorrect errors will be reported.

CAUTION

This command cause a test pattern to be driven onto the I/O pins. Care should be used to ensure that the UUT is isolated from the pins during this test.

Before resuming normal execution, reset each module with the INIT and NEW commands.

Related Commands: STERR?, PINTEST?

***WAI, WAI** ***WAI, WAI**

Purpose: Prevents the VX4820 from overlapping command execution.

Syntax:
```
*WAI
 WAI
```

Parameters: None

Description: The *WAI command is used to prevent the VX4820 from overlapping or simultaneous executing separate commands. When the *WAI command is received, the VX4820 will not begin command execution of a following command until all current operations are completed.

Related Commands: *OPC?

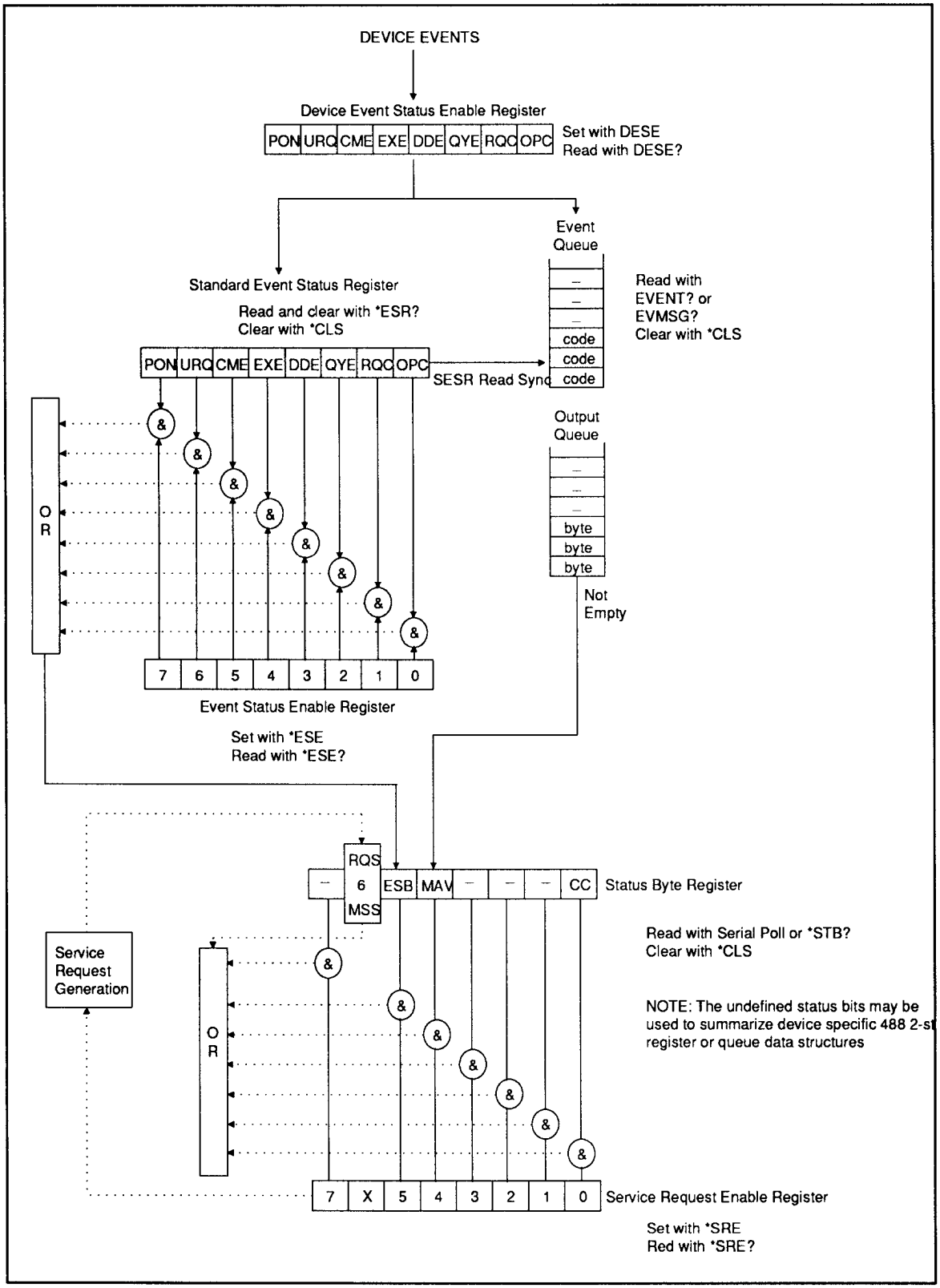


Figure 7-1. Register Bit Definitions

Section 8
Digital Toolbox

Introduction

The Digital Toolbox is a member of the TekTMS family of test management software. It supports the generation and debug of test data for the VX4820 Digital Test Module. *EDIT* allows manual entry of data, the import and export of ASCII files, and Maco capabilities. The output of EDIT is a binary pattern file which may be loaded into VX4820 modules. *VERIFY*checks for correct syntax in user defined macro files. *MERGE*combines multiple pattern files supporting modular design of digital tests. *DEBUG* connects the software tools to the VX4820 hardware allowing interactive test execution and debug. All of these tools run under the Windows$_{TM}$ 3.0 graphical user inteterface.

Features

- Easy generation of digital tests
- User-defined Macros with parameter passing
- Import and Export of ASCII test files
- Support for modular test design
- Download and execute digital tests interactively
- Extensive debug support includes breakpoints, single-step and command trace

Description

The Digital Toolbox contains four application programs: EDIT, a digital test data editor; VERIFY, a macro checker; MERGE, a test merge utility; and DEBUG, a loader/debugger utility. These programs are Windows$_{TM}$ 3.0 applications.

EDIT is a full-featured, mouse-driven digital pattern editor. Standard features are provided such as cut, copy, and paste. Other data manipulation functions include insert rows, fill, and vertical entry. Extensive display formatting is provided supporting grouped data, selectable radix, and user-defined notes. Macros can be included within the test data, allowing test generation at the transaction level rather than the bit level. All pin functions (drive, compare, and inhibit) are supported in a single display.

VERIFY supports the generation of user defined macro libraries. Macro libraries can be generated using a text editor such as Notepad or Write. VERIFY will check the macro text file for errors and provide error messages and online help to correct the errors. Once all errors have been eliminated, the macro text file can be read and applied within EDIT.

MERGE combines multiple pattern files into a single test output test file. EDIT can be used to build test modules which are non-overlapping. A group of these test modules can then be combined into a single downloadable file using MERGE.

DEBUG forms a connection between the test generation tools and the VX4820 module. Test files can be associated with particular VX4820 Module and selectively loaded. Once the test file is loaded, the test can be executed or single stepped. Breakpoints may be set within the test. Test progress and status is continuously displayed. An integrated line editor allows simple modifications to be made to the test data interactively.

What is required

To utilize EDIT, VERIFY and MERGE you must have a IBM PC or true compatible with Windows$_{TM}$ 3.0 installed. The Digital Toolbox applications require at least one megabyte of hard disk space plus .5 megabyte for each VX4820 module.

The DEBUG application requires a VX4530/5 or VX5530/5 embedded controller with EPConnect 3.04 and Windows$_{TM}$ 3.0 installed.

Installation

To install the Digital Toolbox:

1. Start Windows 3.0 by typing **win** and then ENTER.

2. Put the Digital Toolbox disk in the drive you want to use for the installation and close the drive door.

3. In the Windows 3.0 program manager, pull down the **File** menu and select **Run....**

4. On the RUN Command Line type the drive path for the selected floppy drive followed by **install** . If you are using drive A you would type A:INSTALL and press ENTER.

5. Follow the instructions provided by the Install application. INSTALL will load all of the toolbox applications in a single group window. Each of these applications may be executed by double-clicking their icon.

NOTE

If you install the Digital Toolbox on a PC which already has a copy of the toolbox, two sets of icons will be displayed. The second set of icons can be removed by clicking the duplicate icon and typing <Del> <Ret>. The origional and duplicate icon are identical in function and point to the same executable file.

Using Digital Toolbox

Generating Test data

The digital test data defines the I/O function of each pin during each step of the test. The VX4820 provides seven pin functions: Drive HI (1), Drive LO (0), Inhibit (X), Compare HI (H), Compare LO (L), Drive HI with compare HI (S), and Drive LO with compare LO (R).

VX4820 Data Representation

CHAR	DRIVER	COMPARE
1	Drive the pin HI	Mask - no comparison
0	Drive the pin LO	Mask - no comparison
X	Inhibit - high impedance	Mask - no comparison
H	Inhibit - high impedance	Compare with HI
L	Inhibit - high impedance	Compare with LO
S	Drive the pin HI	Compare with HI
R	Drive the pin LO	Compare with LO

Digital Toolbox Edit provides three methods to generate test data. The test data may be directly entered, entered as macro transactions, or imported from an ASCII file. In direct entry mode, the user types the actual pin functions to be executed. To make direct data entry manageable, extensive data-formating capabilities such as pin grouping, selectable group radix, and user defined notes are provided.

Data may also be entered as macro transactions. Macros define multi-step I/O transactions. These transactions may form complete bus read or write cycles. Parameters such as address and data can be passed into a macro so that each instance of the macro is unique. Macros support a level of absraction to the user from the bit level to the transaction level.

The user can define his own unique macro transactions by generating macro libraries. These libraries are ASCII text files that can be entered or edited with a text editor such as Notepad or Write. They are written in a simple macro language. Once written, a Macro Library may be checked with VERIFY for correct syntax.

The third method for generating test data is to import an ASCII file. ASCII files must contain only the pin function characters (1, 0, X, H, L, S, R) and each sequence should be terminated with the line feed character (10 decimal). The text file may be the translated output of a digital simulator or it may be alogorithmically generated by a computer program. Tek Swave format can also be imported into EDIT.

Once the test data is in EDIT, it can be saved to a downloadable file. EDIT generates one master pattern file. It also generates one download file for each VX4820 module.

Modular Test Generation

MERGE supports modular test generation. Tests can be broken up into functional modules, each module being generated separately using the EDIT application. The beginning sequence of each test module can be selected to ensure it does not overlap with any other modules to be merged. The MERGE application can combine these separate modules into a single test file.

Downloading, Executing, and Debugging a Test

DEBUG allows the user to interactively execute and debug a test. It associates the download test files with specific modules in the VXI mainframe. Test files can be downloaded into the VX4820 and executed normally or in single step modes. Breakpoints can be set on specific test sequences which freeze the I/O pins on that sequence.

Tools are provided which give direct interaction with VX4820 modules in the mainframe. A talker/listener window allows the user to directly send commands to the VX4820. A trace window displays all communications with the VX4820, providing an online learning tool.

On-line Help

All of the Digital Toolbox applications provide online help. The help menus provide descriptions of the operations, procedures, and definitions needed to use each tool. **Quick help** is provided in EDIT and DEBUG to help the first time user get started.

EDIT

EDIT can be used to generate digital test vectors and save them in a downloadable file. The file can be loaded into the VX4820 Digital Test Module and executed by the DEBUG application or by the system controller during test execution. It also supports the import and export of ASCII files.

Starting the Editor

To start the EDIT application:

1. Open the Digital Toolbox window by double clicking its icon.
2. Double click the EDIT icon.

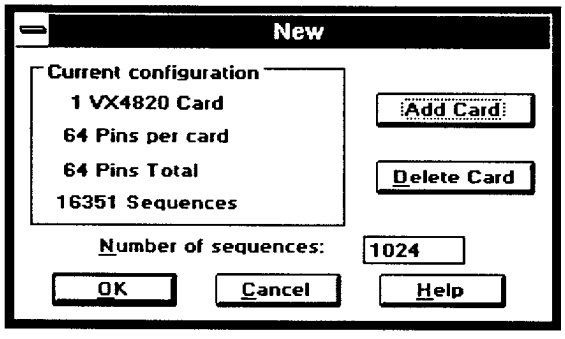
Selecting the Test Configuration

To build a new test pattern, the number of VX4820 modules in the test system must be configured.

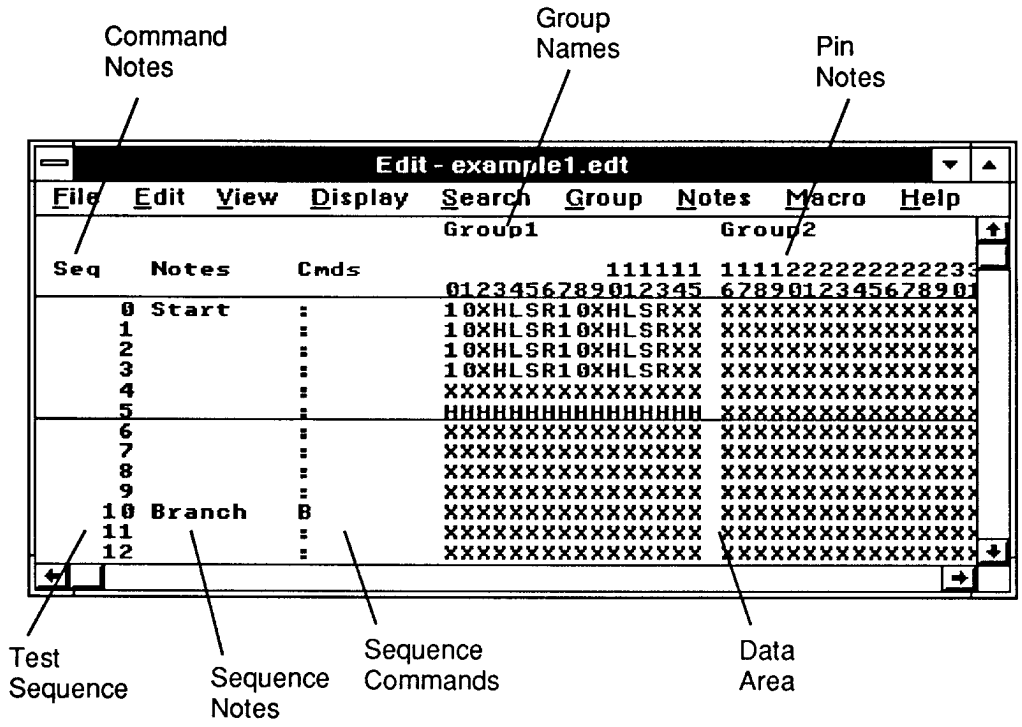
To enter the system configuration:

1. Pull down the **File** menu and select **New...** .
2. In the New dialog box select the number of modules to be included in the test system by clicking the **Add Card** and **Delete Card** buttons. The total number of pins is displayed in the **Current Configuration** information.
3. Select the number of sequences to edit in this file. Multiple files can be merged later to form a complete test.
4. Click **OK** when the selected configuration is correct.

The data area of the screen will be filled with the don't care symbol ('X') for as many pins as has been configured.



Display Areas



Display Formatting

Display Colors

The color of selected display areas and data characteristics may be changed by the user.

To change display colors:

1. Pull down the **Display** menu and select **Color...** .
2. In the Colors dialog box, chose an item from the **Selections** list.
3. Adjust the **Red**, **Green** and **Blue** contents of the item. The sample displays the selected colors.
4. Repeat the above steps for other items.
5. Click on **OK**.

The selected colors are saved in the DIGBOX.INI file. When you start the EDIT application the selected colors are automatically restored.

Grouping Pins

Pins can be grouped, separating them from other test data. Grouped pins may be assigned a name which is displayed above the pin data.

To define a group:

1. Click and drag horizontally across the pin notes area, selecting a group of adjacent pins.
2. Pull down the **Group** menu and select **Group**.
3. Type the group name in the Group dialog box. When displayed this name will be truncated to the number of columns used by the group.
4. Click the **OK** button.

A blank column will be placed between the grouped pins and other adjacent pins. The group name will apear above the group.

To break up a group:

1. Select the group by clicking on the group name.
2. Pull down the **Group** menu and select **Ungroup**.

Changing Group Names

To change a group name:

1. Select the group name by clicking on it.
2. Pull down the **Group** menu and select **Group**.
3. Edit the name in the Group dialog box and click **Ok**.

Hiding Pins or Groups

Pins and groups may be selectivly displayed or hidden.

To hide pins:

1. Click and drag horizonally across the pin notes area, selecting pins to be hidden.
2. Pull down the **View** menu and select **Suppress Pins**.

The selected pins will be removed from the display.

To hide a group:

1. Select the group by clicking on its name.
2. Pull down the **View** menu and select **Suppress Groups**.

Redisplaying Pins or Groups

To restore the display of pins and groups:

1. Pull down the **View** menu and select **Show pins** or **Show groups**.
2. Select the pins or groups to display by clicking on the pin number or group name. Any combination of pins or groups may be selected by holding down the shift key and dragging or clicking.
3. Click the **OK** button.

To restore all pins or groups select **Show all pins** or **Show all groups** in the View menu.

Pin Notes

Pin notes can be paced on each column of data. Either the pin reference number or note may be displayed.

To toggle between the reference number and the user defined pin notes:

Pull down the **Notes** menu and select **Pin notes**.

To define the pin label:

1. If the pin is in a group, set the group radix to binary.
2. Double-click on the pin notes area above the selected pin.
3. Type in a one to three character label for that pin in the **Edit Pin Note** dialog box.
4. Click the **OK** button

To edit a pin notes label double click on the note.

Sequence Notes

Sequence notes may be placed on each row of data. They can be hidden or enabled.

To toggle sequence notes between hidden and enabled:

Pull down the **Notes** menu and select **Seq notes**.

To add/edit sequence notes:

1. Double-click on the sequence notes area of the selected sequence.
2. Enter the sequence note in the **Edit Sequence Note** dialog box.
3. Click the **OK** button.

To edit a seqence note double-click on the note.

Marks

Vertical or horizontal marks (lines) can be added to the display to separate data. vertical marks within a group is displayed only when the group radix is binary.

To add a horizontal mark:

1. Click on the sequence which the mark is to underline.
2. Pull down the **Display** menu and select **Add display marker**.

To add a vertical mark:

1. Click on the pin note which the mark is to follow.
2. Pull down the **Display** menu and select **Add display marker**.

To remove a mark:

1. Click on the sequence which the mark underlines or the pin which preceeds the mark.
2. Pull down the **Display** menu and select **Remove display marker**

Group Radix

Groups can be displayed in binary, octal, or hexadecimal radix.

To select the group radix:

1. Select a group by clicking on its name.
2. Pull down the **Group** menu and select **Radix...** .
3. Click on the desired radix in the **Change Radix** dialog box.

Once displayed the **Change Radix** dialog box remains active. The radix of groups can then be switched by selecting the group name and clicking on a new radix. The dialog box can be moved to a convenient location on the display or removed by clicking **Cancel**.

Editing Test Data

Typing Data in Binary Radix

Data is entered into the test file by typing in characters which represent pin functions. A blicking underline cursor indicates the position at which data is overwritten. The cursor can be positioned by clicking within the data area, or by using the keyboard arrow keys.

The cursor normally moves one column to the right for each keystroke. When the cursor reaches the end of a line it wraps back to the begining of the following line. The editor can also be placed in vertical entry mode. After each keystroke the cursor will drop down one row and remain on the same column.

To switch between vertical entry and normal mode:

Pull down the **Edit** menu and select **Vertical entry**.

Binary Data Representation

CHAR	DRIVER	COMPARE
1	Drive the pin HI	Mask - no comparison
0	Drive the pin LO	Mask - no comparison
X	Inhibit - high impedance	Mask - no comparison
H	Inhibit - high impedance	Compare with HI
L	Inhibit - high impedance	Compare with LO
S	Drive the pin HI	Compare with HI
R	Drive the pin LO	Compare with LO

Typing Data in Octal and Hex Radix

Grouped pin data can be displayed in binary, octal, or hexadecimal radix. In octal and hex radix, three or four pins are combined into a single character. Because each pin has seven possible states (functions), a simple octal or hexadecimal data representation is not possible.

In octal and hex radix, pattern data is always displayed in a numeric format (0 ... F, X). Each digit represents the values of three or four pins, respecively. If all pins represented by the character are drive values (1, 0), the character is displayed in the selected drive color. If all pins represented by the character are compare values (H, L, S, R) it is displayed in the compare color. If the digit is mixed (it contains a combination of drive, compare, or Inhibit characters), it is displayed in grey. To edit grey characters, the group radix must be changed to binary. When the radix is octal or hexadecimal, inhibit pins are interpreted numerically as zero.

When data is typed in the octal or hex radix, the characters (0 ... F) are interpreted as drive values (1, 0). To directly enter compare values, hold the CNTL key as you type. The keystrokes will then be interpreted as compare values (H, L). To input inhibit, drive with compare characters (X, S, R) or mixed values, the radix must be changed to binary.

Viewing Test Data

The data area can be scrolled horizontally or vertically with the horizontal and vertical scroll bars. You can also jump directly to a particular test seqence or sequence note.

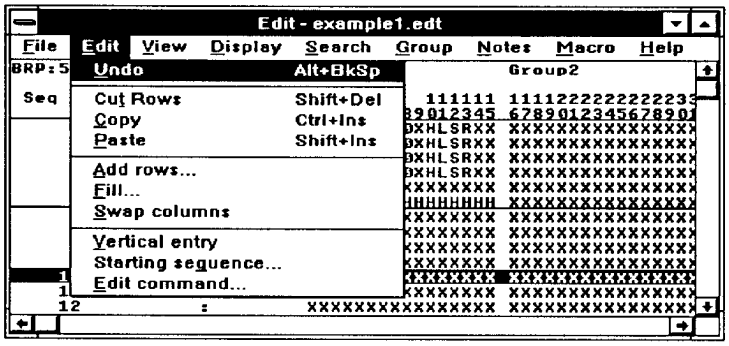
To jump to a sequence or note:

1. Pull down the **Search** menu and select **Go to...** .
2. In the **Go To** dialog box select the **Target** item. To go to a sequence type in the desired sequence value. To go to a **Command/Macro** select an item from the **Cmd/Macro** list box. To go to a **Seq Note** select an item from the **Seq Notes** list box.
3. Click on **OK**.

The selected item will be placed on the top line of the EDIT display.

Copy and Paste Data

You can copy blocks of data from one location to another using the copy and paste edit commands.



To copy a block of data:

1. Select the block you want to copy. Point to the first data character you want to select. Drag the insertion point to the last character you want to select. Release the mouse button.
2. Pull down the **Edit** menu and select **Copy**.
3. Move the data cursor to the point where the copied data is to be placed by clicking on that character.
4. Pull down the **Edit** menu and select **Paste**.

Cut, Copy, and Paste Test Sequences

You can cut, copy, and paste entire sequences.

To cut or copy and paste single or multiple test sequence steps:

1. Select the sequences to cut or copy. Point to the first sequence you want to select. Drag the insertion point to the last sequence you want to select. Release the mouse button.

2. Pull down the **Edit** menu and select **Cut** or **Copy**.

The cut or copied data may be pasted by selecting an insertion point and selecting **Paste**.

To add additional blank rows:

1. Select a sequence by clicking on a sequence number. The new rows will be added below this sequence.

2. Pull down the **Edit** menu and select **Add rows...** .

3. Enter the number of rows to be added in the **Add Rows** dialog box.

4. Click on **OK**. The rows are added after the insertion sequence.

Fill Data Pattern

The Fill option allows you to define a multi-line fill pattern which will be repeated over the selected range of test steps.

To Fill a range of sequence with a data pattern:

1. Select the sequences to fill. Point to the first sequence you want to select. Drag the insertion point to the last sequence you want to select. Release the mouse button.

2. Pull down the **Edit** menu and select **Fill...** .

3. In the **Fill With Pattern** dialog box **Add** or **Del** lines until the desired fill pattern length is achieved.

4. Edit the fill pattern using the **Valid Characters** displayed. Dot defines no change to existing data.

5. If the selected range of sequence is to be filled click on **Fill**. If the entire file is to be filled click on **Fill All**.

Swapping Pin Data

A specific relationship exists between the EDIT pin reference number and the physical pin on the VX4820 pods.. EDIT pin reference numbers are displayed when the pin notes are turned off. Pin reference number 0 is connected to pin 0 of pod 0 on module 0(data file *.ed0), reference 1 to pin 1 of pod 0, etc. In systems with multiple modules, this ordering repeats every 64 pins. Pin reference number 64 connects to pin 0 of pod 0 on module 2 (data file *.ed1).

The data associated with the physical pins may be rearanged by swapping.

To swap the data between two pins:

1. Select one of the two pins whose data is to be swapped.
2. Pull down the **Edit** menu and select **Swap columns**.
3. The cursor will change to a crosshair. Select the pin to swap with.

Find and Change

The Find and Change option allows you to search for multiline patterns and selectivly replace those patterns with a new one.

To search for a multi-line pattern:

1. Pull down the **Search** menu and select **Find...** .
2. In the **Find a Pattern** dialog box click **Add** and **Del** to display the desired number of lines for the search pattern.
3. Edit the search pattern using the **Valid Characters** displayed. Dot matches any digit.
4. Select a search range.
5. Click on **Find Next**.

The found pattern will be displayed on the top line of the data area. To remove the **Find a Pattern** dialog box click **Cancel**.

To find and change a multi-line pattern:

1. Pull down the **Search** menu and select **Change...** .
2. In the **Change a Pattern** dialog box click **Add** and **Del** to display the desired number of lines for the **Match** and **Change** pattern.
3. Edit the **Match** and **Change** pattern using the **Valid Characters** displayed. Dot matches any digit in the **Match**. In the **Change** dot leaves the digit unmodified.
4. To select the search range click **Range**.

5. Click on **Find** to go to the next match. Click on **Change** to modify the pattern match with the current find. To find the next match and change to the replace pattern click **Find+ Change**. To change all matching patterns within the search range click **Change All**.

To remove the **Change a Pattern** dialog box click **Cancel**

To jump directly to a sequence number, note, or command:

1. Pull down the **Search** menu and select **Goto...** .
2. Select the **Target** object type: **Sequence, Seq Note, Command/Macro**.
3. If the Target is a Sequence, type in the sequence number. Otherwise, select a target object from the appropriate list box.
4. Click **Ok**.

Selecting the Starting Sequence

Test files can be designed as a collection of test modules and later combined using MERGE. The range of test sequences within each module must not overlap. One test module might start at sequence 0, the next at sequence 100, etc.

To change the starting sequence of the test file:

1. Pull down the **Edit** menu and select **Starting sequence...** .
2. Fill in the new starting sequence in the **Set Starting Sequence** dialog box.
3. Click **Ok**.

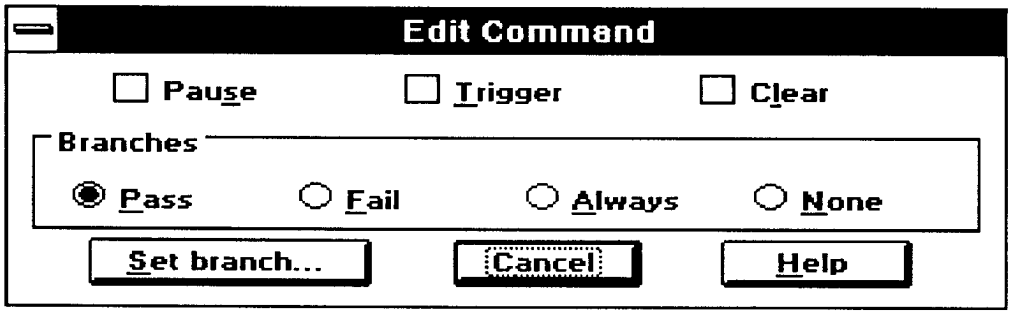
When an EDIT file contains branch instructions and the starting sequence is modified, branch destinations that exist within the current file are adjusted relative to the new starting sequence. The branch destination will maintain the same relationship it had with the rest of the file prior to the change. If the branch destination sequence is outside the range in the current file, it is left unmodified.

Sequence Commands

Sequence commands control functions that execute when the sequence is executed during the test. In the VX4820 these include PAUSE, BRANCH, CLEAR test state and TRIGGER out.

To place a sequence command:

1. Double-click on the **Cmds** area next to the desired sequence.
2. Click on the desired sequence command in the **Edit Command** dialog box.
3. If a branch was selected, enter the desired branch to sequence in the **Add/Change Branch Target** dialog box.



Branch commands on the VX4820 require that the previous seven test sequences be executed linearly.

Whenever a sequence is selected, the sequence commands set on that sequence are displayed in the sequence command area.

Test Files

Saving and Retrieving Test Files

Test patterns are saved to one or more test files.

To save a test to a new file:

1. Pull down the **File** menu and select **Save**
2. Select the test data rate and scale factor.
3. Select the starting and ending sequence for the test
4. Enter the file name.
4. Click on **OK**.

To retrieve a pattern file:

1 Pull down the **File** menu and select **Open**.

2. Type the file name or click on a file name in the select box. EDIT file names have a .EDT extension.
3. Click on **Ok**.

Printing Test Files

Test files or portions of the file can be printed.

To print a test file:

1. Pull down the **File** menu and select **Print...** .
2. Select the print range and click **Print**.

To print the currently displayed screen:

Pull down the **File** menu and click **Screen dump**.

Macros

Macros allow a complete I/O transaction to be defined and hidden behind a single display line. Parameters can be passed into the macro such as address or data values. Complete microprocessor read and write cycles may be defined.

Macros are defined by macro libraries. The macro library is an ASCII file with defined syntax. It can be input and edited with Windows Write or Notepad. The macro language defines the macro name, the number of sequence steps contained, which line is displayed and how the parameters are passed into the macro.

Using Macros

The macro library must be loaded into the editor before it can be used.

To load a macro library:

1. Pull down the **Macro** menu.
2. Select one of the displayed macro libraries.
3. If no other macro libraries have been loaded, or if the currently loaded libraries should be removed, select **Open** . All current references of macros within the test file and all prior macro definitions will be deleted. If the new macro library is to be added to the currently loaded and defined macros, slect **Append.**

When multiple macro librariy files are loaded into EDIT, each macro name must be unique throughout all files. Once the macro library has been loaded you may apply the macros by name.

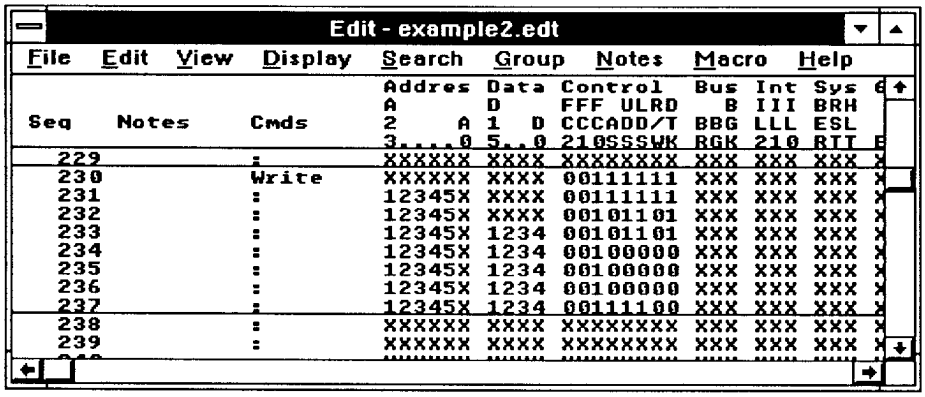
To apply a macro:

1. Select the sequence where the macro is to be applied by clicking on the sequence number.
2. Pull down the **Macro** menu and select **Apply/Remove marco...** .
3. In the **Apply/Remove Macro** dialog box select the desired macro name.
4. Select **Overlay** or **Insert**. Overlay overwrites existing data, insert adds new sequence steps.
5. Click on **Apply**.
6. In the **Enter Macro Parameters** dialog box edit the input parameter data.
7. Click on **OK**.

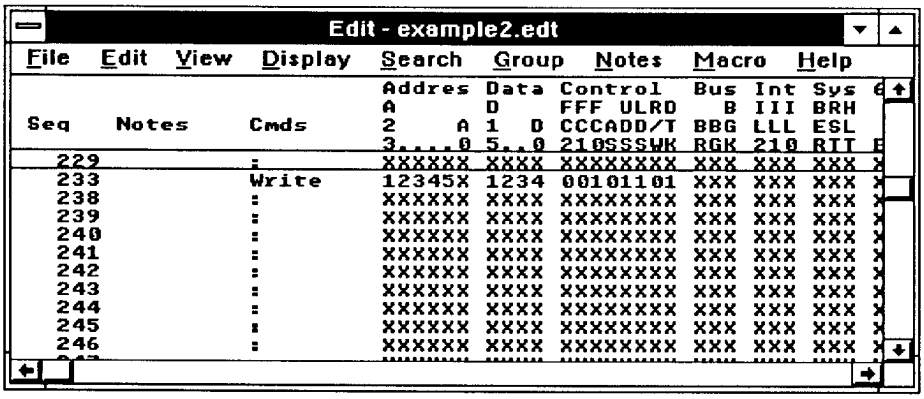
Macros can be hidden behind a single display line, or expanded to show all macro lines.

To toggle between these two display modes:

Pull down the **Display** menu and select **Expand macro cycles...** .



Expanded Macro



Hidden Macro

Defining Macros

Macro libraries are ASCII files which contain macro language commands. Windows Write can be used to edit these files. Its output file should be selected to be ASCII.

The language elements are listed below.

```
NAME <macro name>
```

The NAME command begins a macro definition and provides the name that is selected in EDIT. The macro name is a character string of 1 — 8 characters and must be unique within the library file. The end of a macro definition occurs when a NAME command starts a new macro definition or when the end of the file is encountered.

```
DISPLAYLINE <line>
```

The DISPLAYLINE command is optional. It defines which line of the macro will be displayed in the EDIT application when the macro is displayed as a single line. The sequence must be an integer between 0 and the number of lines defined within the macro (data lines only). If this command is omitted, the display line will be 0 (The first line of the macro).

```
DATA <pin data>,<pause>,<clear>,<trigger>,<branch>,
    <branch destination>
```

The DATA command defines a macro line that will be overlaid or inserted into the test data. The value passed is a string of pin data characters (1, 0, H, L, X, S, R) followed by 5 sequence command parameters. The parameters and the pin data must be comma separated. The parameters are:

pause ::= {ON | OFF},
clear ::= {ON | OFF},
trigger ::= {ON | OFF},
branch ::= {PASS | FAIL | ALWAYS | NONE},
branch destination ::= integer, 0 ... 16350

If the data provided is less than the number of pins in the editor the right most pins will be left un-modified.

Example: DATA XXXX1111SSSRRRHHHLLL,OFF,ON,OFF,NONE,0

```
PARAM <name>,<radix>,<location>
```

The PARAM command controls parameter substitution within the macro for a particular line. Any number of PARAM commands may be included within a macro definition.

The name parameter is required. It supplies the name of the parameter which is displayed in the EDIT application.

The radix parameter is required. It defines the radix that the parameter will be entered within the EDIT application. The radix can also be changed in EDIT. Radix values can be HEX, OCT or BIN. If compare values are to be entered the radix should be set to BIN.

The location values are required. They tell EDIT where to make parameter substitutions. The location values consist of a line number followed by a series of comma separated column values. The line number must exist within the macro (the first line is 0). The column numbers determine the placement of individual bits, passed in by the named parameter, within the selected line.

Parameter bits are sequentially placed, starting with the most significant parameter bit, proceeding to the least significant parameter bit. Column values can be a single integer, or an integer range separated by a dash. A negative column value causes input parameter bits to be discarded. The number of bits discarded is the absolute value of the negative column value.

In the example below, the passed in parameter is 15.

```
DATA XXXXXXXX,OFF,OFF,OFF,NONE,0
PARAM  DATA,HEX,0,0-3,-3,7
```

the resulting pattern is: 0001XXX1

`REM [string]`

The REM command defines a user defined remark. This can be any text useful for documenting your macro file. Edit ignores REM lines.

Example:

```
REM Last change date: April 12, 1991, MSH
```

Macro File Example

```
REM Macro Example
NAME  write
DISPLAYLINE 4
DATA  XXXXXXXXXXXXXXXXXXXXXXXX011,OFF,OFF,OFF,NONE,0
DATA  ........................................001,OFF,OFF,OFF,NONE,0
DATA  ........................................001,OFF,OFF,OFF,NONE,0
DATA  ........................................000,OFF,OFF,OFF,NONE,0
DATA  ........................................000,OFF,OFF,OFF,NONE,0
DATA  XXXXXXXXXXXXXXXXXXXXXXXX011,OFF,OFF,OFF,NONE,0
PARAM  ADDRESS,HEX,1,0-15
PARAM  ADDRESS,HEX,2,0-15
PARAM  ADDRESS,HEX,3,0-15
PARAM  ADDRESS,HEX,4,0-15
PARAM  DATA,HEX,1,16-23
PARAM  DATA,HEX,2,16-23
PARAM  DATA,HEX,3,16-23
PARAM  DATA,HEX,4,16-23
```

Import and Export of ASCII Files

ASCII data files can be loaded into the EDIT data area. EDIT can also store its test data in a DOS file as ASCII characters.

The output file will contain a string of characters (1, 0, H, L, X, S, R) ,one for each pin, followed by a Linefeed character (dec 10). The number of characters must exactly match the selected EDIT configuration (64 chars per module) and must only contain 1, 0, H, L, X, S, and R characters followed by a Linefeed character (dec 10).

To export a test pattern in ASCII from the EDIT application:

1. Pull down the **File** menu and select **Export...** .
2. Set the sequence range and enter the filename to write to in the **Export** dialog box.
3. Click on **OK**.

The file will be written and can be viewed and edited by a text editor such as Notepad.

To import a text file:

1. Pull down the **File** menu and select **Import...** .
2. Click the **ASCII** button and select a filename from the **Files** list box.
3. Click on **OK**.

Simulator data can be imported to EDIT by first translating the simulation file to a TekSwave file utilizing the Tektronix CADLink software product.

To import a TekSwave file:

1. Pull down the **File** menu and select **Import...** .
2. Click the **TekSwav** button and select a filename from the **Files** list box.
3. Select a file.
4. Click on **OK**.

VERIFY

Macro files can be generated by using any text editor that can produce an ASCII output format. Once the macro file has been typed in, it should be checked with VERIFY. All errors should be eliminated before the macro file is used within EDIT.

Starting Verify

To start the VERIFY application:

1. Open the Digital Toolbox window by double-clicking its icon.
2. Double-click the VERIFY icon.

Checking Macro Files

To check a macro file:

1. Pull down **File** and select **Open...**
2. Select a macro file and click **Ok**.

Verify will scan the macro file, checking it for correct structure and syntax. Any errors found will be listed in the Verify window. **Help** contains all rules and syntax requirements for macros.

To clear the error message list:

Pull down **File** and select **Clear screen**

MERGE

Merge supports modular generation of digital test files. It combines multiple, non-overlapping source files into a single ouput test file. The output file may be viewed by EDIT or downloaded into a VX4820 test system.

STARTING THE MERGE UTILITY

To start the MERGE application:

1. Open the Digital Toolbox window by double-clicking its icon.
2. Double-click the MERGE icon.

SELECTING INPUT FILES

Multiple .EDT source files may be combined into a single output .EDT file.

To select the input files:

1. Pull down **File** and select **Merge files...**
2. Select each file to merge and click **Add**. Repeat this step until all files have been added..
3. Once all source files have been successfully added click **Merge**.
4. Input a destination file name and select the test range.
5. Click **Ok**.

To view the source files sequence range and number of pins:

1. Pull down **File** and select **Merge files...**
2. Select a .EDT file.
3. Click **Info...**

Saving the Merge Configuration

The test file configuration can be saved and later recalled. This allows test files to easily be rebuilt after one of the source files has been modified.

To save the MERGE configuration:

1. Pull down File and select **Save merge definition as ...** .
2. Type in the file name and click **Ok**.

The output merge file does not have any pin groups. It will have the pin notes from the input file which contains the lowest sequence values.

DEBUG

DEBUG provides an interactive environment for downloading and debugging digital test files. Debug requires that a VX4820 be installed in the system. Debug should only be run on an appropriate embedded controller (i.e. VX4530/5 or VX5530/5).

Starting the Loader/Debugger

To start the Debug application:

1. Open the Digital Toolbox window by double-clicking its icon
2. Double-click the DEBUG icon.

Configuring the Test

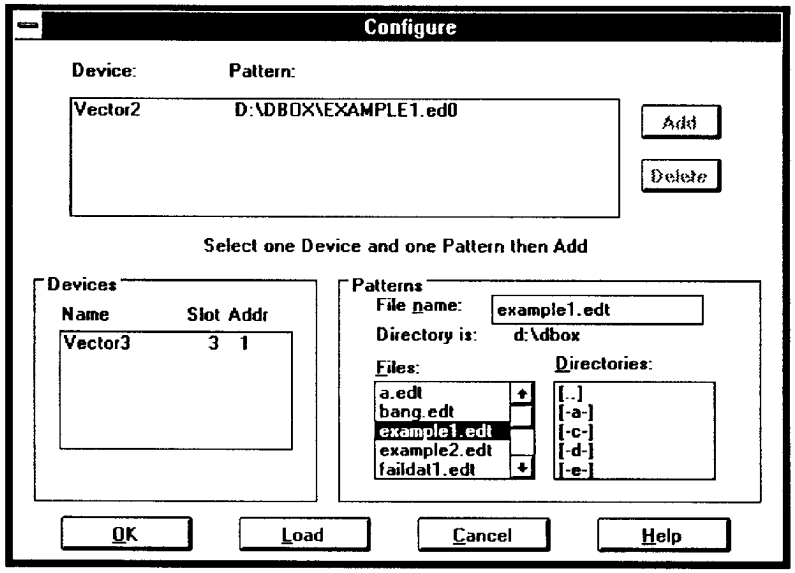
The first task that must be accomplished to utilize DEBUG is to select which test files will be downloaded and which VX4820 modules will be used in the test.

To select test files and VX4820 modules:

1. Pull down **File** and select **New...**
2. Select a module from the devices list and a file from the patterns list and click **Add**.
3. When all devices and files have been added click **Ok** or **Load** to immediately load the test file.

NOTE

Modules may be added to the configuration without an associated file.



Downloading / Uploading Test Data

Once configured, test pattern data can be loaded into the VX4820 module(s). This can be accomplished during configuration in the **Configure** dialog box or at any time it is desired to reload.

To load a test file into the VX4820 module:

Pull down **File** and select **Load**.

Test files can also be uploaded from the VX4820 module(s) to a test file. Either a single module or all of the configured modules are uploaded.

To upload a test data from all configured modules:

Pull down **File** and select **Store**.

To upload individual module test files:

1. Pull down **File** and select **Store module**.
2. Select the module to upload and click **Ok** .

Executing the Test

Once the Debug has been configured and the test file(s) loaded, the test can be executed. Two modes of execution are provided: Run and Debug. Run immediately executes the test. Debug provides interactive control of the test.

To immediately execute a loaded test:

Pull down **Run** and select **Run** or type **F4**.

Executing the test under debug provides more control over test execution. The starting sequence of the test can be modified, the test can be single-steped, and breakpoints can be set.

To execute in debug mode:

1. Pull down **Run** and select **Run debug...**
2. Select the starting test sequence, run mode and set any desired breakpoints. To remove breakpoints, backpace over the number.
3. Click **Run** to immediatly execute the test or **Ok** to return to the main screen.

If single step is selected, test execution will pause on each test step. If breakpoints are set, test execution will pause at each breakpoint.

To continue a paused test:

Pull down the **Run** menu and select **Continue** of type **F5**.

A test can be stopped at any time. When a test is stopped, by the STOP command or as a result of executing the SEQ:END sequence, the drive values on the pins will be that of the last test sequence executed. However, he pins can be forced to the INHIBIT state by selecting Panic stop.

To stop a running test:

Pull down the **Run** menu and select **Stop** or **Panic stop**, or type **F6**.

The Debug Display

The Debug display contains information about the state of the current test. This information is periodically updated when a test is executing. The instrument state can be Stopped, Armed, Running, or Paused. The Armed state is entered when the start command is received by the VX4820. If external clocks or external STST protocol have been programmed, the module will remain in the Armed state until the appropriate signals (clock or start) are received. If pause commands, break-points or single-step has been programmed, the Paused state will be entered.

The **Current seq** and **Total Vectors**, and **Pass/Fail** fields describe test progress. The Pass/Fail information takes seven clock cycles to propigate throughout the VX4820 module(s). This limits the Pass/Fail display to indicating correct status within the last 7 executed test sequences. For example, if a compare failure occurs on sequence 100, the Pass/Fail display would not reflect that failure until sequence 106 is executed. This becomes readily apparant when executing tests in single-step mode. When a test terminates by executing the SEQ:END sequence or is manually stoped, the Pass/Fail display is updated to the last executed test sequence.

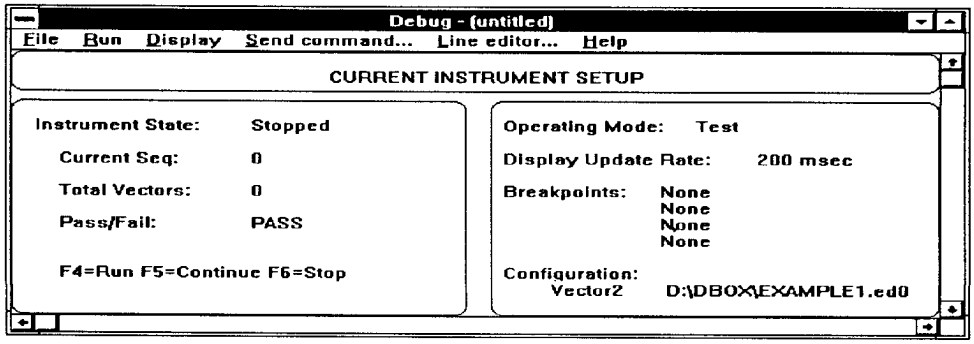
If a test failure occurs, information about the failure sequence and pin data can be retrieved. The Instrument state must be Stopped. Test failure information is available only for the first test failure.

To view test failure information:

Pull down the **Display** menu and select **Display fail ...** .

In the **Fail Data** dialog box, actual and expect data are displayed. The actual data is the actual value sampled on the pin during execution of the test. The expected data is the programmed test value for that test step. Where the actual data and expected data do not match (would cause a failed comparison), the actual data pin value is displayed in red.

The update rate of the test status can be changed to fit your needs. The continuous update only occurs when the test is not stopped.



To change the update rate:

1. Pull down **Display** and select **Update rate...**
2. Select **Continuous** or **On demand**.
3. If Continuous was selected, set the desired **Update rate:**
4. Click **Ok**.

The Operating mode of a test can be either Test or Learn. In Test mode, drive and compare functions test the UUT for expected operation. In Learn mode, the VX4820 drives values onto the test pins and captures the response from the UUT (see SECTION 1, Learn Mode).

To toggle between learn and test modes:

Pull down the **Run** menu and select **Learn**.

Saving the Debug Configuration

Once a test configuration has been establised, it can be saved and retrieved. The configuration includes the test files and module selections.

To save the current configuration:

Pull down **File** and select **Save** or **Save as...**

To retrieve a previously saved configuration:

1. Pull down **File** and select **Open...**
2. Select a file with the extension **.DBG** and click **Ok**

Trace Display

The trace display provides visibility of the communications between Debug and the VX4820 modules. All messages sent to and received from configured modules is displayed in the **Trace** window. Messages sent to the module are displayed in blue and messages received are displayed in red.

To start the trace display:

Pull down the **Display** menu and select **Trace...** .

To remove the trace window:

click the **Cancel** button in the Trace window.

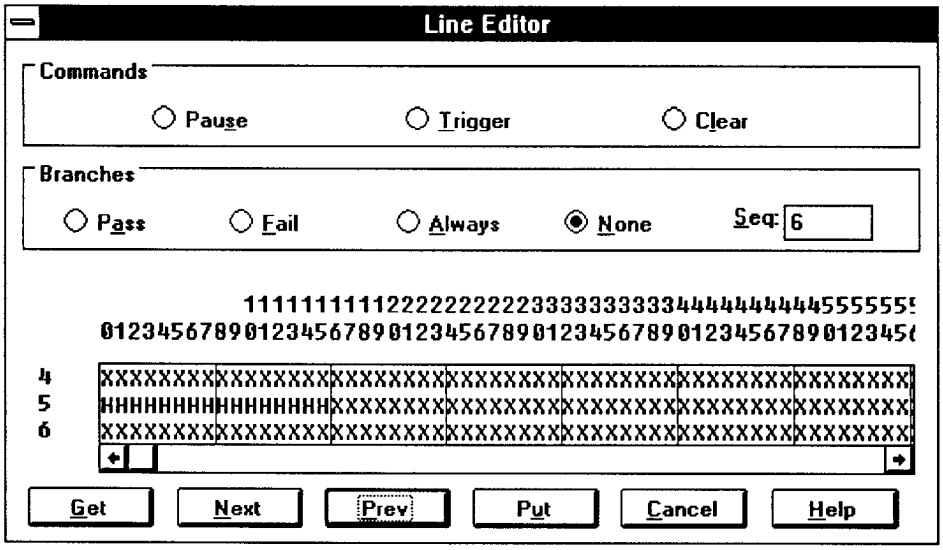
Sequence Line Editor

Debug provides an online editor that can be used to make minor changes to the digital test pattern. The editor operates directly on the pattern stored in the VX4820 module. The test sequence that is to be edited must first be retrieved from the module. It is then displayed, along with the prior and following line, and can be modified by the user. Once the displayed line is modified, *it must be put back into the VX4820* to have an affect on the executing test. Tests that have been modified with this editor can be uploaded to the origional test file after errors have been removed.

To edit a paticular test step:

1. Pull down **Line editor...**
2. Click on **Get**
3. Enter the desired sequence value and click **Ok**.
4. Edit the line.
5. Click on **Put** to load the change into the VX4820. If Put is not clicked and another line is requested or the editor is exited, the changes made will not be reflected in the loaded test pattern.
6. Edit other lines by repeating Steps 2 — 5 or by clicking **Next** or **Prev** buttons.

The line editor displays any commands or branches set on the currently displayed sequence. These can be modified by clicking the associated button. Care should be taken when setting branches not to violate any of the branching rules (see *Section 1*).



Talker / Listener

Debug provides an online talker/listener window that allows commands to be sent to any configured VX4820.

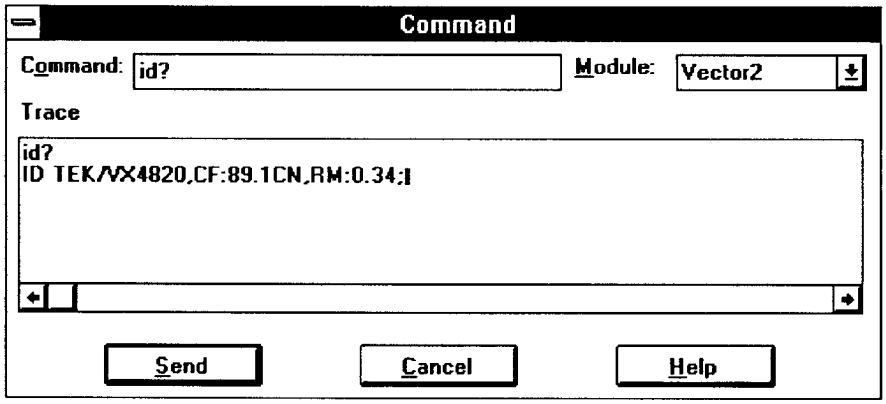
To bring up the talker/listener window:

1. Pull down **Send command...**

2. Select a VX4820 module from the **Module** list.

To send commands or queries to the VX4820, type the correct command syntax into the **Command** text box. Then click on Send, or press the Enter key. Commands sent to the VX4820 are displayed in blue. If the command ends in a '?' key, the command response will automatically be retrieved and displayed in red.

If there is an error in command syntax, the VX4820 will log the error. Error messages can be retrieved with the EVMSG? command. If the sytax error occured on a query, no response will be generated and the talker / listener will time out with the error: "Time out: No response from VX4820".



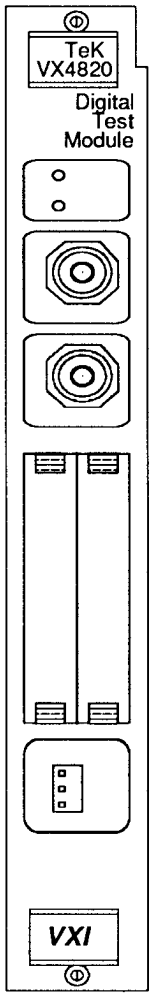
Warning

The following servicing instructions are for use only by qualified personnel. To avoid personnel injury, do not perform any servicing other than that contained in the operating instructions unless you are qualified to do so. Refer to General Safety Summary and Service Safety Summary prior to performing any service.

Section 9
Module Service

General Information

The VX4820 Digital Test Module provides digital stimulus and response compare for automated test of digital circuits and assemblies. One or two UUT interface pods are attached to front panel connectors which provide high fidelity digital I/O pin electronics.



VX4820

Standard Accessories

TTL I/O Pod(s) with 1.8-meter cable

VX4820 Users Manual

Digital Toolbox Software

Service Safety Summary

Only qualified personnel should perform service procedures. This Service Safety Summary and the General Safety Summary should be read before performing service procedures.

Do Not Service Alone

Do not perform internal service or adjustment of this product unless another person capable of rendering first aid and resuscitation is present.

Use Care when Servicing with Power On

To avoid personal injury from high current, remove jewelry such as rings, watches, and other metallic objects, before servicing the instrument. Do not touch exposed connections and components while power is on. Disconnect power before soldering, removing protective panels, or replacing components.

Performance Check

Equipment List

Oscilloscope (Tek 2465)

Dual Channel
BW > = 250 MHz

Pulse Generator (Tek PG502)

20 MHz repetition rate
TTL output levels (V_{OL} < .8V, V_{OH} > 2.4V)

VXI System (Tek VX1405, VX4530, TekTMS)

C or D size mainframe
VXI slot 0 module
Controller with talker/listener capability

Power Supply

0 — 3V capable of sinking and sourcing current
(alternate method if sink/source current power
supply is not available requires a 2V power supply)

Test Procedure Descriptions

NOTE: It is assumed that this procedure is executed from beginning to end.

A. Check for proper power-up and diagnostic execution

A.1 Ensure that pod1 (and pod 2, if provided) are correctly connected to the VX4820 front panel.

A.2 Power-up the VX4820. Observe the READY and ACCESSED LEDs. The ACCESSED LED should flash during system power-up as a result of the execution of the VXI Resource Manager. The READY LED should illuminate within 5 seconds of power-up. This indicates that power-up diagnostics found no problems.

A.3 Execute the runtime extended diagnostics. Type the following commands:

TST?

Check for the response: **0;** .

A.4 Check pod LED operation. Send the following command:

```
IDPOD 0,OFF
IDPOD 0,ON
IDPOD 1,OFF
IDPOD 1,ON
```

As these commands are typed, check that the pod 0 and pod 1 LEDs illuminate and extinguish appropriately.

A.5 Check for correct pod ID response. Send the following commands:

```
PODTYPE? 0
```

Check for response: **PODTYPE 289;** if the pod is installed or **-1** if not.

Send the commands:

```
PODTYPE? 1
```

Check for response: **PODTYPE 289;** if the pod is installed or **-1** if not.

B. Check operation of the pod I/O pins for ability to drive and compare and generation of internal clocks.

B.1 Send the command:

```
PINTEST? 0
```

Check the response: **PINTEST PASS,#HFFFFFFFE,#HFFFFFFFE;** .

Send the command:

```
PINTEST? 1
```

Check the response: **PINTEST PASS,#HFFFFFFFE,#HFFFFFFFE;** .

B.2 Type the following commands:

```
INIT
NEW
TSTPAT
SEQ:START 0
SEQ:END 8
INTCLKRATE 50
ARM
START
```

B.3 Using the oscilloscope with a ≥ 1 MΩ probe, view each of the data I/O pins on each pod. Make sure to use the adjacent pod ground pin to correctly ground the scope probe. Check that a square wave exists with a 50 ns high level and a 350 ns low level. Check that the low level is less than +0.5 volts and the high level is greater than +2.4 volts.

B.4 Using the oscilloscope with a ≥ 1 MΩ probe, view the strobe out on each pod. Check that the low level is less than +0.5 volts and the high level is greater than +2.4V. Check that the output period is 50 ns.

B.5 Using the oscilloscope with a ≥ 1 MΩ probe, view the front panel CAL OUT BNC. Check that the low level is less than +0.8 volts and the high level is greater than +2.4V. Check that the output period is 50 ns.

B.6 Type the following commands:

```
STOP
SEQ:START 10
SEQ:END 18
ARM
START
```

B.7 Using the oscilloscope, view the pin 0 of pod 0 rising edge on Channel 1 and the pin 1 of pod 0 rising edge on Channel 2. Use Channel 1 as the trigger source. Measure the delay from the Channel 1 trace to the Channel 2 trace at 2V. Record this value. (If the Channel 2 edge is prior to the Channel 1 edge, the value will be negative.)

B.8 Move the Channel 2 oscilloscope probe to the next I/O pin on the pod. Repeat the delay measurement in **B.7** and record the value.

B.9 Repeat **B.8** for the remaining pins on pod 0 (and pod 1 if present).

B.10 Determine the high and low delay values. (The pin 0 of pod 0 has a value of 0 delay.) The skew is the high value minus the low value, which should be ≤ 10 ns.

B.11 Change the oscilloscope slope to negative edge triggered and repeat Step **B.7** — **B.10** to calculate the falling edge skew.

B.12 Type the following commands:

```
STOP
INHIBIT
```

B.13 Adjust the power supply to 0.8V. Connect all the pod data I/O pins 3 — 31 for both pods (if present) to the power supply. (Connect the power supply negative (-) to the pod ground and the power supply positive (+) to the pod I/O pins.) (The power supply will need to sink current for this measurement. If the power supply is not capable of sinking current, connect the data I/O pins to ground.)

B.14 Type the following commands:

```
SEQ:START 20
SEQ:END 27
ARM
START
```

B.15 Send the following command:

```
STATE?
```

Check the response for **STATE STOPPED,27,8,PASS,PASS;**

B.16 Adjust the power supply to 2.0V. (If, in Step **B.12**, the pins were connected to ground, disconnect the ground connection and connect all of the I/O pins 0 — 31 to the power supply.)

B.17 Type the following commands:

```
SEQ:START 30
SEQ:END 37
ARM
START
```

B.18 Send the following command:

```
STATE?
```

Check the response for **STATE STOPPED,37,8,PASS,PASS;**

B.19 Type the following commands:

```
INTCLKRATE 3000000
ARM
START
```

With the oscilloscope, check the CAL OUT BNC for a square wave output with a period of 3.0 ms.

C. Check operation from external clocks.

C.1 Connect one end of a 50 ohm coax cable to the pulse generator. Connect the opposite end of the cable to a 50 ohm terminator. Using the oscilloscope, adjust the period of the pulse generator to 50 ns with a 50% duty cycle. Set the low output level to +0.8V and the high output level to +2.0V.

C.2 Connect the terminated end of the 50 ohm cable to the VX4820 CLK IN BNC.

C.3 Send the following commands:

```
STOP
CONNECT:CLOCK  CLKBNC
ARM
START
```

C.4 Using the oscilloscope, view pin 0 of pod 0 and pin 0 of pod 1. Make sure to use the adjacent pod ground pin to correctly ground the scope probe. Check that a square wave exists with a 50 ns high level and a 350 ns low level.

C.5 Send the following commands:

```
STOP
```

C.6 Remove the terminated coax cable from the VX4820 front panel CLK IN BNC. Connect the center of this terminated cable to the CLK IN pin on pod 0. Connect the shield of the cable to an adjacent pod ground pin.

C.7 Send the following commands:

```
CONNECT:CLOCK  EXTPOD0
ARM
START
```

C.8 Using the oscilloscope, view pin 0 of pod 0 and pin 0 of pod 1. Make sure to use the adjacent pod ground pin to correctly ground the scope probe. Check that a square wave exists with a 50 ns high level and a 350 ns low level.

C.9 If pod 1 does not exist, skip steps C.10 through C.12

C.10 Send the following commands:

```
STOP
```

Connect the center of the terminated end of the 50 ohm cable to the CLK IN pin on pod 1. Connect the shield of the cable to an adjacent pod ground pin.

C.11 Send the following commands:

```
CONNECT:CLOCK  EXTPOD1
ARM
START
```

C.12 Using the oscilloscope, view pin 0 of pod 0 and pin 0 of pod 1. Make sure to use the adjacent pod ground pin to correctly ground the scope probe. Check that a square wave exists with a 50 ns high level and a 350 ns low level.

D. Check the VX4820 fail register and learn mode.

D.1 Send the following commands:

```
STOP
INIT
NEW
SEQ:VECTOR "0X",0
SEQ:VECTOR "1X",1
SEQ:VECTOR "0X",2
SEQ:VECTOR "1X",3
SEQ:START 0
SEQ:END 3
```

D.2 Connect I/O pin 0 of pod 0 to pin 1 of pod 0.

D.3 Send the following commands.

```
MODE LEARN
ARM
START
STATE?
```

Check the response: **STATE 3,4,PASS,PASS;**

D.4 Send the following command:

```
SEQ:VECTOR? 0
```

Check the response the last two characters in the response data is **0L**.

D.5 Send the following command:

```
SEQ:VECTOR? 1
```

Check the response the last two characters in the response data is **1H**.

D.6 Type the following commands:

```
MODE TEST
SEQ:VECTOR "0H",1
ARM
START
STATE?
FAILDATA?
```

Check that the STATE? command returns **STATE STOPPED,3,4,FAIL, FAIL;**

Check that the seq parameter is **1 (FAILDATA 1,...)** and that the last two characters returned in the fail data are **00**.

Troubleshooting

When the VX4820 is first powered, power-up diagnostics are executed to verify module functionality. These diagnostics execute in less than 5 seconds. If the diagnostics find no errors, the green READY LED will be illuminated. If a failure is detected the LED will remain off and the VME SYSFAIL line will continue to be asserted. The VXI Resource Manager can detect the SYSFAIL assertion and read the fail bit in the VX4820 registers. If the READY LED illuminates, the VX4820 has passed its diagnostics and is waiting for instructions from its commander.

Failure of the READY LED to illuminate can also be caused by a mainframe failure such as a +5V supply fault or failure to provide the required backplane clocks. Usually other modules within the mainframe will exhibit problems also if this is the case.

The VX4820 contains fuses on each power supply provided by the mainframe or user. These fuses are listed in the Switch and Jumper Settings within this section. These fuses are soldered into the ECB. Repair of blown fuses should be referred to qualified service personnel.

The Pod I/O pins may be exposed to damaging potentials by a failed UUT. If after a test is executed a failed UUT is detected (and a UUT failure could cause damage to the pins), it is recommended that the PINTEST command be executed after the failure. The PINTEST command exercises all I/O pins within a pod to verify their functionality. No UUT should be attached to the tester when PINTEST is executed as all pins will be driven HI and LO.

Diagnostics

When the module is first powered, power-up diagnostics are executed. Below is a list of circuits tested:

CPU RAM
NVRAM
VXI REGISTERS
INTERNAL CLOCKS
SEQUENCE GENERATOR
POD0 (limited)
POD1 (limited)

Some of the power-up diagnostics are limited in their coverage because they must finish in less than 5 seconds. Better coverage is obtained when the diagnostics are executed at runtime. Runtime diagnostics are executed when the TST? command is received.

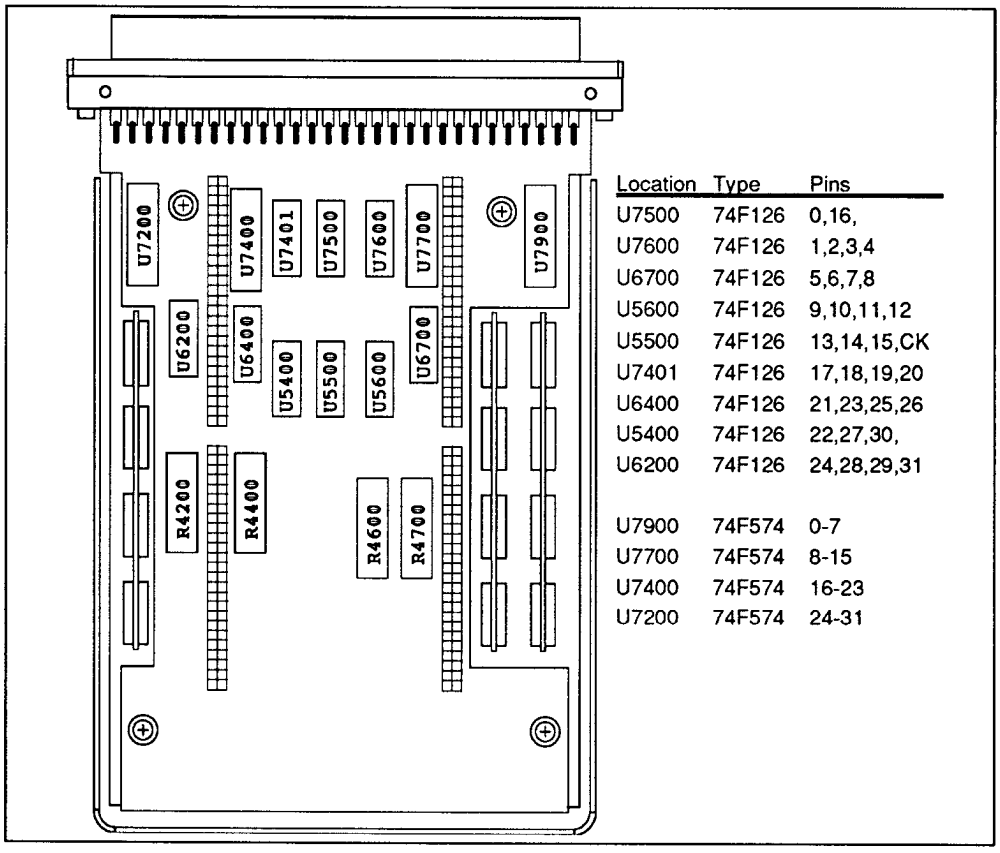
When power-up or runtime diagnostics are executed, the I/O pins will be driven HI and LO to verify their functionality. Care should be taken to ensure that no UUT is connected to the pins and that the pins are not held LO or HI by test fixture circuits.

Replacing Pin Driver and Receiver ICs

The VX4820 TTL Pod contains the pin driver and receiver ICs. If these ICs are exposed to excessive potentials as the result of testing a failed UUT, these ICs may be damaged and fail to operate correctly. If UUT failures can potentially damage the pin driver/receiver, the PINTEST command should be executed after each UUT failure to verify correct operation.

The pod driver and receiver ICs are easily replaceable. It is recommended that all the ICs be replaced as a set. The Pod should be removed from the test system and delivered to a qualified repair station. To replace the driver/receiver ICs:

1. Remove the 7 pod cover screws.
2. Remove the Pod cover.
3. Remove and replace the damaged driver/receiver ICs.
4. Replace the Pod cover.
5. Replace the 7 cover screws.
6. Connect the pod to the VX4820 module and execute PINTEST.



Location	Type	Pins
U7500	74F126	0,16,
U7600	74F126	1,2,3,4
U6700	74F126	5,6,7,8
U5600	74F126	9,10,11,12
U5500	74F126	13,14,15,CK
U7401	74F126	17,18,19,20
U6400	74F126	21,23,25,26
U5400	74F126	22,27,30,
U6200	74F126	24,28,29,31
U7900	74F574	0-7
U7700	74F574	8-15
U7400	74F574	16-23
U7200	74F574	24-31

Driver/Receiver IC Locations

Module Removal/Replacement Procedure

Removal

Make sure power to the VXI mainframe is turned off. Loosen the captive screws at the top and bottom of the VX4820 front panel. Simultaneously push both the top and bottom ejector assemblies to the eject position (top assembly is pushed up, bottom assembly is pushed down). Once the VX4820 is disengaged from the VXI backplane connectors, it can be removed from the mainframe.

Replacement

Make sure the power to the VXI mainframe is turned off. Align the VX4820 with the upper and lower card cage guides, and slide the module in until resistance is felt. Firmly push on the top and bottom of the front panel to seat the module into the VXI backplane connectors. The front panel should be flush with the mainframe front panel when the VX4820 is fully installed. Tighten the captive screws at the top and bottom of the VX4820 front panel.

Preparing the Module for Exchange

The VX4820 module is exchanged complete.

Contact your local Tektronix, Inc. Field Office for ordering and shipping instructions for the exchange module.

Switch and Jumper Settings

Below is shown the factory switch, jumper, and fuse configuration.

Switch/Jumper	Name	Default Setting
S7100	Logical Address Switch	All OPEN (Autoconfigure)
J9700, J9800	Anchor/Expander Protocol Select	EXP
J5900, J5902	Auxiliary POD power select	INT
F9500	+5V fuse	Installed
F9602	-5.2V fuse	Installed
F9600	-2V fuse	Installed
F5900	-24V fuse	Installed
F6900	+24V fuse	Installed
F9400	-12V fuse	Installed (B010136 & up)

Replaceable Parts

Replacement parts are available from or through your local Tektronix, Inc. Field Office or representative.

Changes to Tektronix instruments are sometimes made to accommodate improved components as they become available. To give you the benefit of the latest circuit improvements developed in our engineering department, it is important, when ordering parts, to include the following information in your order: part number, instrument type or number, serial number, and modification number if applicable.

If a part you have ordered has been replaced with a new or improved part, your local Tektronix, Inc. Field Office or representative will contact you concerning any change in part number.

Replaceable Parts List, Module

Tektronix Part Number	Qty.	Name and Description
333-3953-00	1	Panel, Front, VX4820
367-0411-00	1	Handle, Ejector, Top, with Hardware
367-0410-00	1	Handle, Ejector, Bottom, with hardware
334-8179-00	1	Marker, Identification, Tek VX4820
334-7519-00	1	Label, VXI, Handle, Ejector
213-1036-00	2	Nut Block, m2.5 x 10
213-1040-00	2	Screw, Machine, m2.5 x 10
386-6261-00	1	Key Plate, ECL CLASS, Top
211-0101-00	2	Screw, Key Plate
166-0670-00	2	Sleeve, Plastic, Grey, 0.23 in., Insulating
213-1035-00	2	Screw, Collar, m2.5 x 11
200-3958-00	1	Cover, Left Side
200-3957-00	1	Cover, Right Side
334-8192-00	1	Marker, Ident
129-1398-00	7	Standoff
211-0105-00	13	Screw, Cover
210-1307-00	7	Lock Washer
210-0845-00	2	Washer, BNC
220-0497-00	2	Nut, BNC
671-1985-00	1	Circuit Board Assembly, Module; B010100—B010135
671-2435-00	1	Circuit Board Assembly, Module; B010136—
159-0146-00	3	Fuse, 7A 125V Fast; B010100—B010135
159-0146-00	4	Fuse, 7A 125V Fast; B010136—
159-0203-00	2	Fuse, 2A 125V Fast

Replaceable Parts List, Pod

Tektronix Part Number	Qty.	Name and Description
200-3959-00	1	Cover, Right
200-3960-00	1	Cover, Left
334-8193-00	1	Marker Ident, Pod
337-3781-00	1	Insulator Plate
129-1399-00	4	Standoff
210-1307-00	4	Lock Washer
211-0408-00	4	Screw, ECB
211-0105-00	7	Screw, Cover
671-1983-00	1	Circuit Board Assy, Pod Control
671-1984-00	1	Circuit Board Assy, UUT I/F
156-3252-00	9	IC, 74F126, Pin Driver
156-2953-00	4	IC, 74F574, Pin Receiver
175-4066-02	1	Cable

Appendix A
ASCII/GPIB Code Chart

B7 B6 B5 / BITS / B4 B3 B2 B1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		NUMBERS SYMBOLS		UPPER CASE		LOWER CASE	
0 0 0 0	0 NUL 0	20 DLE 16	40 SP 0	60 0 16	100 @ 0	120 P 16	140 ` 0	160 p 16
	0 0	10 16	20 32	30 48	40 64	50 80	60 96	70 112
0 0 0 1	1 GTL SOH 1	21 LLO DC1 17	41 ! 1	61 1 17	101 A 1	121 Q 17	141 a 1	161 q 17
	1 1	11 17	21 33	31 49	41 65	51 81	61 97	71 113
0 0 1 0	2 STX 2	22 DC2 18	42 " 2	62 2 18	102 B 2	122 R 18	142 b 2	162 r 18
	2 2	12 18	22 34	32 50	42 66	52 82	62 98	72 114
0 0 1 1	3 ETX 3	23 DC3 19	43 # 3	63 3 19	103 C 3	123 S 19	143 c 3	163 s 19
	3 3	13 19	23 35	33 51	43 67	53 83	63 99	73 115
0 1 0 0	4 SDC EOT 4	24 DCL DC4 20	44 $ 4	64 4 20	104 D 4	124 T 20	144 d 4	164 t 20
	4 4	14 20	24 36	34 52	44 68	54 84	64 100	74 116
0 1 0 1	5 PPC ENQ 5	25 PPU NAK 21	45 % 5	65 5 21	105 E 5	125 U 21	145 e 5	165 u 21
	5 5	15 21	25 37	35 53	45 69	55 85	65 101	75 117
0 1 1 0	6 ACK 6	26 SYN 22	46 & 6	66 6 22	106 F 6	126 V 22	146 f 6	166 v 22
	6 6	16 22	26 38	36 54	46 70	56 86	66 102	76 118
0 1 1 1	7 BEL 7	27 ETB 23	47 ' 7	67 7 23	107 G 7	127 W 23	147 g 7	167 w 23
	7 7	17 23	27 39	37 55	47 71	57 87	67 103	77 119
1 0 0 0	10 GET BS 8	30 SPE CAN 24	50 (8	70 8 24	110 H 8	130 X 24	150 h 8	170 x 24
	8 8	18 24	28 40	38 56	48 72	58 88	68 104	78 120
1 0 0 1	11 TCT HT 9	31 SPD EM 25	51) 9	71 9 25	111 I 9	131 Y 25	151 i 9	171 y 25
	9 9	19 25	29 41	39 57	49 73	59 89	69 105	79 121
1 0 1 0	12 LF 10	32 SUB 26	52 * 10	72 : 26	112 J 10	132 Z 26	152 j 10	172 z 26
	A 10	1A 26	2A 42	3A 58	4A 74	5A 90	6A 106	7A 122
1 0 1 1	13 VT 11	33 ESC 27	53 + 11	73 ; 27	113 K 11	133 [27	153 k 11	173 { 27
	B 11	1B 27	2B 43	3B 59	4B 75	5B 91	6B 107	7B 123
1 1 0 0	14 FF 12	34 FS 28	54 , 12	74 < 28	114 L 12	134 \ 28	154 l 12	174 | 28
	C 12	1C 28	2C 44	3C 60	4C 76	5C 92	6C 108	7C 124
1 1 0 1	15 CR 13	35 GS 29	55 - 13	75 = 29	115 M 13	135] 29	155 m 13	175 } 29
	D 13	1D 29	2D 45	3D 61	4D 77	5D 93	6D 109	7D 125
1 1 1 0	16 SO 14	36 RS 30	56 . 14	76 > 30	116 N 14	136 ^ 30	156 n 14	176 ~ 30
	E 14	1E 30	2E 46	3E 62	4E 78	5E 94	6E 110	7E 126
1 1 1 1	17 SI 15	37 US 31	57 / 15	77 ? UNL 63	117 O 15	137 — UNT 95	157 o 15	177 DEL (RUBOUT)
	F 15	1F 31	2F 47	3F 63	4F 79	5F	6F 111	7F 127
	ADDRESSED COMMANDS	UNIVERSAL COMMANDS	LISTEN ADDRESSES		TALK ADDRESSES		SECONDARY ADDRESSES OR COMMANDS (PPE)	(PPD)

KEY

octal 25 PPU — GPIB code
NAK — ASCII character
hex 15 21 — decimal

Tektronix
COMMITTED TO EXCELLENCE

REF: ANSI STD X3. 4-1977
IEEE STD 488-1978
ISO STD 646-1973

Appendix B
VXIbus Glossary

This glossary contains terms defined according to their use in the VXIbus System. Although some terms may be used in other systems with different meanings, it is important to apply them only to VXIbus applications. In any instance in which a term is applicable only to a particular instrument module, that fact is so noted.

ACCESSED Indicator

An amber LED indicator that illuminates when the module identity is selected by the Resource Manager module, and flashes during any I/O operation for the module.

Backplane

The printed circuit board mounted in a VXIbus Mainframe to provide the interface between VXIbus modules, and between the Mainframe power supplies and the modules.

CLK10

A 10 MHz, individually buffered, differential ECL system clock sourced from Slot 0 and distributed to each slot (1 — 12) through P2. It is distributed to each module as as a single source, single destination signal with a matched delay of under eight (8) nanoseconds total.

Commander

A device controlling another device (a servant). A commander may be a servant of another commander.

Command

Any communication from a commander to a message based servant, consisting of a write to the servant's Data Low register, possibly preceded by a write to the data register.

Communication Registers

A set of device registers accessible to the commander of the device. Such registers are used for inter-device communications, and are required on all VXIbus message-based devices.

Configuration Registers

A set of registers allowing the system to identify a (module) device type, model, manufacturer, address space, and memory requirements. In order to support automatic system and memory configuration, the VXIbus Specification requires all VXIbus devices have a set of such registers, all accessible from P1 on the VXIbus.

C-Size Card

A VXIbus instrument module 340 x 233.4 x 30.48 mm (13.4 x 9.2 x 1.2").

D-Size Card

A VXIbus instrument module 340 x 366.7 x 30.48 mm (13.4 x 14.4 x 1.2").

ECLTRG

Six single-ended ECL trigger lines (two on P2 and four on P3) functioning as inter-module timing resources, and bussed across the VXIbus subsystem backplane. Any module, including the Slot 0 module, may drive and receive information from these lines. These lines have an impedance of 50 Ω and the asserted state is logical High.

Embedded Address

An address in a communications protocol in which the destination of the message is included in the message.

ESTST

Extended STart/STop protocol; used to synchronize VXIbus modules.

Extended Self Test

Any self test or diagnostic power-up routine executing after the initial kernel self-test program.

External System Controller

The host computer or other external controller exerting overall control of VXIbus operations.

Instrument Module

A plug-in printed circuit board, with associated components and shields, that can be installed in a VXIbus mainframe. An instrument module can contain more than one device, and/or can occupy more than one mainframe slot.

Interface Device

A VXIbus device providing an interface to external equipment.

Local Bus

A daisy-chained bus connecting adjacent VXIbus slots.

Logical Address

An 8-bit number uniquely identifying each VXIbus Device in a system. It defines a device's A16 register address, and indicates Commander/Servant relationships.

Mainframe

A rigid framework providing mechanical support for modules inserted into a VXIbus backplane. It ensures connectors mate properly, adjacent modules do not contact each other, and modules do not disengage from the backplane due to vibration or shock. It also may provide mechanical support and housing for power supplies and their distribution wiring to the backplane.

Message

A series of data bytes treated as a single communication element, with a well defined message body and terminator.

Message Based Device

A VXIbus device supporting VXI configuration and communication registers. Such devices support the Word Serial Protocol, and possibly other message-based protocols.

MODID Lines

Module/system identity lines.

P1

The top-most backplane connector for a given module slot in a vertical Mainframe, such as the Tektronix VX1500. The left-most backplane connector for a given slot in a horizontal Mainframe.

P2

The middle backplane connector for a given slot in a Mainframe.

P3

The bottom backplane connector for a given module slot in a vertical Mainframe, such as the Tektronix VX1500. The right-most backplane connector for a given slot in a horizontal Mainframe.

Query

A form of command allowing for inquiry to obtain status or data.

READY Indicator

A green LED indicator that illuminates when the power-up diagnostic routines have been successfully completed. An internal failure or failure of +5V power will extinguish this indicator.

Register Based Device

A VXIbus device supporting VXI register maps, but not high level VXIbus communication protocols.

Resource Manager

A VXIbus device providing configuration management services such as address map configuration, determining system hierarchy, allocating shared system resources, performing system self-test diagnostics, and initializing system commanders.

Self Test

A set of routines testing the operational functionality of the the instrument module. These routines are performed on power-up, and on command.

Servant

A device controlled by a Commander. There are message-based and register-based servants.

Slot 0 Controller

See Slot 0 Module. Also see Resource Manager.

Slot 0 Module

A VXIbus device providing the minimum VXIbus Slot 0 services to Slots 1 — 12 (CLK10 and the MODID module identity lines), but that can also provide other VXIbus services such as CLK100, SYNC100, STARBUS, and trigger control.

STST

STart/STop protocol; used to synchronize modules.

Synchronous Communications

A communications system following the "command-response" cycle model. In this model, a device issues a command to another device; the second device executes the command; then returns a response. Synchronous commands are executed in the order received.

SYSFAIL*

A signal line on the VMEbus used to indicate a failure by a device. The device failing asserts this line.

TTLTRG

Open collector TTL lines used for inter-module timing and communication.

VXIbus Subsystem

One Mainframe with modules installed. The installed modules include one module in Slot 0 to perform the Slot 0 functions and a given complement of instrument modules in Slots 1 — 12. The subsystem also may include a Resource Manager.

Word Serial Protocol

A word oriented, bidirectional, serial protocol for VXIbus communications between message-based devices (devices that include communication registers in addition to configuration registers).

Word Serial Communications

Inter-device communications using the Word Serial Protocol.

WSP

See Word Serial Protocol.

10 MHz Clock

A 10 MHz timing reference. Also see CLK10.

488-To-VXIbus Interface

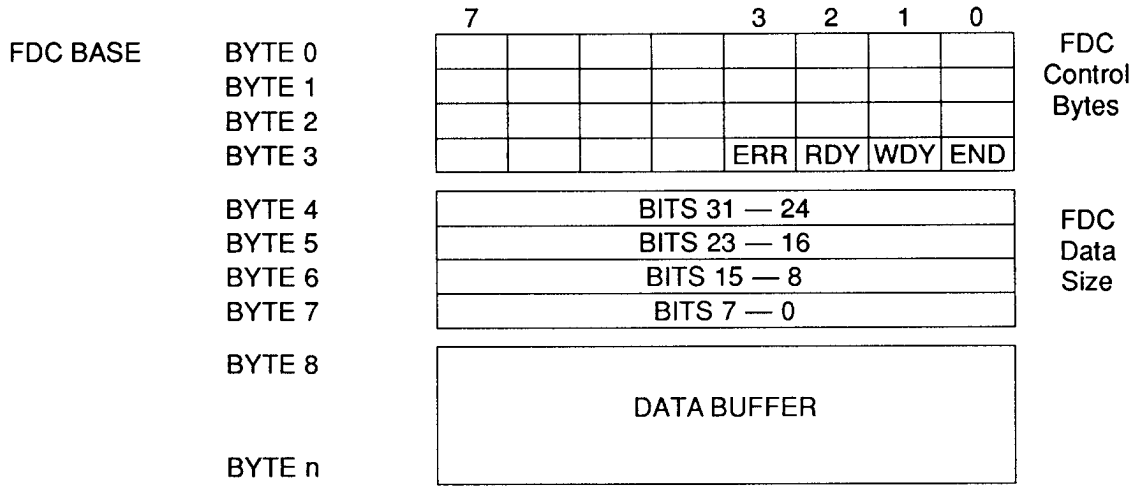
A message based device providing for communication between the IEEE 488 bus and VXIbus instrument modules.

Appendix C
Fast Data Channel Reference

OVERVIEW

Fast Data Channel (FDC) protocol provides a mechanism for quickly transferring test data to or from the VX4820. It is designed to be a simple and efficient protocol providing the highest possible throughput with a minimum of required software overhead.

The protocol utilizes the shared memory provided on each VX4820 module as the media for transferring data. A portion of this memory is negotiated utilizing the *FDCBASE?* and *FDCSIZE?* commands. Actual transfer of data is initiated by the *FDCLOAD* or *FDCLOAD?* commands.



The download or upload of data utilizes FDC memory. FDC memory is a shared memory area that can be accessed by both the VX4820 and from the VXI bus. The first 4 bytes in the FDC memory are control bytes, the next 4 bytes indicate actual data size, and the rest of the memory is the data buffer. FDCBASE is required to be word aligned.

In the control bytes, currently only bits 0 — 3 of byte 3 are defined. Bit 0 is the END bit, bit 1 is the Write Ready bit (WDY) and bit 2 is the Read Ready bit (RDY). If the END bit is set to 1, this is the last block of data to transfer. If the END bit is set to 0, this is not the last data block. If the WDY bit is set to 1, the VX4820 is ready to receive data. If the WDY bit is set to 0, it is not ready to receive any data. If the RDY bit is set to 1, the VX4820 has put the data in the data buffer and is ready for the system controller to read the data. If the RDY bit is set to 0, the data has been read by the system controller. The ERR bit indicates a transfer error has occurred. Either the controller or the module may set this bit. When detected the FDCLOAD or FDCLOAD? command is aborted and any data transferred must be considered to be corrupt.

The data size contains the number of bytes in the buffer to transfer in that particular data block. The data buffer contains the data to transfer to or from the VX4820. If multiple buffers of data are transferred, the data size of all but the last transfer must be even (on word boundaries).

Download Procedure

FDC download from the system commander to the VX4820 is as follows:

Step 1: Commander gets the FDC address space and FDC size from the VX4820 using the *FDCBASE?* and *FDCSIZE?* commands.

Step 2: Commander sets END bit to 0, WDY bit to 0.

Step 3: Commander sends *FDCLOAD* command to the VX4820.

Step 4: When VX4820 receives the *FDCLOAD* command, and it is ready to receive data, it will set the WDY bit to 1.

Step 5: When the WDY bit is set to 1, the commander can put data into the FDC data buffer, set the data size, and set the WDY bit to 0. If it is the last block of data, the commander must also set the END bit to 1.

Step 6: When the WDY bit is 0, the VX4820 will read the data from the FDC memory, and set the WDY bit to 1.

Step 7: When the WDY bit is 1, the Commander can repeat step 5 to step 6 until all data has been transferred.

System Commander	VX4820
Get FDC address and space Set END and WDY bit to 0 Send *FDCLOAD* command	
	Receive *FDCLOAD* command Set WDY to 1
Put data to FDC data buffer Set data size Set END bit Set WDY bit to 0 to 1 for last data block.	
	Read data from FDC data buffer Set WDY bit to 1
.	
.	
.	

Download Example

Download 2 blocks of sequence data to the VX4820:

FDCBASE?

Retrieve FDC memory address

FDCSIZE?

Retrieve FDC memory size

Set END and WDY bit to 0

FDCLOAD

Wait for WDY bit become 1

Put first data block into FDC data buffer

Put number of bytes into data size

Set WDY bit to 0

Wait for WDY bit become 1

Put second data block into FDC data buffer

Put number of bytes into data size

Set END bit to 1

Set WDY bit to 0

Wait for WDY bit become 1

Upload Procedure

The FDC transfer mechanism from the VX4820 to the system commander is as follows:

Step 1: Commander gets the FDC address space and FDC size from the VX4820.

Step 2: Commander set END bit to 0, RDY bit to 0.

Step 3: Commander sends *FDCLOAD?* command to the VX4820.

Step 4: When the VX4820 receives the *FDCLOAD?* command, it will respond with the number of bytes to transfer. It will then put data in the FDC data buffer, set the data size and set the RDY bit to 1. The VX4820 will set the END bit to 1 if it is the last block of data.

Step 5: When the RDY bit is set to 1, the commander can read the FDC data buffer, and set the RDY bit to 0.

Step 6: When the RDY bit is 0, the VX4820 can repeat Steps 4 and 5 and send the rest of the data to the commander.

System Commander	VX4820
Get FDC address and space Set END and RDY bit to 0 Send FDCLOAD? command	
	Receive FDCLOAD? command Return the byte count
Get the byte count. Terminate if 0	
	Put data to FDC data buffer Set END bit to 1 for the last data block Set RDY to 1
Wait for RDY to become 1 Read data from FDC data buffer Set RDY bit to 0 . . .	

Upload Example

Upload 25 sequence steps to the commander starting from sequence address 10.

BLRANGE 10,35

Set the VX4820 response range

FDCBASE?

Retrieve FDC memory address

FDCSIZE?

Retrieve FDC memory size

Set END and RDY bit to 0

FDCLOAD?

Retrieve number of sequence steps in transfer

Wait for RDY bit become 1

read data from FDC data buffer

Set RDY bit to 0

Wait for RDY bit become 1

Read data from FDC data buffer

Set RDY bit to 0

Repeat until END bit is set to 1

VX4820 Data Buffer Format

Data Buffer

Byte Offset	Data	Size
0 - 31	WSLOAD Header	CHAR [32]
32 - 33	FDC Version #	CHAR [2]
34 - 39	Target Object ("VX4820")	CHAR [6]
40 - 43	Offset to pin data	LONG
44 - 47	Starting Sequence	LONG
48 - 51	Sequence Count	LONG
52 - ...	Control list	CHAR []
.....	binary-coded pin data	WORD []

WSLOAD Header

String containing WSLOAD command syntax backfilled with spaces. The format is: "WSLOAD #9nnnnnnnnn"+ 18 space characters; nnnnnnnnn is a 9 digit decimal number which specifies the number of following bytes.

FDC Version #

The first byte is the version major number and the second byte is the minor number. Both of them are represented in binary format.

Target Object

A fixed string "VX4820"

Offset to pin data

32 bit offset (required even) of the pin data from the FDC Version #.

Sequence Starting Address

Starting sequence of the pin data.

Sequence Count

Total number of sequences in the pin data area.

Last Sequence Address

The last sequence in the test.

Internal Clock Cycle Rate

Clock rate in nanoseconds (50-3276700 binary)

Control List

The control list format is: | size | tag | param1 | - | paramN |

SIZE	TAG	PARAMETER 1	PARAMETER 2	PARAMETER 3	SIZE	...

Control List Format

size

1 byte long, indicates the size of this control information block. So the address of the < size> field plus the size will be the address of the next control information block.

tag

1 byte long, indicates the type of control information. The tag and parameter fields for the VX4820 can be:

* The addresses and branch type are represented in binary data format.
* The Branch, Clear, Pause and Trigger tag specify setting the

tag (1 byte)	Description	Param1 (2 bytes)	Param2 (2 bytes)	Param3 (2 bytes)
1	Branch	From Seq	Dest Seq	Branch type 0 - increment 1 - BRA Always 2 - BRC Fail 3 - BRC Pass
2	Clear Event	Seq Step		
3	Pause	Seq Step		
4	Trigger	Seq Step		
60	Seq Start	Seq Step		
61	Seq End	Seq Step		
62	Int Clk Rate	HI WORD	LO WORD	
255	End			

corresponding condition in a sequence address.
* The End tag specifies the end of control information list.

Pin Data Format

The binary coded pin data has the following format:

n	Pod 0	Drive/Compare 15-0
n+1	Pod 0	Drive/Compare 31-16
n+2	Pod 0	Output Enable 15-0
n+3	Pod 0	Output Enable 31-16
n+4	Pod 0	Compare Mask*/Input 15-0
n+5	Pod 0	Compare Mask*/Input 31-16
n+6	Pod 1	Drive/Compare 15-0
n+7	Pod 1	Drive/Compare 31-16
n+8	Pod 1	Output Enable 15-0
n+9	Pod 1	Output Enable 31-16
n+10	Pod 1	Compare Mask*/Input 15-0
n+11	Pod 1	Compare Mask*/Input 31-16

Output/ Cmpr	Output Enable	Cmpr Mask	Pin I/O type	Learn Mode
0	1	1	DRIVE LO	yes
1	1	1	DRIVE HI	yes
0	0	1	INHIBIT	yes
0	0	0	COMPARE LO	no
1	0	0	COMPARE HI	no
0	1	0	DRIVE LO WITH COMPARE	no
1	1	0	DRIVE HI WITH COMPARE	no
1	0	1	INHIBIT	yes

Pin Programming (Bit Level)

FDC Commands

The FDC commands are described in the remaining pages of this appendix.

FDCBASE? **FDCBASE?**

Purpose: *FDCBASE?* gets the starting address of the Fast Data Channel (FDC) buffer.

Query Syntax: `FDCBASE?`

Query Parameters: None

Query Response `FDCBASE <address>;`

`address`: numeric hex value #H200000 ... EF0000

-1 if not FAILED

`data`: string of char (64)

Only valid if test state is FAILED

Description *FDCBASE* returns the absolute A24 VXI address of the shared memory used by the FDC protocol. FDC protocol is used to download or upload test sequence data.

Example: In this example, the current FDC base address is 200000h:

```
FDCBASE?

#H200000;
```

Related Commands: FDCSIZE?, FDCLOAD, FDCLOAD?

FDCEVENT, FDCEVENT? **FDCEVENT, FDCEVENT?**

Purpose: *FDCEVENT* controls Fast Data Channel (FDC) EVENT generation. *FDCEVENT?* returns the value of the FDCEVENT switch.

Command Syntax:

```
FDCEVENT {ON | OFF}
```

Command Parameters:

ON

Causes the module to send the data available EVENT.

OFF

Disables the generation of the data available EVENT.

Query Syntax:

```
FDCEVENT?
```

Query Parameters: None

Query Response

```
{ON | OFF};
```

Description *FDCEVENT* controls the generation of VXI events during FDC transfers. FDC events are generated to signal its commander that its FDC buffer contains valid data. VXI user-defined event #H81nn (where nn = VX4820 logical address) is used to report the event.. Using FDC Events reduces the need for the commander to poll for data buffer status.

FDCEVENT? returns the current state of the FDCEVENT control.

Example: In this example, the FDC Events are enabled.

```
FDCEVENT ON

FDCEVENT?
ON
```

Related Commands: FDCBASE?, FDCLOAD, FDCLOAD?

FDCLOAD, FDCLOAD? **FDCLOAD, FDCLOAD?**

Purpose: *FDCLOAD* downloads binary test data to the instrument.
FDCLOAD? uploads binayr test data from the instrument.

Command Syntax:
```
FDCLOAD
```

Command Parameters: None

Query Syntax:
```
FDCLOAD?
```

Query Parameters: None

Query Response
```
<test steps>;
```

test steps: number of test steps transferred
numeric value 1 ... 16351 or 0, if error

Description *FDCLOAD* transfers test data from the system commander to the VX4820 module.

FDCLOAD? transfers test data from the VX4820 module to the system controller. The transfer is accomplished using the Fast Data Channel (FDC) protocol for high-speed data communication. Data transfer can also be accomplished with the SEQ:VECTOR and SEQ:VECORINC commands using the VXI Word Serial Protocol.

NOTE

When the FDCLOAD(?) command is received, the command focus moves from the VXI Word Serial Protocol to the Fast Data Channel. If the appropriate drivers for FDC are not invoked, the communications with the VX4820 module will hang. The only way to recover communications is via a soft reset of the VX4820 or a system reset.

Related Commands: BLRANGE, FDCBASE?, FDCSIZE?, WSLOAD

FDCSIZE? **FDCSIZE?**

Purpose: *FDCSIZE?* returns the size of the Fast Data Channel (FDC) buffer.

Query Syntax:

```
FDCSIZE?
```

Query Parameters: None

Query Response

```
<buffer size>;
```

`buffer size:` number of bytes

Description *FDCSIZE* returns the absolute size of the FDC memory. FDC protocol is used to download or upload test sequence data.

Example: In this example, the FDC memory area is 64 Kbytes:

```
FDCSIZE?

65536
```

Related Commands: FDCBASE?, FDCLOAD, FDCLOAD?

Appendix D
VX4820 Diagnostic Routines

Below is a list of the diagnostic tests performed by the VX4820. POWER_UP tests are executed each time the module is powered up or reset. All diagnostics are invoked when the TST? command is received. STERR? returns the test number.

Test #	Group #	Test Type	Test Description
200	2		ROM checksum test
400	4	POWER_UP	NVRAM checksum test
500	5	POWER_UP	ACFAIL interrupt test
600	6	POWER_UP	Shared Memory data line test
610	6	POWER_UP	Shared Memory address line test
620	6		Shared Memory cell test
800	8	POWER_UP	VXI ID register test
805	8	POWER_UP	VXI Device Type register test
820	8	POWER_UP	VXI Protocol register test
900	9	POWER_UP	Tick timer test
1300	13	POWER_UP	Sequence Nibblizer test
1305	13	POWER_UP	SCR Data Line test
1310	13		State Control RAM Cell Test
1315	13	POWER_UP	Sequence counter test
1320	13	POWER_UP	Internal clock test
1325	13	POWER_UP	Vector Counter interrupt test
1330	13	POWER_UP	Sequencer test
1335	13	POWER_UP	Clear Compare event test
1340	13	POWER_UP	Pod 5V test
2000	20	POWER_UP	Pod 0 Address Register Test
2001	20	POWER_UP	Pod 0 Data Line Test
2002	20	POWER_UP	Pod 0 Address Line Test
2003	20		Pod 0 Data RAM Cell Test
2010	20		Pod 0 Output pin test
2015	20		Pod 0 Pod FIFO test
2020	20		Pod 0 compare event unassert test
2025	20		Pod 0 compare event assert test
2100	21	POWER_UP	Pod 1 Address Register Test
2101	21	POWER_UP	Pod 1 Data Line Test
2102	21	POWER_UP	Pod 1 Address Line Test
2103	21		Pod 1 Data RAM Cell Test
2110	21		Pod 1 Output pin test
2115	21		Pod 1 Pod FIFO test
2120	21		Pod 1 compare event unassert test
2125	21		Pod 1 compare event assert test