# EX1629

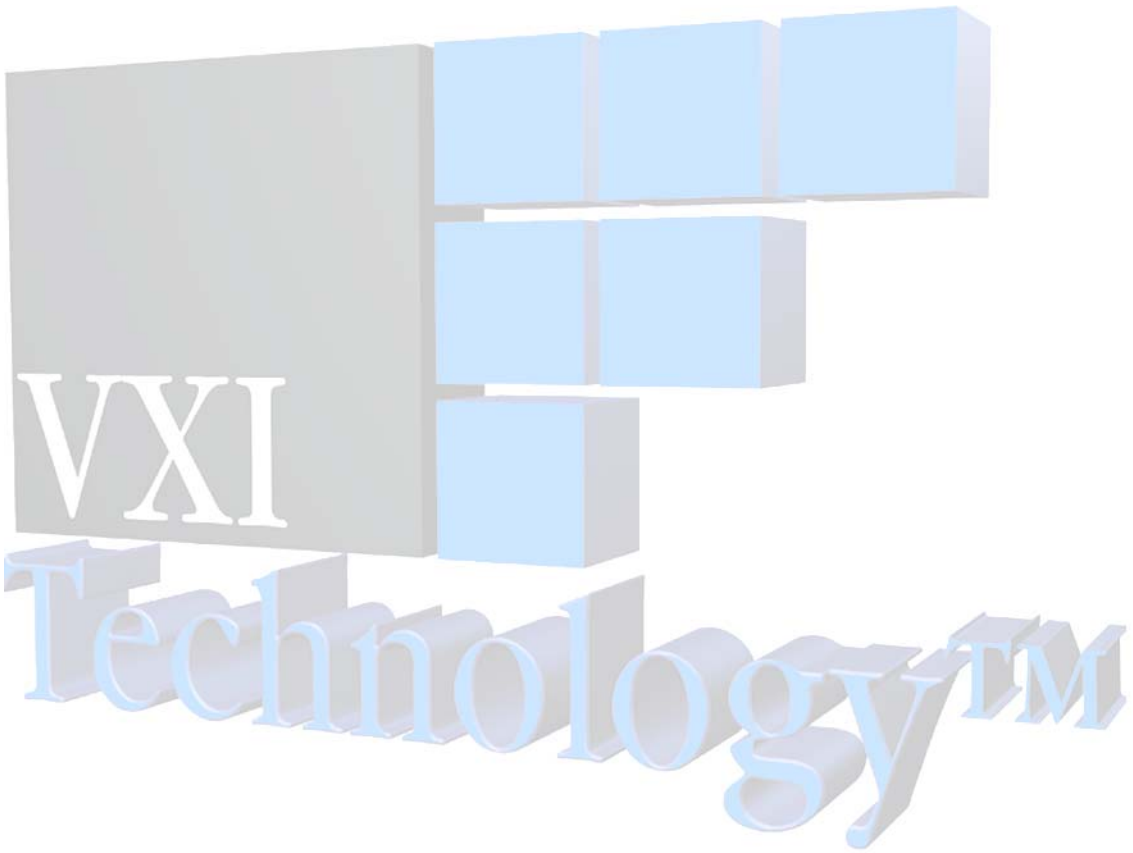
## 48-CHANNEL STRAIN GAGE INSTRUMENT

## USER'S MANUAL

**P/N: 82-0109-000**
**Released February 23, 2006**

**VXI Technology, Inc.**

**2031 Main Street**
**Irvine, CA 92614-6509**
**(949) 955-1894**


VXI
Technology™

# TABLE OF CONTENTS

**INTRODUCTION**
Certification ..... 10
Warranty ..... 10
Limitation of Warranty ..... 10
Restricted Rights Legend ..... 10
DECLARATION OF CONFORMITY ..... 11
GENERAL SAFETY INSTRUCTIONS ..... 12
Terms and Symbols ..... 12
Warnings ..... 12
SUPPORT RESOURCES ..... 14
**SECTION 1 ..... 15**
INTRODUCTION ..... 15
Overview ..... 15
Features ..... 15
A Complete High-Density Solution ..... 15
Data Integrity ..... 15
Excitation Source ..... 16
Programmable Bridge Configurations ..... 16
Self-calibration ..... 16
Confidence Measurement System ..... 16
Shunt Calibration ..... 16
Wideband Output ..... 16
Multiple Gain Ranges ..... 16
Sampling Rate ..... 17
Digital Filtering ..... 17
Triggering ..... 17
Input Connector ..... 17
TEDS Transducer Support ..... 17
LXI Trigger Bus ..... 17
EX1629 Specifications ..... 18
Explanation of Specifications ..... 22
Sampling Rate ..... 22
Bridge Excitation ..... 22
Bridge Completion ..... 22
Shunt Calibration ..... 23
Quarter-Bridge Strain Measurement ..... 23
Quarter-Bridge Strain Measurement/Full-Bridge Strain Measurement ..... 23
Wideband Outputs ..... 24
Confidence Measurements ..... 24
Maximizing Measurement Performance ..... 24
Utilize self-calibration ..... 24
Utilize excitation measurement ..... 25
Utilize proper strain gage wiring techniques ..... 25
Compensate for lead wire desensitization error ..... 25
Allow for thermal stabilization of the bridge ..... 26
**SECTION 2 ..... 27**
PREPARATION FOR USE ..... 27
Overview ..... 27
Unpacking ..... 27
Installation Location ..... 27
Warm-Up Time ..... 27
Input Connections / Wiring ..... 28

Bridge Configurations ..... 29
Voltage Measurement Configurations ..... 31
Driver Installation ..... 33
Network Configuration ..... 33
**SECTION 3 ..... 35**
BASIC OPERATION ..... 35
Introduction ..... 35
Engineering Unit (EU) Conversion ..... 35
Quarter-Bridge 350, Quarter-Bridge 120, Quarter-Bridge User ..... 35
Half-Bridge Bending ..... 36
Half-Bridge Poisson ..... 37
Full-Bridge Bending ..... 37
Full-Bridge Poisson ..... 38
Full-Bridge Bending Poisson ..... 38
Voltage ..... 39
Ratiometric ..... 39
Linear ..... 39
Nonstandard ..... 39
Completion Resistor ..... 40
Input Multiplexer ..... 40
Completion Resistor/Input Multiplexer Default Settings ..... 41
Gage Factor / Poisson Ratio ..... 41
Measurement Range / Gain ..... 42
Excitation Source ..... 42
Excitation Source Measurement ..... 43
Unstrained Voltage Measurement ..... 43
Scan List Configuration ..... 44
Sampling Rate ..... 44
Units ..... 45
Tare ..... 45
Digital Filter ..... 45
Triggering ..... 46
Data Format ..... 46
Shunt Calibration ..... 46
Self-Calibration ..... 47
Locking ..... 48
Confidence Scan List Configuration ..... 48
Configuration Storage ..... 49
Wideband Output ..... 49
Digital I/O ..... 51
LXI Trigger Bus ..... 52
TEDS Transducer Support ..... 52
Reset Button - LXI LAN Configuration Initialize (LCI) Mechanism ..... 54
**SECTION 4 ..... 55**
TRIGGERING ..... 55
Overview ..... 55
Acquisition Data and FIFO ..... 56
Confidence Measurement System ..... 57
ADC Clock and Synchronization ..... 57
Synchronizing Multiple Instruments ..... 57
**SECTION 5 ..... 59**
WEB PAGE OPERATION ..... 59
Introduction ..... 59
Opening the Web Page ..... 59
General Web Page Operation ..... 59
Password ..... 60

VXI Technology Logo....60
EX1629 Strain Gage Measurement Unit....60
Reset....60
Reboot....61
Network Configuration....61
Time Configuration....63
Upgrade....64
**SECTION 6....65**
PROGRAMMING....65
Introduction....65
Default Settings....65
Opening an Instrument Session....67
Closing an Instrument Session....67
Configuring the Acquisition Channels....67
Setting Bridge Limits....69
Lead Wire Compensation....69
Configure Trigger and ADC Clock....70
ADC Sample Clock....70
ADC Synchronization....71
Trigger Source....71
Arm Source....72
Standalone (Single Instrument) Example Configuration....72
Multiple Instruments (Master/Slave) Example Configuration....73
Retrieving Data (Read FIFO and Streaming Data)....76
Read FIFO....77
Asynchronous Streaming Data....79
Calibration Data....83
Starting/Stopping Acquisition....84
**SECTION 7....85**
FUNCTION CALLS....85
Introduction....85
Function Return Value....85
Function Tree....85
Initialize....85
Limit Checking....85
Configuration Calls....85
Lock Function Calls....86
Digital Input/Output Calls....86
LXI Trigger Bus Calls....86
Scanlist Calls....86
Trigger System Calls....86
Filter Configuration Calls....87
Excitation Voltage Calls....87
EU Conversion Calls....87
Shunt Configuration Calls....88
TEDS Calls....88
Data Retrieval Calls....88
Data Retrieval Calls - Advanced....88
Self-Calibration Calls....88
Internal Calibration Source Calls....88
Utility Function Calls....88
Lead Wire Calls....89
Calibration File Query....89
Close....89
Alphabetical Function Set....90
Sample Function Definition....94

EX1629 Function Set....95
vtex1629_abort....95
vtex1629_allow_all_channels....96
vtex1629_break_lock....97
vtex1629_check_lock....98
vtex1629_clear_stored_config....99
vtex1629_close....100
vtex1629_compare_digests....101
vtex1629_dio_clear_event....102
vtex1629_dio_clear_events_all....103
vtex1629_disable_logging....104
vtex1629_disable_streaming_data....105
vtex1629_enable_logging....106
vtex1629_enable_streaming_data....107
vtex1629_enable_streaming_dataEx....109
vtex1629_erase_teds_data....110
vtex1629_error_message....111
vtex1629_error_query....112
vtex1629_findinstr....113
vtex1629_get_arm_count....114
vtex1629_get_arm_delay....115
vtex1629_get_arm_source....116
vtex1629_get_bridge_limit....118
vtex1629_get_bridge_limit_enabled....120
vtex1629_get_cal_coefficients....121
vtex1629_get_cal_file....124
vtex1629_get_cal_file_size....126
vtex1629_get_cal_source....127
vtex1629_get_completion_resistor....128
vtex1629_get_conf_scanlist....129
vtex1629_get_confidence_limit....130
vtex1629_get_confidence_reporting_mode....132
vtex1629_get_current_config_digest....133
vtex1629_get_dio_bank0_direction....134
vtex1629_get_dio_bank0_pullup....135
vtex1629_get_dio_bank1_direction....136
vtex1629_get_dio_bank1_pullup....137
vtex1629_get_dio_config_events....138
vtex1629_get_dio_input....139
vtex1629_get_dio_output....140
vtex1629_get_dsp_version....141
vtex1629_get_EU_conversion....142
vtex1629_get_euconv_dynamic_excitation_enabled....143
vtex1629_get_euconv_excitation....144
vtex1629_get_excitation....145
vtex1629_get_excitation_enabled....146
vtex1629_get_fifo_count....147
vtex1629_get_gain....148
vtex1629_get_gauge_factor....149
vtex1629_get_half_bridge_lead_wire_desensitization....150
vtex1629_get_IIR_filter_configuration....151
vtex1629_get_input_multiplexer....153
vtex1629_get_instrument_serial_number....154
vtex1629_get_lead_wire_resistance....155
vtex1629_get_linearscaling_configuration....156
vtex1629_get_lxibus_configuration....157
vtex1629_get_lxibus_input....159

vtex1629_get_lxibus_output ..... 160
vtex1629_get_pattern_arm_configuration ..... 161
vtex1629_get_pattern_trig_configuration ..... 163
vtex1629_get_poisson_ratio ..... 165
vtex1629_get_sample_clock_source ..... 166
vtex1629_get_sample_count ..... 167
vtex1629_get_sample_frequency ..... 168
vtex1629_get_scanlist ..... 169
vtex1629_get_selfcal_status ..... 170
vtex1629_get_settling_time ..... 171
vtex1629_get_shunt_enabled ..... 172
vtex1629_get_shunt_source ..... 173
vtex1629_get_shunt_value ..... 174
vtex1629_get_stored_config_digest ..... 175
vtex1629_get_strain_units ..... 176
vtex1629_get_synch_source ..... 177
vtex1629_get_tare ..... 178
vtex1629_get_teds_data ..... 179
vtex1629_get_trigger_count ..... 180
vtex1629_get_trigger_delay ..... 181
vtex1629_get_trigger_source ..... 182
vtex1629_get_trigger_timer ..... 183
vtex1629_get_unstrained_voltage ..... 184
vtex1629_identify_sensor ..... 185
vtex1629_init ..... 186
vtex1629_load_stored_config ..... 187
vtex1629_lock ..... 188
vtex1629_measure_confidence ..... 189
vtex1629_measure_excitation_voltage ..... 191
vtex1629_measure_lead_wire_resistance ..... 193
vtex1629_measure_unstrained_voltage ..... 195
vtex1629_read_fifo ..... 196
vtex1629_read_fifoEx ..... 198
vtex1629_read_teds_MLAN ..... 200
vtex1629_read_teds_URN ..... 201
vtex1629_reset ..... 202
vtex1629_reset_fifo ..... 203
vtex1629_reset_tare ..... 204
vtex1629_reset_trigger_arm ..... 205
vtex1629_revision_query ..... 206
vtex1629_self_cal_clear ..... 207
vtex1629_self_cal_clear_stored ..... 208
vtex1629_self_cal_get_status ..... 209
vtex1629_self_cal_init ..... 210
vtex1629_self_cal_is_running ..... 212
vtex1629_self_cal_is_stored ..... 213
vtex1629_self_cal_load ..... 214
vtex1629_self_cal_store ..... 215
vtex1629_self_test ..... 216
vtex1629_self_test_get_status ..... 217
vtex1629_self_test_init ..... 218
vtex1629_send_dio_pulse ..... 219
vtex1629_send_lxibus_pulse ..... 220
vtex1629_set_arm_count ..... 221
vtex1629_set_arm_delay ..... 222
vtex1629_set_arm_source ..... 223
vtex1629_set_bridge_limit ..... 224

vtex1629_set_bridge_limit_enabled....226
vtex1629_set_cal_out....227
vtex1629_set_cal_source....228
vtex1629_set_completion_resistor....229
vtex1629_set_conf_scanlist....230
vtex1629_set_confidence_limit....231
vtex1629_set_confidence_reporting_mode....233
vtex1629_set_dio_bank0_direction....234
vtex1629_set_dio_bank0_pullup....235
vtex1629_set_dio_bank1_direction....236
vtex1629_set_dio_bank1_pullup....237
vtex1629_set_dio_config_events....238
vtex1629_set_dio_output....240
vtex1629_set_EU_conversion....241
vtex1629_set_euconv_dynamic_excitation_enabled....242
vtex1629_set_euconv_excitation....243
vtex1629_set_excitation....244
vtex1629_set_excitation_enabled....245
vtex1629_set_gain....246
vtex1629_set_gauge_factor....247
vtex1629_set_half_bridge_lead_wire_desensitization....248
vtex1629_set_IIR_filter_configuration....249
vtex1629_set_input_multiplexer....251
vtex1629_set_lead_wire_resistance....252
vtex1629_set_linearscaling_configuration....253
vtex1629_set_lxibus_configuration....254
vtex1629_set_lxibus_output....256
vtex1629_set_pattern_arm_configuration....257
vtex1629_set_pattern_trig_configuration....259
vtex1629_set_poisson_ratio....261
vtex1629_set_sample_clock_source....262
vtex1629_set_sample_count....263
vtex1629_set_sample_frequency....264
vtex1629_set_scanlist....265
vtex1629_set_shunt_enabled....266
vtex1629_set_shunt_source....267
vtex1629_set_shunt_value....269
vtex1629_set_strain_units....270
vtex1629_set_synch_source....271
vtex1629_set_tare....273
vtex1629_set_teds_data....274
vtex1629_set_trigger_count....275
vtex1629_set_trigger_delay....276
vtex1629_set_trigger_source....277
vtex1629_set_trigger_source_timer....278
vtex1629_set_trigger_timer....279
vtex1629_set_unstrained_voltage....280
vtex1629_soft_arm....281
vtex1629_soft_synch....282
vtex1629_soft_trig....283
vtex1629_store_current_config....284
vtex1629_trig_init....285
vtex1629_unlock....286
vtex1629_write_teds_MLAN....287
vtex1629_zero_cal....288
Error Messages....289

**APPENDIX A** ..... **297**
MULTI-INSTRUMENT OPERATION ..... 297
Introduction ..... 297
Distributing Sample Clock and Synchronization Signals ..... 297
Triggering ..... 300
**APPENDIX B** ..... **301**
EX1629 FILTERING ..... 301
Introduction ..... 301
Analog Anti-Aliasing Filter ..... 301
Digital Filters ..... 302
CIC Filter ..... 303
DSP Filters ..... 303
Group Delay ..... 304
Transformations ..... 305
The Bilinear Transform ..... 305
The Matched Z-transform ..... 305
**APPENDIX C** ..... **307**
MICROLAN (MLAN) PRIMER ..... 307
Introduction ..... 307
Programming MLAN ..... 308
DS2430 Commands ..... 311
WRITE_SCRATCHPAD_2430 ..... 311
READ_SCRATCHPAD_2430 ..... 313
COPY_SCRATCHPAD_2430 ..... 314
READ_MEMORY_2430 ..... 315
WRITE_AND_COPY_SCRATCHPAD_2430 ..... 316
DS2431 Commands ..... 317
WRITE_SCRATCHPAD_2431 ..... 317
READ_SCRATCHPAD_2431 ..... 319
COPY_SCRATCHPAD_2431 ..... 320
READ_MEMORY_2431 ..... 321
WRITE_AND_COPY_SCRATCHPAD_2431 ..... 322
Additional Notes ..... 324
Checksums ..... 324
Sending & Receiving ..... 324
Printing Packets ..... 325
CRC Checking ..... 326
Version Information ..... 326
**INDEX** ..... **327**

## CERTIFICATION

VXI Technology, Inc. (VTI) certifies that this product met its published specifications at the time of shipment from the factory. VTI further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology (formerly National Bureau of Standards), to the extent allowed by that organization's calibration facility, and to the calibration facilities of other International Standards Organization members.

## WARRANTY

The product referred to herein is warranted against defects in material and workmanship for a period of one year from the receipt date of the product at customer's facility. The sole and exclusive remedy for breach of any warranty concerning these goods shall be repair or replacement of defective parts, or a refund of the purchase price, to be determined at the option of VTI.

For warranty service or repair, this product must be returned to a VXI Technology authorized service center. The product shall be shipped prepaid to VTI and VTI shall prepay all returns of the product to the buyer. However, the buyer shall pay all shipping charges, duties, and taxes for products returned to VTI from another country.

VTI warrants that its software and firmware designated by VTI for use with a product will execute its programming when properly installed on that product. VTI does not however warrant that the operation of the product, or software, or firmware will be uninterrupted or error free.

## LIMITATION OF WARRANTY

The warranty shall not apply to defects resulting from improper or inadequate maintenance by the buyer, buyer-supplied products or interfacing, unauthorized modification or misuse, operation outside the environmental specifications for the product, or improper site preparation or maintenance.

VXI Technology, Inc. shall not be liable for injury to property other than the goods themselves. Other than the limited warranty stated above, VXI Technology, Inc. makes no other warranties, express or implied, with respect to the quality of product beyond the description of the goods on the face of the contract. VTI specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

## RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (b)(3)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

VXI Technology, Inc.
2031 Main Street
Irvine, CA 92614-6509 U.S.A.

# DECLARATION OF CONFORMITY

**Declaration of Conformity According to ISO/IEC Guide 22 and EN 45014**

| | |
|---|---|
| **MANUFACTURER'S NAME** | VXI Technology, Inc. |
| **MANUFACTURER'S ADDRESS** | 2031 Main Street<br>Irvine, California 92614-6509 |
| **PRODUCT NAME** | 48-Channel Strain Gage Instrument |
| **MODEL NUMBER(S)** | EX1629 |
| **PRODUCT OPTIONS** | All |
| **PRODUCT CONFIGURATIONS** | All |

*VXI Technology, Inc. declares that the aforementioned product conforms to the requirements of the* Low Voltage Directive 73/23/EEC *and the* EMC Directive 89/366/EEC *(inclusive* 93/68/EEC*) and carries the "CE" mark accordingly. The product has been designed and manufactured according to the following specifications:*



| | |
|---|---|
| **SAFETY** | EN61010 (2001) |
| **EMC** | EN61326 (1997 w/A1:98) Class A<br>CISPR 22 (1997) Class A<br>VCCI (April 2000) Class A<br>ICES-003 Class A (ANSI C63.4 1992)<br>AS/NZS 3548 (w/A1 & A2:97) Class A<br>FCC Part 15 Subpart B Class A<br>EN 61010-1:2001 |

*I hereby declare that the aforementioned product has been designed to be in compliance with the relevant sections of the specifications listed above as well as complying with all essential requirements of the Low Voltage Directive.*

**February 2007**

CE

***Steve Mauga, QA Manager***

# GENERAL SAFETY INSTRUCTIONS

Review the following safety precautions to avoid bodily injury and/or damage to the product. These precautions must be observed during all phases of operation or service of this product. Failure to comply with these precautions, or with specific warnings elsewhere in this manual, violates safety standards of design, manufacture, and intended use of the product.

***Service should only be performed by qualified personnel.***

## TERMS AND SYMBOLS

These terms may appear in this manual:

**WARNING** Indicates that a procedure or condition may cause bodily injury or death.

**CAUTION** Indicates that a procedure or condition could possibly cause damage to equipment or loss of data.

These symbols may appear on the product:

**ATTENTION -** Important safety instructions

Frame or chassis ground

Indicates that the product was manufactured after August 13, 2005. This mark is placed in accordance with *EN 50419, Marking of electrical and electronic equipment in accordance with Article 11(2) of Directive 2002/96/EC (WEEE)*. End-of-life product can be returned to VTI by obtaining an RMA number. Fees for take-back and recycling will apply if not prohibited by national law.

## WARNINGS

Follow these precautions to avoid injury or damage to the product:

**Use Proper Power Cord** To avoid hazard, only use the power cord specified for this product.

**Use Proper Power Source** To avoid electrical overload, electric shock, or fire hazard, do not use a power source that applies other than the specified voltage.

**Use Proper Fuse** To avoid fire hazard, only use the type and rating fuse specified for this product.

## WARNINGS (CONT.)

**Avoid Electric Shock**

To avoid electric shock or fire hazard, do not operate this product with the covers removed. Do not connect or disconnect any cable, probes, test leads, etc. while they are connected to a voltage source. Remove all power and unplug unit before performing any service. ***Service should only be performed by qualified personnel.***

**Ground the Product**

This product is grounded through the grounding conductor of the power cord. To avoid electric shock, the grounding conductor must be connected to earth ground.

**Operating Conditions**

To avoid injury, electric shock or fire hazard:

- Do not operate in wet or damp conditions.
- Do not operate in an explosive atmosphere.
- Operate or store only in specified temperature range.
- Provide proper clearance for product ventilation to prevent overheating.
- DO NOT operate if any damage to this product is suspected. ***Product should be inspected or serviced only by qualified personnel.***

**Improper Use**

The operator of this instrument is advised that if the equipment is used in a manner not specified in this manual, the protection provided by the equipment may be impaired.
Conformity is checked by inspection.

# SUPPORT RESOURCES

Support resources for this product are available on the Internet and at VXI Technology customer support centers.


**VXI Technology**
**World Headquarters**

VXI Technology, Inc.
2031 Main Street
Irvine, CA 92614-6509

Phone: (949) 955-1894
Fax: (949) 955-3041

**VXI Technology**
**Cleveland Instrument Division**

5425 Warner Road
Suite 13
Valley View, OH 44125

Phone: (216) 447-8950
Fax: (216) 447-8951

**VXI Technology**
**Lake Stevens Instrument Division**

VXI Technology, Inc.
1924 - 203 Bickford
Snohomish, WA 98290

Phone: (425) 212-2285
Fax: (425) 212-2289

**Technical Support**

Phone: (949) 955-1894
Fax: (949) 955-3041
E-mail: support@vxitech.com

*Visit http://www.vxitech.com for worldwide support sites and service plan information.*

# SECTION 1

# INTRODUCTION

## OVERVIEW

The EX1629 is a 48-channel high-performance strain gage measurement instrument. Its combination of measurement performance and integrity, configuration flexibility, package density, and network connectivity make it the most powerful, yet easy-to-use, instrument of its kind. The EX1629 is a complete, self-contained strain measurement system that communicates over Ethernet. Unlike other data acquisition offerings in its class, the EX1629 offers a tightly integrated solution that frees the user from the complexity of marrying terminal blocks, signal conditioning cards, digitizer, excitation source, and chassis together.

## FEATURES

### *A Complete High-Density Solution*

The EX1629 provides 48 channels of strain conditioning, bridge completion, and excitation in a single 19-inch rack-mount enclosure. Its density and integration simplify the task of assembling a test station. Most applications require only the simple connection of power, Ethernet communication, and input connections. Moreover, test consistency and reliability is greatly increased because its base configuration requires no accessory modules or other equipment to be connected or cabled together.

The design of the EX1629 provides full configuration flexibility, with all bridge completion and excitation source configurations set programmatically. There is no need to manually reconfigure hardware to make measurement changes.

The EX1629 can operate independently or, for large data acquisition applications, multiple instruments can be synchronized via an external trigger bus. This design allows for numerous units to be controlled by a single host computer by utilizing a programmable master-slave relationship.

### *Data Integrity*

The design of the EX1629 placed paramount importance on maintaining data integrity under all measurement conditions. Each input channel is an independent measurement system, with discrete signal conditioning circuitry, excitation source, and 24-bit ADC. There is no sharing of source and measurement circuitry or sampling bandwidth among input channels. This makes each channel's performance completely insensitive to the state of all other channels, whether they are in normal operation, shorted, or overloaded.

Each channel is individually protected against shorts to ground, across the gage, or to another gage, as well as overvoltage. Trifilar input transformers provide superior high-frequency common mode noise rejection. Finally, an analog filter provides anti-alias protection for dynamic applications.

### *Excitation Source*

The EX1629 features bridge excitation, programmable and regulated on a per channel basis. The positive and negative excitation voltages are independently programmed from 0 V to +8 V and 0 V to -8 V, respectively, with current capability of 50 mA per channel. This programming independence provides the flexibility of balanced or imbalanced excitation. Finally, remote sense lines on each input connector can be employed for improved half-bridge and full-bridge performance.

### *Programmable Bridge Configurations*

Complete bridge configuration support is selectable under program control on a per channel basis. Options include full, half, quarter120, quarter350, and quarterUser bridges. Moreover, a high impedance voltage mode provides direct voltage measurements up to ±15 V.

### *Self-calibration*

In order to deliver high measurement accuracy over a wide ambient operating temperature range, the EX1629 provides the ability to perform an instrument self-calibration. During self-calibration, the input signal conditioning paths are disconnected from the input jacks and connected instead to a calibration bus that is driven by an internal calibration source. Through measurement of the conditioning paths at multiple calibration source points, software compensation for circuitry drift since the last full calibration is conducted. This provides a significant accuracy improvement without the burden of connecting external equipment. Moreover, the self-calibration process completes quickly and does not require removal of the actual input connections, making it convenient to run often.

### *Confidence Measurement System*

In addition to the main bridge measurement path, the EX1629 provides a unique, secondary measurement system that operates in parallel. This “confidence” measurement system provides the key bridge parameters of excitation voltage, excitation current, and common mode voltage and can provide verification of the integrity of the bridge and cable connections. Because the confidence system employs its own ADC, it has no effect on the sampling characteristics of the main bridge input.

### *Shunt Calibration*

The traditional shunt calibration process is supported to ensure correct bridge performance. Each input channel provides a unique, precision 55 kΩ resistor that can be programmatically connected internally for quarter-bridge shunting or externally for full- or half-bridge shunting. Moreover, an external resistor may be connected into each of three front panel connectors, one for each group of 16 input channels. Similarly, this front panel resistor may be programmatically connected internally or externally.

### *Wideband Output*

Each channel of the EX1629 has a high-performance analog wideband buffered output that can be connected to a high-speed digitizer for measuring structural vibration levels. The wideband outputs are accessible on the rear panel of the instrument via three 44-pin D-Sub connectors.

### *Multiple Gain Ranges*

Each channel is independently programmable with a signal conditioning gain of 1, 10, or 100. This provides application flexibility to make high resolution strain measurements as well as direct voltage measurements.

### *Sampling Rate*

The EX1629 can be configured for a sampling rate from 1 Sa/s to 10000 Sa/s in 30 discrete settings, regardless of the number of channels included in the scan list. This permits the tailoring of the data load to the dynamic requirements of the test application. If the number of channels are limited, however, a sample rate of 25000 Sa/s may be achieved.

### *Digital Filtering*

Each input provides a programmable digital filter with selectable type, order, and cut-off frequency. This provides the flexibility to customize the measurement response to the dynamic and noise rejection requirements of the application.

### *Triggering*

The EX1629 supports a full function trigger model with a separate arm source and trigger source event structure. Trigger and arm source events can be independently programmed from a variety of sources including Immediate, Timer, Digital I/O, and the Trigger Bus.

### *Input Connector*

The EX1629 features a standard RJ-45 telecom connector for each input channel. Not only are these connectors reliable, but low-cost construction of custom length cables is also readily available. Reconfiguration or replacement of strain gage connections is as easy as connecting a telephone.

### *TEDS Transducer Support*

The EX1629 provides support to read TEDS-equipped sensors to allow direct input of bridge configuration parameters and input configuration management.

### *LXI Trigger Bus*

The EX1629 features an LXI *(LAN eXtensions for Instrumentation)* compatible 8-channel trigger bus on the rear panel of the instrument. This differential-pair LVDS (*Low-Voltage Differential Signal*) bus consists of two identical ports connected in parallel. The primary use of the trigger bus is the transmission of high-speed signals for multiple-unit triggering and synchronization.

# EX1629 SPECIFICATIONS

| GENERAL SPECIFICATIONS | |
|---|---|
| **CHANNELS** | |
| | 48 differential inputs |
| **FUNCTIONS (STRAIN)** | |
| | Quarter 120 Ω, Quarter 350 Ω, Quarter User-Defined,<br>Half-Bending, Half-Poisson,<br>Full-Bending, Full-Poisson, Full-Bending Poisson |
| **FUNCTIONS (NON-STRAIN)** | |
| | Voltage, Ratiometric, Linear |
| **SAMPLING RATE** | |
| **For All Channels** | 30 settings from 1 Sa/s to 10000 Sa/s per channel |
| **For 16 Channels** | Up to 25000 Sa/s (see *Explanation of Specifications* for limitations.) |
| **A/D CONVERSION** | |
| | 24-bit ΔΣ converter per channel |
| **GAINS** | |
| | 1, 10, or 100, software selectable |
| **NETWORK CONNECTION** | |
| | 10/100 Base-T Ethernet |
| **INPUT CONNECTOR** | |
| | RJ-45 |

| BRIDGE EXCITATION | |
|---|---|
| **REGULATION** | |
| | Independent high-side and low-side control on a per channel basis |
| **HIGH-SIDE RANGE** | |
| | 0 to +8 V |
| **LOW-SIDE RANGE** | |
| | 0 to -8 V |
| **RESOLUTION** | |
| | 14-bit (500 µV) |
| **SENSE** | |
| | Local or remote |
| **CURRENT OUTPUT** | |
| | 50 mA per channel, short circuit limited to 60 mA |
| **SET POINT ACCURACY** | |
| | ±(0.04% + 2 mV) typical, ±(0.07% + 10 mV) maximum, for $\lvert V_{EXC} \rvert$ > 20 mV |
| **STABILITY** | |
| | ±(20 ppm/°C + 20 µV/°C) typical, ±(30 ppm/°C + 60 µV/°C) maximum |

| BRIDGE COMPLETION | |
|---|---|
| **RESISTOR VALUE** | |
| | 120 Ω, 350 Ω, and user-specified, software selectable<br>User value available as factory option. |
| **RESISTOR STABILITY** | |
| | 5 ppm/°C |
| **INPUT CONNECTOR LEAD RESISTANCE** | |
| | 30 mΩ minimum, 60 mΩ maximum, matched to <5 mΩ within each channel |
| **BACK-HALF RESISTORS** | |
| | 10 kΩ/10 kΩ, 0.1%, 2 ppm/°C |

| SHUNT CALIBRATION | |
|---|---|
| **INTERNAL RESISTOR** | |
| | 55 kΩ (0.1%, 25 ppm/°C) per channel standard (0.05%, 5 ppm/°C optional) |
| **EXTERNAL RESISTOR** | |
| | Front panel connection shared among 16 channels<br>Connection resistance: 16 Ω typical |
| **RESISTOR CONNECTION** | |
| | Software selectable: local (across completion resistor) or remote |

**QUARTER-BRIDGE STRAIN MEASUREMENTS**

| Excitation | Gain | Range[1] | Gain Accy[2] | Offset Accy[2,3] | Gain TC[4] | Offset TC[4] |
|---|---|---|---|---|---|---|
| 10 V | 100 | +30927 με/-29126 με | ±0.12% | ±25 με | ±50 ppm/ºC | ±4 με/ºC |
| 5 V | 100 | +63829 με/-56603 με | ±0.12% | ±25 με | ±50 ppm/ºC | ±4 με/ºC |

*Note 1: Nominal for balanced bridge*
*Note 2: Conditions:*
*GF = 2.0, $R_{comp}$ = 350 Ω, balanced excitation*
*<30 days, ±5 ºC from last self-calibration*
*15 ºC to 35 ºC, 1 year from full calibration*
*Assumes the excitation voltage is measured and used in the conversion. Valid for 30 days, ±5 ºC.*
*Includes the stability effects of the excitation source.*
*60 minute warm-up*
*Exclusive of lead wire desensitization errors*
*Exclusive of gage errors*
*Exclusive of noise*
*Note 3: <30 days, ±5 ºC from unstrained voltage measurement.*
*Note 4: Only applies outside of self-calibration window*

**FULL-BRIDGE STRAIN MEASUREMENTS**

| Excitation | Gain | Range[1] | Gain Accy[2] | Offset Accy[2,3] | Gain TC[4] | Offset TC[4] |
|---|---|---|---|---|---|---|
| 5 V | 100 | ±15000 με | ±0.05% | ±1.5 με | ±50 ppm/ºC | ±0.2 με/ºC |
| 2.5 V | 100 | ±30000 με | ±0.06% | ±3 με | ±60 ppm/ºC | ±0.4 με/ºC |

*Note 1: Nominal for balanced bridge*
*Note 2: Conditions:*
*GF = 2.0, balanced excitation, remote sense*
*<30 days, ±5 ºC from last self-calibration*
*15 ºC to 35 ºC, 1 year from full calibration*
*Assumes the excitation voltage is measured and used in the conversion. Valid for 30 days, ±5 ºC.*
*Includes the stability effects of the excitation source.*
*60 minute warm-up*
*Exclusive of gage errors*
*Exclusive of noise*
*Note 3: <30 days, ±5 ºC from unstrained voltage measurement.*
*Note 4: Only applies outside of self-calibration window*

| CONFIDENCE MEASUREMENTS | |
|---|---|
| **TOTAL EXCITATION VOLTAGE** | |
| | ±(0.012% + 500 μV) |
| **±EXCITATION VOLTAGE** | |
| | ±(0.012% + 2.5 mV) |
| **±EXCITATION CURRENT** | |
| | ±(0.1% + 50 μA) |
| **COMMON MODE VOLTAGE** | |
| | ±(0.1% + 2.5 mV) |
| **SAMPLING RATE** | |
| | Approximately 500 Sa/s |

| VOLTAGE MEASUREMENTS | | | | | |
|---|---|---|---|---|---|
| **Gain** | **Range** | **Gain Accy[1]** | **Offset Accy[1]** | **Gain TC[2]** | **Offset TC[2]** |
| 100 | ±150 mV | ±0.025% | ±15 µV | ±30 ppm/ºC | ±2 µV/ºC |
| 10 | ±1.5 V | ±0.025% | ±100 µV | ±30 ppm/ºC | ±12.5 µV/ºC |
| 1 | ±15 V | ±0.025% | ±700 µV | ±30 ppm/ºC | ±125 µV/ºC |

*Note 1: Conditions:*
*<30 days, ±5 ºC from last self-calibration*
*15 ºC to 35 ºC, 1 year from full calibration*
*60 minute warm-up*
*Exclusive of noise*
*Note 2: Only applies outside of self-calibration window.*

| INPUT CHARACTERISTICS | |
|---|---|
| **INPUT IMPEDANCE (dc)** | |
| | 10 GΩ |
| **INPUT BIAS CURRENT** | |
| | 10 nA maximum |
| **INPUT PROTECTION** | |
| | ±25 V |
| **COMMON MODE INPUT RANGE** | |
| | ±15 V |
| **CMRR (dc TO 60 Hz)** | |
| | 120 dB typical, 110 dB minimum (Gain = 100) |

| FILTERING | |
|---|---|
| **ANALOG ANTI-ALIAS LPF** | |
| | 60 kHz 1-pole per channel |
| **DIGITAL FIR FILTERING** | |
| **Passband ripple** | Achieves desired sampling frequency by decimation from 50000 Sa/s. |
| **$f_s \geq 3125$ Hz** | ±0.01 dB |
| **$f_s < 3125$ Hz** | ±0.001 dB |
| **Alias rejection** | 100 dB |
| **DIGITAL IIR FILTERING** | |
| **Configuration options per channel** | Type (Bessel, Butterworth, None)<br>Cut-off frequency (0.001 Hz to 4005 Hz)<br>Transform (Bilinear or Matched Z)<br>Order (1-10) |

| WIDEBAND OUTPUTS | |
|---|---|
| **CHANNELS** | |
| | 1 per input channel |
| **CONNECTORS** | |
| | (3) 44-pin male D-sub |
| **MAXIMUM OUTPUT VOLTAGE** | |
| | ±15 V |
| **OUTPUT IMPEDANCE** | |
| | 150 Ω |

| Gain | Gain Accy | Offset Accy (RTI) | Bandwidth (-3 dB) |
|---|---|---|---|
| 100 | ±0.15% | ±150 µV | > 100 kHz |
| 10 | ±0.15% | ±500 µV | > 150 kHz |
| 1 | ±0.15% | ±5 mV | > 150 kHz |

| DIGITAL I/O | |
|---|---|
| **CHANNELS** | |
| | 16 |
| **CONNECTOR** | |
| | (1) 44-pin female D-sub |
| **ELECTRICAL** | |
| $V_{INPUT}$ | -0.5 V to 5.5 V |
| $V_{IH}$ | 2 V minimum |
| $V_{IL}$ | 0.8 V maximum |
| $V_{OH}$ ($I_{OH}$ = -5.2 mA) | 2.5 V minimum |
| $V_{OL}$ ($I_{OL}$ = 48 mA) | 0.5 V maximum |

| TRIGGER BUS | |
|---|---|
| **CHANNELS** | |
| | 8 |
| **CONNECTORS** | |
| | (2) Micro DB-25 |
| **ELECTRICAL** | |
| **Logic type** | M-LVDS Type 1 |
| $V_{IT+}$ | 50 mV maximum |
| $V_{IT-}$ | -50 mV minimum |
| $V_{OS}$ | 1 V typical |

| POWER REQUIREMENTS | |
|---|---|
| **LINE VOLTAGE** | |
| | (90 – 264) V ac, (47 – 440) Hz |
| **INPUT POWER** | |
| | 200 VA maximum |

| ENVIRONMENTAL | |
|---|---|
| **OPERATING TEMPERATURE** | |
| | -5 °C to +55 °C |
| **HUMIDITY** | |
| | 5% to 85% relative humidity |

| MECHANICAL | |
|---|---|
| **HEIGHT** | |
| | 3.5 in (8.89 cm) |
| **WIDTH** | |
| | 19 in (48.26 cm) |
| **DEPTH** | |
| | 22 in (55.88 cm) |

## EXPLANATION OF SPECIFICATIONS

This section provides explanatory notes to certain elements of the EX1629 specifications that may be confusing or easily misunderstood.

### *Sampling Rate*

The EX1629 ADCs run at 50 kSa/s and data is decimated down by an integer factor to the user-selected sample rate. The EX1629 is capable of supporting a sampling rate of 10 kSa/s on all 48 channels simultaneously. Over a limited number of channels, however, the EX1629 is capable of supporting a sample rate of 25 kSa/s. In order to realize this higher sampling rate, the number of operating channels must be limited to 16 maximum and these channels must all exist on the same analog board (i.e. channels 0 through 15, 16 through 31, or 32 through 47 can be selected). If these conditions are not met, an error will occur.

### *Bridge Excitation*

Performance of the excitation source is quantified in two ways. *Set point accuracy* refers to the absolute accuracy of the excitation output compared to its nominal programmed value. Conversely, *stability* refers to the drift characteristics of the source once it has been programmed to a specific value. Since the EX1629 provides the ability to measure the excitation source and use the measured value in the EU conversion, it is the source's stability that ultimately effects strain measurement accuracy, not its set point accuracy.

While the source's performance characteristics are provided, they should not be added to the listed quarter-bridge and full-bridge accuracy tables, as these accuracy tables already contain the effects of the excitation source stability. The source characteristics are listed for reference and for the possibility that a user might use the EX1629 to provide bridge excitation, but another piece of equipment to measure the bridge output. In that case, the excitation performance is required in order to calculate the total system uncertainty. For this analysis, it must be noted that the listed characteristics are for each source independently.

### *Bridge Completion*

The characteristic *input connector lead resistance* refers to the residual resistance that the input connector represents in a quarter-bridge configuration. Specifically, referring to Figure 2-3, this is the resistance between Pin 1 (+Excitation) of the connector and the local connection of the +Excitation Sense line. Similarly, it is the resistance between Pin 2 (-Excitation) and the point at which the completion resistor is shunted. These resistances serve to slightly desensitize a quarter-bridge configuration, even in the absence of external lead wire resistances. This resistance is specified from 30 mΩ to 60 mΩ, depending on the selected channel. This results in an uncompensated gain error on a 350 Ω bridge of 86 ppm to 172 ppm.

This error is not reflected in the quarter-bridge accuracy table, however, because it is assumed that lead wire compensation will be done to remove the effects of external lead wire resistance. That process will simultaneously and completely compensate for this resistance, as it is matched within each channel to within 5 mΩ. If the lead wire compensation is done via shunt calibration, the 30 mΩ possible difference between channels is unimportant, as each channel will undergo a unique shunt calibration. If, however, the lead wire compensation is done theoretically (as one might do if the connecting cable is well characterized), an average compensation of 129 ppm should be used for the internal resistance. This would leave a possible uncompensated error of only 43 ppm.

Automatic measurement is possible, conversely, using the traditional shunt calibration process. Shunt calibration is the process of placing a known resistance in parallel with one of the bridge elements to create a known simulated strain. For quarter-bridge configuration, this element is usually the internal completion resistor. In this method, the deviation of the actual measured strain

from the theoretical strain is then considered to be due to lead wire desensitization, and a compensation factor is determined. However, some strain purists argue that shunt calibration is actually meant to be a verification step and should not be used for this purpose. What is needed, then, is an automated way to determine the lead wire desensitization error without using shunt calibration. This can be accomplished using the vtex1629_set_lead_wire_resistance and vtex1629_set_half_bridge_lead_wire_desensitization functions. An additional API is available to help measure the lead wire resistance, beginning with firmware version 1.0: vtex1629_measure_lead_wire_resistance.

| **NOTE** | This functionality was added to the EX1629 after September 2006 in firmware version 1.0. To ensure compatibity, use the instrument's embedded web page interface or use the EX1629's vtex1629_revision_query function. |
|---|---|

***Shunt Calibration***

Shunt calibration can be performed with either the internal 55 kΩ resistor or an external resistor inserted into the front panel shunt connector. For highest accuracy, the value of this external resistor should be precisely known. The *connection resistance* characteristic refers to the series resistance of the switch mechanisms used to route the external resistor into the requested bridge circuit. This resistance must be considered in conjunction with the raw resistor value when determining the theoretical simulated strain of the shunt calibration process. The connection resistance is the same for local or remote connection.

***Quarter-Bridge Strain Measurement***

The strain dynamic range is slightly different for tension vs. compression. While the dynamic range of the voltage measurement circuitry is a balanced ±160 mV, the transfer function of strain-to-voltage is nonlinear for quarter-bridge configuration, and that results in the small disparity.

Quarter-bridge measurement accuracy is noted as being exclusive of lead wire desensitization errors. This refers to the gain error that is generated by lead wire resistance in series with the actual strain gage. Depending on the length and gauge of the wire employed, the resultant resistance can cause errors that are much larger than the underlying instrument accuracy. Precision measurements consequently demand that lead wire compensation is conducted to eliminate this error. This compensation can be calculated theoretically if the resistance of the connection wire is characterized or can be inferred through the shunt calibration process. Reference the *Bridge Completion* paragraph above for information on how to compensate for the lead wire resistance that is internal to the EX1629.

***Quarter-Bridge Strain Measurement/Full-Bridge Strain Measurement***

The key concepts regarding the conditions on the strain measurement accuracy tables are:

- The stability effects of the excitation source are already captured in the listed accuracies. It is thus not necessary to combine the two to create a total system accuracy. The listed strain accuracies are already for the entire EX1629 measurement system.

- The accuracies do assume that the excitation voltage is measured and used in the EU conversion before the initiation of strain measurements. This eliminates its set point accuracy from being an error source.

- The restriction of 30 days, ±5 °C is used throughout the conditions. Essentially the intention is to describe a strain test setup that is initiated with a self-calibration, excitation voltage measurement, and unstrained voltage measurement. From that point, continuous strain measurements are considered to be taken over ambient conditions of 30 days, ±5 ºC.

The advantage to specifying in this way is that it provides maximum values that correspond to real world conditions. For in a production environment, it is impractical to allow, for example, only a ±1 ºC temperature swing. Similarly, since many strain tests can be very long in duration, it is impractical to demand a daily self-calibration or excitation voltage measurement.

The disadvantage, however, is that these assumptions result in measurement uncertainties that are overstated for test sequences that are significantly shorter in duration and subject to less environmental movement. However, since the gain and offset temperature coefficients are provided, the resultant performance improvements can be interpolated.

#### *Wideband Outputs*

The offset accuracy of the wideband outputs is provided referred to input (RTI). This is the accuracy of the calculated input signal. When considered in its raw form at the connector, namely referred to output (RTO), the listed accuracies must be multiplied by the gain of the range used.

The bandwidth specifications for each range were generated for an output sine wave signal of 2 V peak-to-peak. Very large signals (as a percentage of range) will encounter slew rate limiting and have a lower effective bandwidth.

#### *Confidence Measurements*

There are two specifications listed for excitation voltage measurement. The ±*excitation voltage* accuracy refers to the uncertainty of each excitation source measurement as an independent measurement. Conversely, the *total excitation voltage* accuracy refers to the uncertainty of the combined excitation source measurement, specifically the difference between the +excitation voltage and the –excitation voltage. This quantity has a lower uncertainty, because, as a difference measurement, it is subject to fewer error sources. While these performance characteristics are provided, they should not be added to the listed quarter-bridge and full-bridge accuracy tables, as these accuracy tables already contain the effects of the excitation voltage measurement.

The excitation current measurements are defined such that current flowing out of the source is positive current and current flowing into the source is negative current. Consequently, the +*excitation current* quantity is nominally a positive number, and the –*excitation current* quantity is nominally a negative number. Moreover, the current measurements are defined to be the total current of the source, not just that flowing in the external bridge. As shown in Figure 2-3, there is a 20 kΩ resistance, represented by the back-half resistors, that is always connected between the excitation sources. As a result, nonzero excitation source values will generate nonzero excitation currents even if the input channel is open. For example, a total excitation voltage of 5 V on an unpopulated channel will nominally display excitation currents of ±250 µA.

## MAXIMIZING MEASUREMENT PERFORMANCE

This section discusses tips and procedures that can help maximize the actual performance realized with the EX1629 and aid the user in avoiding some common pitfalls associated with strain gage measurement.

#### *Utilize self-calibration*

Self-Calibration should be conducted as often as practical, especially if the ambient environment has changed significantly since the previous calibration. However, fast ambient environmental changes should ideally be followed by a period of thermal stabilization before conducting self-calibration. The self-calibration process completes quickly and does not require removal of the actual input connections, making it convenient to run often.

***Utilize excitation measurement***

Measurement accuracy is notably improved by utilizing the EX1629's ability to measure its excitation source and use the measurement in the EU conversion. By doing so, the set point accuracy of the excitation source ceases to be an error source. The strain accuracy tables are based on the assumption that excitation measurement is performed.

While a tightly regulated supply, the excitation source does, nonetheless, have a temperature drift characteristic. For this reason, it is best to conduct the excitation source measurement just prior to the initiation of strain measurements, making the source measurement as current as possible.

***Utilize proper strain gage wiring techniques***

In addition to the accuracy of the measuring instrument, the total system accuracy of the strain measurement is a function of the gage characteristics and the connection wiring of the gage to the measuring instrument. Nonideal wiring techniques can create measurement inaccuracies far above those of the measuring instrument.

For half- and full-bridge configurations, it is highly recommended that the remote excitation sense lines be used on the excitation source, as shown in Figure 2-4 and Figure 2-5, respectively. These lines should be connected at the same point that the ±Excitation lines are connected to the bridge. Ultimately, the excitation source regulates based on the voltage present on its sense lines. Without remote sense, this regulation point is at the EX1629 input connector. This is a nonideal connection because the lead wire resistance between the EX1629 and the bridge will create a voltage drop, lowering the effective excitation value at the bridge. Remote sense inherently compensates for the lead wire resistance and delivers the correct excitation value.

For quarter-bridge configurations, it is highly recommended that the full 3-wire connection be used, as shown in Figure 2-3. Specifically, it is important that the –Sense line be connected at the gage, instead of locally at the EX1629 input connector. The 3-wire connection reduces the total lead wire resistance seen by the gage by putting half of it in series with the completion resistor. Not only does this reduce the static lead wire desensitization error, but it also provides an inherent level of temperature compensation. Specifically, since the same lead wire resistance is in the active leg as well as the completion leg, any variation of the resistance due to temperature naturally occurs in both legs and cancels.

***Compensate for lead wire desensitization error***

Even if the 3-wire connection is employed in quarter-bridge configuration, there may still be significant error in the measurement, as there is lead wire resistance that does not move with the underlying strain, but is indistinguishable from the actual gage resistance. This results in the measured strain being systematically in error. Depending on the length and gauge of the wire employed, this error may be much larger than the underlying instrument accuracy. Precision measurements consequently demand that lead wire compensation is conducted to eliminate this error. The error arises from the fact that the measuring instrument cannot distinguish between the resistance of the lead wires and the resistance of the strain gage. Specifically, when the resistance of the gage changes under load, the measuring instrument reads a strain value lower than the true value, because part of the total resistance it considers to be the gage is not changing. The extent of the desensitization error is dependent on the resistance values of the gage and the lead wire, related by this equation:

$$error = \frac{R_{lead}}{R_{gage} + R_{lead}}$$

For example, lead wire resistance of 1 Ω on a 350 Ω gage causes a desensitization error of:

$$error = \frac{1}{350+1} = 0.02849$$

Because it is a gain error, it is easily compensated if the desensitization can be measured or calculated prior to the commencement of the strain testing. Depending on the abilities of the data collection mechanism, the compensation can be done as a post-acquisition mathematical operation, through manipulation of the gage factor in the calculation equation, or through internal gain compensation based on an inputted lead wire value.

But, regardless of how the compensation is conducted, the first step in the process is to determine the value of the desensitization for each measurement channel in the system. Depending on the consistency of the test setup and the accuracy desired, there are two common ways of determining the error. The most basic way is simply to measure the resistance value of the lead wire with a DMM during the gage installation process. However, for a large channel count application with little consistency in the distance between the sensors and the measuring instrument, this would require the manual measurement and tracking of hundreds of wires, an obviously arduous task. This compensation can be calculated theoretically if the resistance of the connection wire is characterized or can be inferred through the shunt calibration process, or by calling the vtex1629_set_lead_wire_resistance and vtex1629_set_half_bridge_lead_wire_desensitization functions. Please reference the *Bridge Completion* subsection in Section 1 for information on how to compensate for the lead wire resistance that is internal to the EX1629.

***Allow for thermal stabilization of the bridge***

Compared to other sensor measurements, bridge measurements inherently involve relatively high amounts of power dissipation. Power is dissipated in the excitation source, the completion resistor, as well as the gage itself. The power dissipation is not so much a problem as a change in power. Specifically, when the excitation source value is changed, the amount of power being dissipated in the various bridge elements changes. A power dissipation change then leads to a temperature change through the thermal impedance of each element. Temperature change then ultimately results in performance characteristic drift. Since bridge power is directly related to the excitation current amount, this issue is worsened by increasing the excitation voltage or decreasing the bridge resistance.

To achieve maximum performance, it is best to allow the system elements to thermally stabilize once the bridge configuration has been programmed and the excitation source set and enabled. This time will be largely driven by the characteristics of the strain gage chosen for the application, and should be determined empirically. Only after this time should the excitation and unstrained voltage measurements be conducted.

# SECTION 2

# PREPARATION FOR USE

## OVERVIEW

This section provides a step-by-step process for setting up the EX1629 for use. It covers hardware installation, input connections, and software installation.

## UNPACKING

When the EX1629 is unpacked from its shipping carton, the contents should include the following items:

- EX1629 High-Performance Strain Gage Instrument
- Power line cord
- *EX1629 User's Manual* (this manual)
- *VXI Technology, Inc. Drivers and Product Manuals* CD

All components should be immediately inspected for damage upon receipt of the unit.

## INSTALLATION LOCATION

The EX1629 is designed to be largely insensitive to external electrical, magnetic, and thermal disturbances. However, as with all precision instrumentation, certain precautions, if taken into consideration, can help achieve maximum performance.

1) The unit, particularly its front panel, should be located away from sources of extreme high or low temperatures.
2) The unit should be located away from sources of high magnetic fields such as motors, generators, and power transformers.

The EX1629 employs active cooling for maximum product reliability. Fans located at the rear of the instrument pull ambient air through holes on the sides of the chassis and exhaust it out through the rear. The unit's installation with regards to other instrumentation and mounting cabinetry should ensure that the intake holes and exhaust fans remain unobstructed.

## WARM-UP TIME

The specified warm-up time of the EX1629 is 60 minutes. If, however, the unit is being subjected to an ambient temperature change greater than 10 °C, extra stabilization time is recommended to achieve maximum performance.

## INPUT CONNECTIONS / WIRING

Extensive testing has resulted in the qualification of the standard RJ-45 telecom connector as the ideal low-cost connector for strain gages. Not only are these connectors reliable, but low-cost construction of custom length cables is also readily available. Reconfiguration or replacement of strain gage connections is as easy as connecting a telephone. An example cable connection is illustrated in Figure 2-1.

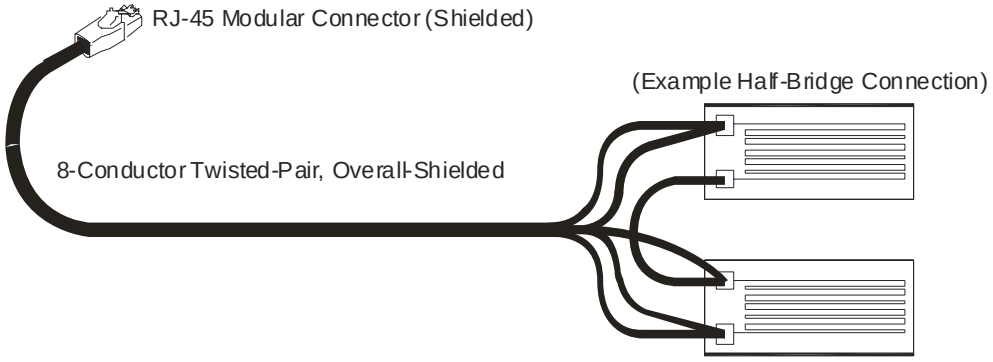



**FIGURE 2-1 CONNECTING STRAIN GAGES**

Figure 2-2 shows the pin assignment for each EX1629 strain gage connector. Depending on the bridge configuration employed, anywhere from three to all eight of the signal connections will be actively used, as illustrated in the bridge configuration diagrams that follow. For proper operation, unused input connections must be left open, as opposed to being grounded or tied together.

| **NOTE** | Unused user connections must be left open. |
|---|---|

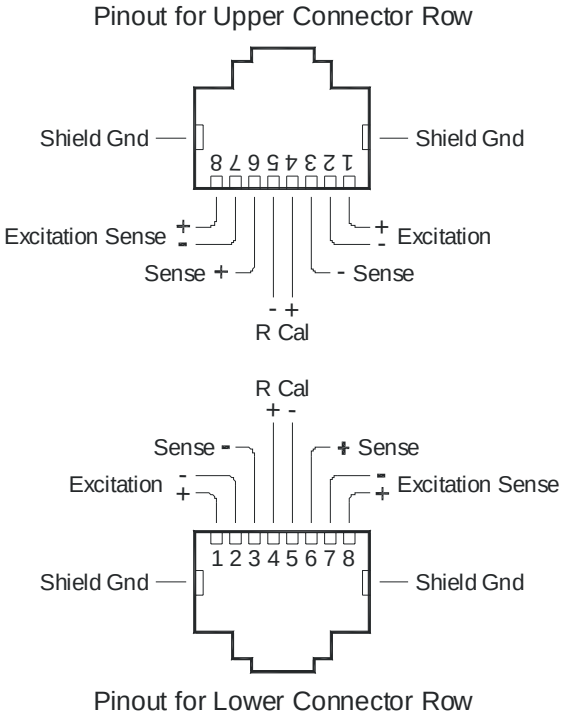



**FIGURE 2-2 INPUT CONNECTOR PIN ASSIGNMENT**

Depending on the bridge configuration and cable length employed, the wire gauge (and thus resistance) of the input connections can be a factor in determining ultimate system performance. For example, in a full-bridge configuration using excitation remote sense, the measurements are insensitive to the resistance of all connections. However, in a quarter-bridge configuration, the resistance of the +Excitation and –Excitation connections can be a significant error source if not properly compensated. For more details, see *Maximizing Measurement Performance* in Section 1.

## BRIDGE CONFIGURATIONS

Figure 2-3 illustrates how to connect a single strain gage in quarter-bridge configuration. As explained in *Maximizing Measurement Performance* in Section 1, the –Sense line should ideally be connected at the gage, instead of locally at the EX1629 input connector. Moreover, the wire length and gauge of the connections to Pins 1 and 2 should be matched. Fortunately, this is typically guaranteed by routing both lines as part of the same cable. For this configuration, the ±Excitation Sense lines are not used and must be left open, as opposed to being grounded or tied together. Whether the gage will be in tension or compression, the connection is the same.

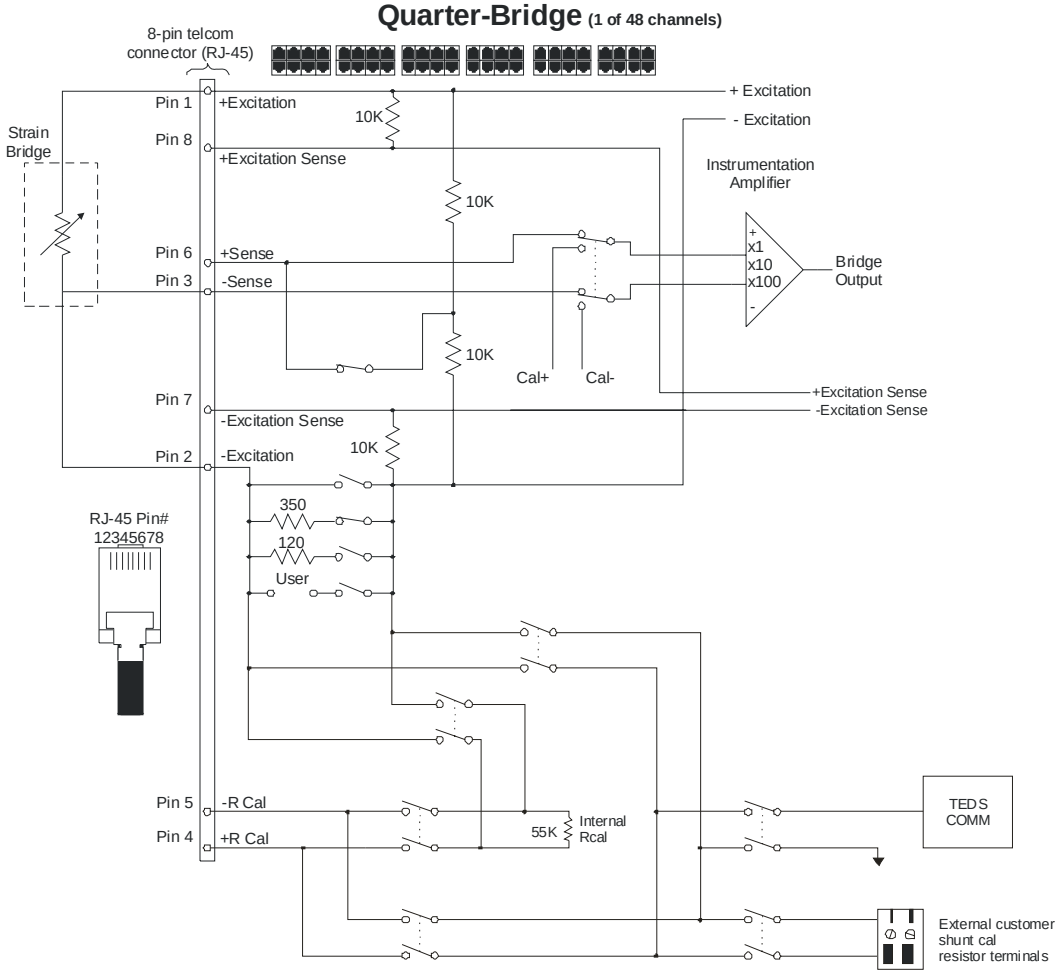



**FIGURE 2-3: QUARTER-BRIDGE CONFIGURATION**

| **NOTE** | The ±Excitation Sense lines must be left open. |
|---|---|

Figure 2-4 illustrates how to connect two strain gages in basic half-bridge bending configuration. As explained in *Maximizing Measurement Performance* in Section 1, the remote excitation sense lines should ideally be used on the excitation source, connected at the same point that the ±Excitation lines are connected to the bridge. The shunt calibration lines (Pins 4 and 5), however, are only necessary if that functionality is required. Note that the gage in tension is connected from Pin 1 to Pin 3 and the gage in compression is connected from Pin 2 to Pin 3.

Furthermore, it is critical to understand how the EX1629 defines its measurement in this configuration. Specifically, it defines the measured strain to be the strain that is present in each gage, not the total of the two gages. For example, if the upper gage is subjected to tensile strain of 1000 με and the lower gage is subjected to compressive strain of 1000 με, the EX1629 will measure a value of +1000 με, not +2000 με.

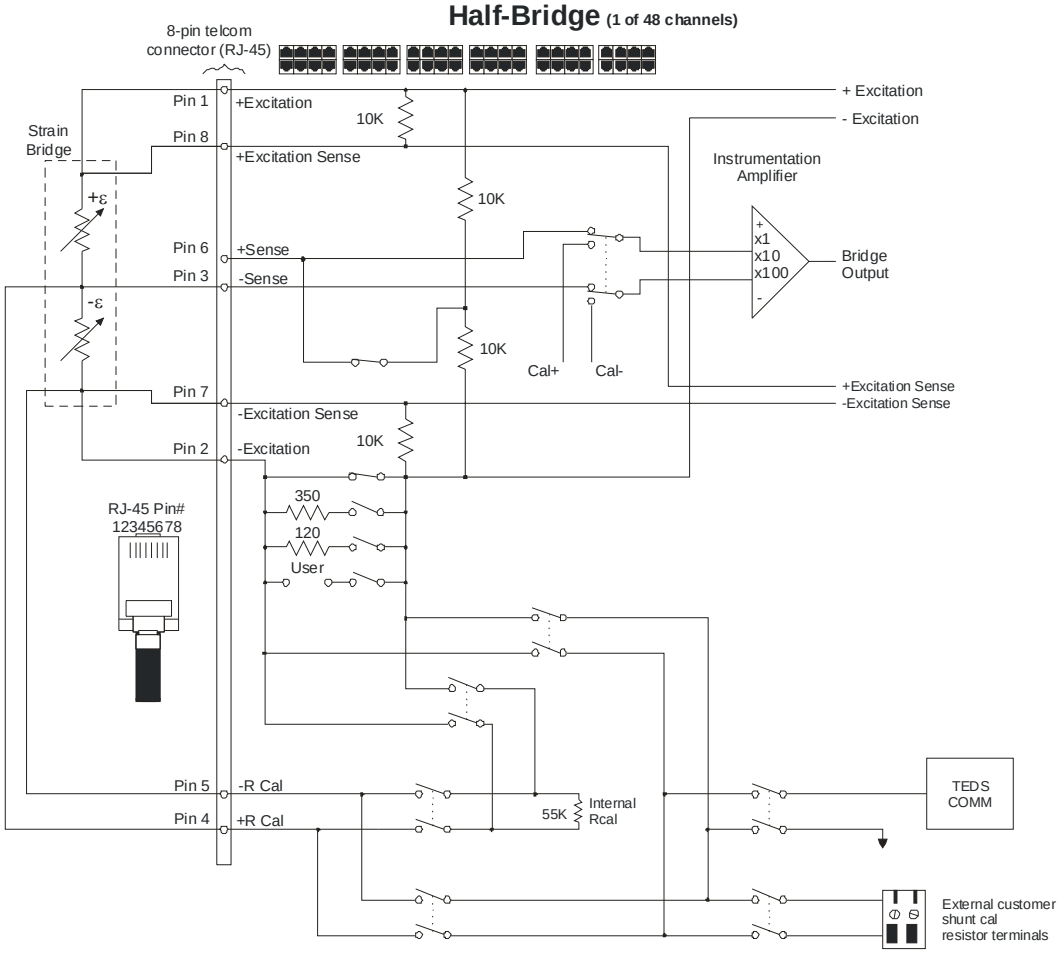



**FIGURE 2-4: HALF-BRIDGE CONFIGURATION**

Figure 2-5 illustrates how to connect four strain gages in basic full-bridge bending configuration. As explained in *Maximizing Measurement Performance* in Section 1, the remote excitation sense lines should ideally be used on the excitation source, connected at the same point that the ±Excitation lines are connected to the bridge. The shunt calibration lines (Pins 4 and 5), however, are only necessary if that functionality is required. Note that the gages in tension are connected from Pin 1 to Pin 6 and from Pin 2 to Pin 3 and the gages in compression are connected from Pin 2 to Pin 6 and from Pin 1 to Pin 3.

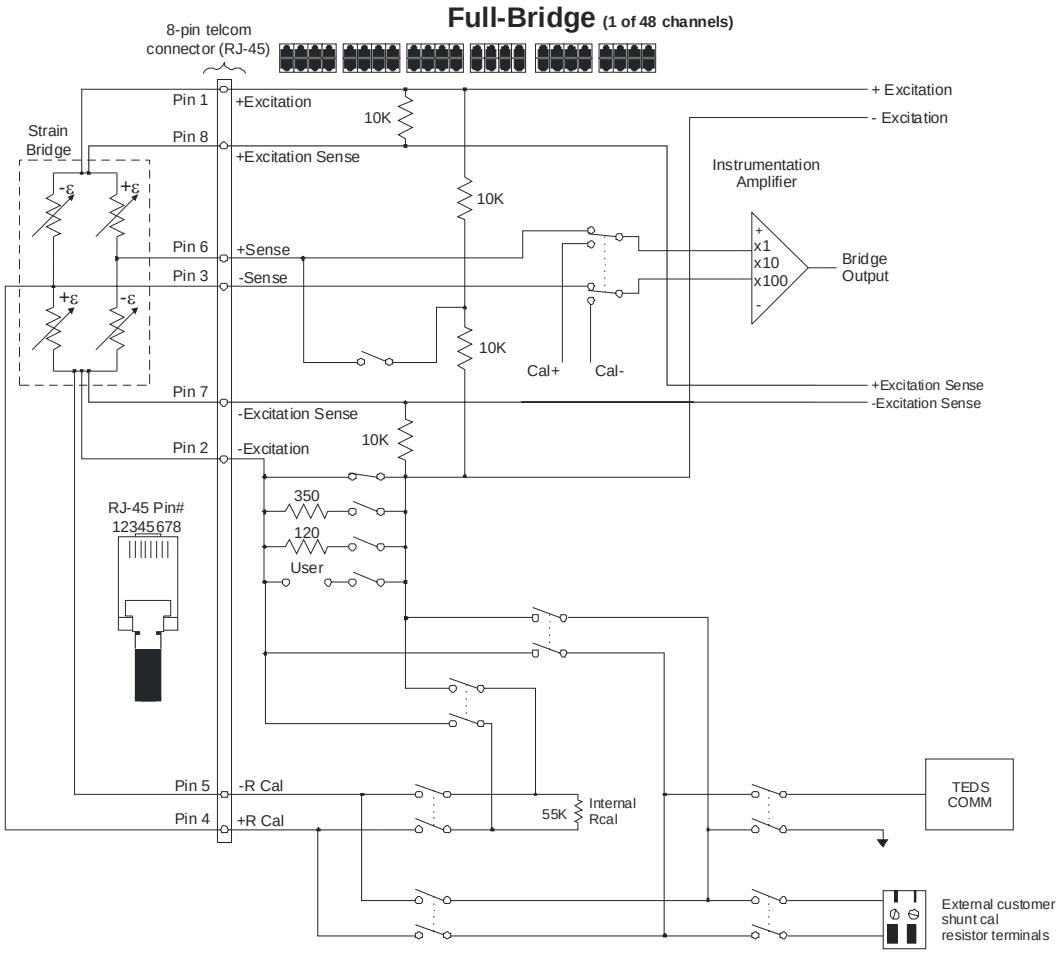



**FIGURE 2-5: FULL-BRIDGE CONFIGURATION**

Furthermore, it is critical to understand how the EX1629 defines its measurement in this configuration. Specifically, it defines the measured strain to be the strain that is present in each gage, not the total of the four gages. For example, if the positive gages are subjected to tensile strain of 1000 με and the negative gages are subjected to compressive strain of 1000 με, the EX1629 will measure a value of +1000 με, not +4000 με.

## VOLTAGE MEASUREMENT CONFIGURATIONS

The EX1629 main input channels can also be used for general voltage measurement. For this application, the channel is effectively configured for Full-Bridge measurements (i.e., no completion resistor or "back-half" of the bridge is enabled). The signal to be measured should be connected to the +Sense and –Sense lines. The excitation source may be used in this mode, if the measurement requires it, but the excitation source limits should be kept in mind.

For fully differential (floating) inputs, the configuration of Figure 2-6 should be used. For situations that require a grounded connection, the configuration of Figure 2-7 should be used, with the negative Excitation enabled and set to 0 V.

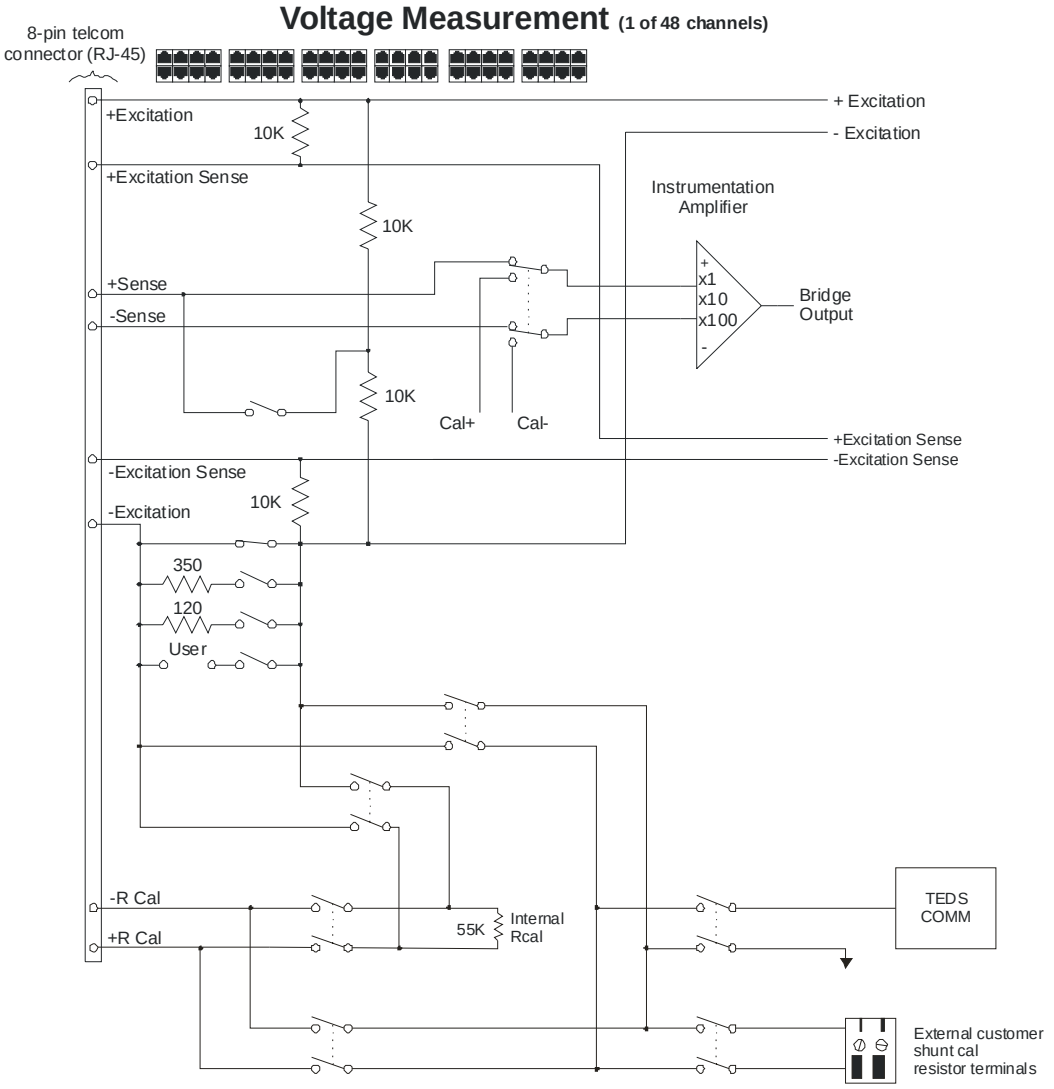
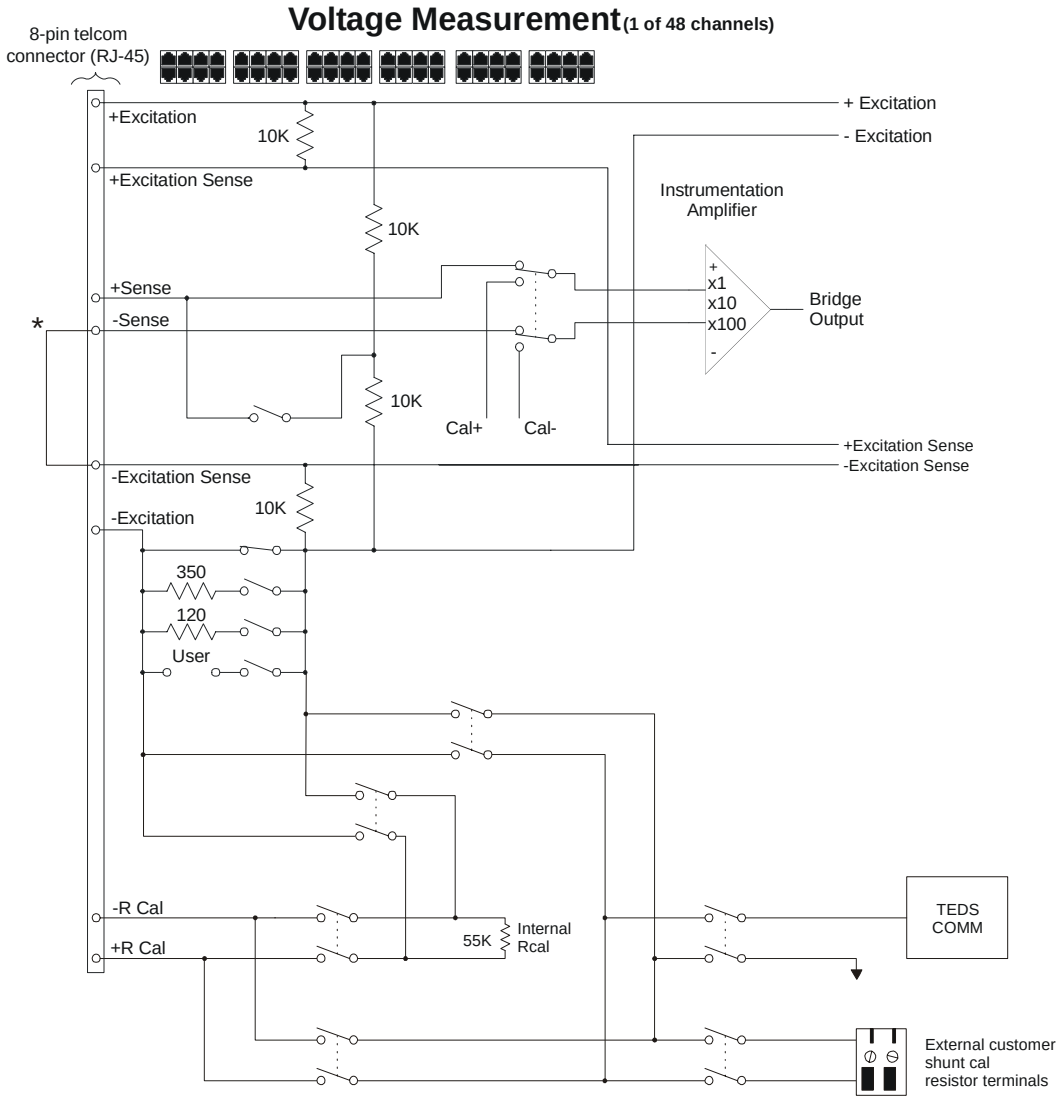



**FIGURE 2-6: VOLTAGE MEASUREMENT CONFIGURATION (FLOATING INPUT)**




* The **negativeExcitationVoltage** parameter of the vtex1629_set_excitation function must be set to 0 V.

**FIGURE 2-7: VOLTAGE MEASUREMENT CONFIGURATION (GROUNDED INPUT)**

## DRIVER INSTALLATION

The EX1629 is shipped with a *VXI Technology, Inc. Drivers and Product Manuals* CD which includes software drivers, user's manuals, the VXI Technology Product Catalog, as well as some third-party software. Please refer to the ReadMe.txt file included on the Distribution CD for installation instructions specific to revision of the CD provided.

## NETWORK CONFIGURATION

With its default network configuration, the EX1629 will attempt to locate a DHCP server. If one is found, the IP address assigned by the DHCP server will be used. Otherwise, after a timeout of 20 seconds, the unit will attempt to obtain an IP address by using AutoIP.

| **NOTE** | At any time, the EX1629 can be returned to a known, default network configuration by using the LCI (LAN Configuration Initialize) mechanism. See *Reset Button - LXI LAN Configuration Initialize (LCI) Mechanism* for more information. |
|---|---|

AutoIP is a mechanism for finding an unused IP address in the range 169.254.X.Y, where X is in the range 1 - 254 and Y is in the range 0 - 255. The device will first attempt to obtain the specific address 169.254.X.Y, where X and Y are the second-to-last and last octets (bytes) of the device's MAC address. However, X will be set to 1 if it is 0 in the MAC address, and to 254 if it is 255 in the MAC address. This is in accordance with the AutoIP standard (RFC 3927). If this address is already in use, the unit will attempt to obtain other IP addresses in a pseudorandom fashion until it finds one that is available.

To illustrate the AutoIP mechanism, Table 2-1 lists the AutoIP default address for some example MAC addresses.

| MAC Address | AutoIP Default Address |
|---|---|
| 00:0D:3F:01:00:01 | 169.254.1.1 |
| 00:0D:3F:01:01:01 | 169.254.1.1 |
| 00:0D:3F:01:A3:28 | 169.254.163.40 |
| 00:0D:3F:01:FE:FE | 169.254.254.254 |
| 00:0D:3F:01:FF:FE | 169.254.254.254 |

**TABLE 2-1: AUTOIP DEFAULT ADDRESS ASSIGNMENT**

If a static IP address assignment is preferred, one can be optionally assigned via the embedded web page interface. This is done by clicking the **Network Configuration** link, disabling DHCP and AutoIP, enabling Static, and then assigning a static IP address, subnet mask, and gateway address, and, optionally up to three DNS servers (see Figure 5-2). For more information, see *Network Configuration* in Section 5.

# SECTION 3

# BASIC OPERATION

## INTRODUCTION

This section expands on the description of the EX1629's features and explains how to best use them.

## ENGINEERING UNIT (EU) CONVERSION

Each EX1629 input channel can be individually configured for one of eleven different preset, standard EU conversions. Each of these conversions is described below. Setting a specific conversion not only controls the mathematical operations applied to the acquisition data, but also automatically configures elements of the signal conditioning path. For example, setting a quarter-bridge 350 conversion enables the 350 Ω completion resistor and connects the back-half resistors to the +Sense measurement line, as illustrated in Figure 2-3. The automatic configuration, however, can be overridden, as these elements can be configured independently.

Common terms used in the conversion equations are the following:

ε = strain
$V_{backhalf}$ = the voltage sensed from the back-half resistor network
$V_{-sense}$ = the voltage sensed on the –Sense input pin
$V_{+sense}$ = the voltage sensed on the +Sense input pin
$V_{unstrained}$ = the unstrained voltage (measured or manually entered)
$V_{excitation}$ = the excitation voltage (measured or manually entered)
GF = gage factor
ν = Poisson ratio

### *Quarter-Bridge 350, Quarter-Bridge 120, Quarter-Bridge User*

These conversions apply to the following bridge configuration:

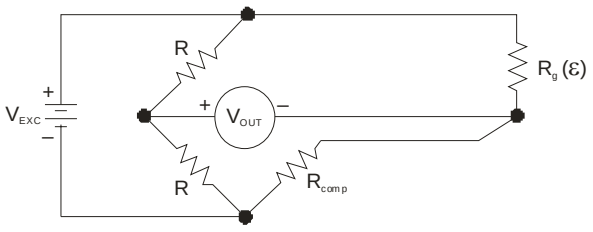



**FIGURE 3-1: QUARTER-BRIDGE CONFIGURATION**

The specific conversion is selected to match the nominal resistance of the active gage.

The quarter-bridge strain conversion is calculated according to:

$$V_{\text{diff}} = V_{\text{backhalf}} - V_{-\text{sense}}$$

$$V_r = \frac{V_{\text{diff}} - V_{\text{unstrained}}}{V_{\text{excitation}}}$$

$$\varepsilon = \frac{4 \cdot V_r}{GF \cdot (1 - 2 \cdot V_r)}$$

Setting this conversion automatically configures the input path for quarter-bridge mode, in which the required completion resistor is enabled and the back-half resistors are connected. This configuration is illustrated in Figure 2-3.

Any of these quarter-bridge conversions also apply to the following bridge configuration, where the dummy gage is utilized for temperature compensation of the active gage:

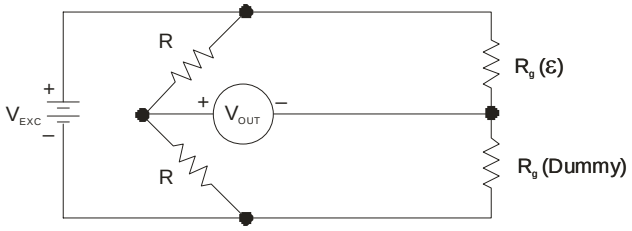



**FIGURE 3-2: QUARTER-BRIDGE WITH DUMMY GAGE**

This is simply a quarter-bridge configuration where the completion resistor is external to the instrument (since only one leg of the bridge is active, it is a quarter-bridge, not a half-bridge which would contain two strain gage resistors). To configure a channel for this use, the channel's EU conversion should be set to quarter-bridge (any of the 120, 350, or User configurations can be used, since the nominal resistance only effects which completion resistor is selected), the input multiplexer should be set to half-bridge, and then the completion resistor should be set to full. While this configuration appears similar to a half-bridge configuration, it is different in that the quarter-bridge conversion equation is used. This configuration may be used with any external completion resistor.

### *Half-Bridge Bending*

This conversion applies to the following bridge configuration:

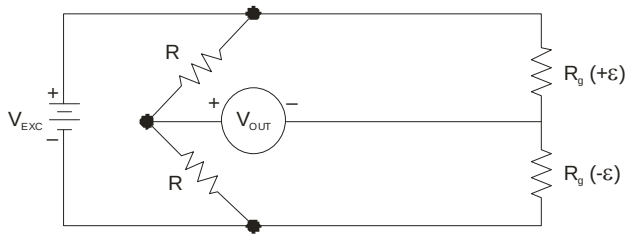



**FIGURE 3-3: HALF-BRIDGE BENDING CONFIGURATION**

The half-bridge strain conversion is calculated according to:

$$V_{\text{diff}} = V_{\text{backhalf}} - V_{\text{-sense}}$$
$$V_r = \frac{V_{\text{diff}} - V_{\text{unstrained}}}{V_{\text{excitation}}}$$
$$\varepsilon = \frac{2 \cdot V_r}{GF}$$

Setting this conversion automatically configures the input path for half-bridge mode, in which the completion resistor is shorted and the back-half resistors are connected. This configuration is illustrated in Figure 2-4.

### *Half-Bridge Poisson*

This conversion applies to the following bridge configuration:

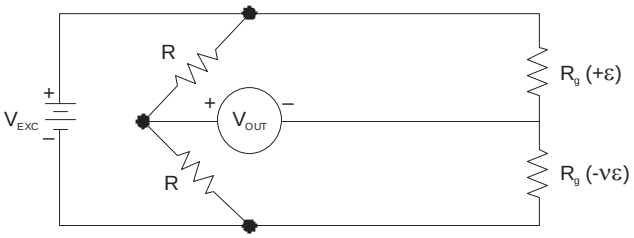



**FIGURE 3-4: HALF-BRIDGE POISSON CONFIGURATION**

The half-bridge strain conversion is calculated according to:

$$V_{\text{diff}} = V_{\text{backhalf}} - V_{\text{-sense}}$$
$$V_r = \frac{V_{\text{diff}} - V_{\text{unstrained}}}{V_{\text{excitation}}}$$
$$\varepsilon = \frac{4 \cdot V_r}{GF \cdot \left((\upsilon + 1) + 2 \cdot V_r \cdot (\upsilon - 1)\right)}$$

Setting this conversion automatically configures the input path for half-bridge mode, in which the completion resistor is shorted and the back-half resistors are connected. This configuration is illustrated in Figure 2-4.

### *Full-Bridge Bending*

This conversion applies to the following bridge configuration:

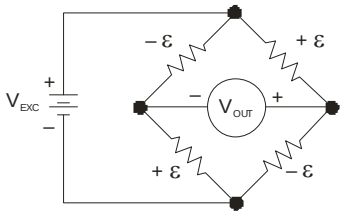



**FIGURE 3-5: FULL-BRIDGE BENDING CONFIGURATION**

The full-bridge strain conversion is calculated according to:

$$V_{\text{diff}} = V_{\text{+sense}} - V_{\text{-sense}}$$
$$V_r = \frac{V_{\text{diff}} - V_{\text{unstrained}}}{V_{\text{excitation}}}$$
$$\varepsilon = \frac{-V_r}{GF}$$

Setting this conversion automatically configures the input path for full-bridge mode, in which the completion resistor is shorted and the back-half resistors are disconnected. This configuration is illustrated in Figure 2-5.

### *Full-Bridge Poisson*

This conversion applies to the following bridge configuration:

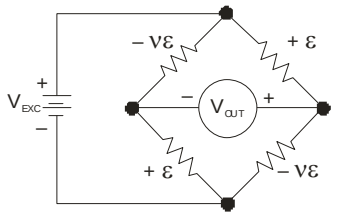



**FIGURE 3-6: FULL-BRIDGE POISSON CONFIGURATION**

The full-bridge strain conversion is calculated according to:

$$V_{\text{diff}} = V_{\text{+sense}} - V_{\text{-sense}}$$
$$V_r = \frac{V_{\text{diff}} - V_{\text{unstrained}}}{V_{\text{excitation}}}$$
$$\varepsilon = \frac{-2 \cdot V_r}{GF \cdot \left((\nu + 1) - V_r \cdot (\nu - 1)\right)}$$

Setting this conversion automatically configures the input path for full-bridge mode, in which the completion resistor is shorted and the back-half resistors are disconnected. This configuration is illustrated in Figure 2-5.

### *Full-Bridge Bending Poisson*

This conversion applies to the following bridge configuration:

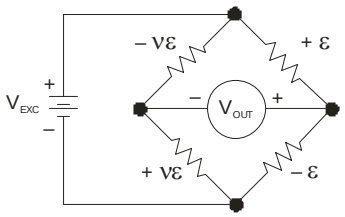



**FIGURE 3-7: FULL-BRIDGE BENDING POISSON CONFIGURATION**

The full-bridge strain conversion is calculated according to:

$$V_{\text{diff}} = V_{\text{+sense}} - V_{\text{-sense}}$$
$$V_r = \frac{V_{\text{diff}} - V_{\text{unstrained}}}{V_{\text{excitation}}}$$
$$\varepsilon = \frac{-2 \cdot V_r}{GF \cdot (\nu + 1)}$$

Setting this conversion automatically configures the input path for full-bridge mode, in which the completion resistor is shorted and the back-half resistors are disconnected. This configuration is illustrated in Figure 2-5.

***Voltage***

This simply returns the differential voltage, $V_{diff}$.

$$V_{\text{diff}} = V_{\text{+sense}} - V_{\text{-sense}}$$

Setting this conversion automatically configures the input path for full-bridge mode, in which the completion resistor is shorted and the back-half resistors are disconnected. This configuration is illustrated in Figure 2-6.

***Ratiometric***

This performs a scaling of the differential voltage, $V_{diff}$, according to:

$$V_{\text{diff}} = V_{\text{+sense}} - V_{\text{-sense}}$$
$$V_r = \frac{V_{\text{diff}} - V_{\text{unstrained}}}{V_{\text{excitation}}}$$

Setting this conversion automatically configures the input path for full-bridge mode, in which the completion resistor is shorted and the back-half resistors are disconnected.

***Linear***

This performs a scaling of the differential voltage, $V_{diff}$, according to:

$$V_{\text{diff}} = V_{\text{+sense}} - V_{\text{-sense}}$$
$$V = GF \cdot V_{\text{diff}} + V_{\text{unstrained}}$$

Setting this conversion automatically configures the input path for full-bridge mode, in which the completion resistor is shorted and the back-half resistors are disconnected.

***Nonstandard***

As previously mentioned, there are potentially desirable configurations that are not covered by the standard conversions. An example of this is a quarter-bridge configuration that provides voltage output instead of strain. Fortunately, the EX1629 provides the flexibility to create virtually any bridge measurement configuration. Creating a nonstandard EU conversion begins by selecting the standard conversion that provides the desired conversion equation. Following that, the nominal settings of the completion resistor and input multiplexer can be changed to fit the requirements of the application.

The default selection is voltage.

## COMPLETION RESISTOR

In a standard quarter-bridge strain application, the completion resistor value must match the nominal resistance of the active strain gage. The required completion resistor is normally enabled through the appropriate setting of the EU conversion. For example, setting a quarter-bridge 350 conversion automatically enables the 350 Ω completion resistor. However, there are potentially desirable configurations that are not covered by the standard conversions. An example of this is a quarter-bridge configuration that provides voltage output instead of strain (this is sometimes useful as a "sanity" check). In that case, the completion resistor must be specifically configured.

Creating a nonstandard EU conversion begins by selecting the standard conversion that provides the desired conversion equation. Following that, the nominal settings of the completion resistor and input multiplexer can be changed to fit the requirements of the application.

The available settings for the completion resistor are full, 120, 350, user, and off. 120 and 350 refer to the 120 Ω and 350 Ω resistor paths. Full refers to a low-impedance connection between the –excitation source and the –Excitation connection (Pin 2 of the input connector). The typical resistance in full mode is 125 mΩ. User refers to the user-specified resistor path, which may be populated as a factory option. Off refers to the disabling of all of the resistor paths.

The default value of the completion resistor is full.

## INPUT MULTIPLEXER

The connection of the EX1629's signal conditioning circuitry is governed by the setting of its input multiplexer, which connects the measurement path to the input measurement lines (±Sense), the back-half resistors, the calibration source, or ground. The required input multiplexer configuration is normally controlled through the appropriate setting of the EU conversion. For example, setting a quarter-bridge 350 conversion automatically configures the input multiplexer for quarter mode, as shown in Figure 2-3. However, there are potentially desirable configurations that are not covered by the standard conversions. An example of this is a quarter-bridge configuration that provides voltage output instead of strain. In that case, the input multiplexer must be specifically configured.

Creating a nonstandard EU conversion begins by selecting the standard conversion that provides the desired conversion equation. Following that, the nominal settings of the completion resistor and input multiplexer can be changed to fit the requirements of the application.

The available settings for the input multiplexer are shown in Table 3-1, mapped to the specific connections of the instrumentation amplifier inputs.

| Setting | (+) Input | (-) Input |
|---|---|---|
| full | +Sense | -Sense |
| half | back-half | -Sense |
| quarter | back-half | -Sense |
| cal‡ | cal source | cal source |
| gnd‡ | ground | Ground |
| ‡ *For factory use only.* | | |

**TABLE 3-1: INPUT MULTIPLEXER SETTINGS**

The default value of the input multiplexer is full.

## COMPLETION RESISTOR/INPUT MULTIPLEXER DEFAULT SETTINGS

Table 3-2 contains the default input multiplexer and completion resistor values for each EU conversion.

| EU Conversion | Completion Resistor | Input Multiplexer |
|---|---|---|
| Voltage | Full | Full |
| Quarter 120 Ω | 120 Ω | Back-half |
| Quarter 350 Ω | 350 Ω | Back-half |
| Quarter User | User | Back-half |
| Half Bending | Full | Back-half |
| Half Poisson | Full | Back-half |
| Full Bending | Full | Full |
| Full Poisson | Full | Full |
| Full Bending Poisson | Full | Full |
| Ratiometric | Full | Full |
| Linear | Full | Full |

**TABLE 3-2: DEFAULT COMPLETION RESISTOR/INPUT MULTIPLEXER VALUES**

## GAGE FACTOR / POISSON RATIO

As illustrated in the *Engineering Unit (EU)* Conversion subsection, there are two constants utilized in the EU strain conversions: gage factor and Poisson ratio.

The gage factor (GF), a measure of strain gage sensitivity, is a dimensionless quantity defined as the ratio of the fractional change in resistance to the fractional change in length along the primary axis of the strain gage. Mathematically, this is expressed as:

$$GF = \frac{\Delta R / R}{\Delta L / L} = \frac{\Delta R / R}{\varepsilon}$$

The gage factor value for a specific strain gage is provided by the strain gage manufacturer. The default gage factor is 2.0.

The Poisson ratio (ν), in simple terms, is a measure of the extent to which a material contracts as it is being stretched. In engineering terms, it is a dimensionless quantity defined as the ratio of transverse contraction strain to longitudinal extension strain in the direction of the stretching force. Mathematically, this is expressed as:

$$\nu = \frac{-\varepsilon_{trans}}{\varepsilon_{long}}$$

The Poisson ratio value for a specific material should be obtained from a mechanical engineering reference. The default Poisson ratio is 0.3.

## MEASUREMENT RANGE / GAIN

Each EX1629 input channel can be individually configured with respect to its signal conditioning gain. The differential voltage measurement range of the EX1629 is ±150 mV, ±1.5 V, or ±15 V, for gain settings of 100, 10, and 1, respectively. As strain measurements will nearly always be conducted at a gain of 100, its measurement range is primarily a function of bridge configuration and excitation voltage level. The extent of initial bridge imbalance, which is reflected in the unstrained voltage measurement, is a secondary factor, but it is normally not large enough to be significant. Nominal measurement ranges for some common bridge configurations are listed in Table 3-3.

| Bridge Configuration | Excitation | Range |
|---|---|---|
| Quarter | 10 V | +30927 με/-29126 με |
| Quarter | 5 V | +63829 με/-56603 με |
| Half | 10 V | ±15000 με |
| Half | 5 V | ±30000 με |
| Full | 5 V | ±15000 με |
| Full | 2.5 V | ±30000 με |

**TABLE 3-3: EX1629 MEASUREMENT RANGE**

For quarter-bridge configuration, note that the dynamic range is slightly different for tension vs. compression. While the dynamic range of the voltage measurement circuitry is a balanced ±150 mV, the transfer function of strain-to-voltage is nonlinear, and that results in the small disparity.

The default gain setting is 1, although most strain gage measurements are taken using a gain of 100.

## EXCITATION SOURCE

Each EX1629 input channel features an independent excitation source that is not only programmable on a per channel basis, but also with respect to the positive and negative supplies that compose the total excitation voltage. This programming independence provides the flexibility of balanced or unbalanced excitation. Specifically, the positive and negative excitation voltages are programmable from 0 V to +8 V and 0 V to -8 V, respectively, with a total current capability of 50 mA per channel. Overcurrent protection is 60 mA. Moreover, each source is independently regulated, such that an overcurrent condition on one channel does not affect the regulation of any other channels.

The operations to program and enable each excitation voltage are discrete. Excitation voltages that are not enabled output an actual value of 0 V, regardless of their programmed value.

When the excitation source is changed, the nominal value of the total excitation voltage is updated in the EU strain conversions. However, for highest accuracy, the excitation voltage should be measured and the measurement used in the EU conversion (see the *Excitation Source Measurement* section that follows). The strain accuracy tables are based on the requirement that excitation measurement is performed.

For highest accuracy in half-bridge and full-bridge configurations, each excitation source has a remote sense connection. In order to properly remove the effects of lead wire resistance, these lines should be connected at the same point that the ±Excitation lines are connected to the bridge. The remote sense lines are always active in the circuitry; there is no control to turn on/off remote sense. Because of this, it is critical that they be left open (unconnected) in quarter-bridge configuration, where their connection would be invalid.

As explained in *Maximizing Measurement Performance* in Section 1, because of the relatively high levels of power dissipation involved, it is best to allow the bridge system elements to thermally stabilize after an excitation source change. Excitation and unstrained voltage measurements taken after an appropriate delay will demonstrate improved stability during the subsequent strain testing.

| **NOTE** | For maximum measurement performance, an excitation source change should be followed by a thermal stabilization delay before the excitation and unstrained voltage measurements are performed. |
|---|---|

The default programmed values for the excitation sources are 0 V. The default enable states for the excitation sources are off.

## EXCITATION SOURCE MEASUREMENT

For highest measurement accuracy, the EX1629 provides the ability to measure its excitation source and use the measurement in the EU conversion. By doing so, the set point accuracy of the excitation source ceases to be an error source. The strain accuracy tables are based on the requirement that excitation measurement is performed.

An excitation source measurement is composed of two confidence system measurements: a measurement of the +excitation voltage followed by a measurement of the –excitation voltage. The total excitation voltage is then calculated as $(+V_{exc}) - (-V_{exc})$. Through program control, the user dictates whether the measurement is used in the EU strain conversions, taking the place of the nominal excitation value. However, because the measurement system accuracy exceeds the set point accuracy of the source, overall strain accuracy is always improved by using the source measurement in the EU conversion.

The excitation source measurement also provides control over whether the local sense or remote sense lines of the excitation source are measured. If the remote sense lines are not connected to the external strain bridge, such as in quarter-bridge configuration, either setting can be used. The values in either case are the same. However, if the remote sense lines are connected to the bridge, as they ideally should be in half- or full-bridge configuration, the remote sense lines should be measured, as they represent the true source output seen by the bridge.

While a tightly regulated supply, the excitation source does, nonetheless, have a temperature drift characteristic. For this reason, it is best to conduct the excitation source measurement just prior to the initiation of strain measurements, making the source measurement as fresh as possible. Moreover, to achieve maximum performance, it is best to allow the system elements to thermally stabilize following an excitation source change before conducting its measurement. For more details, see *Maximizing Measurement Performance* in Section 1.

| **NOTE** | For maximum measurement performance, an excitation source change should be followed by a thermal stabilization delay before the excitation source measurement is performed. |
|---|---|

While measuring the excitation source is the conventional method of providing a non-nominal value of the excitation voltage to the EU conversion, it is also possible to manually enter a value. This would normally only be done for system diagnostic purposes.

## UNSTRAINED VOLTAGE MEASUREMENT

Integral to strain measurement, the unstrained voltage measurement mathematically removes the effects of initial bridge imbalance by measuring the bridge voltage with the gage(s) in an unstrained state and using the resultant value in the EU conversion. For while an unstrained bridge would ideally have an output of 0 V, component tolerances of the gages and, if employed, the completion resistor produce a nonzero output. Performing an unstrained voltage measurement

eliminates these tolerances as an error source. The strain accuracy tables are based on the requirement that unstrained voltage measurement is performed.

The unstrained voltage measurement should only be taken once the bridge configuration and excitation source parameters are established. It is critical that the operating conditions under which the unstrained voltage measurement is taken be identical to those present during strain measurement. That is, an unstrained voltage measurement taken before an excitation source change would be mathematically invalid.

| **NOTE** | The unstrained voltage measurement should only be taken once the bridge configuration and excitation source parameters are established. |
|---|---|

The EX1629 components responsible for the unstrained voltage are primarily the completion resistor and the main bridge measurement circuitry. And while they are of extremely high quality, there is, nonetheless, a small temperature drift characteristic. For this reason, it is best to perform the unstrained voltage measurement just prior to the initiation of strain measurements, making it as fresh as possible. Moreover, to achieve maximum performance, it is best to allow the system elements to thermally stabilize following an excitation source change before conducting its measurement. For more details, see *Maximizing Measurement Performance* in Section 1.

| **NOTE** | For maximum measurement performance, an excitation source change should be followed by a thermal stabilization delay before the unstrained voltage measurement is performed. |
|---|---|

While measuring the unstrained voltage is the conventional method of providing a value to the EU conversion, it is also possible to manually enter a value. This would normally only be done for system diagnostic purposes.

### SCAN LIST CONFIGURATION

The EX1629 can be configured to include from 1 to all 48 of its input channels in the scan list. Because of the channel independence present in the EX1629 design, there are no accuracy, noise, or speed ramifications from the structure of the scan list. Its channel entries can consequently be solely dictated by the user's application requirements. Moreover, since the EX1629 features an independent A/D converter per channel, input channels are, in a sense, not "scanned" at all. Instead, the scan list simply dictates the specific data to be returned. The term "scan list" is borrowed from traditional scanning (multiplexed) data acquisition systems. A valid scan list consists of:

- at least one channel
- not more than 48 channels
- no repeated channels

Each EX1629 input channel maintains its high input impedance and operational independence from the other channels regardless of its inclusion in the scan list.

### SAMPLING RATE

The EX1629 can be configured for a sampling rate from 1 Sa/s to 10000 Sa/s in 30 discrete settings, regardless of the number of channels in the scan list. This permits the tailoring of the data load to the dynamic requirements of the test application. The valid sample rates are 1, 2, 4, 5, 8, 10, 16, 20, 25, 33.33, 40, 50, 80, 100, 125, 166.67, 200, 250, 400, 500, 625, 833.33, 1000, 1250, 2000, 2500, 3125, 5000, 6125, 10000 and 12500 Sa/s. A rate of 25000 Sa/s can be achieved if 1) the number of active channels is limited to a maximum of 16 and 2) all active channels are on the same analog board (i.e. channels 0 through 15, 16 through 31, or 32 through 47 can be selected). Requested rates that fall between valid ones are rounded to the closest valid one.

Unique in its design, the EX1629 features a secondary "confidence" measurement system that runs in parallel with the main bridge measurement system. This confidence system, if enabled, produces data at a fixed rate of approximately 500 Sa/s. If the sampling rate is set at or below 500 Sa/s, the confidence system can produce fresh data for every main bridge output reading. If, however, the sampling rate is set above 500 Sa/s, the returned data will periodically be returned without confidence data.

The default setting is 1000.

## UNITS

Each EX1629 input channel can be individually configured with respect to the units of its strain conversions, either strain (ε) or microstrain (με). The default units setting is strain. The results of voltage and linear conversions are always in volts, and ratiometric conversions are unitless.

If the tare feature is employed, it should be noted that an entered tare value is unaffected by a change in the units setting. For example, if a tare value of 100 is entered when the units are microstrain, it will be considered as 100 με. If the unit's setting is then changed to strain, this tare value will be considered as 100 ε. To avoid this confusion, it is best to set the desired units first and then enter any tare value.

## TARE

Each EX1629 input channel can be assigned a tare, or relative, value. The tare value is subtracted from the nominal conversion results to produce the final readings, yielding measurements that are referenced to a nonzero point. Tare can be used on strain as well as non-strain EU conversions.

A subtle characteristic of the tare value is that it is not linked to the units setting of the instrument. For example, if a tare value of 100 is entered when the units are microstrain, it will be considered as 100 με. However, if the units setting is then changed to strain, this tare value will be considered as 100 ε. Similarly, if instead the conversion is changed to voltage, this tare value will be considered as 100 V. To avoid this confusion, it is best to set the desired EU conversion and units first and then enter any tare value.

Tare values apply only to main bridge conversions, not to confidence system measurements.

The default tare value is 0.

## DIGITAL FILTER

The EX1629 finite impulse response (FIR) filters provide near Nyquist bandwidths as indicated in Table B-1. The passband ripple for sample frequencies ($f_s$) greater than or equal to 3125 Hz is -0.01 dB, while it is ±0.001 dB for $f_s < 3125$ Hz. The filters provide an alias rejection of 100 dB and have linear phase. Some sampling frequencies result in very high attenuation of 60 Hz signals. The FIR filters are solely a function of sampling and are not user configurable.

Each EX1629 channel can be individually configured with respect to infinite impulse response (IIR) filtering. The user can select either Butterworth or Bessel type filters, or may choose no IIR filter. If IIR filtering is turned on, the -3 dB frequencies can be chosen continuously from $f_s/1000$ to $f_c$ max as indicated in Table B-1. The order of the filters can be set by the user as well. Butterworth filters can be set from 0 - 10 and Bessel filters can be set from 1 to 10. When the user order is specified as 0, the EX1629 calculates the order based on an analog prototype Butterworth filter with -200 dB attenuation at $f_s/2$, given the sampling frequency and the -3 dB frequency. The on-board DSP will design IIR filters on-the-fly given a sampling frequency, the -3 dB frequency, a user order between 1 and 10, the filter type, and the transform.

The default settings are: Butterworth filter with -3 dB freq = 10 Hz, bilinear transform, and user order = 0 (calculated order = 6). More information on filtering can be found in *Appendix B*.

## TRIGGERING

The EX1629 supports a full function trigger model with a separate arm source and trigger source event structure. For a complete explanation of the trigger model, see Section 4. In summary, an acquisition sequence is enabled with a trigger initialize function. Measurement data is then acquired upon the receipt of the programmed arm source event followed by the receipt of the programmed trigger source event. Trigger and arm source events can be independently programmed from a variety of sources including Immediate, Timer, Digital I/O, and the Trigger Bus.

## DATA FORMAT

By default, the data returned during data retrieval is limited to the main bridge readings and the absolute time of scan initiation. If enabled, the EX1629 can also return data from its confidence measurement system, such as the excitation source voltages and excitation source currents.

## SHUNT CALIBRATION

The EX1629 features an extremely capable and fully programmable shunt calibration architecture to ensure correct bridge performance. Three discrete shunt calibration modes are supported that can be employed in local and remote connections. Figure 3-8 illustrates the shunt calibration design.

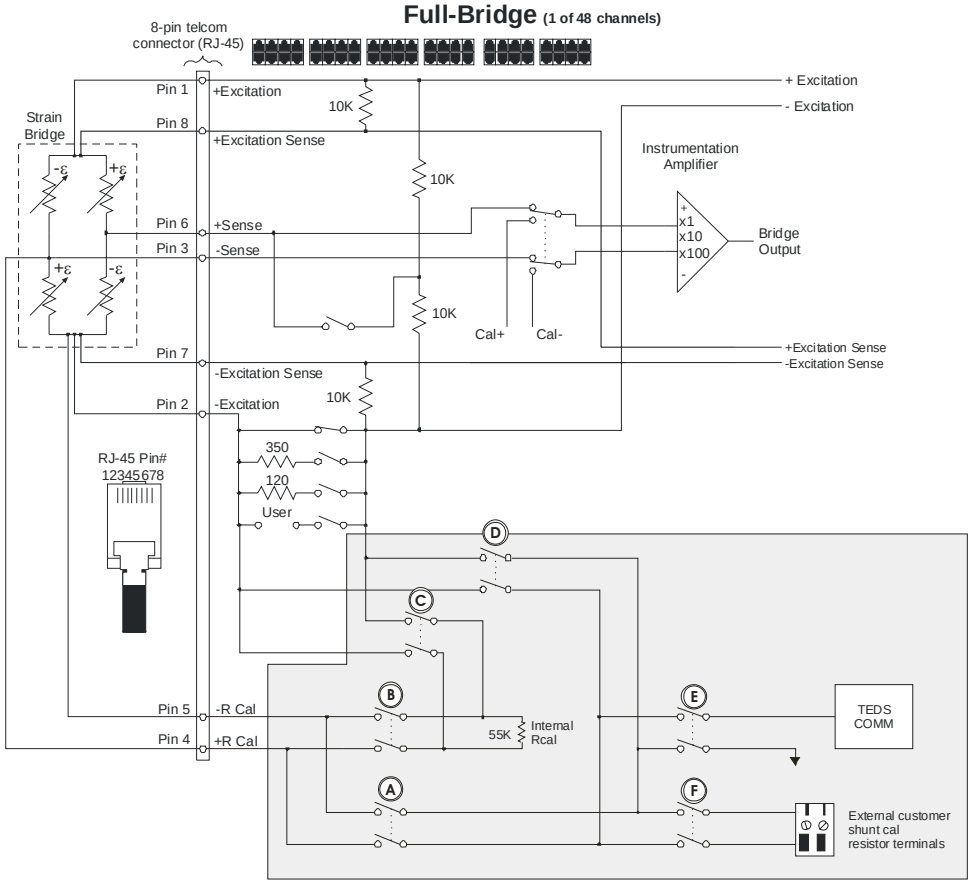



**FIGURE 3-8: SHUNT CALIBRATION CONFIGURATION**

Each input channel provides a unique, precision 55 kΩ resistor that can be connected internally (switch C) for quarter-bridge shunting or externally (switch B) for full- or half-bridge shunting. Since there is a resistor per channel, all 48 input channels may be shunted simultaneously.

If an alternate shunt resistor is required, an external resistor may be connected into each of three front panel connectors, one for each group of 16 input channels. Each resistor may be connected internally (switches D, F) for quarter-bridge shunting or externally (switches A, F) for full- or half-bridge shunting. Up to three front panel resistors may be used simultaneously, but not for more than one channel in each 16-channel block.

For highest accuracy, the value of this external resistor should be precisely known. The *connection resistance* characteristic in the specifications refers to the total resistance of the connection switches. This resistance must be considered in conjunction with the raw resistor value when determining the theoretical simulated strain of the shunt calibration process. The connection resistance is the same for local or remote connection.

Finally, the EX1629 offers the ability to read and control a TEDS-equipped remote resistor (switches A, E) for full- or half-bridge shunting. Up to three TEDS remote resistors may be used simultaneously, but not for more than one channel in each 16-channel block.

Table 3-4 summarizes the shunt calibration capabilities and the specific connection switches they utilize.

| Setting | Usage | Switches |
|---|---|---|
| front_panel_remote | 1 per 16 channels | A, F |
| front_panel_local | 1 per 16 channels | D, F |
| internal_remote | every channel | B |
| internal_local | every channel | C |
| teds_remote | 1 per 16 channels | A, E |

**TABLE 3-4: SHUNT CALIBRATION SETTINGS**

Shunt calibration configuration involves two discrete and disjoint operations. Each channel is assigned a shunt calibration mode, as described above. Separately, each shunt is enabled (connected) or disabled (disconnected).

The default shunt calibration mode is internal_remote. The default shunt enable setting is disabled.

## SELF-CALIBRATION

In order to deliver high measurement accuracy over a wide ambient operating temperature range, the EX1629 provides the ability to perform an instrument self-calibration. During self-calibration, the input signal conditioning paths are internally disconnected from the input jacks and connected instead to a calibration bus that is driven by an internal, precision calibration source. Through measurement of the conditioning paths at multiple calibration source points, software compensation for circuitry drift since the last full calibration is conducted. Once self-calibration is performed, the accuracy specifications listed in Section 1 are valid for 30 days and over a ±5 ºC ambient temperature change. Note, however, that the unit must still undergo a full calibration at least once a year, regardless of the use of self-calibration.

Self-calibration should be conducted as often as practical, especially if the ambient environment has changed significantly since the previous calibration. However, fast ambient environmental changes should ideally be followed by a period of thermal stabilization before conducting self-calibration to allow the internal circuitry to stabilize to a new thermal operating condition. The self-calibration process completes quickly and does not require removal of the actual input connections, making it convenient to run often.

Similarly, self-calibration should only be performed after the EX1629 has been allowed to warm up from a cold start for at least 60 minutes. To protect the user, the instrument will return a warning if self-calibration is initiated prior to the completion of this warm-up time. However, it is only a warning and can be overridden by repeating the calibration function. An override would be completely acceptable, for example, in cases where a) the unit is already fully warmed-up and is quickly moved from one physical location to another or b) instrument line power is briefly lost due to a facility power outage.

Self-calibration does not overwrite, modify, or take the place of the instrument's nonvolatile calibration constants generated by a full calibration. Instead, it generates an additional set of calibration constants that are applied to the measurement calculation after the full calibration constants. By default, self-calibration data is volatile, meaning that it is not saved through instrument reboots or power cycles. This ensures that the instrument always initializes with calibration constants generated by a full calibration. This feature is particularly important when the instrument is being shared among multiple users. Each user is consequently sheltered from the actions of others.

Despite having the ability to conduct self-calibration at any time, there may be user applications that require the use of self-calibration, but demand that it create nonvolatile data. The EX1629 supports that operation as well. Once self-calibration is performed, the data can be stored to nonvolatile memory through a separate function. Similarly, previously stored self-calibration data can be loaded or cleared from nonvolatile memory.

Self-calibration offers a convenient way to mitigate the effects of time and temperature on the signal conditioning circuitry of the EX1629, resulting in significant performance improvement. However, it cannot compensate or correct for potential drift errors caused by an excitation source change that is not followed by a thermal equilibrium delay. Similarly, any error caused by lead wire desensitization is outside of the calibration loop and is not eliminated. Finally, the use of self-calibration does not remove the necessity of excitation source measurement to achieve stated accuracies. For more details, see *Maximizing Measurement Performance* in Section 1.

## LOCKING

By default, the EX1629 allows unrestricted operation from multiple hosts. While this offers a high level of user flexibility, there are instances where protected operation is desirable, if not required. For these applications, the EX1629 can be "locked," meaning that it will accept function calls from only the host IP address that issued the lock function call. With the EX1629 in this mode, other host connections that attempt function calls will be denied.

By design, the locking mechanism is able to be overridden by a secondary host that issues a break lock function. Thus, the lock provides a warning to other users that the unit is in a protected operation state, but not absolute security. This allows for instrument recovery if the host or application should become disabled. Self-calibration requires the acquisition of a lock prior to its initiation.

## CONFIDENCE SCAN LIST CONFIGURATION

The Confidence Measurement System allows several signals within a bridge to be measured, in addition to the voltage across the bridge (main input channels), including the excitation voltages and currents. The confidence measurement system is configured by a scanlist, analogous to the main input channel scanlist, except that, in the case of the confidence measurement system, the measurements are actually taken sequentially (scanned) by a number of ADCs, whereas the main input channels each have a dedicated ADC.

The confidence scanlist contains the list of confidence channels to be measured on each main input channel. There is a single confidence scanlist for the entire instrument, so all channels in the main input channel scanlist have the same confidence measurements taken. That is, if there are 16 channels enabled in the main input scanlist, say channels 0 through 15, and the confidence scanlist is configured with 4 channels, say excitation current (positive and negative) and excitation output voltage (positive and negative), 64 confidence measurements will be taken (16 x 4).

The confidence system runs at a fixed rate (500 Sa/s), regardless of the sampling rate of the main input channels, and asynchronously to the main input measurements. Confidence measurement data is inserted into the same data stream as the main input channel data. When the main input sampling rate is equal to or less than 500 Sa/s, all scans of data include confidence data. Between 500 Sa/s and 1000 Sa/s, confidence data is included in the scans when available – the data stream is self-describing a manner such that clients can tell when confidence data is available and when it is not. Above 1000 Sa/s, the confidence data is not available.

| | |
|---|---|
| **NOTE** | The key restriction that MUST be enforced is that any confidence scan list that includes +V_SENSE and -V_SENSE must also include +EXCITEOUT and -EXCITELOW as well. |

## CONFIGURATION STORAGE

By default, all configuration options on the EX1629 are at factory defaults when the instrument is powered-on or is reset. The instrument does support the ability to save user configurations to non-volatile storage, however. This option saves all configuration variables, and automatically restores them at power-on or reset, instead of using the factory defaults.

To aid developers, the instrument can generate a "digest" (also known as a "hash") for the instrument's configuration. This digest is a statistically unique value, 16 bytes long, that serves as a signature or fingerprint for the current configuration state of the instrument. By configuring the instrument with the desired values and then saving the configuration to non-volatile storage, the configuration will be available whenever the instrument powers on or is reset. Additionally, by retaining the digest of the configuration when it is stored, an application can quickly verify that the instrument is configured identically to the desired state, without laboriously retrieving all configuration variables and verifying them against the desired state.

| | |
|---|---|
| **NOTES** | **1)** The configuration storage (and digest) pertains only to the acquisition configuration of the instrument (e.g., scanlist, EU conversions, excitation voltages, etc.). Other instrument configuration, such as network and time configuration are not covered by this mechanism.<br>**2)** Due to the way in which the internal configuration state of the instrument is stored, digest values for the factory default configuration as well as digest values for saved configurations may change between firmware versions. |

## WIDEBAND OUTPUT

The wideband output connector carries a buffered image of the main channel signal. It is a single ended signal, referenced to GND, and shielded to the chassis. The magnitude of this signal is ±15 V. Assmann Electronics' P/N: AHDS44LL-Z is one example of a mating connector.

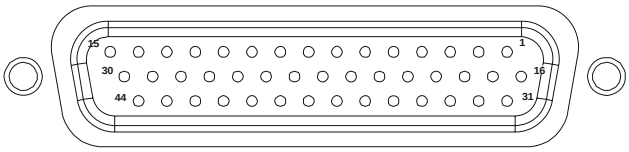



**FIGURE 3-9: WIDEBAND OUTPUT CONNECTOR**

| WIDEBAND OUT CH 0-15 | |
|---|---|
| **Pin #** | **Signal** |
| 1 | CH 7 HI |
| 2 | CH 6 HI |
| 3 | CH 5 HI |
| 4 | CH 4 HI |
| 5 | CH 3 HI |
| 6 | CH 2 HI |
| 7 | CH 1 HI |
| 8 | CH 0 HI |
| 9 | CH 15 HI |
| 10 | CH 14 HI |
| 11 | CH 13 HI |
| 12 | CH 12 HI |
| 13 | CH 11 HI |
| 14 | CH 10 HI |
| 15 | CH 9 HI |
| 16 | CH 7 LO |
| 17 | CH 6 LO |
| 18 | CH 5 LO |
| 19 | CH 4 LO |
| 20 | CH 3 LO |
| 21 | CH 2 LO |
| 22 | CH 1 LO |
| 23 | CH 0 LO |
| 24 | CH 15 LO |
| 25 | CH 14 LO |
| 26 | CH 13 LO |
| 27 | CH 12 LO |
| 28 | CH 11 LO |
| 29 | CH 10 LO |
| 30 | CH 9 LO |
| 31 | CH 7 Shield |
| 32 | Ch 6 Shield |
| 33 | CH 5 Shield |
| 34 | CH 4 Shield |
| 35 | CH 3 Shield |
| 36 | CH 2 Shield |
| 37 | CH 1 & CH 0 Shield |
| 38 | CH 8 HI |
| 39 | CH 15 & CH 14 Shield |
| 40 | CH 8 LO |
| 41 | CH 13 & CH 8 Shield |
| 42 | CH 12 Shield |
| 43 | CH 11 Shield |
| 44 | CH 10 & CH 9 Shield |

| WIDEBAND OUT CH 16-31 | |
|---|---|
| **Pin #** | **Signal** |
| 1 | CH 23 HI |
| 2 | CH 22 HI |
| 3 | CH 21 HI |
| 4 | CH 20 HI |
| 5 | CH 19 HI |
| 6 | CH 18 HI |
| 7 | CH 17 HI |
| 8 | CH 16 HI |
| 9 | CH 31 HI |
| 10 | CH 30 HI |
| 11 | CH 29 HI |
| 12 | CH 28 HI |
| 13 | CH 27 HI |
| 14 | CH 26 HI |
| 15 | CH 25 HI |
| 16 | CH 23 LO |
| 17 | CH 22 LO |
| 18 | CH 21 LO |
| 19 | CH 20 LO |
| 20 | CH 19 LO |
| 21 | CH 18 LO |
| 22 | CH 17 LO |
| 23 | CH 16 LO |
| 24 | CH 31 LO |
| 25 | CH 30 LO |
| 26 | CH 29 LO |
| 27 | CH 28 LO |
| 28 | CH 27 LO |
| 29 | CH 26 LO |
| 30 | CH 25 LO |
| 31 | CH 23 Shield |
| 32 | Ch 22 Shield |
| 33 | CH 21 Shield |
| 34 | CH 20 Shield |
| 35 | CH 19 Shield |
| 36 | CH 18 Shield |
| 37 | CH 17 & CH 16 Shield |
| 38 | CH 24 HI |
| 39 | CH 31 & CH 30 Shield |
| 40 | CH 24 LO |
| 41 | CH 29 & CH 24 Shield |
| 42 | CH 28 Shield |
| 43 | CH 27 Shield |
| 44 | CH 26 & CH 25 Shield |

| WIDEBAND OUT CH 32-47 | |
|---|---|
| **Pin #** | **Signal** |
| 1 | CH 39 HI |
| 2 | CH 38 HI |
| 3 | CH 37 HI |
| 4 | CH 36 HI |
| 5 | CH 35 HI |
| 6 | CH 34 HI |
| 7 | CH 33 HI |
| 8 | CH 32 HI |
| 9 | CH 47 HI |
| 10 | CH 46 HI |
| 11 | CH 46 HI |
| 12 | CH 44 HI |
| 13 | CH 43 HI |
| 14 | CH 42 HI |
| 15 | CH 41 HI |
| 16 | CH 39 LO |
| 17 | CH 38 LO |
| 18 | CH 37 LO |
| 19 | CH 36 LO |
| 20 | CH 35 LO |
| 21 | CH 34 LO |
| 22 | CH 33 LO |
| 23 | CH 32 LO |
| 24 | CH 47 LO |
| 25 | CH 46 LO |
| 26 | CH 45 LO |
| 27 | CH 44 LO |
| 28 | CH 43 LO |
| 29 | CH 42 LO |
| 30 | CH 41 LO |
| 31 | CH 39 Shield |
| 32 | Ch 38 Shield |
| 33 | CH 37 Shield |
| 34 | CH 36 Shield |
| 35 | CH 35 Shield |
| 36 | CH 34 Shield |
| 37 | CH 33 & CH 32 Shield |
| 38 | CH 40 HI |
| 39 | CH 47 & CH 46 Shield |
| 40 | CH 40 LO |
| 41 | CH 45 & CH 40 Shield |
| 42 | CH 44 Shield |
| 43 | CH 43 Shield |
| 44 | CH 42 & CH 41 Shield |

**TABLE 3-5: EX1629 WIDEBAND OUTPUT PIN ASSIGNMENTS**

## DIGITAL I/O

The EX1629 provides sixteen programmable digital input/output signals, modeled on the VT1533A digital input/output signal conditioning plug-on (SCP) module. These sixteen signals are divided into two banks of 8 signals: Bank 0 is digital I/O signals 0 through 7 and Bank 1 is digital I/O signals 8 through 15. Each bank is configurable as inputs (default) or outputs. Further, when configured as outputs, each bank can be configured as a passive (resistor) pull-up or an active (transistor) pull-up. When configured as inputs, the DIO signals (0 through 15) may be used as trigger and/or arm sources.

Several vendors provide appropriate mating connectors. AMP part number 216166-1 (44-pin housing) is one example.

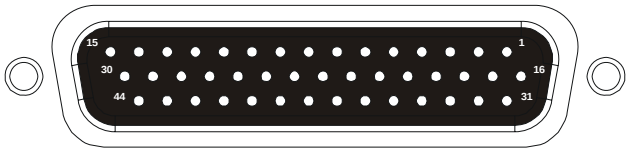



**FIGURE 3-10: DIGITAL I/O DB-44 CONNECTOR**

| Pin | Signal | Pin | Signal |
|---|---|---|---|
| 1 | DIO_BUS0 | 23 | GND |
| 2 | DIO_BUS1 | 24 | GND |
| 3 | DIO_BUS2 | 25 | GND |
| 4 | DIO_BUS3 | 26 | GND |
| 5 | DIO_BUS4 | 27 | GND |
| 6 | DIO_BUS5 | 28 | GND |
| 7 | DIO_BUS6 | 29 | GND |
| 8 | DIO_BUS7 | 30 | GND |
| 9 | DIO_BUS8 | 31 | DIO_TRIG0 |
| 10 | DIO_BUS9 | 32 | GND |
| 11 | DIO_BUS10 | 33 | DIO_TRIG1 |
| 12 | DIO_BUS11 | 34 | GND |
| 13 | DIO_BUS12 | 35 | GND |
| 14 | DIO_BUS13 | 36 | GND |
| 15 | DIO_BUS14 | 37 | GND |
| 16 | GND | 38 | GND |
| 17 | GND | 39 | GND |
| 18 | GND | 40 | GND |
| 19 | GND | 41 | GND |
| 20 | GND | 42 | GND |
| 21 | GND | 43 | GND |
| 22 | GND | 44 | DIO_BUS15 |

**TABLE 3-6: DIGITAL I/O CONNECTOR PIN ASSIGNMENTS**

## LXI TRIGGER BUS

The EX1629 provides an LXI compatible trigger bus connector. For more information on the LXI Trigger Bus, please visit www.lxistandard.org and refer to *LXI Standard Revision 1.1* and *LXI Trigger Bus Cable and Terminator Specifications Rev 1.1*.

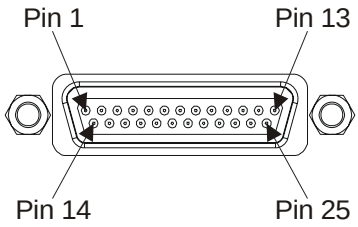



**FIGURE 3-11: TRIGGER BUS DB-25 CONNECTOR**

| Pin | Signal | Pin | Signal |
|---|---|---|---|
| 1 | +3.3 V | 14 | RP_TRIG_P0 |
| 2 | GND | 15 | RP_TRIG_N0 |
| 3 | RP_TRIG_P1 | 16 | RESERVED |
| 4 | RP_TRIG_N1 | 17 | RP_TRIG_P2 |
| 5 | GND | 18 | RP_TRIG_N2 |
| 6 | RP_TRIG_P3 | 19 | GND |
| 7 | RP_TRIG_N3 | 20 | RP_TRIG_P4 |
| 8 | GND | 21 | RP_TRIG_N4 |
| 9 | RP_TRIG_P5 | 22 | GND |
| 10 | RP_TRIG_N5 | 23 | RP_TRIG_P6 |
| 11 | RESERVED | 24 | RP_TRIG_N6 |
| 12 | RP_TRIG_P7 | 25 | RESERVED |
| 13 | RP_TRIG_N7 | | |

**TABLE 3-7: LXI TRIGGER BUS CONNECTOR PIN ASSIGNMENTS**

## TEDS TRANSDUCER SUPPORT

The EX1629 supports reading and writing to Transducer Electronic Data Sheets (TEDS) devices that implement the IEEE 1451.4 standard. Each channel (0 through 47) functions as a 1-Wire bus master, although only one channel can be active at a time, reading, or writing. Only one TEDS device per channel is supported.

There are two software interfaces to support TEDS devices, one that is tailored specifically for the Dallas/Maxim DS2430 part and one that is more general purpose that will work with any 1-Wire TEDS device, implementing the MicroLAN (MLAN) protocol. The former is deprecated; the latter, MLAN interface, is more general purpose and should be used by all new applications. The MLAN protocol is documented in IEEE 1451.4-2004 Annex G. See the *MicroLAN (MLAN) Primer* and associated sections for more information.

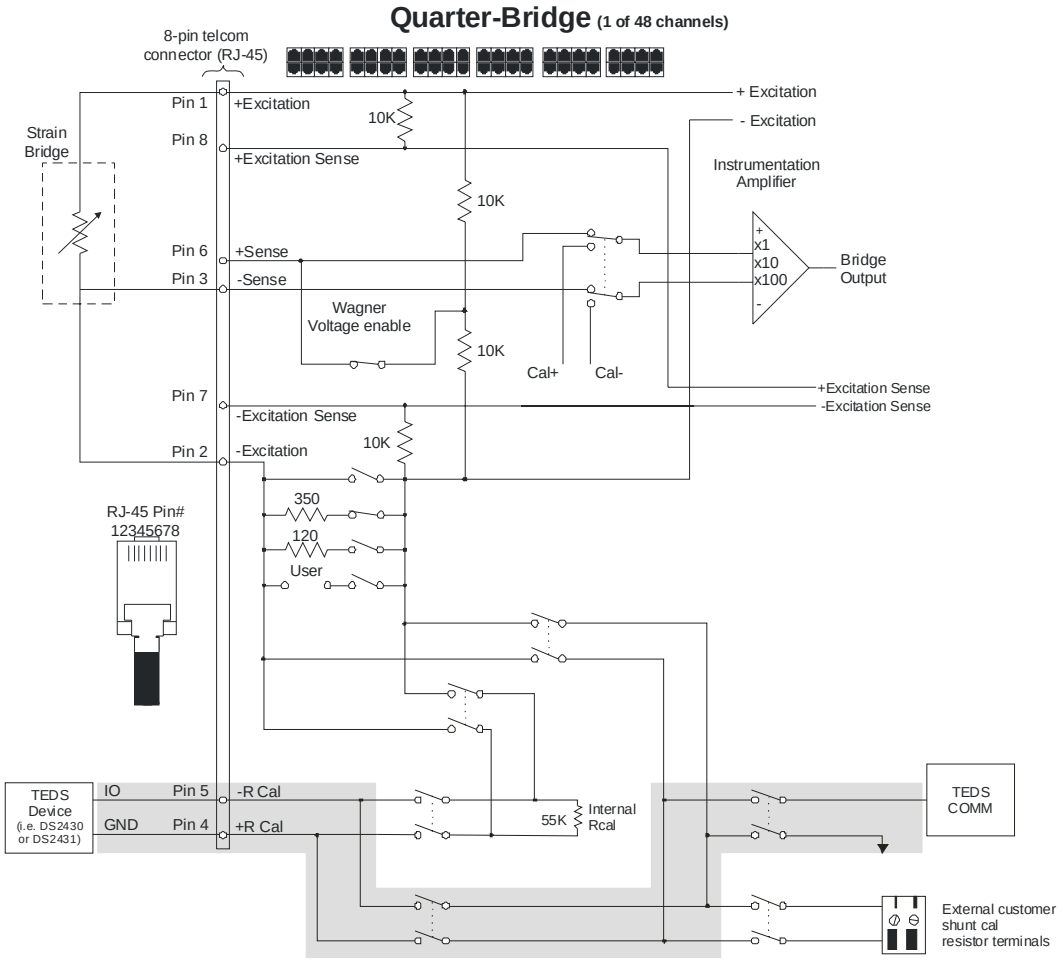



**FIGURE 3-12: TEDS WIRING SCHEMATIC**

| | |
|---|---|
| **NOTE** | Wiring the TEDS device to the EX1629 may not be as expected given the signal names used. The correct wiring is provided above in Figure 3-12 as well as below:<br><br>RJ-45 Pin 4 (RCAL + / TEDS +) to DS2430/1 GND<br>RJ-45 Pin 5 (RCAL - / TEDS -) to DS2430/1 IO<br><br>The "+" and "-" indicators for the signal names indicate that, since negative voltages are used to communicate with the TEDS device, the GND signal is actually more positive than the IO signal. |

## RESET BUTTON - LXI LAN CONFIGURATION INITIALIZE (LCI) MECHANISM

The reset button on the rear panel of the EX1629, implemented according to the LXI LAN Configuration Initialize (LCI) Mechanism specification, can be used to restore default network settings. This is useful for recovery from an incorrect or unknown network configuration. To perform a network reset:

1) Power off the EX1629.
2) Press and hold the reset button.
3) Power on the EX1629.
4) Continue to hold the reset button for at least 30 seconds.
5) Release the reset button.

The EX1629 will power up as usual, but will use the default network configuration (DHCP and AutoIP enabled) instead of its previous settings. VXI-11 Discovery (supported by the VISA IO-Libraries) or the vtex1629_findinstr function can be used to determine the IP address of the instrument.

# SECTION 4

# TRIGGERING

## OVERVIEW

The EX1629 supports a full function trigger model with a separate arm source and trigger source event structure. The trigger model is based on the industry standard SCPI 1999 Trigger Subsystem and is diagramed in Figure 4-1.

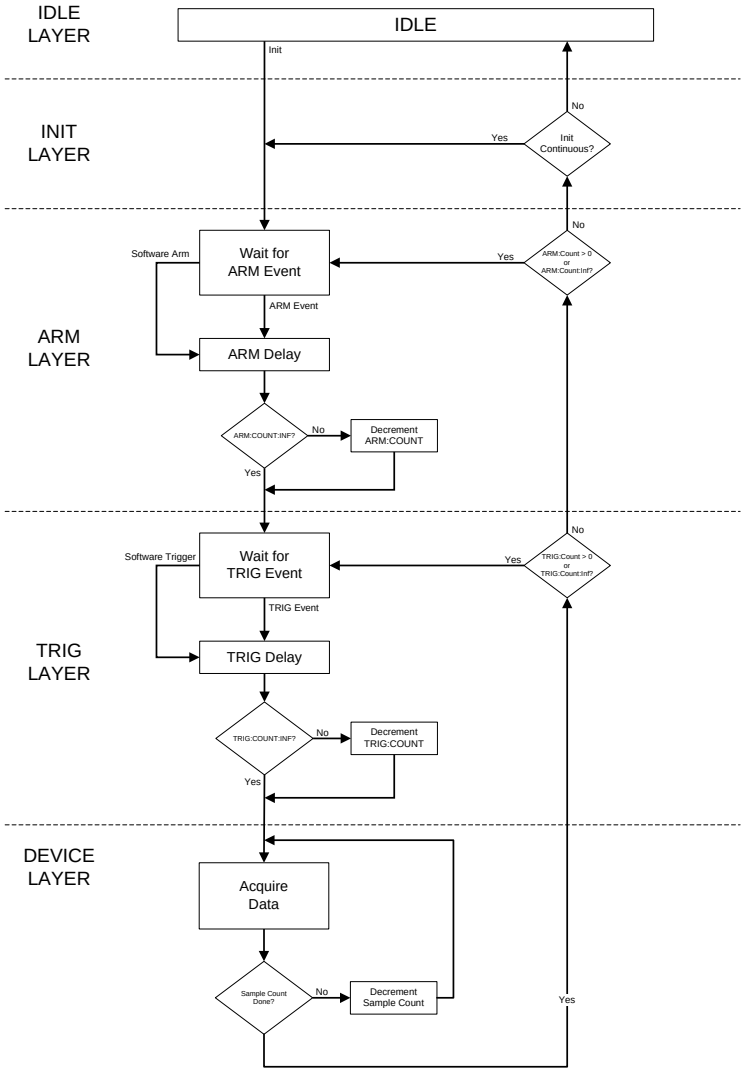



**FIGURE 4-1: EX1629 TRIGGER MODEL**

The trigger model is sectioned into five layers: IDLE, INIT, ARM, TRIG, and DEVICE. The EX1629 reset condition places it in the IDLE state. A trigger initialize command begins the acquisition sequence by transitioning the instrument through the INIT layer into the ARM layer. As this occurs, the reading buffer memory is cleared

Upon entering the ARM layer, the ARM Count is reset to its specified value. The instrument remains in the ARM layer until the specified ARM event occurs or a Software Arm is issued. Once that occurs, the specified ARM Delay (if any) is waited, the ARM Count is decremented, and the instrument transitions into the TRIG layer.

Upon entering the TRIG layer, the TRIG Count is reset to its specified value. The instrument remains in the TRIG layer until the specified TRIG event occurs or a Software Trigger is issued. Once that occurs, the specified TRIG Delay (if any) is waited, the TRIG Count is decremented, and the instrument transitions into the DEVICE layer.

In the DEVICE layer, channels in the scan list are measured the requested sample count number of times, and stored into local memory (FIFO).

If the TRIG Count remains nonzero, the instrument stays in the TRIG layer until the specified TRIG event (and subsequent device action) occurs enough times to decrement it to zero. Once the TRIG Count reaches zero, the instrument then evaluates the remaining ARM Count and repeats the ARM layer action if it is nonzero. However, since each transition into the TRIG layer resets the TRIG Count, each additional ARM layer action results in the full specified number of TRIG Count actions through the TRIG layer and DEVICE layer.

Once the ARM Count reaches zero, the instrument transitions back into the INIT layer. If Init Continuous mode is enabled, the ARM layer is automatically reentered without the issuance of a trigger initialize command. However, unlike with a trigger initialize, the reading buffer memory is not cleared. Conversely, if Init Continuous mode is disabled, the instrument is returned to the IDLE layer and requires the issuance of a new trigger initialize command to begin a new acquisition sequence.

## ACQUISITION DATA AND FIFO

When a trigger event occurs and the instrument transitions into the Device Layer of Figure 4-1, a user-configured number of samples are acquired. Main bridge acquisition data is acquired in parallel from all channels enabled in the scanlist, using a separate ADC per channel. Acquisition data, from both the main bridge measurements as well as the confidence subsystem, is captured, filtered appropriately, calibration compensated, converted to engineering units, and stored in a FIFO in the EX1629's RAM along with a timestamp. In parallel, the instrument's DIO values are sampled and placed into the FIFO as well.

The digital inputs (DIO) are sampled during acquisition as well. The maximum sample rate for DIO data is 1 kSa/s. If the main acquisition sample rate is higher than 1 kSa/s, the expected phase error in terms of samples between DIO & Main bridge shall be ±[Main bridge samp freq/(2*DIO sample freq)].

To maintain current data within the digital filters, data is always acquired from the main bridge channels. When not within the Device Layer of Figure 4-1, this data is discarded before it reaches the RAM FIFO. This serves to keep the digital filter states updated.

Applications may retrieve data from this FIFO using either the Read FIFO or Streaming Data interfaces. Please refer to the *Retrieving Data (Read FIFO and Streaming Data)* section for further details. Once data is retrieved from the FIFO, via either method, it is no longer kept within the FIFO.

## CONFIDENCE MEASUREMENT SYSTEM

The Confidence Measurement System runs in parallel with the main bridge measurement system, but largely asynchronously. The Confidence Measurement System scans the various Confidence Measurement Sources on each main input channel enabled in the scanlist. This scanning occurs at the same rate as the main input channel sampling rate, up to 500 Sa/s. Above 500 Sa/s and below 1000 Sa/s, the Confidence Measurement System continues to sample at 500 Sa/s, inserting data into the FIFO when new data is available. At 1000 Sa/s, confidence data is available with every other dataset. The confidence data is filtered by a transfer function represented by the following differential equation: $y(n) = 0.01x(n) + 0.99y(n-1)$, where $y(n)$ is the filtered confidence data and $x(n)$ is the measured confidence data. This function serves to reduce noise variance.

Confidence data is acquired for every main input channel enabled in the scanlist. Which confidence data is acquired is controlled by the confidence scanlist. There is only one confidence scanlist in the system, and thus the same confidence data is sampled for every main input channel. The amount of confidence data returned is basically the product of the length of the main input channel scanlist with the length of the confidence scanlist. For example, if the main input scanlist is configured to measure channels 0, 1, 3, and 5, and the confidence scanlist is configured to measure +Excite and –Excite, the total number of confidence data values returned will be 8.

To be clear, there is a single FIFO in the EX1629 that holds both main input channel data as well as confidence measurement system data.

## ADC CLOCK AND SYNCHRONIZATION

As was previously described, each input channel is an independent measurement system with its own synchronous analog to digital converter (ADC). Input data is acquired synchronous to a sample clock that is distributed to each ADC. In addition to the sample clock, a synchronization signal is used to reset each of the ADCs and align them to acquire data synchronously. The triggering system must be re-synchronized each time a change is made to the sample clock source, sample frequency, or filtering parameters.

## SYNCHRONIZING MULTIPLE INSTRUMENTS

In cases where larger acquisition systems are required, multiple devices can be configured to utilize the same ADC sample clock and synchronization signal to acquire data that is synchronized across all acquisition channels. This requires configuring one of the devices as a master and the remaining devices as slaves. The devices must be connected together using the LXI Trigger Bus in either a star or daisy chain configuration. The details of how to program the configuration of multi-box systems is described in Section 6.

Please refer to *Configure Trigger and ADC Clock* for further details.

# SECTION 5

# WEB PAGE OPERATION

## INTRODUCTION

The EX1629 offers an embedded web page to control network configuration, time configuration, and firmware upgrades.

## OPENING THE WEB PAGE

Type the EX1629's assigned IP address or host name into an Internet browser application.

## GENERAL WEB PAGE OPERATION

When initial connection is made to the EX1629, the instrument home page, **Index**, appears. This page displays instrument-specific information including:

- Name
- Serial number
- IP address
- MAC address
- Firmware version (also visible in the bottom right corner of every page)
- Date of last full calibration
- Presence of nonvolatile self-calibration data

This page is accessible from any other instrument page by clicking on the EX1629 web page header.

The EX1629 command menu is displayed on the left hand side of every internal web page. The entries on the command menu represent three types of pages:

***Status*** This type of page performs no action and accepts no entries. It provides operational status and information only. The **Index** page is an example of a status page.

***Action*** This type of page initiates a command on the instrument, but does not involve parameter entry. The **Reboot** page is an example of an action page.

***Entry*** This type of page displays and accepts changes to the configuration of the instrument. The **Time Configuration** page is an example of an entry page.

Use of the entry-type web pages in the EX1629 are governed by a common set of operational characteristics:

- Pages initially load with the currently-entered selections displayed.
- Each page contains a *Submit* button to accept newly entered changes. Leaving a page before submitting any changes has the effect of canceling the changes, leaving the instrument in its original state.
- Navigation through a parameter screen is done with the Tab key. The Enter key has the same function as clicking the *Submit* button and cannot be used for navigation.

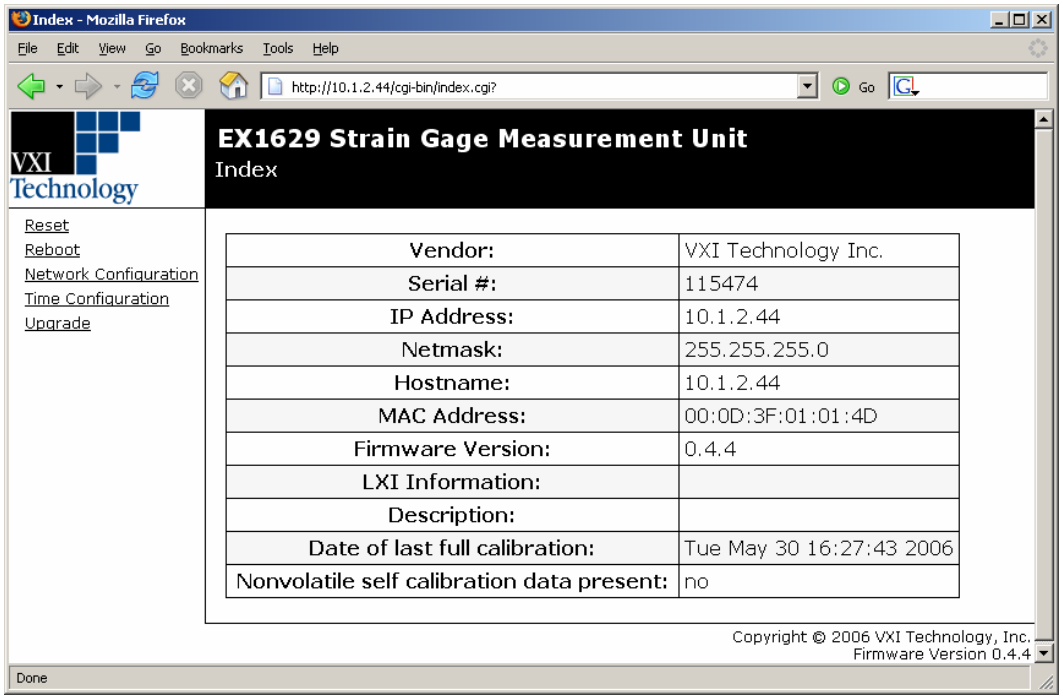



**FIGURE 5-1: EX1629 MAIN WEB PAGE**

## PASSWORD

Pages that modify instrument configuration (e.g., Network Configuration) require a password login. This mechanism is meant to prevent accidental configuration modification. The default password is **ex1629**. Pages that require a password will provide a link to the password login page.

| NOTES | **1)** The instrument password uses lower-case letters.<br>**2)** This password mechanism provides only the most basic security. Applications requiring additional security should consider using private, dedicated subnets, firewalls, etc. to limit access to the instrument on the network. |
|---|---|

## VXI TECHNOLOGY LOGO

The VXI Technology Logo that appears on the upper left of all EX1629 web pages is a link to the VXI Technology, Inc. corporate website: http://www.vxitech.com.

## EX1629 STRAIN GAGE MEASUREMENT UNIT

The title block ("EX1629 Strain Gage Measurement Unit" on a black background at the top of the page) of all EX1629 Web Pages is a link to the Main Web Page.

## RESET

This action page is used to return all of the EX1629's acquisition configuration parameters to their default values. It is most commonly used to return the instrument to a known configuration state prior to the initiation of a new test sequence. Specific affected configuration parameters and their reset values are documented in Table 6-1. An instrument reset only affects acquisition parameters and does not affect self-calibration data. This is equivalent to the vtex1629_reset function.

| NOTE | An instrument reset clears the FIFO reading memory. All desired acquisition data must be retrieved from the FIFO prior to the issuance of this command. |
|---|---|

### REBOOT

This action page is used to perform a complete instrument reboot, equivalent to that which occurs when the instrument is power cycled. It is most commonly used to accept changes that are made to the network configuration or time configuration settings. In addition, it is suggested that a reboot be performed before conducting a firmware upgrade.

### NETWORK CONFIGURATION

The EX1629 Network Configuration page can be seen in Figure 5-2. By default, the EX1629 will attempt to locate a DHCP server. If one is found, the IP address assigned by the DHCP server will be assumed, along with subnet masks, gateway, etc. Otherwise, after a timeout of 20 seconds, the unit will attempt to obtain an IP address by using AutoIP (IPv4 Dynamic Link Local Addressing).

| NOTE | The EX1629 can be returned to a known, default network configuration by using the LCI (LAN Configuration Initialize) Mechanism. See *Reset Button - LXI LAN Configuration Initialize (LCI) Mechanism* for more information. |
|---|---|

AutoIP is a mechanism for finding an unused IP address in the IANA assigned range 169.254.X.Y (169.254/16) where X is in the range 1 - 254 and Y is in the range 0 - 255. The device will first attempt to obtain the specific address 169.254.X.Y, where X and Y are the second-to-last and last octets of the device's MAC address. However, X will be set to 1 if it is 0 in the MAC address, and to 254 if it is 255 in the MAC address for conformance with the AutoIP specifications. If this address is already in use, the unit will attempt to obtain other IP addresses in the 169.254/16 range in a pseudorandom fashion until it finds one that is available.

To illustrate the AutoIP mechanism, Table 5-1 lists the AutoIP default address for some example MAC addresses.

| MAC Address | AutoIP Default Address |
|---|---|
| 00:0D:3F:01:00:01 | 169.254.1.1 |
| 00:0D:3F:01:01:01 | 169.254.1.1 |
| 00:0D:3F:01:A3:28 | 169.254.163.40 |
| 00:0D:3F:01:FE:FE | 169.254.254.254 |
| 00:0D:3F:01:FF:FE | 169.254.254.254 |

**TABLE 5-1: AUTOIP DEFAULT ADDRESS ASSIGNMENT**

If a static IP address assignment is preferred, one can be optionally assigned via the embedded web page interface. This is done by clicking the **Network Configuration** link, disabling DHCP and AutoIP, enabling Static, and then assigning a static IP address, subnet mask, and gateway address, and, optionally up to three DNS servers (see Figure 5-2).

| NOTE | The 169.254/16 subnet is reserved by the IANA for AutoIP usage. It should not be used for either DHCP or static IP configurations. |
|---|---|

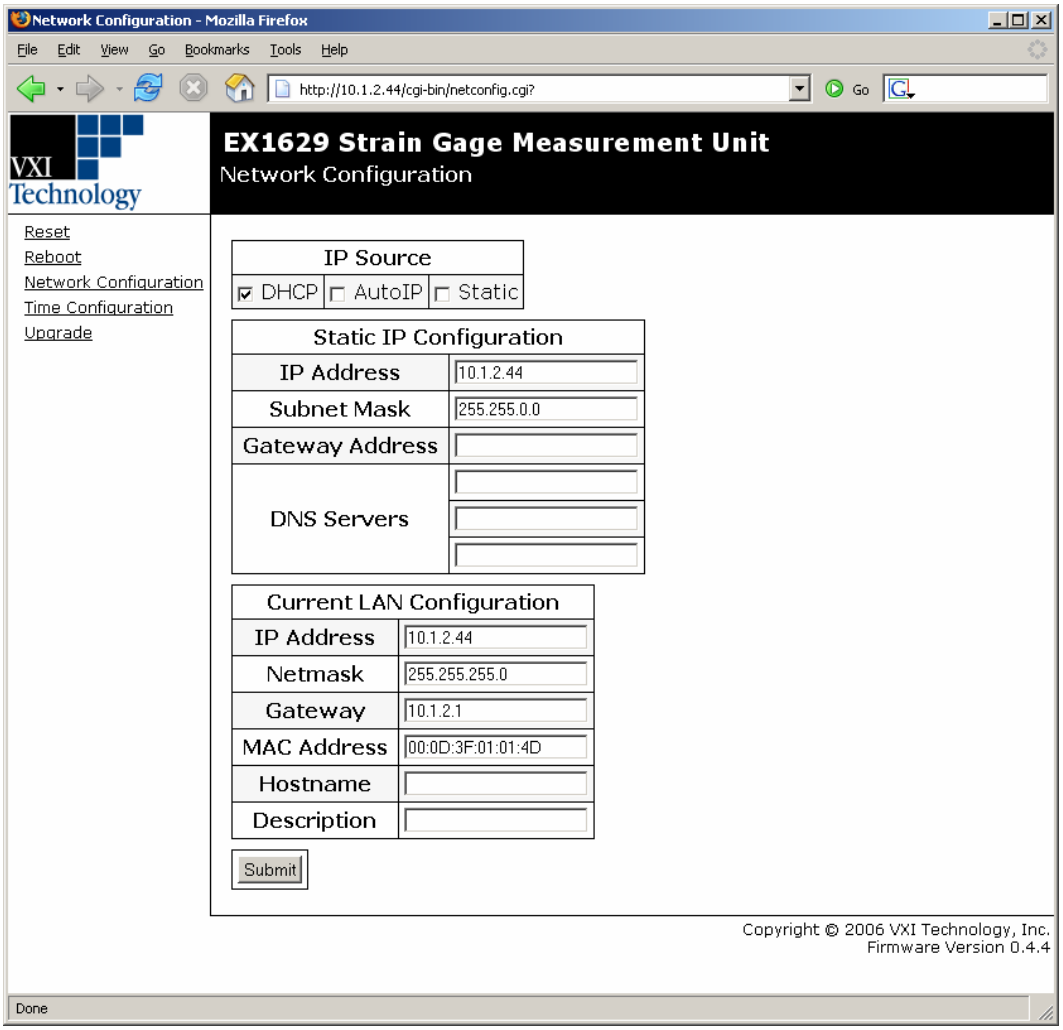



**FIGURE 5-2: EX1629 NETWORK CONFIGURATION**

However, a much more convenient and recommended way to obtain the benefits of a static IP address is to employ DHCP, but assign the instrument a reserved IP address in your company's DHCP server configuration. This reserved address, linked to the EX1629's MAC address on the DHCP server, would be assigned to the EX1629 at power up initialization without having to manually set it on the EX1629. The DHCP server configuration provides a centralized, controlled database of assigned IP addresses, preventing accidental assignment of the same IP address to multiple instruments. Consult your company's Information Technology department for assistance.

VXI-11 Device Discovery is also supported by the EX1629. This allows all EX1629s on a local network to be found without knowledge of their MAC address or IP address with the use of a broadcast message. The vtex1629_findinstr function can be used programmatically to find all available EX1629 instruments on the LAN.

## TIME CONFIGURATION

This entry page is used to change the time configuration of the EX1629. By default, the instrument has no notion of "wall-clock" or calendar time. The instrument has no battery-backed clock or any other mechanism to retain time between reboots and power-cycles. By default, the instrument's time and date at power-up are midnight, January 1, 1970 (the beginning of the "epoch"). The time and date can be manually specified on the Time Configuration page (Figure 5-3). Manual configuration will be necessary if the network environment is such that the instrument cannot reach the Internet. Manual time entry is not affected by the **Zone** control and does not require an instrument reboot to be activated. However, manually-specified time is volatile, and therefore must be reentered upon an instrument reboot or power cycle. It is not, however, affected by the **Reset** page.

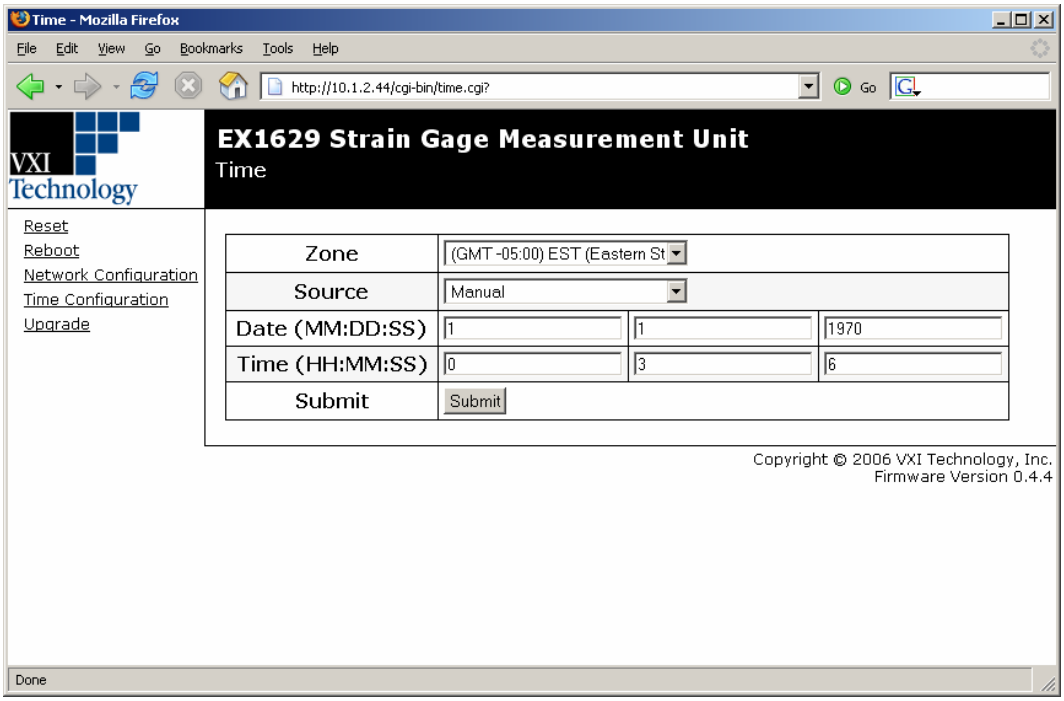



**FIGURE 5-3: EX1629 TIME CONFIGURATION WEB PAGE - MANUAL**

Optionally, the EX1629 supports SNTP (Simple Network Time Protocol), allowing it to receive its time from an SNTP server. The **Zone** control provides a pull-down selection in which the user's specific time zone is selected. SNTP or NTP (Network Time Protocol) servers are specified in the Server configuration panel, by IP Address or hostname. An instrument reboot is then required to activate the new selection.

| **NOTE** | Specifying SNTP or NTP servers by hostname requires that the instrument be configured for DNS, either by DHCP or Static IP. |
|---|---|

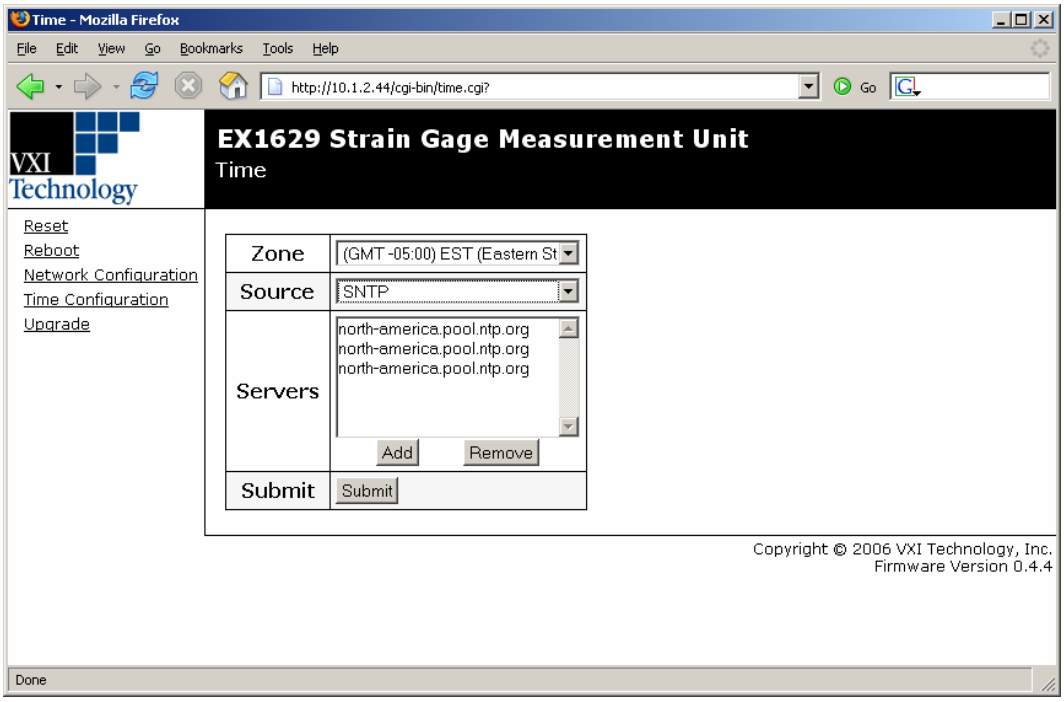



**FIGURE 5-4: TIME CONFIGURATION WEB PAGE - SNTP**

## UPGRADE

This action page is used to upgrade the embedded firmware of the EX1629. Prior to initiating the firmware upgrade process, a new, uncompressed firmware image must be obtained from VXI Technology and be accessible from the computer that is connected to the EX1629. Unless specifically noted by VXI Technology, firmware upgrades do not alter the calibration or non-volatile configuration settings (network configuration, time configuration) of the EX1629.

| **NOTE** | Do not power cycle the EX1629 during the firmware upgrade process. If power is lost during the upgrade, the instrument may be put into an inoperable state, requiring return to the factory. An uninterruptible power supply may be used to avoid this risk. |
|---|---|

Perform the following steps to conduct a firmware upgrade:

1) Perform a **Reboot** or a power cycle.
2) Connect to the EX1629 via the embedded web page.
3) Click on the **Upgrade** link.
4) Click on the *Browse* button and select the firmware image file to be uploaded to the instrument.
5) Click the *Submit* button to initiate the upgrade process.

The upgrade process takes approximately 5 minutes to complete, culminating with an automatic instrument reboot. Once the reboot is complete, reconnect and confirm on the **Index** page that the firmware revision level has been properly updated.

| **NOTE** | Due to the way in which the internal configuration state of the instrument is stored, digest values for the factory default configuration, as well as digest values for saved configurations, may change between firmware versions. |
|---|---|

# SECTION 6

# PROGRAMMING

## INTRODUCTION

This section provides programming examples and rationale for use with the EX1629. While this is not an exhaustive list, it provides a basis for the creation of code for several basic operations.

## DEFAULT SETTINGS

The factory default instrument settings after an instrument reset or a power cycle are listed in Table 6-1. Many programming applications do not require parameter changes from the default settings and can be made far simpler by the elimination of redundant functions. The EX1629 can be returned to the reset state at any time through the issuance of the vtex1629_reset function call.

**NOTE** The EX1629 supports saving user-defined configurations to non-volatile storage. If there is a user-defined configuration stored in non-volatile storage, that configuration is applied after a reset, power cycle, or use of the vtex1629_reset function. The vtex1629_clear_stored_config function can be used to remove a configuration from non-volatile storage, after which the vtex1629_reset function will restore the instrument to the factory defaults.

| SCAN LIST CONFIGURATION RESET VALUES | |
|---|---|
| CONFIGURATION PARAMETER | RESET VALUE |
| **Scan List (enabled channels)** | 0-47 |
| **Gain** | 1 |
| **Completion Resistor** | Full |
| **Input Multiplexer** | Full |
| **Sampling Rate (Sa/s)** | 1000 |
| **Confidence Scan List (elements)** | None |

| EU CONVERSION RESET VALUES | |
|---|---|
| CONFIGURATION PARAMETER | RESET VALUE |
| **EU Conversion** | Voltage |
| **Excitation Voltage** | 0 |
| **Unstrained Voltage** | 0 |
| **Gage Factor** | 2.0 |
| **Poisson Ratio** | 0.3 |
| **Strain Units** | Strain |
| **Tare Value** | 0 |

| EXCITATION SOURCE CONFIGURATION RESET VALUES | |
|---|---|
| CONFIGURATION PARAMETER | RESET VALUE |
| **Programmed Value** | 0 |
| **Output Enable** | Disabled |

TABLE 6-1: DEFAULT SETTINGS

| TRIGGER CONFIGURATION RESET VALUES | |
|---|---|
| CONFIGURATION PARAMETER | RESET VALUE |
| **Sample Count** | 1000 |
| **Arm Source** | Immediate |
| **Arm Count** | 1 |
| **Arm Delay (seconds)** | 0 |
| **Trig Source** | Immediate |
| **Trig Count** | 1 |
| **Trig Delay (seconds)** | 0 |
| **Trig Timer Interval (seconds)** | 0 |

| SHUNT CALIBRATION CONFIGURATION RESET VALUES | |
|---|---|
| CONFIGURATION PARAMETER | RESET VALUE |
| **Shunt Mode** | Internal_remote |
| **Shunt Enable** | Disabled |

| DIGITAL FILTER CONFIGURATION RESET VALUES | |
|---|---|
| CONFIGURATION PARAMETER | RESET VALUE |
| **Type** | 1 (Butterworth) |
| **Cutoff Frequency (hertz)** | 10 |
| **Transform** | 0 (Bilinear) |
| **Specified Order** | 0 (Auto) |
| **Calculated Order** | 6 |

| DIGITAL I/O CONFIGURATION RESET VALUES | |
|---|---|
| CONFIGURATION PARAMETER | RESET VALUE |
| **Bank 0 Direction** | 0 (Input) |
| **Bank 0 Pull-up** | 0 (Passive) |
| **Bank 1 Direction** | 0 (Input) |
| **Bank 1 Pull-up** | 0 (Passive) |
| **Output** | 0 |

| ADC CLOCK RESET VALUES | |
|---|---|
| CONFIGURATION PARAMETER | RESET VALUE |
| **Sample Clock Mode** | Master (Standalone) |
| **Sample Clock Input** | Internal sample clock line |
| **Sample Clock Output** | Internal sample clock line |

| SYNCHRONIZATION RESET VALUES | |
|---|---|
| CONFIGURATION PARAMETER | RESET VALUE |
| **Synch. Mode** | Master (Standalone) |
| **Synch. Input** | Internal synch. line |
| **Synch. Output** | Internal synch. line |
| | |
| **LXI TRIGGER BUS RESET VALUES** | |
| CONFIGURATION PARAMETER | RESET VALUE |
| **Direction** | 0 (Input) |
| **Transmission Scope** | 1 (External and Internal) |
| **Output** | 0 |

**TABLE 6-1: DEFAULT SETTINGS (CONTINUED)**

## OPENING AN INSTRUMENT SESSION

Prior to performing any programmatic actions with an instrument, a session must be opened. The following sample code illustrates this using the EX1629 VXI*plug&play* driver, along with querying the revision of the instrument driver and instrument firmware.

***Sample Code***

```
ViStatus    result = VI_SUCCESS;
ViChar      instrrev[256];
ViChar      driverrev[256];
ViChar      filename[256];
ViSession   vi;
ViChar errDescription[256] = "";

/* open a session */
result = vtex1629_init(instr, VI_ON, VI_ON, &vi);
if(result != VI_SUCCESS) {
     vtex1629_error_message(vi, result, errDescription);
     printf(errDescription);
}

/* query the driver and firmware revision */
result = vtex1629_revision_query(vi, driverrev, instrrev);
if(result != VI_SUCCESS) {
     vtex1629_error_message(vi, result, errDescription);
     printf(errDescription);
}

/* display results */
printf("Driver Revsion: %s\n", driverrev);
printf("Instrument Firmware Revision: %s\n", instrrev);
```

Note that in the sample code above, error code checking and handling is included. The structure used above can be used as a template for all functions. In the examples that follow, error code checking and handling has not been included for brevity.

## CLOSING AN INSTRUMENT SESSION

In order to release system resources, applications should close instrument sessions when finished with them. The following code illustrates this.

***Sample Code***

```
/* close the instrument session*/
result = vtex1629_close(vi);
```

## CONFIGURING THE ACQUISITION CHANNELS

Configuration of the EX1629 is an application-specific process. The following sample code, however, will satisfy many applications and may serve as a template. The sample code does the following:

- Configures the scanlist to contain all 48 channels
- Configures all channels for quarter-bridge 120 Ω strain gage EU (which also configures the input multiplexer and completion resistors appropriately)
- Sets the gain to 100X (most sensitive)
- Disables the IIR filters
- Sets the excitation voltage to ±2.5 V, and enables the excitation supplies
- Sets the sampling rate to 100 Sa/s

- Measures the unstrained voltage and updates the unstrained EU conversion variables
- Measures the excitation voltage with the confidence subsystem and updates the excitation EU conversion variables

***Sample Code***

```
#define MAX_NUM_CHANNELS 48

ViStatus    result = VI_SUCCESS;
ViInt32     channels[MAX_NUM_CHANNELS];
ViInt32     numberOfChannels = MAX_NUM_CHANNELS;
ViInt32     i, numval;

/* initialize a channel array  */
for( i = 0; i < MAX_NUM_CHANNELS; i++ )
  channels[i] = i;

/* set the scanlist */
result = vtex1629_set_scanlist( vi, channels, numberOfChannels);

/* set the EU conversion for all channels*/
result = vtex1629_set_EU_conversion( vi,
                                     channels,
                                     numberOfChannels,
                                     VTEX1629_EUCONV_QTR_BRIDGE_120);

/* set the gain */
result = vtex1629_set_gain( vi,
                            channels,
                            numberOfChannels,
                            VTEX1629_GAIN_HUNDRED);

/* set the excitation */
result = vtex1629_set_excitation( vi,
                                  channels,
                                  numberOfChannels,
                                  2.5,
                                  -2.5);

/* enable the excitation */
result = vtex1629_set_excitation_enabled( vi,
                                          channels,
                                          numberOfChannels,
                                          VI_TRUE);

/* turn off the IIR filters */
result = vtex1629_set_IIR_filter_configuration( vi,
                                          channels,
                                          numberOfChannels,
                                          VTEX1629_IIR_FILT_NONE,
                                          0,
                                          VTEX1629_TRANSFORM_BILINEAR,
                                          1 );

/* set the sample frequency */
result = vtex1629_set_sample_frequency( vi, 100.0 );

/* measure the unstrained voltage */
result = vtex1629_measure_unstrained_voltage( vi,
                                              channels,
                                              numberOfChannels,
                                              50,
```

```
                                                    NULL,
                                                    &numval,
                                                    VI_TRUE);

/* measure the excitation voltage */
result = vtex1629_measure_ excitation _voltage( vi, channels,
numberOfChannels, VTEX1629_EXCITE_SRC_REMOTE, 50, NULL, &numval,
VI_TRUE);
```

***Setting Bridge Limits***

Once the EX1629's channels are configured for data acquisition, the user's can also set limits on the data being acquired so that, if a minimum or maximum value is exceeded, an error message is returned.

***Sample Code***

```
ViStatus status = VI_SUCCESS;
ViChar errMessage[256];

ViInt32 channels[MAX_NUMBER_OF_CHANNELS];
ViInt32 numChannels = MAX_NUMBER_OF_CHANNELS;
ViReal64 minArr[MAX_NUMBER_OF_CHANNELS];
ViReal64 maxArr[MAX_NUMBER_OF_CHANNELS];

int i = 0;

for(i = 0; i < MAX_CHANNELS; i++) {
     channels[i] = i;
}

for(i = 0; i < MAX_NUMBER_OF_CHANNELS; i++) {
     minArr[i] = (-2.0);
}
for(i = 0; i < MAX_NUMBER_OF_CHANNELS; i++) {
     maxArr[i] = 4.0;
}

status = vtex1629_set_bridge_limit(instrumentHandle,
                                   numChannels,
                                   channels,
                                   minArr,
                                   maxArr);

if(status < VI_SUCCESS){
     <inform the user the API call failed>
}
```

***Lead Wire Compensation***

The user can also use the EX1629 to compensate for lead wire  for lead wire resistance. This is a common source of error when making measurements and can be difficult to quantify for large channel count applications. This work can be simplified by utilizing the EX1629s vtex1629_measure_lead_wire_resistance function.

***Sample Code***

```
ViChar errMessage[256];

ViInt32 numberOfChannels = MAX_NUMBER_OF_CHANNELS;
ViInt32 channels[MAX_NUMBER_OF_CHANNELS];
ViReal64 resistance[MAX_NUMBER_OF_CHANNELS];
```

```
ViInt32 sampleCount = 100;

int i = 0;

for(i = 0; i < numberOfChannels; i++) {
     channels[i] = i;
}

memset(resistance, 0x00, sizeof(resistance));

status = vtex1629_measure_lead_wire_resistance(instrumentHandle,
                                                numberOfChannels,
                                                channels,
                                                resistance,
                                                sampleCount,
                                                VI_TRUE);
if(status < VI_SUCCESS){
     <inform the user the API call failed>
}
```

## CONFIGURE TRIGGER AND ADC CLOCK

The EX1629 supports two general use-cases: standalone (one or more instruments that sample independently and asynchronously) and master/slave (one master instrument and one or more slave instruments, sampling synchronously). While the configurations for these two use cases are similar, the master/slave configuration is slightly more complicated, due to the additional requirements for distributing clock and synchronization signals to guarantee synchronous, phase-aligned acquisition.

### *ADC Sample Clock*

The sample clock configuration options on a standalone device and a master device are very similar. In each case, the device is configured as a sample clock master and the internal oscillator is used as the clock source. This sample clock can be distributed within the device and to other devices using any of the LXI Trigger Bus lines. In the case of a standalone device, the internal sample clock line can be also used to distribute the clock without the need to use one of the general-purpose trigger bus lines. This internal sample clock line cannot be used in multi-box configurations since its distribution is limited to within one device.

To allow added flexibility for more complicated multi-box configurations, the sample clock can also be output on one trigger bus line and input on another. This functionality is useful in star multi-box configurations. For example, the master device can be configured to output the clock on LXI0. A trigger bus hub can be utilized to receive this clock on LXI0 and distribute it to the master and slave devices on LXI4. The master device can then be configured to receive its clock on LXI4 instead of using the clock that it is outputting on LXI0. This allows the master and slave devices to use the same clock from the trigger bus hub instead of the master using one clock and the slaves using the same clock but with the added phase delay of the trigger bus hub.

The vtex1629_set_sample_clock_source instrument driver function is used to configure the sample clock source. For a master or standalone device, the **sampleClockMode** parameter should be set to VTEX1629_SAMP_CLK_MODE_MASTER. The **outLine** parameter specifies the trigger bus line that is used to output the clock. This can be either one of the trigger bus lines or can be set to VTEX_LXI_LINE_NONE to use the internal sample clock line. The **inLine** parameter specifies what line is used by the device for its ADC clock. This may or may not be the same as the lines that is specified to output the clock via the outLine parameter. As with the **outLine** parameter, specifying an input line of VTEX_LXI_LINE_NONE will instruct the device to use the internal sample clock line. In the case of a stand alone device that uses the internal sample clock line, both the input and the output lines are set to VTEX_LXI_LINE_NONE.

If a trigger bus line will be used for distributing the sample clock or for receiving a clock back into the device from an external source, it must be configured prior to configuring the sample clock. Regardless of whether the sample clock is only used within the device or if it is distributed to other devices, the trigger bus line that is used to output the sample clock (specified by the **outLine** parameter) must be configured as an output using the vtex1629_set_lxibus_configuration function. If this line will only be used within the device, the transmission scope for the line should be set to internal transmission only. If the sample clock output is intended to be driven out on the external trigger bus, the transmission scope must be set for external and internal transmission. If the sample clock is input on different trigger bus line than it is output, the input trigger bus line must be configured as an input with external and internal transmission scope using the vtex1629_set_lxibus_configuration function. When the internal sample clock line is used, configuration of the trigger bus lines is not required.

Sample clock configuration on a slave device is much simpler than that of a master device. The vtex1629_set_sample_clock_source function is used to specify a **sampleClockMode** parameter of VTEX1629_SAMP_CLK_MODE_SLAVE as well as indicating the trigger bus line that will be used to receive the sample clock. The trigger bus line must be configured as an input with external and internal transmission scope.

### *ADC Synchronization*

Configuration of the ADC synchronization signal is similar to configuration of the ADC sample clock. Standalone and master devices are similar in that they are both configured with a **syncMode** parameter of VTEX1629_SYNC_MODE_MASTER using the vtex1629_set_synch_source function. As a standalone or master device, synchronization pulses are generated on the specified synchronization signal line using the vtex1629_soft_synch function. The **outLine** parameter for this function is used to specify which trigger bus line is used to output the synchronization signal. It can either specify one of the trigger bus lines or a dedicated internal synchronization signal line. As with the ADC sample clock source, the synchronization signal can be configured to only be used within the device or to be output to other devices using the external trigger bus. To allow flexibility, the synchronization signal can be received back into the device on a different line than the one on which it is output. As with the ADC sample clock, any trigger bus lines that are used for the synchronization signal must be properly configured as inputs or outputs and with the proper transmission scope before they can be used for the synchronization signal. For a standalone device, the synchronization signal is typically configured to use the internal dedicated synchronization signal line by setting both the input and output lines to VTEX1629_LXI_LINE_NONE.

The synchronization mode on a slave device is set to VTEX1629_SYNC_MODE_SLAVE and the trigger bus line that is to be used to input the synchronization signal is specified using the vtex1629_set_sample_clock_source **inLine** parameter.

### *Trigger Source*

An EX1629 device can utilize a variety of trigger sources. The simplest is the immediate trigger source. This causes the trigger state machine to bypass the TRIG layer and automatically begin to acquire data. The device can also be configured to receive its trigger signal from either the positive or negative edge transition of one of the LXI Trigger Bus lines. The final, and most complicated, configuration is to generate a trigger signal based on a specified pattern of conditions. A standalone device will either utilize an immediate or pattern trigger source. A slave device will either also utilize an immediate trigger source or will specify a trigger bus line to receive a pattern trigger source that is generated by a master device. The trigger source is specified using the vtex1629_set_trigger_source function.

The vtex1629_set_pattern_trig_configuration function is used to configure the trigger pattern for a master or standalone device that is using a pattern trigger source. This function can be used to specify a combination of LXI Trigger Bus, digital I/O, timer, and software trigger events that will

generate a trigger event. In addition to the LXI Trigger Bus lines that may be used as pattern inputs, a trigger bus line must be used to output the pattern trigger events. This line is specified using the vtex1629_set_pattern_trig_configuration function's **lxiOutput** parameter. This output can either be used within the device or can be output to other devices using the external trigger bus. As with the sample clock and synchronization signal configurations, the same LXI Trigger Bus line can be used by the device for its trigger source or another trigger bus input can be specified using the **lxiInput** parameter. In most cases, the **lxiInput** and **lxiOutput** parameters will be the same. In a star configuration, however, the trigger event might be output on one LXI Trigger Bus line and back in on a different trigger bus line.

The vtex1629_set_lxibus_configuration function must be used to configure the **lxiOutput** line as an output with either internal only scope (if the signal will only be used within the device) or external and internal transmission scope (if the line will be output to other devices). If the **lxiInput** parameter is different from the **lxiOutput** parameter, the trigger bus line specified by **lxiInput** must be configured as an input with external and internal transmission scope. Other LXI Trigger Bus lines that are used for pattern inputs must be properly configured as inputs with external and internal transmission scope.

### *Arm Source*

The arm source is configured in the same manner as the trigger source. It utilizes the vtex1629_set_arm_source and vtex1629_set_pattern_arm_configuration functions.

### *Standalone (Single Instrument) Example Configuration*

The standalone configuration is suitable when only one instrument is required for the data acquisition or if synchronization of multiple instruments is not required. It uses the instrument's internal oscillator for acquisition. The sample code that is provided performs the following tasks:

- Sets the number of samples to acquire on a trigger to 100 (one second's worth with the 100 Sa/s sample rate)
- Reset the trigger system to return the trigger bus to its default configuration
- Sets the Trigger to Immediate
- Sets the Arm to Immediate
- Sets the internal ADC clock
- Sets the internal ADC synchronization

### *Sample Code*

```
ViStatus    result = VI_SUCCESS;

/* Configure the system to acquire 100 samples. */
result = vtex1629_set_sample_count( vi, 0, 100 );

/* Reset the trigger system to return the trigger bus to its default
configuration. */
result = vtex1629_reset_trigger_arm( vi );

/* Set the sample clock source as a standalone device using the
dedicated sample clock line. */

result = vtex1629_set_sample_clock_source( vi,
                                           VTEX1629_SAMP_CLK_MODE_MASTER,
                                          VTEX1629_LXI_LINE_NONE,
                                           VTEX1629_LXI_LINE_NONE);

/* Set the synch source as a standalone device using the dedicated
synch line. */
result = vtex1629_set_synch_source( vi,
```

```
                                                VTEX1629_SYNC_MODE_MASTER,
                                                VTEX1629_LXI_LINE_NONE,
                                                VTEX1629_LXI_LINE_NONE);

    /* Set the arm source to immediate ARM. */
    result = vtex1629_set_arm_source( vi, VTEX1629_TRIG_SRC_IMMEDIATE);

    /* Set the trigger source to immediate trigger. */
    result = vtex1629_set_trigger_source( vi,
                                          VTEX1629_TRIG_SRC_IMMEDIATE);

    /* Issue a synchronization signal since the sample clock source was
    changed. */
    result = vtex1629_soft_synch( vi );
```

***Multiple Instruments (Master/Slave) Example Configuration***

The Master/Slave configuration is suitable for larger acquisition systems, up to several thousand channels (hundreds of instruments). By sharing a single oscillator and utilizing a synchronization signal from the master, the acquisition of all instruments can be coordinated and phase aligned.

One instrument is assigned the role of "master" and its internal oscillator and synchronization signals are routed externally via the LXI Trigger Bus. The remaining "slave" instruments are configured to accept the external clock and synchronization signals from the LXI Trigger Bus. Through the proper sequencing of instrument driver calls, and the resultant hardware events and signals, the ensemble of instruments can be made to behave as one large acquisition system, as illustrated by the sample code below. The following code:

- Sets the number of samples to acquire on a trigger to 100 (one second's worth with the 100 Sa/s sample rate)
- Properly configures the LXI Trigger Bus and DIO banks
- Sets the ADC clock to master/slave on LXI 0
- Sets ADC synchronization to master/slave on LXI 1
- Configures a timer trigger on LXI 2
- Configures a pattern arm on LXI 3

| **NOTE** | Master/Slave configuration requires the use of LXI Trigger Bus Cables, terminators, and, possibly, LXI Trigger Bus hubs, switches, or repeaters. Please talk to your application engineer for further information. |
|---|---|

***Sample Code***

```
    ViStatus    result = VI_SUCCESS;
    ViInt16     trigLxiLines[4] = {0,0,0,0};
    ViInt16     trigDioLines[4] = {0,0,0,0};
    ViInt16     armLxiLines[4] = {0,0,0,0};
    ViInt16     armDioLines[4] = {0,0,0x09,0x06};

    /* Configure the master system to acquire 100 samples. */
    result = vtex1629_set_sample_count( master_vi, 0, 100 );

    /* Configure the slave system to acquire 100 samples. */
    result = vtex1629_set_sample_count( slave_vi, 0, 100 );

    /* Reset the trigger system on the slave device to return it to its
    default configuration. The slave devices should be reset first to
    switch them back to using their internal sample clock oscillators
    before changing the configuration of the master.*/
    result = vtex1629_reset_trigger_arm( slave_vi );
```

```
/* Reset the trigger system on the master device. */
result = vtex1629_reset_trigger_arm( master_vi );

/* Configure the LXI trigger bus lines on the master device. Lines 0
- 3 are external outputs while lines 4 - 7 are outputs that remain
within the device. */
result = vtex1629_set_lxibus_configuration( master_vi,
                                    VTEX1629_LXI_LINE_ZERO,
                                    VTEX1629_LXI_OUTPUT,
                                    VTEX1629_LXI_INTERNAL_EXTERNAL );

result = vtex1629_set_lxibus_configuration( master_vi,
                                    VTEX1629_LXI_LINE_ONE,
                                    VTEX1629_LXI_OUTPUT,
                                    VTEX1629_LXI_INTERNAL_EXTERNAL );

result = vtex1629_set_lxibus_configuration( master_vi,
                                    VTEX1629_LXI_LINE_TWO,
                                    VTEX1629_LXI_OUTPUT,
                                    VTEX1629_LXI_INTERNAL_EXTERNAL );

result = vtex1629_set_lxibus_configuration( master_vi,
                                    VTEX1629_LXI_LINE_THREE,
                                    VTEX1629_LXI_OUTPUT,
                                    VTEX1629_LXI_INTERNAL_EXTERNAL );

result = vtex1629_set_lxibus_configuration( master_vi,
                                    VTEX1629_LXI_LINE_FOUR,
                                    VTEX1629_LXI_OUTPUT,
                                    VTEX1629_LXI_INTERNAL );

result = vtex1629_set_lxibus_configuration( master_vi,
                                    VTEX1629_LXI_LINE_FIVE,
                                    VTEX1629_LXI_OUTPUT,
                                    VTEX1629_LXI_INTERNAL );

result = vtex1629_set_lxibus_configuration( master_vi,
                                    VTEX1629_LXI_LINE_SIX,
                                    VTEX1629_LXI_OUTPUT,
                                    VTEX1629_LXI_INTERNAL );

result = vtex1629_set_lxibus_configuration( master_vi,
                                    VTEX1629_LXI_LINE_SEVEN,
                                    VTEX1629_LXI_OUTPUT,
                                    VTEX1629_LXI_INTERNAL );

/* Configure DIO bank 0 as inputs on the master device. */
result = vtex1629_set_dio_bank0_direction( master_vi,
                                    VTEX1629_DIO_DIRECTION_IN );

/* Configure the sample clock on the master to output on LXI0. */
result = vtex1629_set_sample_clock_source( master_vi,
                                    VTEX1629_SAMP_CLK_MODE_MASTER,
                                    VTEX1629_LXI_LINE_ZERO,
                                    VTEX1629_LXI_LINE_ZERO );

/* Configure the synch source on the master to output on LXI1. */
result = vtex1629_set_synch_source( master_vi,
                                    VTEX1629_SYNC_MODE_MASTER,
                                    VTEX1629_LXI_LINE_ONE,
                                    VTEX1629_LXI_LINE_ONE );
```

```
/* Configure the trigger timer on the master to trigger every 5
seconds. */
result = vtex1629_set_trigger_timer( master_vi, 5 );

/* Configure the trigger pattern to generate an event based on the
timer. */
result = vtex1629_set_pattern_trig_configuration( master_vi,
                                    trigLxiLines,
                                    trigDioLines,
                                    VI_TRUE,
                                    VTEX1629_LXI_LINE_TWO,
                                    VTEX1629_LXI_LINE_TWO );

/* Configure the trigger source on the master as a pattern trigger.
*/
result = vtex1629_set_trigger_source( master_vi,
                                    VTEX1629_TRIG_SRC_PATTERN );

/* Configure the ARM pattern to generate an event on LXI3 when DIO0
and 3 are high and DIO 1 and 2 are low. */
result = vtex1629_set_pattern_arm_configuration( master_vi,
                                    armLxiLines,
                                    armDioLines,
                                    VI_FALSE,
                                    VTEX1629_LXI_LINE_THREE,
                                    VTEX1629_LXI_LINE_THREE );

/* Configure the ARM source on the master as a pattern ARM. */
result = vtex1629_set_arm_source( master_vi,
VTEX1629_TRIG_SRC_PATTERN );


/* Configure the LXI trigger bus lines on the slave device. Lines 0 -
3 are external inputs while lines 4 - 7 are outputs that remain
within the device. */
result = vtex1629_set_lxibus_configuration( slave_vi,
                                    VTEX1629_LXI_LINE_ZERO,
                                    VTEX1629_LXI_INPUT,
                                    VTEX1629_LXI_INTERNAL_EXTERNAL );

result = vtex1629_set_lxibus_configuration( slave_vi,
                                    VTEX1629_LXI_LINE_ONE,
                                    VTEX1629_LXI_INPUT,
                                    VTEX1629_LXI_INTERNAL_EXTERNAL );

result = vtex1629_set_lxibus_configuration( slave_vi,
                                    VTEX1629_LXI_LINE_TWO,
                                    VTEX1629_LXI_INPUT,
                                    VTEX1629_LXI_INTERNAL_EXTERNAL );

result = vtex1629_set_lxibus_configuration( slave_vi,
                                    VTEX1629_LXI_LINE_THREE,
                                    VTEX1629_LXI_INPUT,
                                    VTEX1629_LXI_INTERNAL_EXTERNAL );

result = vtex1629_set_lxibus_configuration( slave_vi,
                                    VTEX1629_LXI_LINE_FOUR,
                                    VTEX1629_LXI_OUTPUT,
                                    VTEX1629_LXI_INTERNAL );

result = vtex1629_set_lxibus_configuration( slave_vi,
                                    VTEX1629_LXI_LINE_FIVE,
                                    VTEX1629_LXI_OUTPUT,
```

```
                                        VTEX1629_LXI_INTERNAL );

    result = vtex1629_set_lxibus_configuration( slave_vi,
                                        VTEX1629_LXI_LINE_SIX,
                                        VTEX1629_LXI_OUTPUT,
                                        VTEX1629_LXI_INTERNAL );

    result = vtex1629_set_lxibus_configuration( slave_vi,
                                        VTEX1629_LXI_LINE_SEVEN,
                                        VTEX1629_LXI_OUTPUT,
                                        VTEX1629_LXI_INTERNAL );

    /* Configure the sample clock on the slave as an input from LXI0. */
    result = vtex1629_set_sample_clock_source( slave_vi,
                                        VTEX1629_SAMP_CLK_MODE_MASTER,
                                        VTEX1629_LXI_LINE_ZERO,
                                        VTEX1629_LXI_LINE_NONE );

    /* Configure the synch source on the slave as an input from LXI1. */
    result = vtex1629_set_synch_source( slave_vi,
                                        VTEX1629_SYNC_MODE_MASTER,
                                        VTEX1629_LXI_LINE_ONE,
                                        VTEX1629_LXI_LINE_NONE );

    /* Configure the trigger source on the slave as an input from LXI2.
    */
    result = vtex1629_set_trigger_source( slave_vi,
                                        VTEX1629_TRIG_SRC_LXI2_POS );

    /* Configure the ARM source on the slave as an input from LXI3. */
    result = vtex1629_set_arm_source( slave_vi,
                                        VTEX1629_TRIG_SRC_LXI3_POS );


    /* Issue a soft synch command to the master to generate a
    synchronization signal since the sample clock source was changed. */
    result = vtex1629_soft_synch( master_vi )
```

## RETRIEVING DATA (READ FIFO AND STREAMING DATA)

The EX1629 stores acquisition data in a large, on-board FIFO in the instruments RAM memory. 48 MB of RAM are reserved for the on-board FIFO. There are two primary mechanisms for retrieving acquisition data from the EX1629 FIFO:

- Read FIFO
- Asynchronous Streaming Data

The Read FIFO mechanism is similar to the way data is returned from traditional data acquisition instruments, with the user application making periodic (polling) queries of the instrument to retrieve data from the instrument's on-board FIFO, while asynchronous data streaming is a more modern, efficient technique, made possible by the instrument's LXI interface, in which the instrument automatically transmits acquisition data to the user application as data becomes available.

The streaming data interface is slightly more complicated to use than the Read FIFO interface, but makes very efficient use of the host computer's processor and the test system's network. As such, the streaming data interface scales well for high channel count and/or high sample rate systems. The two data retrieval mechanisms are mutually exclusive – if the streaming data interface is enabled, Read FIFO requests will return an error.

***Read FIFO***

Use of the Read FIFO mechanism is straight-forward. After properly configuring the system and initializing acquisition (vtex1629_trig_init), a user-application queries the instrument's FIFO for data using the vtex1629_read_fifo function (or the vtex1629_read_fifoEx function – see below). This function takes the number of data scans to retrieve as an argument, along with a timeout value in seconds. The instrument driver attempts to retrieve the requested number of samples from the instrument, returning to the user-application when either the request has been fulfilled, or the timeout elapses. To fulfill the request, the instrument driver may need to make many, repeated queries of the instrument, each query being a network transaction with the instrument. The instrument driver continues "polling" the instrument for data. During the instrument driver call, the user-application "blocks", meaning that the user-application does not continue to the next instructions until the vtex1629_read_fifo function returns.

**Error! Reference source not found.** illustrates the general sequence of events when using Read FIFO. In this diagram, time flows from top to bottom. The two vertical lines represent the two network nodes: the host computer, running the user application, and the EX1629 instrument. The diagonal arrows connecting them represent network messages sent between them (the diagonal arrow, instead of a horizontal arrow, indicates that the message is not received instantaneously). As illustrated in this diagram, each instrument driver function call results in two network messages: one to the EX1629 (a request) and one from the EX1629 (a response). The instrument driver function does not return control to the user application until the response message is received.

As can be seen by the vertical distances, each of the instrument driver function calls takes some finite amount of time, allowing for host computer processing, network transmission and reception, and instrument processing. For instrument setup (e.g., configuring acquisition channels or trigger parameters), this time is typically negligible. For retrieving acquisition data, however, these delays can become significant. This is especially true in high sample count and/or high channel count systems. In such systems, the host computer can waste a significant amount of CPU time in these polling loops, also consuming network bandwidth. The streaming data mechanism offers a more efficient alternative.

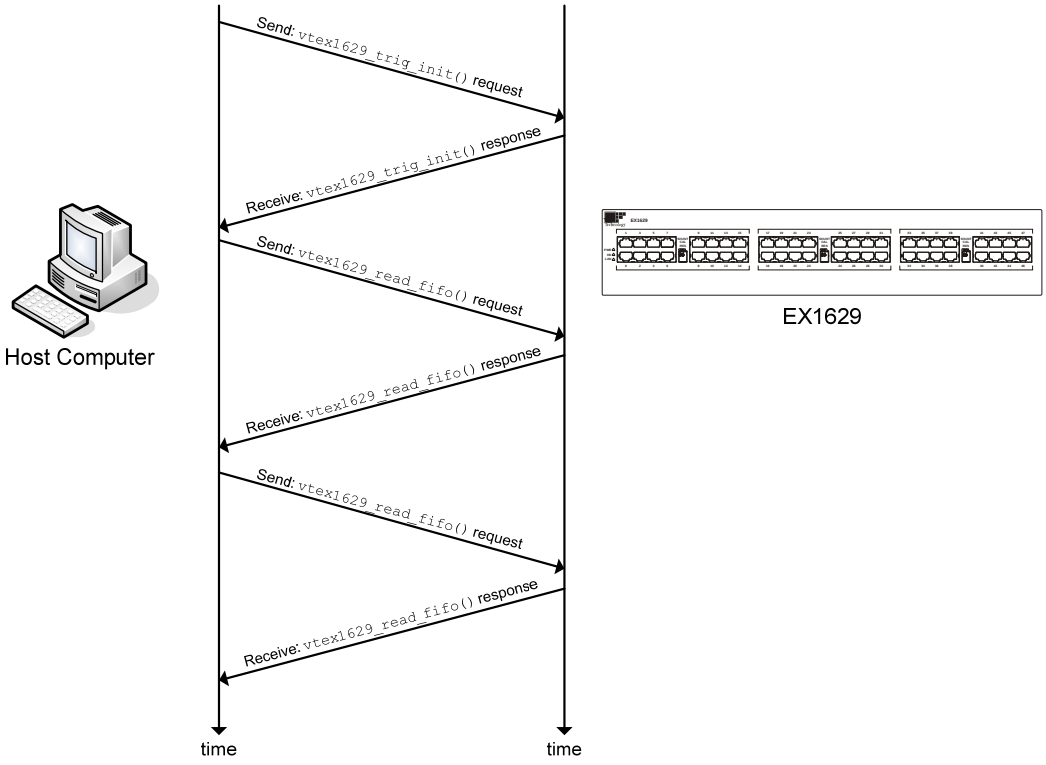



**FIGURE 6-1: READ FIFO NETWORK EXAMPLE**

The vtex1629_read_fifo function provides access to the main channel acquisition data and timestamp information. The following code segment illustrates the basic usage of the vtex1629_read_fifo function, issuing a FIFO query to return 10 scans of data with all 48 channels enabled. The maximum number of scans to return is specified, as is the maximum number of data elements to be returned. The acquisition data, along with the timestamp for each scan, is returned in three arrays of ViReal64 elements, two for the timestamps (seconds and fractional seconds) and one for the acquisition data. Each scan of data will contain 1 to 48 channels worth of data, depending on the scanlist configuration. Scan data is returned sequentially within the acqdata array, so, for the example below, acqdata[0]…acqdata[47] will contain the first scan of data, acqdata[48]….acqdata[95] the second scan of data, etc. The timestamp for the first scan will be contained in seconds[0] and fractseconds[0], the second timestamp in seconds[1] and fractseconds[1], etc.

```
#define NUM_SCANS 10
#define NUM_CHANNELS 48
#define MAX_NUM_SAMPLES (NUM_SCANS * NUM_CHANNELS)
#define TIMEOUT_SECS 5

ViSession vi;
ViReal64  seconds[NUM_SCANS];
ViReal64  fractseconds[NUM_SCANS];
ViReal64  acqdata[MAX_NUM_SAMPLES];
ViInt32   numdata, numscans;

result = vtex1629_read_fifo( vi,
                             NUM_SCANS,
                             seconds,
                             fractseconds,
                             &numscans,
                             MAX_NUM_SAMPLES,
```

```
                                    acqdata,
                                    &numdata,
                                    TIMEOUT_SECS);
```

The actual number of scans and data returned by vtex1629_read_fifo may be equal to or less than the values requested (NUM_SCANS and MAX_NUM_SAMPLES in the example) if the timeout period (TIMEOUT_SECS in the example) expires before the EX1629 has acquired the requested number of samples. The actual number of scans returned and data values returned are stored in numscans and numdata, respectively, by vtex1629_read_fifo.

Additionally, there is a vtex1629_read_fifoEx function that provides access to acquisition data from the confidence measurement system. Please refer to the function references for further details.

***Asynchronous Streaming Data***

The asynchronous streaming data interface optimizes communication between the host computer and the EX1629. The asynchronous streaming data interface allows the EX1629 to transmit acquisition data to the host computer whenever data is available. It "streams" data to the host computer – that is the EX1629 transmits data when available – and is "asynchronous" in that data arrives outside the normal control flow of the user-application. This is in contrast to the Read FIFO mechanism, where the client polls or queries the instrument for data, and the data is returned to the user application when the vtex1629_read_fifo function returns (synchronous with the normal program control flow).

**Error! Reference source not found.** illustrates the general sequence of events when using the streaming data mechanism. This can be compared to **Error! Reference source not found.** for using the Read FIFO mechanism. Prior to initiating the acquisition (vtex1629_trig_init), the streaming data interface must be enabled via the vtex1629_enable_streaming_data function). This configures the streaming data communication on both the instrument and the host computer. It is important that the streaming data interface be enabled prior to initiating acquisition, as the EX1629 prevents streaming data from being enabled after initiating acquisition. As **Error! Reference source not found.** shows, the EX1629 transmits acquisition data to the host computer periodically, whenever data is available, without the host having to request it.

The streaming data interface uses a separate "socket", or communications link, than the one used for other instrument driver functions. Since TCP/IP can support thousands of concurrent sockets, all multiplexed on the same network interface, this does not present a problem for the network.

| **NOTE** | The network communication diagrams provided are oversimplifications. Since TCP/IP is used as the transport layer, there are potentially several Ethernet packets involved (send and receive) in each high-level message. These packets provide, among other things, the reliable data transport feature of TCP. |
|---|---|

The asynchronous nature of the streaming data arrival at the host computer presents the issue of how to deliver the data to the user application. For efficiency, particularly when the acquisition system consists of many instruments, a multi-threaded model was chosen.

Multi-threaded programming is beyond the scope of this manual, but the general idea is that an application can have multiple, concurrent "threads" of control. By default, all applications have one thread, the one that begins executing at the main() function (or similar entry point, depending on the programming language). Optionally, applications may have additional, programmer created threads. These threads all execute in the same memory space, making it very efficient for them to share data. This is different from multi-process programming, wherein each process – basically a memory space with a single, default thread – executes independently.

Threads execute asynchronously to each other by default – that is, their execution relative to other threads within the same application (process) is non-deterministic, and shared data must be protected by design or through suitable inter-thread communication mechanisms (e.g., mutexes) to guarantee consistency. Again, multi-threaded programming is beyond the scope of this manual, but it is important to understand the fundamentals before the streaming data mechanism can be used properly. For more information on this topic, we recommend reviewing a textbook on operating systems (e.g., *Operating System Concepts*, by Silberschatz, Galvin, and Gagne or *Modern Operating Systems*, by Tanenbaum) as well as the Windows SDK information available online.

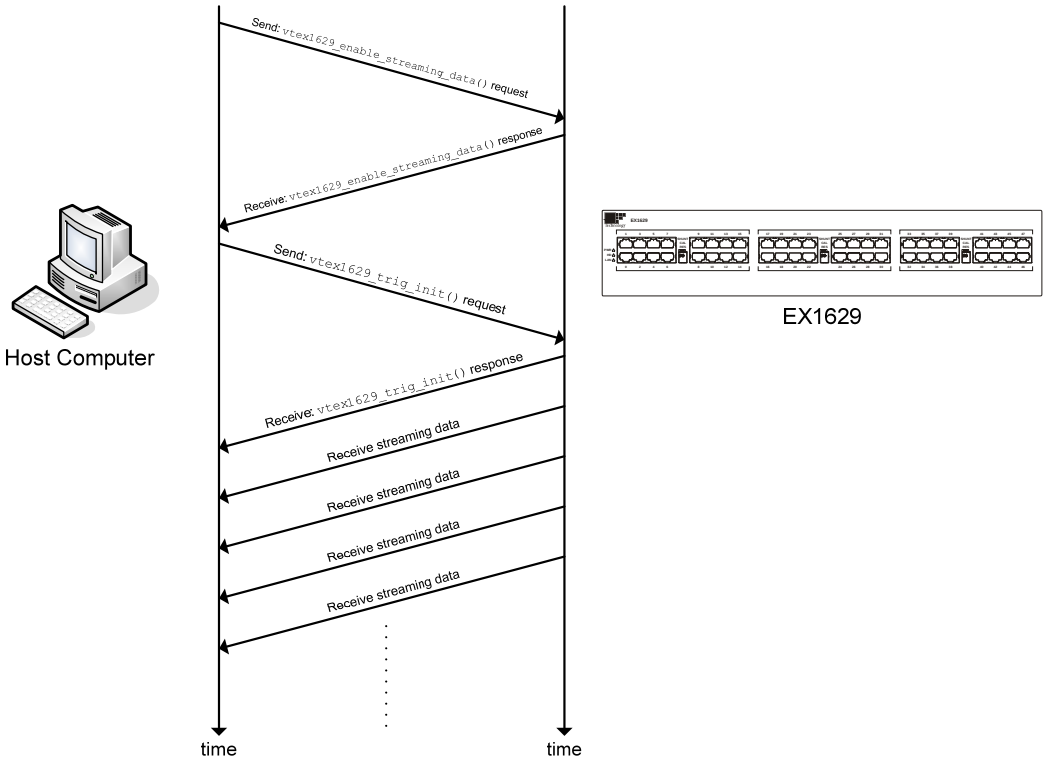



**FIGURE 6-2: STREAMING DATA NETWORK EXAMPLE**

***Basic Streaming Data Usage***

When using the streaming data interface, via the vtex1629_enable_streaming_data function, the user application provides a callback function. Internally, the instrument driver creates a thread and then opens a socket for streaming data between the host computer and the instrument. The newly constructed thread does a "blocking" read on the socket, which causes it to "sleep" (become idle) until data arrives. When acquisition data arrives, the thread begins executing, receives the acquisition data from the instrument, executes the user-provided callback function, passing in the newly arrived data, and then returns to the "sleep" state. The callback function can do whatever is necessary for the application: write the acquisition values to a file on disk, perform limit checking on the acquisition values, update an application-specific data structure (e.g., FIFO) post the acquisition data to a database or spreadsheet, etc. This behavior is illustrated in Figure 6-3, with the reception of streaming data causing the user-provided callback function to be executed.

| NOTE | Since the callback function executes asynchronously in the same process as the main application thread, it is important that any data or data structures used by both threads are suitably protected to guarantee consistency. As with any multi-thread application, care must be taken when using inter-thread communication primitives (e.g., mutexes) to prevent deadlocks and livelocks. Similarly, performing GUI operations (e.g., updating an on-screen graph) within the callback function needs to be implemented carefully. |
|---|---|

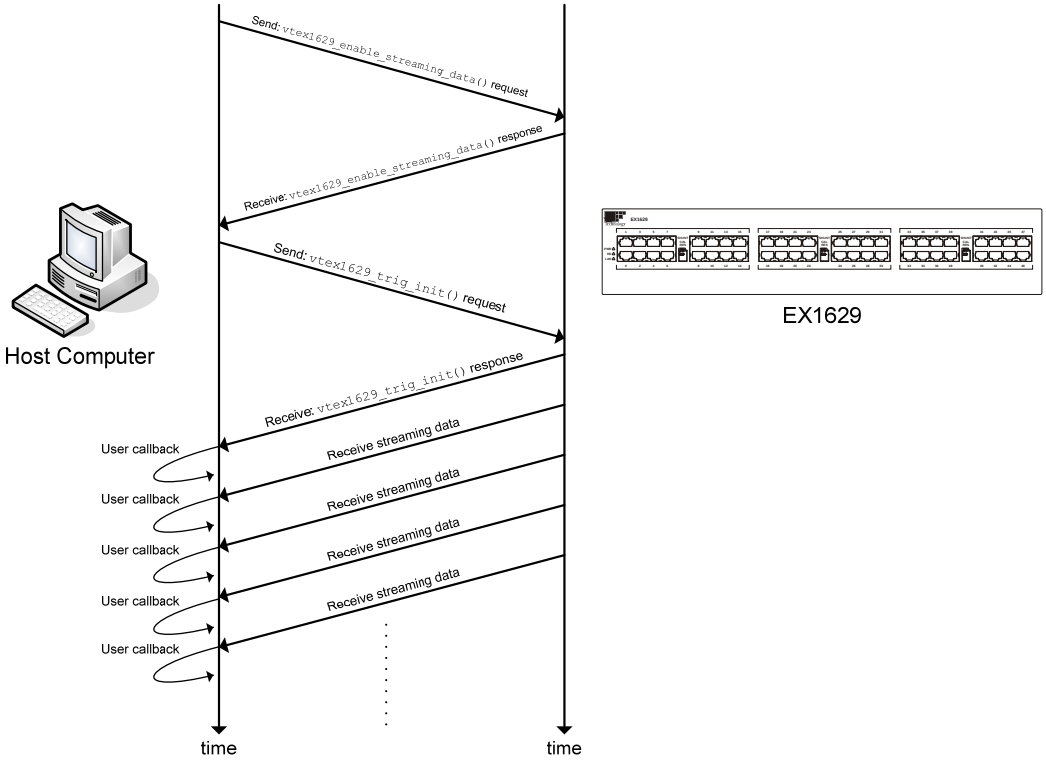



**FIGURE 6-3: STREAMING DATA WITH USER CALLBACK**

The following sample code segment illustrates a very basic use of the streaming data interface. The callback function, stream_callback, just prints the timestamps and data values to a FILE handle. The FILE handle, as well as a sample count total variable, are stored in a user-defined data structure. A pointer to this structure is passed to the vtex1629_enable_streaming_data function, along with a function pointer to the streaming callback function. Later, when streaming data pages (scans) are received, a pointer to the acquisition data, along with the pointer to the user-defined data structure, are passed to the callback function. The user-defined data structure pointer is passed as a void* and should be cast to the appropriate type within the callback function.

```
#define INSTR_LANGUAGE_SPECIFIC
#include<vtex1629.h>

typedef struct {
  FILE      *fout;
  ViInt32   sample_count;
} user_struct;

user_struct my_struct = {0};

ViInt32 stream_callback( void *priv, EX1629_rpc_datapage *data )
{
  user_struct  *priv_struct; // pointer to user structure
```

```
    ViInt32      ds_idx;       // dataset index
    ViInt32      smp_idx;      // sample index

    priv_struct = (user_struct *)priv;

    /* Loop through all of the datasets in the datapage. */
    for( ds_idx = 0; ds_idx < data->dataset.dataset_len && ds_idx <
       MAX_NUM_DATASETS; ds_idx++ ) {
       /* print the timestamp. */
       fprintf( priv_struct->fout,
                "Timestamp: %5u.%09u",
               data->dataset.dataset_val[ds_idx]->timestamp_sec,
               data->dataset.dataset_val[ds_idx]->timestamp_nsec);

       /* print the data */
       for( smp_idx = 0;
            smp_idx < data->dataset.dataset_val[ds_idx]->data.data_len;
            smp_idx++ ) {

          fprintf( priv_struct->fout,
            "\t%i %f\n",
            data->dataset.dataset_val[ds_idx]->data.data_val[smp_idx]);
       }
    }
    if( data->dataset.dataset_len > 0) {
       priv_struct->sample_count++;
    }
  }

  result = vtex1629_enable_streaming_data( vi,
                                           &my_struct,
                                            stream_callback );

  result = vtex1629_trig_init( vi );

  // application code…

  result = vtex1629_abort( vi );

  result = vtex1629_disable_streaming_data( vi );
```

It is imperative that the streaming data interface be enabled prior to initializing acquisition (the vtex1629_trig_init function) and disabled after acquisition completes, or is aborted explicitly (vtex1629_abort).

The streaming callback function extracts the acquisition and timestamp data from the EX1629_rpc_datapage structure. Each datapage contains three data sets, which contains data for a range of channels: the first data set contains data for channels 0 through 15, the second for 16 through 31, and the third contains data for channels 32 through 47. Only data for channels enabled in the scanlist are included. That is, if the main input scanlist contains channels 0 through 15 and 16, the first data set (the zeroeth) will contain 16 samples (channels 0 through 15), the second data set will contain 1 sample (channel 16), and the third data set will contain no samples.

Beyond the timestamp and acquisition values illustrated in the example above, the EX1629_rpc_datapage structure also contains DIO sample data as well as data from the Confidence Measurement Subsystem. The EX1629_rpc_datapage structure is documented in the instrument driver header file.

***Advanced Data Streaming Usage***

In addition to the basic streaming data supported via the vtex1629_enable_streaming_data function, the vtex1629_enable_streaming_dataEx function supports a more advanced streaming interface. Where the vtex1629_enable_streaming_data automatically creates a thread, opens the socket, configures the instrument for streaming data, etc., the vtex1629_enable_streaming_dataEx only configures the instrument for streaming data, making the user application responsible for the other tasks. This provides the application developer more control over the data streaming mechanism than is allowed by the vtex1629_enable_streaming_data function. This, clearly, makes use of the vtex1629_enable_streaming_dataEx function more complicated than the vtex1629_enable_streaming_data function.

For most applications, the basic streaming data interface provides sufficient execution speed and flexibility. If you believe your application would benefit from the advanced streaming interface, please contact your application engineer for further details.

***Calibration Data***

Once calibration is run, a file with this data is stored in memory. To view this data, the following code could be generated. This code determines the size of the calibration file first in order to prevent overwriting any data that may already exist in the external memory location to which the calibration file will be written. In this example, the self-calibration and full calibration files are output in XML format.

***Sample Code***

```
ViStatus status = VI_SUCCESS;
ViChar errMessage[256];

ViInt32 fileType = 0;
ViInt32 bufferSize = 0;
ViString xmlBuffer_combined = 0;
ViInt32 actualSize = 0;


memset(errMessage, 0x00, sizeof(errMessage));
fileType = VTEX1629_CAL_DATA_COMBINED;

status = vtex1629_get_cal_file_size(instrumentHandle,
                                    fileType,
                                    &bufferSize);
if(status < VI_SUCCESS) {
     <inform the user the API call failed>
}

if( (status >= VI_SUCCESS) && (bufferSize > 0) ) {
     xmlBuffer_combined = malloc( bufferSize * sizeof( ViString ) );

     status = vtex1629_get_cal_file(instrumentHandle,
                                    fileType,
                                    bufferSize,
                                    xmlBuffer_combined,
                                    &actualSize);
     if(status < VI_SUCCESS) {
        <inform the user the API call failed>      }

...

     free(xmlBuffer_combined);
}
```

## STARTING/STOPPING ACQUISITION

Acquisition is started by using the vtex1629_trig_init function, which initializes the trigger subsystem. This causes the trigger state machine to transition from the IDLE state to the Waiting for ARM state (see Figure 4-1). Depending on the configuration of the Arm and Trigger sources, data acquisition may begin immediately – if both Arm and Trigger are configured for Immediate (VTEX1629_TRIG_SRC_IMMEDIATE) –, or at some point in the future when the instrument receives the required signals (e.g., a raising edge on LXI 0). In general, after acquisition is initiated by the vtex1629_trig_init function, it will continue until the requested number of samples has been acquired, possibly several times, depending on the configuration of the Arm and Trigger Count variables, and the value of Continuous Init.

```
status = vtex1629_trig_init( vi );
```

The acquisition may be halted at any time by using the vtex1629_abort function. This causes acquisition to stop, and the trigger state machine transitions to the IDLE state (see Figure 4-1).

```
status = vtex1629_abort( vi );
```

# SECTION 7

# FUNCTION CALLS

## INTRODUCTION

This section presents the instrument function set. It begins by listing the APIs according to function and is then followed by an alphabetical listing. With each function is a brief description. The remainder of this section is devoted to describing each function in detail. Each function entry provides the function prototype, the use and range of parameters, and a description of the function's purpose.

## FUNCTION RETURN VALUE

Each function will return a status that will contain either VI_SUCCESS or an error status returned by the function call. Refer to the Error Messages section found later in the chapter for possible error codes. If the vtex1629_error_message function call is used, it will return a description of the error code returned by the last function call made.

## FUNCTION TREE

The function set for the EX1629 has been categorized according to function and is presented below.

### *Initialize*

| | |
|---|---|
| Initialize | vtex1629_init |

### *Limit Checking*

| | |
|---|---|
| Set Bridge Limit Enabled | vtex1629_set_bridge_limit_enabled |
| Get Bridge Limit Enabled | vtex1629_get_bridge_limit_enabled |
| Set Bridge Limit | vtex1629_set_bridge_limit |
| Get Bridge Limit | vtex1629_get_bridge_limit |
| Set Confidence Reporting Mode | vtex1629_set_confidence_reporting_mode |
| Get Confidence Reporting Mode | vtex1629_get_confidence_reporting_mode |
| Set Confidence Limit | vtex1629_set_confidence_limit |
| Get Confidence Limit | vtex1629_get_confidence_limit |

### *Configuration Calls*

| | |
|---|---|
| Store Current Configuration | vtex1629_store_current_config |
| Load Stored Configuration | vtex1629_load_stored_config |
| Clear Stored Configuration | vtex1629_clear_stored_config |
| Get Current Configuration Digest | vtex1629_get_current_config_digest |
| Get Stored Configuration Digest | vtex1629_get_stored_config_digest |
| Compare Digests | vtex1629_compare_digests |

### *Lock Function Calls*

| | |
|---|---|
| Break Lock on Instrument | vtex1629_break_lock |
| Check Lock on Instrument | vtex1629_check_lock |
| Lock Instrument | vtex1629_lock |
| Unlock Instrument | vtex1629_unlock |

### *Digital Input/Output Calls*

| | |
|---|---|
| Send Pulse | vtex1629_send_dio_pulse |
| Configuration Read | |
| Get Bank 0 Direction | vtex1629_get_dio_bank0_direction |
| Get Bank 0 Pull-up | vtex1629_get_dio_bank0_pullup |
| Get Bank 1 Direction | vtex1629_get_dio_bank1_direction |
| Get Bank 1 Pull-up | vtex1629_get_dio_bank1_pullup |
| Get Output State | vtex1629_get_dio_output |
| Get Input State | vtex1629_get_dio_input |
| Configuration Write | |
| Set Bank 0 Direction | vtex1629_get_dio_bank0_direction |
| Set Bank 0 Pull-up | vtex1629_set_dio_bank0_pullup |
| Set Bank 1 Direction | vtex1629_set_dio_bank1_direction |
| Set Bank 1 Pull-up | vtex1629_set_dio_bank1_pullup |
| Set Output State | vtex1629_set_dio_output |
| Event Control | |
| Set DIO Configuration Events | vtex1629_set_dio_config_events |
| Get DIO Configuration Events | vtex1629_get_dio_config_events |
| Clear DIO Events | vtex1629_dio_clear_event |
| Clear All DIO Events | vtex1629_dio_clear_events_all |

### *LXI Trigger Bus Calls*

| | |
|---|---|
| Send LXI Bus Pulse | vtex1629_send_lxibus_pulse |
| Get LXI Bus Configuration | vtex1629_get_lxibus_configuration |
| Get LXI Bus Output | vtex1629_get_lxibus_output |
| Get LXI Bus Input | vtex1629_get_lxibus_input |
| Set LXI Bus Configuration | vtex1629_set_lxibus_configuration |
| Set LXI Bus Output | vtex1629_set_lxibus_output |

### *Scanlist Calls*

| | |
|---|---|
| Get Channel Scanlist | vtex1629_get_scanlist |
| Set Channel Scanlist | vtex1629_set_scanlist |
| Get Gain | vtex1629_get_gain |
| Get Completion Resistor Value | vtex1629_get_completion_resistor |
| Get Input Multiplexer | vtex1629_get_input_multiplexer |
| Get Sampling Frequency | vtex1629_get_sample_frequency |
| Get Confidence Scanlist | vtex1629_get_conf_scanlist |
| Set Gain | vtex1629_set_gain |
| Set Completion Resistor Mode | vtex1629_set_completion_resistor |
| Set Input Multiplexer | vtex1629_set_input_multiplexer |
| Set Sampling Frequency | vtex1629_set_sample_frequency |
| Set Confidence Scanlist | vtex1629_set_conf_scanlist |

### *Trigger System Calls*

| | |
|---|---|
| Trigger Initiate | vtex1629_trig_init |
| Software Arm | vtex1629_soft_arm |
| Software Trigger | vtex1629_soft_trig |
| Software Synchronize | vtex1629_soft_synch |
| Reset | vtex1629_reset_trigger_arm |
| Abort | vtex1629_abort |
| Get Sample Count | vtex1629_get_sample_count |
| Get Arm Source | vtex1629_get_arm_source |

| | |
|---|---|
| Get Arm Count | vtex1629_get_arm_count |
| Get Arm Delay | vtex1629_get_arm_delay |
| Get Trigger Source | vtex1629_get_trigger_source |
| Get Trigger Count | vtex1629_get_trigger_count |
| Get Trigger Delay | vtex1629_get_trigger_delay |
| Get Trigger Timer | vtex1629_get_trigger_timer |
| Get Synchronization Source | vtex1629_get_synch_source |
| Get Sample Clock Source | vtex1629_get_sample_clock_source |
| Get Arm Pattern Configuration | vtex1629_get_pattern_arm_configuration |
| Get Trig Pattern Configuration | vtex1629_get_pattern_trig_configuration |
| Set Sample Count | vtex1629_set_sample_count |
| Set Arm Source | vtex1629_set_arm_source |
| Set Arm Count | vtex1629_set_arm_count |
| Set Arm Delay | vtex1629_set_arm_delay |
| Set Trigger Source | vtex1629_set_trigger_source |
| Set Trigger Count | vtex1629_set_trigger_count |
| Set Trigger Delay | vtex1629_set_trigger_delay |
| Set Trigger Timer | vtex1629_set_trigger_timer |
| Set Synchronization Source | vtex1629_set_synch_source |
| Set Sample Clock Source | vtex1629_set_sample_clock_source |
| Set Arm Pattern Configuration | vtex1629_set_pattern_arm_configuration |
| Set Trig Pattern Configuration | vtex1629_set_pattern_trig_configuration |
| Set Trigger Source Timer | vtex1629_set_trigger_source_timer |

### *Filter Configuration Calls*

| | |
|---|---|
| Get IIR Filter Configuration | vtex1629_get_IIR_filter_configuration |
| Set IIR Filter Configuration | vtex1629_set_IIR_filter_configuration |
| Get Settling Time | vtex1629_get_settling_time |

### *Excitation Voltage Calls*

| | |
|---|---|
| Get Programmed Excitation Voltage | vtex1629_get_excitation |
| Get Excitation Voltage Enabled | vtex1629_get_excitation_enabled |
| Set Programmed Excitation Voltage | vtex1629_set_excitation |
| Set Excitation Voltage Enabled | vtex1629_set_excitation_enabled |

### *EU Conversion Calls*

| | |
|---|---|
| Get EU Conversion Type | vtex1629_get_EU_conversion |
| Get EU Conversion Excitation Voltage | vtex1629_get_euconv_excitation |
| Get Unstrained Voltage | vtex1629_get_unstrained_voltage |
| Get Gage Factor | vtex1629_get_gauge_factor |
| Get Poisson Ratio | vtex1629_get_poisson_ratio |
| Get Strain Units | vtex1629_get_strain_units |
| Get Tare Value | vtex1629_get_tare |
| Set EU Conversion Type | vtex1629_set_EU_conversion |
| Set EU Conversion Excitation Voltage | vtex1629_set_euconv_excitation |
| Set Unstrained Voltage | vtex1629_set_unstrained_voltage |
| Set Gage Factor | vtex1629_get_gauge_factor |
| Set Poisson Ratio | vtex1629_set_poisson_ratio |
| Set Strain Units | vtex1629_set_strain_units |
| Set Tare Values | vtex1629_set_tare |
| Get Linear Scaling Coefficients | vtex1629_get_linearscaling_configuration |
| Set Linear Scaling Coefficients | vtex1629_set_linearscaling_configuration |
| Measure Unstrained Voltage | vtex1629_measure_unstrained_voltage |
| Measure Excitation Voltage | vtex1629_measure_excitation_voltage |
| Measure Confidence | vtex1629_measure_confidence |
| Reset Tare Values | vtex1629_reset_tare |
| Set Dynamic Excitation EU Enabled | vtex1629_set_euconv_dynamic_excitation_enabled |
| Get Dynamic Excitation EU Enabled | vtex1629_set_euconv_dynamic_excitation_enabled |

### *Shunt Configuration Calls*

| | |
|---|---|
| Get Shunt Source | vtex1629_get_shunt_source |
| Get Shunt Enabled | vtex1629_get_shunt_enabled |
| Get Shunt Value | vtex1629_get_shunt_value |
| Set Shunt Source | vtex1629_set_shunt_source |
| Set Shunt Enabled | vtex1629_set_shunt_enabled |
| Set Shunt Value | vtex1629_set_shunt_value |

### *TEDS Calls*

| | |
|---|---|
| Get TEDS Data | vtex1629_get_teds_data |
| Set TEDS Data | vtex1629_set_teds_data |
| Erase TEDS Data | vtex1629_erase_teds_data |
| Read TEDS MLAN | vtex1629_read_teds_MLAN |
| Write TEDS MLAN | vtex1629_write_teds_MLAN |
| Read TEDS URN | vtex1629_read_teds_URN |

### *Data Retrieval Calls*

| | |
|---|---|
| Get FIFO Count | vtex1629_get_fifo_count |
| Read FIFO | vtex1629_read_fifo |
| Read FIFO Extra | vtex1629_read_fifoEx |
| Reset FIFO | vtex1629_reset_fifo |

### *Data Retrieval Calls - Advanced*

| | |
|---|---|
| Enable Streaming Data | vtex1629_enable_streaming_data |
| Disable Streaming Data | vtex1629_disable_streaming_data |
| Enable Streaming Data Expert Mode | vtex1629_enable_streaming_dataEx |

### *Self-Calibration Calls*

| | |
|---|---|
| Initialize Self-Calibration | vtex1629_self_cal_init |
| Initialize Zero Calibration | vtex1629_zero_cal |
| Query Self-Calibration File in Non-vol Memory | vtex1629_self_cal_is_stored |
| Load Self-Calibration File from Non-vol Memory | vtex1629_self_cal_load |
| Store Self-Calibration File to Non-vol Memory | vtex1629_self_cal_store |
| Clear Self-Calibration File from Non-vol Memory | vtex1629_self_cal_clear_stored |
| Clear Current Self-Calibration Image | vtex1629_self_cal_clear |
| Get Self-Calibration Status | vtex1629_self_cal_get_status |
| Get Self-Calibration Failure Status | vtex1629_get_selfcal_status |
| Query Self-Calibration isRunning | vtex1629_self_cal_is_running |

### *Internal Calibration Source Calls*

| | |
|---|---|
| Get Calibration Source | vtex1629_get_cal_source |
| Set Cal Source | vtex1629_set_cal_source |
| Set Cal Out | vtex1629_set_cal_out |

### *Utility Function Calls*

| | |
|---|---|
| Reset | vtex1629_reset |
| Self-Test Functions | |
| Self-Test | vtex1629_self_test |
| Self Test Init | vtex1629_self_test_init |
| Self Test Get Status | vtex1629_self_test_get_status |
| Error Message | vtex1629_error_message |
| Error Query | vtex1629_error_query |
| Revision Query | vtex1629_revision_query |
| Instrument Discovery | vtex1629_findinstr |
| Enable Logging | vtex1629_enable_logging |
| Disable Logging | vtex1629_disable_logging |

| | |
|---|---|
| Get Serial Number | vtex1629_get_instrument_serial_number |
| Get DSP Version | vtex1629_get_dsp_version |

### *Lead Wire Calls*

| | |
|---|---|
| Measure Lead Wire Resistance | vtex1629_measure_lead_wire_resistance |
| Set Lead Wire Resistance | vtex1629_set_lead_wire_resistance |
| Get Lead Wire Resistance | vtex1629_get_lead_wire_resistance |
| Set Half-Bridge Lead Wire Desensitization | vtex1629_set_half_bridge_lead_wire_desensitization |
| Get Half-Bridge Lead Wire Desensitization | vtex1629_get_half_bridge_lead_wire_desensitization |

### *Calibration File Query*

| | |
|---|---|
| Get Calibration File Size | vtex1629_get_cal_file_size |
| Get Calibration File | vtex1629_get_cal_file |
| Calibration Coefficient Query | vtex1629_get_cal_coefficients |

### *Close*

| | |
|---|---|
| Close | vtex1629_close |

## ALPHABETICAL FUNCTION SET

The following table provides a summary of the function calls used by the EX1629 along with an abbreviated description of the function. The pages following this table are function definitions that provide in-depth detail for each function. A sample function definition is provided immediately following this table to illustrate what each section of the definition describes.

| Command | Description |
|---|---|
| vtex1629_abort | Aborts data acquisition. |
| vtex1629_allow_all_channels | Allows the user to include channels that failed calibration in the scanlist for data acquisition. |
| vtex1629_break_lock | Releases a lock on the instrument. |
| vtex1629_check_lock | Queries the lock status of the instrument. |
| vtex1629_clear_stored_config | Erases the stored configuration from nonvolatile storage. |
| vtex1629_close | Closes an instrument programming session. |
| vtex1629_compare_digests | Compares the two provided digests byte-by-byte. |
| vtex1629_dio_clear_event | Clears the DIO event configuration for the specified **inputLine**. |
| vtex1629_dio_clear_events_all | Clears the DIO event configuration for all events. |
| vtex1629_disable_logging | Stops the logging of driver calls. |
| vtex1629_disable_streaming_data | Stops streaming data from instrument. |
| vtex1629_enable_logging | Allows an application to log messages to a file for later review. |
| vtex1629_enable_streaming_data | Starts data streaming from instrument. |
| vtex1629_enable_streaming_dataEx | Start data streaming from instrument (expert mode). |
| vtex1629_erase_teds_data | Erases the data on a TEDS device for one particular channel. |
| vtex1629_error_message | Outputs the error message associated with the statusCode parameter. |
| vtex1629_findinstr | Scans the LAN for available EX1629 instruments. |
| vtex1629_get_arm_count | Queries and returns the arm count for the EX1629. |
| vtex1629_get_arm_delay | Queries and returns the arm delay for the EX1629. |
| vtex1629_get_arm_source | Queries and returns the current setting for the arm source. |
| vtex1629_get_bridge_limit | Queries and returns the minimum and maximum bridge limit values. |
| vtex1629_get_bridge_limit_enabled | Queries and returns the enabled status of the bridge limit function. |
| vtex1629_get_cal_coefficients | Queries and returns the value of a selected calibration coefficient for one or more channels. |
| vtex1629_get_cal_file | Reads up to **bufferSize** characters from the EX1629 and places them in the XML buffer. |
| vtex1629_get_cal_file_size | Returns the total buffer size required to read the cal data. |
| vtex1629_get_cal_source | Queries and returns the current setting for the calibration input source. |
| vtex1629_get_completion_resistor | Queries and returns the mode and the value of the completion resistor for a specific channel. |
| vtex1629_get_conf_scanlist | Queries and returns a list of confidence data values that are currently configured to be measured and stored with data acquisition. |
| vtex1629_get_confidence_limit | Queries and retrieves the current confidence limit settings. |
| vtex1629_get_confidence_reporting_mode | Queries and retrieves the reporting mode for confidence limit checking. |
| vtex1629_get_current_config_digest | Retrieves the digest for the current instrument configuration. |
| vtex1629_get_dio_bank0_direction | Indicates whether bank zero of the digital I/O is configured as input or output. |
| vtex1629_get_dio_bank0_pullup | Queries and returns the pull-up mode for bank zero of the digital I/O. |
| vtex1629_get_dio_bank1_direction | Indicates whether bank one of the digital I/O is configured as input or output. |
| vtex1629_get_dio_bank1_pullup | Queries and returns the pull-up mode for bank one of the digital I/O. |
| vtex1629_get_dio_config_events | Queries and returns the current setting for DIO event transitions. |
| vtex1629_get_dio_input | Queries and returns the current input state of both banks of the digital I/O. |
| vtex1629_get_dio_output | Queries and returns the current programmed output state of both banks of the digital I/O. |
| vtex1629_get_dsp_version | Returns the DSP version information for a given analog board. |
| vtex1629_get_EU_conversion | Reads the EU conversion type for a specific channel. |
| vtex1629_get_euconv_dynamic_excitation_enabled | Queries and returns the dynamic excitation EU conversion state |

| Command | Description |
|---|---|
| vtex1629_get_euconv_excitation | Queries and returns the current value used in EU conversions for the excitation voltage for a given channel. |
| vtex1629_get_excitation | Queries and returns the programmed excitation voltage for a given channel. |
| vtex1629_get_excitation_enabled | Queries and returns the enabled status of the excitation voltage for a specific channel. |
| vtex1629_get_fifo_count | Queries the EX1629 for the current FIFO page count. |
| vtex1629_get_gain | Reads the specified channel's current signal conditioning gain. |
| vtex1629_get_gauge_factor | Queries and returns the gage factor for a specific channel. This is one of the parameters used in EU conversion calculations. |
| vtex1629_get_half_bridge_lead_wire_desensitization | Queries and returns the lead wire desensitization factor for the specified channel. |
| vtex1629_get_IIR_filter_configuration | Queries and returns the IIR filter configuration parameters for a given channel. |
| vtex1629_get_input_multiplexer | Queries and returns the input multiplexer source. |
| vtex1629_get_instrument_serial_number | Returns the instrument's serial number |
| vtex1629_get_lead_wire_resistance | Queries and returns the currently defined lead wire resistance value. |
| vtex1629_get_linearscaling_configuration | Returns the slope (**m**) and intercept (**b**) parameters for a given channel. |
| vtex1629_get_lxibus_configuration | Queries and returns information pertaining to a specified LXI Trigger Bus channel. |
| vtex1629_get_lxibus_input | Queries and returns the input state of each of the channels on the LXI Trigger Bus. |
| vtex1629_get_lxibus_output | Queries and returns the output state of each of the channels on the LXI Trigger Bus. |
| vtex1629_get_pattern_arm_configuration | Queries and returns the EX1629's current configuration for the pattern arm mode of operation. |
| vtex1629_get_pattern_trig_configuration | This queries and returns the EX1629's current configuration for the pattern trigger mode of operation. |
| vtex1629_get_poisson_ratio | Queries and returns the Poisson ratio for a specific channel. This is one of the parameters used in EU conversion calculations. |
| vtex1629_get_sample_clock_source | Queries and returns the configured sample clock source. |
| vtex1629_get_sample_count | Queries and returns both the pre- and post-trigger sample counts for the EX1629. |
| vtex1629_get_sample_frequency | Queries and returns the currently configured sampling frequency for all channels on the EX1629. |
| vtex1629_get_scanlist | Queries and returns a list of channels currently configured to be sampled in the data acquisition process. |
| vtex1629_get_selfcal_status | Queries and returns self-calibration failure status for the selected channels |
| vtex1629_get_settling_time | Queries and returns the current settling time for a particular channel. |
| vtex1629_get_shunt_enabled | Queries and returns the enabled status of a particular channel's shunt resistor. |
| vtex1629_get_shunt_source | Queries and returns the shunt source for a particular channel. |
| vtex1629_get_shunt_value | Queries and returns a shunt resistor value based on a given channel and shunt source. |
| vtex1629_get_stored_config_digest | Retrieves the digest of the instrument configuration saved in non-volatile memory. |
| vtex1629_get_strain_units | Queries and returns the configured strain units for a given channel. |
| vtex1629_get_synch_source | Queries and returns the synchronization source. |
| vtex1629_get_tare | Reads the currently configured tare value for a specific channel |
| vtex1629_get_teds_data | Returns the TEDS data for a given channel. |
| vtex1629_get_trigger_count | Queries and returns the currently configured trigger count for the EX1629. |
| vtex1629_get_trigger_delay | Queries and returns the trigger delay for the EX1629. |
| vtex1629_get_trigger_source | Queries and returns the current setting for the trigger source. |
| vtex1629_get_trigger_timer | Queries and returns the trigger system timer for the EX1629. |
| vtex1629_get_unstrained_voltage | Queries and returns the unstrained voltage currently configured for a given channel. |
| vtex1629_identify_sensor | Controls the activity of a sensor-linked LED. |

| Command | Description |
|---|---|
| vtex1629_init | Opens a session with the instrument and returns a session handle. |
| vtex1629_load_stored_config | Applies the stored configuration to the instrument. |
| vtex1629_lock | Attempts to acquire a lock on the instrument. |
| vtex1629_measure_confidence | Measures the indicated bridge parameter to indicate measurement confidence. |
| vtex1629_measure_excitation_voltage | Measures the total excitation voltage for a list of channels. |
| vtex1629_measure_lead_wire_resistance | Measures the lead wire resistance that exists in a strain gage set up |
| vtex1629_measure_unstrained_voltage | Measures the unstrained voltage for a particular list of channels. |
| vtex1629_read_fifo | This function is the means by which data is retrieved from the instrument. |
| vtex1629_read_fifoEx | Returns data and its confidence elements from the instrument. |
| vtex1629_read_teds_MLAN | Reads a different sized EEPROM from TEDS device. |
| vtex1629_read_teds_URN | Reads the unique registration number (URN) from a TEDS device. |
| vtex1629_reset | Commands the instrument to assume the default settings. |
| vtex1629_reset_fifo | Clears all the currently stored data from the FIFO. |
| vtex1629_reset_tare | Resets the tare values for all channels. |
| vtex1629_reset_trigger_arm | Resets the trigger system configuration settings to their default values. |
| vtex1629_revision_query | Outputs the driver revision and the instrument's firmware revision. |
| vtex1629_self_cal_clear | Clears the current self-calibration image. |
| vtex1629_self_cal_clear_stored | Erases the self-calibration file that is stored in nonvolatile memory. |
| vtex1629_self_cal_get_status | Returns the status of the self-calibration process. |
| vtex1629_self_cal_init | Initializes the self-calibration routine on the EX1629. |
| vtex1629_self_cal_is_running | This functions queries the existence of a previously saved self-calibration file within non-volatile memory. |
| vtex1629_self_cal_is_stored | This functions queries the existence of a previously saved self-calibration file within nonvolatile memory. |
| vtex1629_self_cal_load | Takes a currently stored self-calibration file and loads it as the current self-calibration file to be used in data acquisition. |
| vtex1629_self_cal_store | Takes the current self-calibration image and stores it to nonvolatile memory. |
| vtex1629_self_test | Causes the instrument to perform a self test. |
| vtex1629_self_test_get_status | Obtains the status of the self-test. |
| vtex1629_self_test_init | Initiates a self-test. |
| vtex1629_send_dio_pulse | Sends a pulse out on the selected DIO channels. |
| vtex1629_send_lxibus_pulse | Sends a pulse out on the desired LXI Trigger Bus channels. |
| vtex1629_set_arm_count | Sets the arm count for the EX1629. |
| vtex1629_set_arm_delay | Sets the arm delay for the EX1629. |
| vtex1629_set_arm_source | Sets the arm source on the EX1629. |
| vtex1629_set_bridge_limit | Sets the minimum and maximum bridge limit values for an array of channels |
| vtex1629_set_bridge_limit_enabled | Sets the enabled status of the bridge limit function. |
| vtex1629_set_cal_out | Sets the calibration input source to a specified voltage. |
| vtex1629_set_cal_source | Sets the current setting for the calibration input source. |
| vtex1629_set_completion_resistor | Sets the mode of the completion resistor for a set of channels. |
| vtex1629_set_conf_scanlist | Defines the list of confidence data items to be returned with the measurements. |
| vtex1629_set_confidence_limit | Sets the minimum and maximum values for confidence data limit checking |
| vtex1629_set_confidence_reporting_mode | Sets the reporting mode for confidence limit checking. |
| vtex1629_set_dio_bank0_direction | Sets the direction of bank zero of the digital I/O as input or output. |
| vtex1629_set_dio_bank0_pullup | Sets the pull-up mode for bank zero of the Digital I/O to passive or active. |
| vtex1629_set_dio_bank1_direction | Sets the direction of bank one of the digital I/O as input or output. |
| vtex1629_set_dio_bank1_pullup | Sets the pull-up mode for bank one of the Digital I/O to passive or active. |
| vtex1629_set_dio_config_events | Sets the conditions under which DIO event transitions will occur. |
| vtex1629_set_dio_output | Sets the programmed output state for both digital I/O banks. |
| vtex1629_set_EU_conversion | Set the EU conversion type for a given list of channels. |
| vtex1629_set_euconv_dynamic_excitation_enabled | Sets the dynamic excitation EU conversion state. |

| Command | Description |
|---|---|
| vtex1629_set_euconv_excitation | Manually sets the excitation voltage to be used in EU conversions for a particular list of channels. |
| vtex1629_set_excitation | Sets the programmed excitation voltages for a given list of channels. |
| vtex1629_set_excitation_enabled | Enables or disables the excitation voltages for a list of channels. |
| vtex1629_set_gain | Sets the signal conditioning gain for a given list of channels. |
| vtex1629_set_gauge_factor | Sets the gage factor for a list of channels. |
| vtex1629_set_half_bridge_lead_wire_desensitization | Sets the lead wire desensitization factor a given list of channels. |
| vtex1629_set_IIR_filter_configuration | Configures the IIR filters for a given list of channels. |
| vtex1629_set_input_multiplexer | Sets the input multiplexer source. |
| vtex1629_set_lead_wire_resistance | Sets the resistance of the lead wire. |
| vtex1629_set_linearscaling_configuration | Sets the coefficients for linear scaling for multiple channels |
| vtex1629_set_lxibus_configuration | Configures several characteristics of a specific LXI Trigger Bus channel. |
| vtex1629_set_lxibus_output | Configures the output state of each of the LXI Trigger Bus channels. |
| vtex1629_set_pattern_arm_configuration | Configures the EX1629's pattern arm mode of operation. |
| vtex1629_set_pattern_trig_configuration | Configures the EX1629's pattern trigger mode of operation. |
| vtex1629_set_poisson_ratio | Sets the Poisson ratio for a list of channels |
| vtex1629_set_sample_clock_source | Sets the sample clock source. |
| vtex1629_set_sample_count | Sets both the pre-trigger and the post-trigger sample count for the EX1629. |
| vtex1629_set_sample_frequency | Sets the sampling frequency of all channels of the EX1629. |
| vtex1629_set_scanlist | Defines a list of channels which will be sampled in the data acquisition process. |
| vtex1629_set_shunt_enabled | Enables or disables the shunt resistors for a particular list of channels. |
| vtex1629_set_shunt_source | Sets the shunt source for a given list of channels. |
| vtex1629_set_shunt_value | Sets the value of a shunt resistor based on a given channel and shunt source. |
| vtex1629_set_strain_units | Determines whether the EX1629 will return strain measurements in units of strain (ε) or microstrain (με) for a given list of channels. |
| vtex1629_set_synch_source | Sets the sample clock source. |
| vtex1629_set_tare | Sets the tare values for a list of channels. |
| vtex1629_set_teds_data | Writes data to a TEDS device on a particular channel. |
| vtex1629_set_trigger_count | Sets the trigger count for the EX1629. |
| vtex1629_set_trigger_delay | Sets the trigger delay for the EX1629. |
| vtex1629_set_trigger_source | Sets the trigger source on the EX1629. |
| vtex1629_set_trigger_source_timer | A convenience functions that makes calls to several other driver functions. |
| vtex1629_set_trigger_timer | Sets the trigger timer for the EX1629. |
| vtex1629_set_unstrained_voltage | Sets the unstrained voltage for a list of channels |
| vtex1629_soft_arm | Sends a software generated ARM event to the EX1629. |
| vtex1629_soft_synch | Sends a software generated synchronization event to the device. |
| vtex1629_soft_trig | Sends a software-generated TRIG event to the EX1629. |
| vtex1629_store_current_config | Stores the current configuration of the instrument in the nonvolatile storage. |
| vtex1629_trig_init | Initiates the trigger system. |
| vtex1629_unlock | Unlocks the EX1629 instrument. |
| vtex1629_write_teds_MLAN | Writes a variable sized block to a TEDS EEPROM. |
| vtex1629_zero_cal | Resets the offset values of the unit before a measurement is taken. |

## SAMPLE FUNCTION DEFINITION

### Function_Name

#### FUNCTION PROTOTYPE

This section provides the exact syntax of the function as it would be written in a program.

#### FUNCTION PARAMETERS

This section identifies the parameters that are associated with the function. A description of the parameter will be provided and, when appropriate, the range of values that the parameter will accept without creating an error. Ranges are assumed to be inclusive unless otherwise specified.

#### DATA ITEM RESET VALUE

This section provides the values the data items associated with this function assume after a reset condition. This section is only applicable to “set” functions.

#### DESCRIPTION

This section details what occurs when this function is called.

#### EXAMPLE

This section provides an example of how this function might appear in an application.

## EX1629 FUNCTION SET

# vtex1629_abort

### FUNCTION PROTOTYPE

ViStatus vtex1629_abort (ViSession **vi**);

### FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

### DATA ITEM RESET VALUE

Not applicable to this function.

### DESCRIPTION

This function aborts data acquisition. Specifically, calling this function moves the trigger system from its current state into the IDLE state.

### EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_abort(instrumentHandle);
```

# vtex1629_allow_all_channels

## FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex1629_allow_all_channels (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function allows the user to include channels that failed calibration in the scanlist for data acquisition.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_allow_all_channels(instrumentHandle);
```

# vtex1629_break_lock

## FUNCTION PROTOTYPE

ViStatus vtex1629_break_lock (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function releases a lock on the instrument, regardless of its owner. This allows for instrument recovery if the locking client (application or computer) becomes disabled, without rebooting or cycling power on the instrument.

| **NOTE** | Breaking a lock on the instrument does not automatically acquire it. Acquisition must be done with a separate vtex1629_lock function call. |
|---|---|

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_break_lock(instrumentHandle);
```

# vtex1629_check_lock

## FUNCTION PROTOTYPE

ViStatus vtex1629_check_lock (ViSession **vi**, ViPBoolean **locked**, ViPBoolean **mine**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**locked** = a Boolean return value indicating if the EX1629 is locked. A return value of "1" indicates that the EX1629 is locked.
**mine** = a Boolean return value indicating if the session that called the vtex1629_check_lock function owns the lock. A value of "1" returned indicates that the EX1629 is locked and that the current session owns that lock.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function checks the lock status of the EX1629 instrument. It reports whether it is locked and if so whether the current session owns the lock. When locked, the EX1629 will only accept function calls from the session handle that issued the lock function call. When not locked, the EX1629 will accept function calls from any client.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViBoolean locked, mine;
…
status = vtex1629_check_lock (instrumentHandle, &locked, &mine);

if( locked == VI_TRUE && mine == VI_TRUE ){
    printf("Instrument locked by this client!\n");
}
```

# vtex1629_clear_stored_config

## FUNCTION PROTOTYPE

ViStatus vtex1629_clear_stored_config (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function erases the stored configuration from non-volatile storage. This function does not modify the current configuration of the device.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_clear_stored_config (instrumentHandle);
```

# vtex1629_close

## FUNCTION PROTOTYPE

ViStatus vtex1629_close (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function closes the current instrument programming session. This command should be performed at the conclusion of the test application. If the current session locked the instrument, vtex1629_close will unlock, leaving it in the proper state for the next application.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_close (instrumentHandle);
```

# vtex1629_compare_digests

## FUNCTION PROTOTYPE

ViStatus vtex1629_compare_digests (ViInt32 **digestArraySize**, ViInt8 _VI_FAR **digestA[]**,ViInt8 _VI_FAR **digestB[]**, ViPBoolean **equal**);

## FUNCTION PARAMETERS

**digestArraySize** = defines how many bytes from the two digests will be compared. For consistency, this number should be VTEX1629_MAX_DIGEST_LENGTH bytes.
**digestA[]**= configuration digest (obtained from system)
**digestB[]**= configuration digest (obtained from system).
**equal** = a pointer to a return Boolean value indicating whether all the bytes in the configuration digests are the same. A value of VI_TRUE indicates that the digests are equal, where a value of VI_FALSE indicates otherwise.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function compares the two provided digests byte-by-byte. If VI_TRUE is returned in **equal**, all bytes in **digestA** and **digestB** are equal. A digest is a digital signature, or a fingerprint, representing the actual configuration data.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt8 active[VTEX1629_MAX_DIGEST_LENGTH], stored[VTEX1629_MAX_DIGEST_LENGTH];
ViInt32 actualDigestSize;
ViBoolean equal;
…
status = vtex1629_get_current_config_digest    // Read active configuration signature
    (instrumentHandle,
    VTEX1629_MAX_DIGEST_LENGTH,
    active,
    actualDigestSize);

if (status >= VI_SUCCESS)                      // Read stored configuration signature
{
    status = vtex1629_get_stored_config_digest (instrumentHandle,
                                                VTEX1629_MAX_DIGEST_LENGTH,
                                                stored,
                                                actualDigestSize);
}

if (status >= VI_SUCCESS)                      // Compare the two
{
    status = vtex1629_compare_digests (VTEX1629_MAX_DIGEST_LENGTH,
                                       active,
                                       stored,
                                       &equal);
}

if (status >= VI_SUCCESS)
{
    if (equal == VI_TRUE)
    {
        <the current configuration is the same as the stored one>
    } else
        <the current configuration differs from the stored one>
    }}
```

# vtex1629_dio_clear_event

## FUNCTION PROTOTYPE

ViStatus vtex1629_dio_clear_event (ViSession **vi**, ViInt32 **inputLine**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**inputLine** = the DIO Input Line whose event action entries are being cleared.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function clears the DIO event configuration for the specified **inputLine**.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 dioLine;
…
…
dioLine = 2;
status = vtex1629_dio_clear_event (instrumentHandle, dioLine);
```

# vtex1629_dio_clear_events_all

## FUNCTION PROTOTYPE

ViStatus vtex1629_dio_clear_events_all (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function clears the DIO event configuration for all events.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
…
status = vtex1629_dio_clear_events_all(instrumentHandle);
```

# vtex1629_disable_logging

## FUNCTION PROTOTYPE

ViStatus vtex1629_disable_logging (void);

## FUNCTION PARAMETERS

No parameters are defined for this function.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function stops the logging of driver calls.

## EXAMPLE

```
ViStatus status;
…
status = vtex1629_disable_logging ();
if (status < VI_SUCCESS)
{
    <inform the user the API call failed>
}
```

# vtex1629_disable_streaming_data

## FUNCTION PROTOTYPE

ViStatus vtex1629_disable_streaming_data (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function causes streaming data from the EX1629 to cease. In order to use this function, the macro INSTR_LANGUAGE_SPECIFIC must be defined in the application program.

## EXAMPLE

```
#define INSTR_LANGUAGE_SPECIFIC
#include<vtex1629.h>

ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_disable_streaming_data(instrumentHandle );
```

# vtex1629_enable_logging

## FUNCTION PROTOTYPE

ViStatus vtex1629_enable_logging (ViChar _VI_FAR **filename[]**, ViBoolean **append**);

## FUNCTION PARAMETERS

**filename** = an input string that specifies the file to which logging data will be recorded. The format may be simply a file name (“ex1629_log”) or an absolute path (“C:\vxipnp\vtex1629\ex1629_log.txt”). If a file name (relative path) is provided, the file will be created in the current working directory of the application.
**append** = a Boolean input value that specifies if log results will be appended to an existing file. A value of VI_TRUE will cause logged information to be appended to an existing file. Otherwise, any previous data will be overwritten.

## DATA ITEM RESET VALUE

Not applicable to this function.

## EXAMPLE

```
ViStatus status;
ViString fileName = “myTemporaryFile”;
ViBoolean append = VI_TRUE;
…
status = vtex1629_enable_logging (fileName, append);
if (status < VI_SUCCESS)
{
    <inform the user the API call failed>
}
```

# vtex1629_enable_streaming_data

## FUNCTION PROTOTYPE

ViStatus vtex1629_enable_streaming_data (ViSession **vi**, void ***private_data**, EX1629_STREAM_CALLBACK **callback**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**private_data** = a user-defined object. It will be passed as a parameter to the callback function.
**callback** = pointer to user-defined routine which should be in charge of handling the data.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function starts data streaming from the EX1629. In order to implement this function, the macro INSTR_LANGUAGE_SPECIFIC must be defined.

It should be noted that the data page created by this function contains an error code field which should equal zero. In the event that this error code equals 28, this is an indication that the instrument is no longer synchronized with the LXI clock and it is now utilizing the EX1629's internal oscillator as its clock source. Possible causes of this error include the accidental removal of the LXI cable or a missing clock. To clear this error, use the vtex1629_reset_trigger_arm function. The EX1629 will continue to use the internal clock after this error is cleared, so it will be necessary to correct the cause of the error by reconfiguring the trigger subsystem.

## EXAMPLE

```
#define INSTR_LANGUAGE_SPECIFIC
#include<vtex1629.h>

ViSession instrumentHandle;
ViStatus status;

typedef struct {
    FILE      *fout;
    ViInt32   sample_count;
} user_struct;

user_struct my_struct = {0};

ViInt32 stream_callback( void *priv, EX1629_rpc_datapage *data )
{
    user_struct  *priv_struct;    // pointer to user structure
    ViInt32      ds_idx;          // dataset index
    ViInt32      smp_idx;         // sample index

    priv_struct = (user_struct *)priv;

    /* Loop through all of the datasets in the datapage. */
    for( ds_idx = 0;
         ds_idx < data->dataset.dataset_len && ds_idx < MAX_NUM_DATASETS;
        ds_idx++ ) {

       /* check error code */
       if (dataset.dataset_val[ds_idx]->error_code != 0 ) {
          /* handle error in an application-appropriate manner…*/
          fprintf( stderr,
                   "Error %d in data stream!\n",
                    dataset.dataset_val[ds_idx]->error_code)
       }
```

```
       /* print the timestamp. */
       fprintf( priv_struct->fout,
                "Timestamp: %5u.%09u",
                data->dataset.dataset_val[ds_idx]->timestamp_sec,
                data->dataset.dataset_val[ds_idx]->timestamp_nsec);

       /* print the data */
       for( smp_idx = 0;
            smp_idx < data->dataset.dataset_val[ds_idx]->data.data_len;
            smp_idx++ ) {

          fprintf( priv_struct->fout,
                   "\t%i %f\n",
                    data->dataset.dataset_val[ds_idx]- >data.data_val[smp_idx]);
       }
    }
    if( data->dataset.dataset_len > 0) {
       priv_struct->sample_count++;
    }
}

status = vtex1629_enable_streaming_data(instrumentHandle,
                                        &my_struct,
                                        stream_callback );

status = vtex1629_trig_init(instrumentHandle);

… // application code
```

# vtex1629_enable_streaming_dataEx

## FUNCTION PROTOTYPE

ViStatus vtex1629_enable_streaming_dataEx (ViSession **vi**, ViInt32 **port**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**port** = indicates the TCP/IP port that will be used to stream data. Valid input values: 0 to 65535. (Note that some of these ports are reserved and it is recommended values between 1024 to 65535 be used.)

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function starts data streaming from instrument (expert mode). In order to implement this function the macro INSTR_LANGUAGE_SPECIFIC must be defined.

It should be noted that the data page created by this function contains an error code field which should equal zero. In the event that this error code equals 28, this is an indication that the instrument is no longer synchronized with the LXI clock and it is now utilizing the EX1629's internal oscillator as its clock source. Possible causes of this error include the accidental removal of the LXI cable or a missing clock. To clear this error, use the vtex1629_reset_trigger_arm function. The EX1629 will continue to use the internal clock after this error is cleared, so it will be necessary to correct the cause of the error by reconfiguring the trigger subsystem.

## EXAMPLE

For examples of using the vtex1629_enable_streaming_dataEx function, please contact your application engineer.

# vtex1629_erase_teds_data

## FUNCTION PROTOTYPE

ViStatus vtex1629_erase_teds_data (ViSession **vi**, ViInt32 **channel**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value indicating the channel for which the completion resistor is desired. Valid input values: 0 to 47.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function erases the data on the TEDS device indicated by the **channel** parameter. The only supported EEPROM is DS2430A. For other EEPROMs, the vtex1629_write_teds_MLAN and vtex1629_read_teds_MLAN functions should be used.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_erase_teds_data( instrumentHandle, 0);
```

# vtex1629_error_message

## FUNCTION PROTOTYPE

ViStatus vtex1629_error_message (ViSession **vi**, ViStatus **statusCode**, ViChar _VI_FAR **errMessage[]**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**statusCode**= an input status code corresponding to the error for which the error message is desired.
**errMessage** = a return string that contains the error message. This string should be at least 256 characters long.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function returns the error message text associated with the **statusCode** parameter.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status, code;
ViChar errorMessage[256];
…
status = vtex1629_error_message (instrumentHandle, code, errorMessage);
```

# vtex1629_error_query

## FUNCTION PROTOTYPE

ViStatus vtex1629_error_query (ViSession **vi**, ViPInt32 **error_code**, ViChar _VI_FAR **error_message[]**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**error_code** = Instrument Error Code.
**error_message[]** = Error Message.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function is intended to query system errors generated by an instrument. It returns VI_WARN_NSUP_ERROR_QUERY. It is provided as part of the *plug&play* standard. When errors occur on the EX1629, the vtex1629_error_message function should be used.

# vtex1629_findinstr

## FUNCTION PROTOTYPE

ViStatus vtex1629_findinstr (ViPString **instruments**, ViInt32 **maxinstr**, ViPInt32 **numinstr**, ViInt32 **timeout_secs**);

## FUNCTION PARAMETERS

**instruments** = a return pointer to an array of strings, each of which contains the *Plug&Play* compliant resource descriptor of an EX1629 found on the network. (Example descriptor: "TCPIP::192.168.1.1::INSTR".)
**maxinstr** = an integer input value that specifies the maximum number of instruments to return.
**numinstr** = a returned integer indicating the number of instruments found.
**timeout_secs** = an integer input value, in seconds, indicating the amount of time to search before timing out.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function scans the LAN for available EX1629 instruments and returns their resource descriptors, including IP addresses, as an array of strings, suitable for use with the function.

## EXAMPLE

```
#define MAX_INSTRUMENTS (500)
#define TIMEOUT (1000)
ViStatus status;
ViString instrumentIDs[MAX_INSTRUMENTS];
ViInt32 i, numberDiscovered;
…
…
status = vtex1629_findinstr (&instrumentIDs[0],
                             MAX_INSTRUMENTS,
                             &numberDiscovered,
                             TIMEOUT);
if (status < VI_SUCCESS)
{
    <inform the user the API call failed>
} else {
    for (i=0; i < numberDiscovered; i++)
    {
       <handle this instrument>
    }
}
```

# vtex1629_get_arm_count

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_arm_count (ViSession **vi**, ViPInt32 **armCount**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**armCount** = an integer output value that specifies the currently configured arm count for the EX1629. Valid return values: 1 to 2,147,483,647 ($2^{31}$-1).

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the arm count for the EX1629. This count represents the number of times the EX1629 will wait for ARM events to occur after the trigger state machine leaves the IDLE layer. Trigger counts should be kept in mind when considering this trigger state machine. If the state machine is configured with both arm and trigger counts set greater than one, then, after an ARM event is received, the state machine will go through all trigger counts before returning to the ARM layer to wait for the next ARM event.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 armcount;
…
status = vtex1629_get_arm_count( instrumentHandle, &armcount);
```

# vtex1629_get_arm_delay

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_arm_delay (ViSession **vi**, ViPReal64 **delay**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**delay** = a real output value, in seconds, indicating the arm delay. Valid return values: 0 s to 4294.967295 s

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the arm delay for the EX1629. This indicates the amount of time, in seconds, that the EX1629 will wait after receiving an ARM event before it transitions the trigger state machine from the ARM layer into the TRIG layer. Note that the value this function returns may not be identical to the value set by the vtex1629_set_arm_delay function, as the actual delay time will vary with the set sample frequency (i.e., there is some inherent quantization of the value).

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViReal64 armdelay;
…
status = vtex1629_get_arm_delay( instrumentHandle, &armdelay);
```

# vtex1629_get_arm_source

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_arm_source (ViSession **vi**, ViPInt32 **armSource**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**armSource** = an integer return value that indicates the current source the EX1629 monitors for ARM events. See the *Description* section below for more information. Valid return values: 0 to 17.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the current arm source setting. Possible values for the **armSource** parameter are:

| Decimal Value | Hex Value | #define | armSource Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_TRIG_SRC_IMMEDIATE | Immediate |
| 1 | 0x01 | VTEX1629_TRIG_SRC_PATTERN | Pattern |
| 2 | 0x02 | VTEX1629_TRIG_SRC_LXI0_POS | LXI 0 Positive Edge |
| 3 | 0x03 | VTEX1629_TRIG_SRC_LXI1_POS | LXI 1 Positive Edge |
| 4 | 0x04 | VTEX1629_TRIG_SRC_LXI2_POS | LXI 2 Positive Edge |
| 5 | 0x05 | VTEX1629_TRIG_SRC_LXI3_POS | LXI 3 Positive Edge |
| 6 | 0x06 | VTEX1629_TRIG_SRC_LXI4_POS | LXI 4 Positive Edge |
| 7 | 0x07 | VTEX1629_TRIG_SRC_LXI5_POS | LXI 5 Positive Edge |
| 8 | 0x08 | VTEX1629_TRIG_SRC_LXI6_POS | LXI 6 Positive Edge |
| 9 | 0x09 | VTEX1629_TRIG_SRC_LXI7_POS | LXI 7 Positive Edge |
| 10 | 0x0A | VTEX1629_TRIG_SRC_LXI0_NEG | LXI 0 Negative Edge |
| 11 | 0x0B | VTEX1629_TRIG_SRC_LXI1_NEG | LXI 1 Negative Edge |
| 12 | 0x0C | VTEX1629_TRIG_SRC_LXI2_NEG | LXI 2 Negative Edge |
| 13 | 0x0D | VTEX1629_TRIG_SRC_LXI3_NEG | LXI 3 Negative Edge |
| 14 | 0x0E | VTEX1629_TRIG_SRC_LXI4_NEG | LXI 4 Negative Edge |
| 15 | 0x0F | VTEX1629_TRIG_SRC_LXI5_NEG | LXI 5 Negative Edge |
| 16 | 0x10 | VTEX1629_TRIG_SRC_LXI6_NEG | LXI 6 Negative Edge |
| 17 | 0x11 | VTEX1629_TRIG_SRC_LXI7_NEG | LXI 7 Negative Edge |

**Immediate (0)**: an immediate ARM source. After initialization of the trigger system, the trigger state machine will bypass the ARM layer and will automatically transition into the TRIG layer.

**Pattern (1)**: this arm source allows the EX1629 to accept ARM events from multiple sources. Specifically, the EX1629 can be configured to accept ARM events from any of the LXI Trigger Bus channels, from any of the digital I/O channels, from an internal timer, or from software arm commands. The instrument can be configured to accept any combination of these events simultaneously. The specific pattern is queried with the vtex1629_get_pattern_arm_configuration call.

**LXI *n* Positive Edge (2 – 9)**: these arm sources refer to ARM events coming from the LXI Trigger Bus. More specifically, these arm sources will cause the EX1629 to arm on the positive edge of signals coming into the LXI Trigger Bus.

**LXI *n* Negative Edge (10 – 17)**: these arm sources refer to ARM events coming from the LXI Trigger Bus. More specifically, these arm sources will cause the EX1629 to arm on the negative edge of signals coming into the LXI Trigger Bus.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 armsource;
…
status = vtex1629_get_arm_source (instrumentHandle, &armsource);
```

# vtex1629_get_bridge_limit

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_bridge_limit (ViSession **vi**, ViInt32 **channel**, ViPReal64 **min**, ViPReal64 **max**);

## FUNCTION PARAMETERS

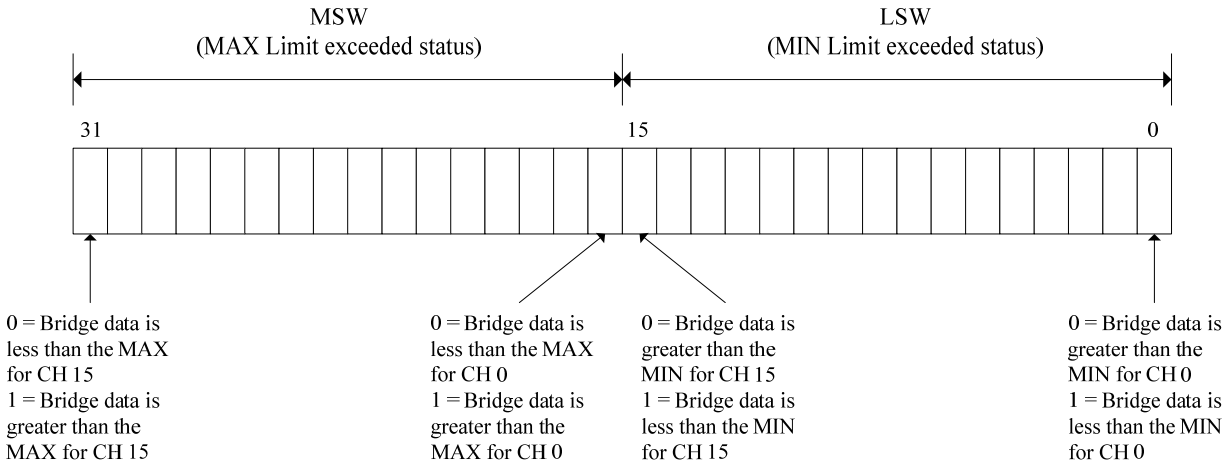
**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value indicating the channel for which the completion resistor is desired. Valid input values: 0 to 47.
**min** = the returned minimum bridge limit value.
**max** = the returned minimum bridge limit value.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the minimum and maximum bridge limit values. The limit check data is part of the data page along with the bridge data. If the bridge data exceeds the maximum or minimum limit values set for any channel, the corresponding flags are set in the limit check result field in a data page.




The “limits” field is a bit-field. This UINT32 has two bits per channel (16-channels per analog board), one to represent MAX limit exceeded and one to represent MIN limit exceeded. The MSW (upper 16-bits) represent the MAX Limit Exceeded status for each of the 16-channels, and the LSW (lower 16-bits) represent the MIN Limit Exceeded status for each of the 16-channels. Bit 0 represents the MIN Limit Exceeded status for channel 0 (channels 0, 16, 32). Bit 16 represents the MAX Limit Exceeded status for Channel 0 (channels 0, 16, 32). Bit 15 represents the MIN Limit Exceeded status for channel 15 (channels 15, 31, and 47). Bit 31 represents the MAX Limit Exceeded status for channel 15 (channels 15, 31, and 47). The rest of the channels follow the same pattern.

| **NOTE** | The channel-to-bit mapping is constant, regardless of scanlist configuration. For example, whether or not channels 0 and 1 are enabled in the scanlist, for instance, channel 2’s MIN Limit Exceeded Bit and MAX Limit Exceeded Bit are always bits 2 and 18, respectively. |
|---|---|

This mode is valid for main bridge sampling frequencies of 1 kHz or less. If the sampling frequency exceeds 1 kHz a value of 0x0 is reported. Also, the bit fields corresponding to inactive channels in the scanlist will be 0.

Limit checking is performed on the output of the EU conversion. So, if the specified EU conversion is in Strain (quarter-, half-, or full-bridge) the limit values are in strain (or microstrain). If the specified EU conversion is volts, then the limit values are in volts.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channel = 5;
ViReal64 min = 0;
ViReal64 max = 0;
…
status = vtex1629_get_bridge_limit (instrumentHandle, channel, &min, &max);
```

# vtex1629_get_bridge_limit_enabled

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_bridge_limit_enabled (ViSession **vi**, ViPBoolean **enabled**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**enabled** = a Boolean return value indicating the enabled status of the excitation source. A returned value of "1" indicates that the excitation source is enabled.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the enabled status of the bridge limit function.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViBoolean armsource;
…
status = vtex1629_get_bridge_limit_enabled (instrumentHandle, &enabled);
```

# vtex1629_get_cal_coefficients

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_cal_coefficients (ViSession **vi**, ViInt32 **caltype**, ViInt32 **coefficientSelector**, ViInt32 _VI_FAR **channelList[]**,ViInt32 **numberOfChannels**, ViReal64 _VI_FAR **coefficientOutputArray[]**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**calType** = file type for which to query the coefficients. Valid values are 1 through 3. See *Description* below for more details.
**coefficientSelector** = an integer input value indicating the desired coefficient to query. Valid values are from VTEX1629_CALSRC_0 (0x00) through VTEX1629_SUPPLY_EXCITE_OUT_POS_OFFSET (0x3F).
**channelList[]** = an integer array containing a list of channel numbers for which to query the coefficient. Acceptable values for this array are VTEX1629_MIN_CHANNEL(0) to VTEX1629_MAX_CHANNEL(47), inclusive.

| | |
|---|---|
| **NOTE** | Since the calibration sources VTEX1629_CALSRC_0 (0x00) through VTEX1629_CALSRC_ALL (0x13) are global across all channels, if the value of the **coefficientSelector** parameter is within the aforementioned range, the API will not query the channels listed in this **channelList** array. |

**numberOfChannels** = a return integer value indicating the number of channels currently included in the scan list. Valid return values: 1 to 48.
**coefficientOutputArray[]** = Return array of the queried coefficients. See the *Description* section below for valid return values.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the value of a selected calibration coefficient for one or more channels.

The **calType** parameter has the following valid input values:

| Decimal Value | Hex Value | #define Symbol | Description |
|---|---|---|---|
| 1 | 0x01 | VTEX1629_CAL_DATA_SELF | Returns self-calibration data |
| 2 | 0x02 | VTEX1629_CAL_DATA_FACTORY | Returns factory calibration data |
| 3 | 0x03 | VTEX1629_CAL_DATA_COMBINED_XML | Returns factory and self-calibration data in XML format. |

If one of the above values is not defined for the **calType** parameter, an error of `VI_ERROR_PARAMETER2` will be returned.

The **coefficientSelector** parameter has the following valid input values:

| Decimal Value | Hex Value | #define Symbol |
|---|---|---|
| 19 | 0x13 | VTEX1629_CALSRC_ALL |
| 20 | 0x14 | VTEX1629_MAIN_ADC_X1_GAIN |
| 21 | 0x15 | VTEX1629_MAIN_ADC_X1_OFFSET |
| 22 | 0x16 | VTEX1629_MAIN_ADC_X10_GAIN |
| 23 | 0x17 | VTEX1629_MAIN_ADC_X10_OFFSET |
| 24 | 0x18 | VTEX1629_MAIN_ADC_X100_GAIN |
| 25 | 0x19 | VTEX1629_MAIN_ADC_X100_OFFSET |
| 26 | 0x1A | VTEX1629_WAGNER_VOLT |
| 27 | 0x1B | VTEX1629_COMP_RESISTOR_350 |
| 28 | 0x1C | VTEX1629_COMP_RESISTOR_120 |

| Decimal Value | Hex Value | #define Symbol |
|---|---|---|
| 29 | 0x1D | VTEX1629_COMP_RESISTOR_USER |
| 30 | 0x1E | VTEX1629_COMP_RESISTOR_SHORT |
| 31 | 0x1F | VTEX1629_SHUNT_RESISTOR_INTERNAL |
| 32 | 0x20 | VTEX1629_SHUNT_RESISTOR_FRONT_PANEL |
| 33 | 0x21 | VTEX1629_SHUNT_RESISTOR_TEDS |
| 34 | 0x22 | VTEX1629_CONF_AGND_GAIN |
| 35 | 0x23 | VTEX1629_CONF_AGND_OFFSET |
| 36 | 0x24 | VTEX1629_CONF_CAL_POS_GAIN |
| 37 | 0x25 | VTEX1629_CONF_CAL_POS_OFFSET |
| 38 | 0x26 | VTEX1629_CONF_BUFF_IN_POS_GAIN |
| 39 | 0x27 | VTEX1629_CONF_BUFF_IN_POS_OFFSET |
| 40 | 0x28 | VTEX1629_CONF_BUFF_V_CMD_GAIN |
| 41 | 0x29 | VTEX1629_CONF_BUFF_V_CMD_OFFSET |
| 42 | 0x2A | VTEX1629_CONF_BUFF_IN_NEG_GAIN |
| 43 | 0x2B | VTEX1629_CONF_BUFF_IN_NEG_OFFSET |
| 44 | 0x2C | VTEX1629_CONF_EXCITE_OUT_NEG_GAIN |
| 45 | 0x2D | VTEX1629_CONF_EXCITE_OUT_NEG_OFFSET |
| 46 | 0x2E | VTEX1629_CONF_EXCITE_OUT_POS_GAIN |
| 47 | 0x2F | VTEX1629_CONF_EXCITE_OUT_POS_OFFSET |
| 48 | 0x30 | VTEX1629_CONF_V_SENSE_NEG_GAIN |
| 49 | 0x31 | VTEX1629_CONF_V_SENSE_NEG_OFFSET |
| 50 | 0x32 | VTEX1629_CONF_V_SENSE_POS_GAIN |
| 51 | 0x33 | VTEX1629_CONF_V_SENSE_POS_OFFSET |
| 52 | 0x34 | VTEX1629_CONF_EXCITE_OUT_CURR_POS_GAIN |
| 53 | 0x35 | VTEX1629_CONF_EXCITE_OUT_CURR_POS_OFFSET |
| 54 | 0x36 | VTEX1629_CONF_EXCITE_OUT_CURR_POS_COMMON_MODE_RESISTANCE |
| 55 | 0x37 | VTEX1629_CONF_EXCITE_OUT_CURR_NEG_GAIN |
| 56 | 0x38 | VTEX1629_CONF_EXCITE_OUT_CURR_NEG_OFFSET |
| 57 | 0x39 | VTEX1629_CONF_EXCITE_OUT_CURR_NEG_COMMON_MODE_RESISTANCE |
| 58 | 0x3A | VTEX1629_SENSE_RESISTOR_NEG |
| 59 | 0x3B | VTEX1629_SENSE_RESISTOR_POS |
| 60 | 0x3C | VTEX1629_SUPPLY_EXCITE_OUT_NEG_GAIN |
| 61 | 0x3D | VTEX1629_SUPPLY_EXCITE_OUT_NEG_OFFSET |
| 62 | 0x3E | VTEX1629_SUPPLY_EXCITE_OUT_POS_GAIN |
| 63 | 0x3F | VTEX1629_SUPPLY_EXCITE_OUT_POS_OFFSET |

If the value of the **coefficientSelector** parameter is between VTEX1629_CALSRC_0 (0x00) and VTEX1629_CALSRC_N_14_0 (0x12), the resultant coefficient will be placed in element 0 of this array (i.e. **coefficientOutputArray[0]**). See note with the **channelList** parameter for more information.

If the value of coefficientSelector is VTEX1629_CALSRC_ALL (0x13), then the resultant data will be stored in the first 0x13 elements of this array. Each element can subsequently be queried by simply using its “constant” value as the index. For example, the value for VTEX1629_CALSRC_N_0_7 is located at coefficientOutputArray[VTEX1629_CALSRC_N_0_7], the value for VTEX1629_CALSRC_N_14_0 is located at coefficientOutputArray[VTEX1629_CALSRC_N_14_0], etc.

If the value of the **coefficientSelector** parameter is between VTEX1629_MAIN_ADC_X1_GAIN (0x14) and VTEX1629_SUPPLY_EXCITE_OUT_POS_OFFSET (0x3F), each element of this array corresponds to the equivalent index in the **channelList** array. For example, the coefficient value at element i in this array corresponds to the channel designated in element i of the **channelList** array.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status = VI_SUCCESS;
ViInt32 CalType = -1;
ViInt32 coefficientSelector = -1;
ViInt32 channelList[MAX_CHANNELS];
ViInt32 numberOfChannels = 0;
ViReal64 coefficientOutputArray[MAX_CHANNELS];
int i = 0;
…
memset(channelList, 0x00, sizeof(channelList));
memset(coefficientOutputArray, 0x00, sizeof(coefficientOutputArray));

CalType = VTEX1629_CAL_DATA_SELF;
coefficientSelector = VTEX1629_MAIN_ADC_X1_GAIN;

numberOfChannels = MAX_NUMBER_OF_CHANNELS;
for(i = 0; i < numberOfChannels; i++) {
    channelList[i] = i;
}

status = vtex1629_get_cal_coefficients(instrumentHandle,
                                       CalType,
                                       coefficientSelector,
                                       channelList,
                                       numberOfChannels,
                                       coefficientOutputArray);
```

# vtex1629_get_cal_file

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_cal_file (ViSession **vi**, ViInt32 **fileType**, ViInt32 **bufferSize**, ViString **xmlBuffer**, ViPInt32 **actualSize**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**fileType** = deinfes the file type that will be acquired. Valid input values: 0 through 3. See the *Description* section below for more information.
**bufferSize** = defines the size of the **xmlBuffer** allocated array.
**xmlBuffer** = defines the location where the requested calibration file will be loaded.
**actualSize** = returns the number of bytes actually read by the API. The API reads up to **bufferSize** characters from the EX1629 and places them in the buffer.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function reads up to **bufferSize** characters from the EX1629 and places them in the XML buffer. It returns, in the **actualSize** parameter**,** the number of bytes read.

The **fileType** parameter has the following valid input values:

| Decimal Value | Hex Value | #define Symbol | Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_CAL_DATA_COMBINED | Both self and factory calibration data |
| 1 | 0x01 | VTEX1629_CAL_DATA_SELF | Self-calibration data |
| 2 | 0x02 | VTEX1629_CAL_DATA_FACTORY | Factory calibration data |
| 3 | 0x03 | VTEX1629_CAL_DATA_COMBINED_XML | Both self and factory calibration data in XML format. |

The **bufferSize** parameter represents the size of the **xmlBuffer** parameter. Ideally, the user should first call the vtex1629_get_cal_file_size function to determine size of the desired calibration file and, hence, the appropriate value for this parameter. If this parameter is not set to one of the valid values listed above, an error of `VI_ERROR_PARAMETER2` will be returned.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status = VI_SUCCESS;
ViChar errMessage[256];

ViInt32 fileType = 0;
ViString buffer = 0;
ViInt32 bufferSize = 0;
ViInt32 actualSize = 0;
…
fileType = VTEX1629_CAL_DATA_SELF;

status = vtex1629_get_cal_file_size(instrumentHandle, fileType, &bufferSize);
if(status < VI_SUCCESS) {
    Log("Error occurred when getting cal file size");
    vtex1629_error_message (vi, status, errMessage);
    Log(errMessage);
}

if(status > VI_SUCCESS) {
    buffer = malloc( bufferSize * sizeof(ViString) );

    status = vtex1629_get_cal_file(instrumentHandle,
```

```
                                    fileType,
                                    bufferSize,
                                    buffer,
                                    &actualSize);

    if(status < VI_SUCCESS) {
        <inform the user the API call failed>
    }
}
```

# vtex1629_get_cal_file_size

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_cal_file_size (ViSession **vi**, ViInt32 **fileType**, ViPInt32 **fileSize**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**fileType** = deinfes the file type that will be acquired. Valid input values: 0 through 3. See the *Description* section below for more information.
**fileSize** = returned file size for the specified calibration file type.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function returns the total buffer size required to read the cal data, including terminating nulls, etc. The client application should use this size to allocate a sufficiently large buffer.

The **fileType** parameter has the following valid input values:

| Decimal Value | Hex Value | #define Symbol | Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_CAL_DATA_COMBINED | Both self and factory calibration data |
| 1 | 0x01 | VTEX1629_CAL_DATA_SELF | Self-calibration data |
| 2 | 0x02 | VTEX1629_CAL_DATA_FACTORY | Factory calibration data |
| 3 | 0x03 | VTEX1629_CAL_DATA_COMBINED_XML | Both self and factory calibration data in XML format. |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status = VI_SUCCESS;
ViInt32 bufferSize = 0;
fileType = VTEX1629_CAL_DATA_COMBINED;
…
status = vtex1629_get_cal_file_size(instrumentHandle, fileType, &bufferSize);
```

# vtex1629_get_cal_source

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_cal_source (ViSession **vi**, ViPInt32 **calSource**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**calSource** = an integer output value that indicates the calibration source voltage. See the *Description* section below for valid return values. Valid return values: 0 to 18.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the current voltage setting of the calibration source. Valid return values are:

| Decimal Value | Hex Value | #define Symbol | Nominal Voltage (V) |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_CALSRC_0 | 0 |
| 1 | 0x01 | VTEX1629_CALSRC_P_0_07 | +0.07 |
| 2 | 0x02 | VTEX1629_CALSRC_N_0_07 | -0.07 |
| 3 | 0x03 | VTEX1629_CALSRC_P_0_11 | +0.11 |
| 4 | 0x04 | VTEX1629_CALSRC_N_0_11 | -0.11 |
| 5 | 0x05 | VTEX1629_CALSRC_P_0_14 | +0.14 |
| 6 | 0x06 | VTEX1629_CALSRC_N_0_14 | -0.14 |
| 7 | 0x07 | VTEX1629_CALSRC_P_0_7 | +0.7 |
| 8 | 0x08 | VTEX1629_CALSRC_N_0_7 | -0.7 |
| 9 | 0x09 | VTEX1629_CALSRC_P_1_1 | +1.1 |
| 10 | 0x0A | VTEX1629_CALSRC_N_1_1 | -1.1 |
| 11 | 0x0B | VTEX1629_CALSRC_P_1_4 | +1.4 |
| 12 | 0x0C | VTEX1629_CALSRC_N_1_4 | -1.4 |
| 13 | 0x0D | VTEX1629_CALSRC_P_7_0 | +7.0 |
| 14 | 0x0E | VTEX1629_CALSRC_N_7_0 | -7.0 |
| 15 | 0x0F | VTEX1629_CALSRC_P_11_0 | +11.0 |
| 16 | 0x10 | VTEX1629_CALSRC_N_11_0 | -11.0 |
| 17 | 0x11 | VTEX1629_CALSRC_P_14_0 | +14.0 |
| 18 | 0x12 | VTEX1629_CALSRC_N_14_0 | -14.0 |

**NOTE** This function is intended for factory use only.

# vtex1629_get_completion_resistor

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_completion_resistor (ViSession **vi**, ViInt32 **channel**, ViPInt32 **completionResistorMode**, ViPReal64 **calibratedValue**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value indicating the channel for which the completion resistor is desired. Valid input values: 0 to 47.
**completionResistorMode** = an integer return value indicating the completion resistor mode. Valid return values: 0, 3, 4, 120, or 350.
**calibratedValue** = a real return value that indicates the calibrated value of the currently configured completion resistor. This value is retrieved from the non-volatile, factory calibration file that is stored within the device.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the mode and value of the completion resistor for a specific channel. The valid return values are as follows:

| Decimal Value | Hex Value | #define Symbol | Resistor Mode | calibratedValue |
|---|---|---|---|---|
| 0 | 0x00 | VTEX1629_COMPRES_FULL | Full | 0.0 (N/A) |
| 3 | 0x03 | VTEX1629_COMPRES_USER | User-Defined | Actual value installed, 0.0 (N/A) otherwise |
| 4 | 0x04 | VTEX1629_COMPRES_OFF | OFF | 0.0 (N/A) |
| 120 | 0x78 | VTEX1629_COMPRES_120 | 120 Ω | Actual value |
| 350 | 0x15E | VTEX1629_COMPRES_350 | 350 Ω | Actual value |

Referring to the “Full” completion resistor is a bit of a misnomer – it really represents a short in the leg of the bridge circuit that contains the completion resistor. It is used in Full and Half-Bridge mode.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 compmode;
ViReal64 resvalue;
…
status = vtex1629_get_completion_resistor(instrumentHandle, 0, &compmode, &resvalue);
```

# vtex1629_get_conf_scanlist

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_conf_scanlist (ViSession **vi**, ViInt32 _VI_FAR **confElements[]**, ViPInt32 **numConfElements**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**confElements** = a return array of integers indicating the enabled data elements in the confidence scan list. Valid return values: 0 to 12. This array must be at least 12 elements long.
**numConfElements** = an integer return value indicating the size of the **confElements** list. Valid return values: 0 to 12.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns a list of confidence data elements that will be measured and returned along with the main bridge data. The confidence data elements are the following:

| Decimal Value | Hex Value | #define Symbol | confElements Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_CONFSRC_BRIDGE_POS | Bridge (+) |
| 1 | 0x01 | VTEX1629_CONFSRC_BRIDGE_COMM | Bridge (common mode) |
| 2 | 0x02 | VTEX1629_CONFSRC_BRIDGE_NEG | Bridge (-) |
| 3 | 0x03 | VTEX1629_CONFSRC_EXCITE_POS | Excite (+) |
| 4 | 0x04 | VTEX1629_CONFSRC_EXCITE_NEG | Excite (-) |
| 5 | 0x05 | VTEX1629_CONFSRC_EXCITE_NEG_SENSE | Excite Sense (-) |
| 6 | 0x06 | VTEX1629_CONFSRC_EXCITE_POS_SENSE | Excite Sense (+) |
| 7 | 0x07 | VTEX1629_CONFSRC_EXCITE_POS_CURR | Excite Current (+) |
| 8 | 0x08 | VTEX1629_CONFSRC_EXCITE_NEG_CURR | Excite Current (-) |
| 9 | 0x09 | VTEX1629_CONFSRC_POS_CAL | Calibration Bus (+) |
| 10 | 0x0A | VTEX1629_CONFSRC_NEG_CAL | Calibration Bus (-) |
| 11 | 0x0B | VTEX1629_CONFSRC_GND | Ground |
| 12 | 0x0C | VTEX1629_CONFSRC_EXCITEOUT_BUFF | Excite Out (Buffered) |

**NOTES**
**1)** Confidence elements 9 through 11 are for system diagnostic use only and should not be employed during normal operation.
**2)** Confidence element 12 can only be used on EX1629 with firmware version 1.0 or later.

A value of 0 for the **numConfElements** parameter indicates that the confidence scan list is empty. In this case, any values in the **confElements** parameter are invalid.

**NOTE** The key restriction that MUST be enforced is that any confidence scan list that includes +V_SENSE and -V_SENSE must also include +EXCITEOUT and -EXCITELOW as well.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 conf[12];
ViInt32 numelements;
…
status = vtex1629_get_conf_scanlist(instrumentHandle, conf, &numelements);
```

# vtex1629_get_confidence_limit

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_confidence_limit (ViSession **vi**, ViInt32 **channel**, ViInt32 **confSrcEnum,** ViPReal64 **min**, ViPReal64 **max**);

## FUNCTION PARAMETERS

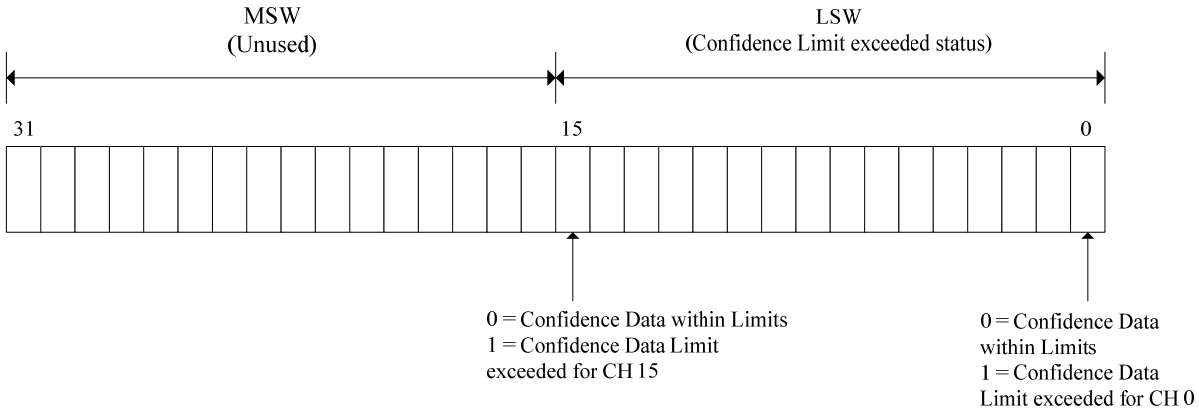
**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value indicating the channel for which the completion resistor is desired. Valid input values: 0 to 47.
**min** = the returned minimum confidence limit value.
**max** = the returned maximum confidence limit value.
**confSrcEnum** = defines the confidence source value that will be queried. Valid input values: 0 to 12.

## DATA ITEM RESET VALUE

Not applicable to this function.

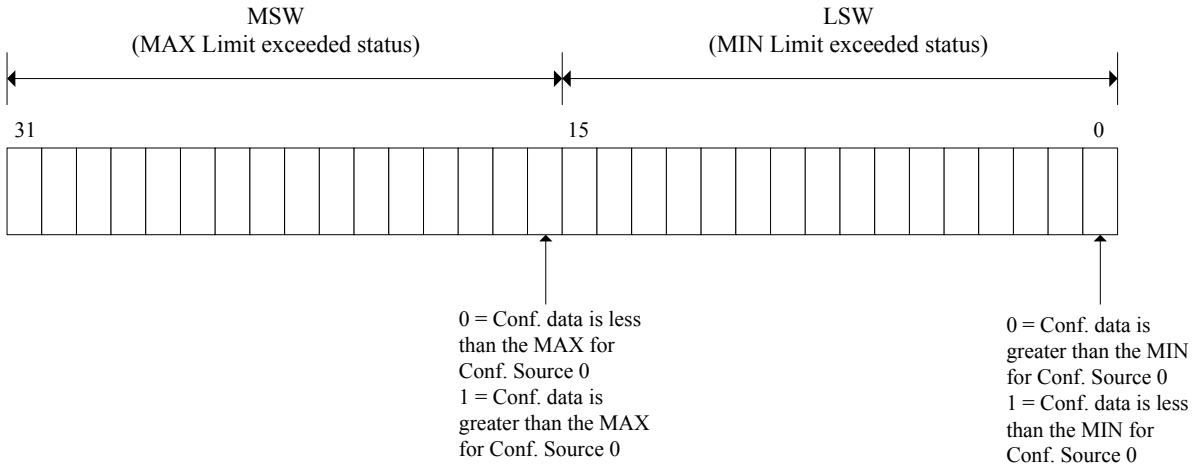
## DESCRIPTION

This function queries and retrieves the minimum and maximum values for confidence data limit checking. Confidence limit checking mode is only valid for main bridge sampling frequencies less than 1 kHz. Returned values in the datapage correspond to the confidence channels for channels in the active scanlist. There exists a confidence limit check result summary field (shown in the diagram below) which indicates if any of the limits on all active confidence sources for a particular channel were exceeded or not. This is a 16-bit field, stored in the least-significant 16 bits of a UINT32 variable.




**Confidence Limit Check Result summary (Available per Analog Board)**

A detailed confidence limit check result (shown in the diagram below) is also available which returns two bits per channel per confidence source – that is, MAX Limit Exceeded and MIN Limit Exceeded, per channel, per confidence source. There is one UINT32 entry per bridge channel in the bridge scanlist. This UINT32 has two bits per confidence source (CONF_NUM_SRC sources per bridge channel), one to represent MAX limit exceeded and one to represent MIN limit exceeded. The MSW (lower CONF_NUM_SRC of the upper 16-bits) represent the MAX Limit Exceeded status for each of the CONF_NUM_SRC confidence sources, and the LSW (lower CONF_NUM_SRC bits of the lower 16-bits) represent the MIN Limit Exceeded status for each of the CONF_NUM_SRC confidence sources. Bit 0 represents the MIN Limit Exceeded status for source 0. Bit 16 represents the MAX Limit Exceeded status for source 0. Bit (CONF_NUM_SRC-1) represents the MIN Limit Exceeded status for source (CONF_NUM_SRC-1). Bit (16+CONF_NUM_SRC-1) represents the MAX Limit Exceeded status for source CONF_NUM_SRC. The rest of the sources follow the same pattern.




**Confidence Limit Check Detailed Result (Available per Bridge channel)**

| **NOTE** | The source-to-bit mapping is constant, regardless of confidence scanlist configuration. For example, whether or not sources 0 and 1 are enabled in the confidence scanlist, for instance, source 2's MIN Limit Exceeded Bit and MAX Limit Exceeded Bit are always bits 2 and 18, respectively. |
|---|---|

The confidence source mapping follows the same ordering as the source # define in vtex1629.h i.e. if sources 3, 8, and 10 are selected then they are reported in that order. Confidence sources that are not part of the confidence scanlist are not reported and will have their bit-fields set to 0.

Confidence values are reported at a maximum frequency of 500 Hz. This mode is supported up to 1 kHz sampling rate. Hence, at 1 kHz, every other packet will contain confidence information. The datapage size is 248 words when it has full confidence information i.e. confidence data and full limit check values, and is 24 words when it has no confidence information. Hence, the total data rate = ((248+24)/2)*4*8*1000 samples/second= 4.352 Mb/s.

**EXAMPLE**

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channel = 10;
ViReal64 min = 0;
ViReal64 max = 0;
ViInt32 confSrcEnum = 0;
…
status = vtex1629_get_confidence_limit (instrumentHandle,
                                        channel,
                                        confSrcEnum,
                                        &min,
                                        &max);
```

# vtex1629_get_confidence_reporting_mode

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_confidence_reporting_mode (ViSession **vi**, ViPInt32 **mode**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**mode** = the returned mode value. Valid input values: 0 through 2.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and retrieves the mode used for confidence limit checking. Valid return values for the **mode** parameter are as follows:

| Decimal Value | Hex Value | #define Symbol | mode Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_CONF_LIMIT_DISABLE_REPORT | Reporting disabled |
| 1 | 0x01 | VTEX1629_CONF_LIMIT_SUMMARY_REPORT_ONLY | Summary report mode selected |
| 2 | 0x02 | VTEX1629_CONF_LIMIT_DETAILED_REPORT | Detailed report mode selected |

If set to VTEX1629_CONF_LIMIT_DISABLE_REPORT, the EX1629 will not collect confidence limit checking data. If set to VTEX1629_CONF_LIMIT_SUMMARY_REPORT_ONLY, an array will be created which indicates the channels that exceeded their limits. VTEX1629_CONF_LIMIT_DETAILED_REPORT, by contrast, provides an array that indicates if the minimum or maximum limit of a channel has been exceeded.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 reportingMode;
…
status = vtex1629_get_confidence_reporting_mode(instrumentHandle, &reportingMode);
```

# vtex1629_get_current_config_digest

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_current_config_digest (ViSession **vi**, ViInt32 **digestArraySize**, ViInt8 _VI_FAR **digest[]**, ViPInt32 **digestActualSize**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**digestArraySize** = contains the size of the allocated digest array. For consistency, the client application should allocate VTEX1629_MAX_DIGEST_LENGTH bytes.
**digest[]** = the current configuration's digest.
**digestActualSize** = the actual configuration digest size.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function retrieves the digest for the current instrument configuration. The digest is a digital signature representing the configuration data.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 digestActualSize;
ViInt8 digest[VTEX1629_MAX_DIGEST_LENGTH];
…
status = vtex1629_get_current_config_digest(instrumentHandle,
                                            VTEX1629_MAX_DIGEST_LENGTH,
                                            &digestActualSize);
```

# vtex1629_get_dio_bank0_direction

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_dio_bank0_direction (ViSession **vi**, ViPInt32 **direction**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**direction** = an integer return value indicating the direction of bank zero of the digital I/O. Valid return values: 0 or 1.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function indicates whether bank zero of the digital I/O is configured as input or output. The **direction** parameter is defined as follows:

0 = input 1 = output

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 dio0dir;
…
status = vtex1629_get_dio_bank0_direction(instrumentHandle, &dio0dir);
```

# vtex1629_get_dio_bank0_pullup

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_dio_bank0_pullup (ViSession **vi**, ViPInt32 **pullup**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**pullup** = an integer return value indicating the pull-up mode for bank zero of the digital I/O. Valid return values: 0 or 1.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the pull-up mode for bank zero of the digital I/O. The **pullup** parameter is defined as follows:

0 = passive pull-up mode 1 = active pull-up mode

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 dio0pullup;
…
status = vtex1629_get_dio_bank0_pullup(instrumentHandle, &dio0pullup);
```

# vtex1629_get_dio_bank1_direction

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_dio_bank1_direction (ViSession **vi**, ViPInt32 **direction**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**direction** = an integer return value indicating the direction of bank one of the digital I/O. Valid return values: 0 or 1.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function indicates whether bank one of the digital I/O is configured as input or output. The **direction** parameter is defined as follows:

0 = input 1 = output

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 dio1dir;
…
status = vtex1629_get_dio_bank1_direction(instrumentHandle, &dio1dir);
```

# vtex1629_get_dio_bank1_pullup

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_dio_bank1_pullup (ViSession **vi**, ViPInt32 **pullup**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**pullup** = an integer return value indicating the pull-up mode for bank one of the digital I/O. Valid return values: 0 or 1.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the pull-up mode for bank one of the digital I/O. The **pullup** parameter is defined as follows:

0 = passive pull-up mode 1 = active pull-up mode

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 dio1pullup;
…
status = vtex1629_get_dio_bank1_pullup(instrumentHandle, &dio1pullup);
```

# vtex1629_get_dio_config_events

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_dio_config_events (ViSession **vi**, ViInt32 **inputLine**, ViInt32 **inputTrigType**, ViInt32 **numActions**, ViReal64 _VI_FAR **outputLineArr[]**, ViReal64 _VI_FAR **outputActionTypeArr[]**, ViPInt32 **numActionsActual**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**inputLine** = defines the DIO input line whose configuration is being queried. Valid input values: 0 through 7.
**inputTrigType** = defines the input trigger type. Valid input values: 0 (high-to-low transition) or 1 (low-to-high transition).
**numActions** = defines the size of the **outputLineArr[]** and **outputActionTypeArr[]** arrays. Valid input values: 0 through 8.
**outputLineArr[]** = an integer array containing a list of digital output lines that are affected by the **inputLine** and **inputTrigType** combination. Valid return values: 0 through 7.
**outputActionTypeArr[]** = an integer array containing a list of the output action that will occur based on the **inputLine** and **inputTrigType** parameters. Valid return values: 0 through 3.
**numActionsActual** = the actual number of actions available. Valid return values: 1 through 8.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the current setting for DIO event transitions.

The **numActions** parameter defines the size of both the **outputLineArr[]** and **outputActionTypeArr[]** arrays. Although any value 0 through 8 is acceptable, to avoid possible errors, it is recommended that this parameter be set to 8.

The **outputActionTypeArr[]** parameter is an array which contains a list of output actions that will occur based on events that occur on the specified **inputLine**. Note that each element of this array corresponds to the equivalent index in the **outputLineArr[]** parameter. For example, the action type at element *i* in this array corresponds to (i.e. will occur on) the line designated in element *i* of the **outputLineArr[]** array.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status = VI_SUCCESS;
ViChar errMessage[256];
ViChar tempString[256] = "";

ViInt32 inputLine = 1;
ViInt32 inputTrigType = 2;
ViInt32 numActions = MAX_DIO_CHANNELS;
ViInt32 outputLineArr[MAX_DIO_CHANNELS];
ViInt32 outputActionTypeArr[MAX_DIO_CHANNELS];
...
memset(outputLineArr, 0x00, sizeof(outputLineArr));
memset(outputActionTypeArr, 0x00, sizeof(outputActionTypeArr));

status = vtex1629_get_dio_config_events(instrumentHandle,
                                        inputLine,
                                        inputTrigType,
                                        numActions,
                                        outputLineArr,
                                        outputActionTypeArr);
```

# vtex1629_get_dio_input

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_dio_input (ViSession **vi**, ViPInt32 **dioIn**);

## FUNCTION PARAMETERS

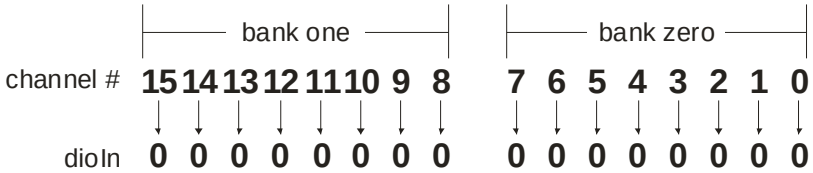
**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**dioIn** = an integer return value indicating the input state of the digital I/O. See the *Description* below for more information concerning this parameter. Valid return values: 0 to 65535.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the current input state of both banks of the digital I/O. The **dioIn** parameter is a decimal value that must be converted to a 16-bit binary value. Once done, the eight most significant bits correspond to the eight channels of bank one (channels 8-15). The eight least significant bits correspond to the eight channels of bank zero (channels 0-7). This is illustrated below.




The upper 16-bits will always be zero.

For example, a user queries the output state and the following was returned:

**dioIn** = 49164 → 0x0000C00C → 11000000 00001100

This indicates that channels 2 and 3 of digital I/O bank zero and channels 14 and 15 of digital I/O bank one are high, while the remaining channels are low.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 dio_in;
…
status = vtex1629_get_dio_input(instrumentHandle, &dio_in);
```

# vtex1629_get_dio_output

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_dio_output (ViSession **vi**, ViPInt32 **dioOut**);

## FUNCTION PARAMETERS

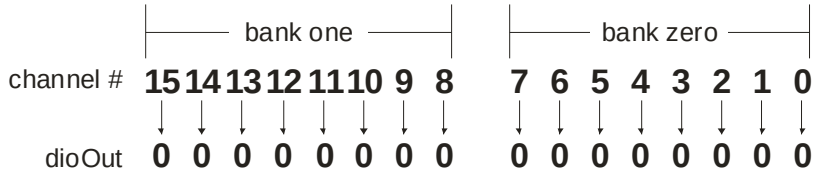
**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**dioOut** = an integer return value indicating the output state of the digital I/O. See the *Description* below for more information concerning this parameter. Valid return values: 0 to 65535.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the current programmed output state of both banks of the digital I/O. The **dioOut** parameter is a decimal value that must be converted to a 16-bit binary value. Once done, the eight most significant bits correspond to the eight channels of bank one (channels 8-15). The eight least significant bits correspond to the eight channels of bank zero (channels 0-7). This is illustrated below.




The upper 16-bits will always be zero.

For example, a user queries the output state and the following was returned:

**dioOut** = 49164 → 0x0000C00C → 11000000 00001100

This indicates that channels 2 and 3 of digital I/O bank zero and channels 14 and 15 of digital I/O bank one are configured high, while the remaining channels are low.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 dio_out;
…
status = vtex1629_get_dio_output(instrumentHandle, &dio_out);
```

# vtex1629_get_dsp_version

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_dsp_version (ViSession **vi**, ViInt32 **board**, ViPInt32 **dspMajor**, ViPInt32 **dspMinor**, ViPInt32 **dspBuild**, ViChar _VI_FAR **date[]**, ViChar _VI_FAR **time[]**, ViPInt32 **FPGAVersion**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**board** = indicates the analog board for which the function will retrieve information. Valid values are 0, 1, or 2.
**dspMajor** = indicates the minor dsp version.
**dspMinor** = indicates the minor DSP version.
**dspBuild** = indicates the DSP build.
**date[]** = indicates the date of the DSP build. The client should allocate an array of 32 characters for this returned string.
**time[]** = indicates the time of the DSP build. The client should allocate an array of 32 characters for this returned string.
**FPGAVersion** = indicates the FPGA version.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function returns the DSP (digital signal processor) version information for a given analog board. This function is intended for factory use only. Customers should refer to the firmware version reported by vtex1629_revision_query function.

# vtex1629_get_EU_conversion

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_EU_conversion (ViSession **vi**, ViInt32 **channel**, ViPInt32 **EUConversionType**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the EU conversion type will be returned. Valid input values: 0 to 47.
**EUConversionType** = an integer return value indicating the EU conversion type. See *Description* below for more information. Valid return values: 0 to 10.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the EU conversion type for a specific channel. Valid return values for the **EUConversionType** parameter are:

| Decimal Value | Hex Value | #define Symbol | EUConversionType Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_EUCONV_VOLT_OUTPUT | Voltage |
| 1 | 0x01 | VTEX1629_EUCONV_QTR_BRIDGE_120 | Quarter-Bridge 120 |
| 2 | 0x02 | VTEX1629_EUCONV_QTR_BRIDGE_350 | Quarter-Bridge 350 |
| 3 | 0x03 | VTEX1629_EUCONV_QTR_BRIDGE_USER | Quarter-Bridge User |
| 4 | 0x04 | VTEX1629_EUCONV_HALF_BRIDGE_BEND | Half-Bridge Bending |
| 5 | 0x05 | VTEX1629_EUCONV_HALF_BRIDGE_POIS | Half-Bridge Poisson |
| 6 | 0x06 | VTEX1629_EUCONV_FULL_BRIDGE_BEND | Full-Bridge Bending |
| 7 | 0x07 | VTEX1629_EUCONV_FULL_BRIDGE_POIS | Full-Bridge Poisson |
| 8 | 0x08 | VTEX1629_EUCONV_FULL_BRIDGE_BPOIS | Full-Bridge Bending Poisson |
| 9 | 0x09 | VTEX1629_EUCONV_RATIOMETRIC | Ratiometric |
| 10 | 0x0A | VTEX1629_EUCONV_LINEAR | Linear |

See the *Engineering Unit (EU)* Conversion section in Section 3 for more details.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 euconv;
…
status = vtex1629_get_EU_conversion(instrumentHandle, 16, &euconv);
```

# vtex1629_get_euconv_dynamic_excitation_enabled

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_euconv_dynamic_excitation_enabled (ViSession **vi**, ViInt32 **channel**, ViPBoolean **enabled**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the EU conversion type will be returned. Valid input values: 0 to 47.
**enabled** = a Boolean return value indicating the enabled status of the excitation source. A returned value of "1" indicates that the excitation source for the given channel is enabled.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the dynamic excitation EU conversion state. In this mode of operation, the EX1629 uses the excitation voltage measured by the confidence ADC (in real time) in its calculations. This mode is available for bridge sampling frequencies ($f_s$) less than 1 kHz. While in this mode, it is advised to give the confidence filters at least 1.5 s to settle, from the time the confidence source for excitation voltage is enabled or the excitation value is changed.

The vtex1629_measure_excitation_voltage interface uses the confidence subsystem to measure the excitation voltage. It returns these voltages to the calling function. Optionally, it can update the excitation voltage value used for the strain EU conversion. The dynamic excitation EU conversion is slightly different. It is a mode of operation that essentially does the same operations as vtex1629_measure_excitation_voltage, measuring the excitation voltage using the confidence subsystem and updating the excitation voltage value used in the EU conversion in real-time.

This is a Boolean mode of operation, selectable per channel. If the user enables this mode, the set excitation voltage EU function should return an error (users should not be able to manually set the excitation voltage EU value when in this automatic mode). If the user queries the excitation voltage EU value, the result is the latest, real-time value.

## EXAMPLE

```
ViSession instrumentHandle;
ViInt32 channel = 5;
ViBoolean get_enabled = VI_TRUE;

status = vtex1629_get_euconv_dynamic_excitation_enabled (instrumentHandle,
                                                         channel,
                                                         &get_enabled);
```

# vtex1629_get_euconv_excitation

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_euconv_excitation (ViSession **vi**, ViInt32 **channel**, ViPReal64 **euConversionVoltage**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** =an integer input value that specifies the channel for which the EU conversion excitation voltage will be returned. Valid input values: 0 to 47.
**euConversionVoltage** = a real return value, in volts, that indicates the EU conversion excitation voltage for the given channel. Valid return values: 0.00000 to +16.00000.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the current value used in EU conversions for the excitation voltage for a given channel.

| | |
|---|---|
| **NOTE** | This value may be different than what is currently programmed on the excitation sources, depending on the system configuration (although in almost all cases it should be the same). This value is the number that is used int the formulas that convert voltage measurements to strain measurements. |

Please refer to vtex1629_get_excitation, vtex1629_get_excitation_enabled, and vtex1629_measure_excitation_voltage for more information.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViReal64 euexcite;
…
status = vtex1629_get_euconv_excitation(instrumentHandle, 47, &euexcite);
```

# vtex1629_get_excitation

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_excitation (ViSession **vi**, ViInt32 **channel**, ViPReal64 **positiveExcitationVoltage**, ViPReal64 **negativeExcitationVoltage**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the programmed excitation voltages will be returned. Valid input values: 0 to 47.
**positiveExcitationVoltage** = a real return value, in volts, indicating the programmed positive excitation voltage for a given channel. Valid return values: 0.000000 through 8.000000.
**negativeExcitationVoltage** = a real return value, in volts, indicating the programmed negative excitation voltage for a given channel. Valid return values: -8.000000 through 0.000000.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the actual excitation voltages for a given channel. The excitation value is quantized with a 14-bit DAC. As a result, excitation value returned may be different from the value that was programmed using the vtex1629_set_excitation function call. The total voltage applied to the bridge is equal to the positive excitation voltage minus the negative excitation voltage. For example, if the positive excitation voltage is +5.0 V and the negative excitation voltage is -5.0 V, the total excitation voltage is 10.0 V (5.0 - -5.0).

| **NOTE** | The control of the excitation voltage values and their enabling are disjoint operations. Thus, the return of a nonzero value does not guarantee that the excitation source is enabled. That must be queried with the vtex1629_get_excitation_enabled call. |
|---|---|

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViReal64 exc_pos, exc_neg;
…
status = vtex1629_get_excitation(instrumentHandle, 0, &exc_pos, &exc_neg);
```

# vtex1629_get_excitation_enabled

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_excitation_enabled (ViSession **vi**, ViInt32 **channel**, ViPBoolean **enabled**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the enabled status of the excitation source will be returned. Valid input values: 0 to 47.
**enabled** = a Boolean return value indicating the enabled status of the excitation source. A returned value of "1" indicates that the excitation source for the given channel is enabled.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the enabled status of the excitation source for a particular channel. An excitation source that is not enabled will output 0 V, regardless of its programmed value.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViBoolean exc_ena;
…
status = vtex1629_get_excitation_enabled(instrumentHandle, 0, &exc_ena);
```

# vtex1629_get_fifo_count

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_fifo_count (ViSession **vi**, ViPInt32 **pageCount**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**pageCount** = an integer return value that indicates the current number of pages stored in the FIFO.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries the EX1629 for the current FIFO memory page count. One page of data in the FIFO corresponds to one scan, or sample of all enabled channels, taken by the instrument.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 fifo_count;
…
status = vtex1629_get_fifo_count(instrumentHandle, &fifo_count);
```

# vtex1629_get_gain

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_gain (ViSession **vi**, ViInt32 **channel**,ViPReal64 **gain**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the gain will be returned. Valid input values: 0 to 47.
**gain** = a real return value indicating the specified channel's currently configured gain setting. Valid return values: 1.00, 10.0, and 100.0.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the specified channel's signal conditioning gain.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViReal64 gain;
…
status = vtex1629_get_gain(instrumentHandle, 31, &gain);
```

# vtex1629_get_gauge_factor

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_gauge_factor (ViSession **vi**, ViInt32 **channel**, ViPReal64 **gageFactor**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the gage factor will be returned. Valid input values: 0 to 47.
**gageFactor** = a real return value indicating the given channel's currently entered gage factor value.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the gage factor for a specific channel. This is one of the parameters used in EU conversion calculations.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViReal64 gf;
…
status = vtex1629_get_gage_factor(instrumentHandle, 0, &gf);
```

# vtex1629_get_half_bridge_lead_wire_desensitization

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_half_bridge_lead_wire_desensitization (ViSession **vi**, ViInt32 **channel**, ViPReal64 **factor**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the gage factor will be returned. Valid input values: 0 to 47.
**factor** = returned factor value for the specified channel.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the lead wire desensitization factor for the specified channel. The **factor** parameter is defined as follows:

$$factor = 1 + \frac{R_{lead}}{R_{gage}}$$

where $R_{\text{lead}}$ represents the resistance of the lead and $R_{\text{gage}}$ is the resistance of the strain gage.

## EXAMPLE

```
ViSession instrumentHandle;
ViInt32 channel = 5;
ViReal64 factorVal = 0;

status = vtex1629_get_half_bridge_lead_wire_desensitization(instrumentHandle,
                                                            channel,
                                                            &factorVal);
```

# vtex1629_get_IIR_filter_configuration

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_IIR_filter_configuration (ViSession **vi**, ViInt32 **channel**, ViPInt32 **filterType**, ViPReal64 **cutoffFreq**, ViPInt32 **transform**, ViPInt32 **filterOrder**, ViPReal64 **groupDelay**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the IIR filter configuration will be returned. Valid input values: 0 to 47.
**filterType** = an integer return value indicating the type of filter that is currently configured to be used for the given channel. See *Description* below for more information. Valid return values: 0, 1, or 2.
**cutoffFreq** = a real return value indicating the current cutoff frequency, in hertz (Hz), currently configured in filters on the given channel. This parameter is only relevant for Bessel and Butterworth filter types. Valid return values: 0.001 to 4005.
**transform** = an integer return value indicating the type of filter transform currently configured for the given channel. This parameter is only relevant for Bessel and Butterworth filter types. Valid return values: 0 or 1.
**filterOrder** = an integer return value indicating the order of the filter on a given channel. This parameter is only relevant for Bessel and Butterworth filter types. Valid return values: 1 to 10.
**groupDelay** = a real return value indicating the group delay, in number of samples, of the currently configured filter for the given channel.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the IIR filter configuration parameters for a given channel. The **filterType** parameter has three allowed values:

| Decimal Value | Hex Value | #define Symbol | filterType Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_IIR_FILT_NONE | None |
| 1 | 0x01 | VTEX1629_IIR_FILT_BUTTERWORTH | Butterworth |
| 2 | 0x02 | VTEX1629_IIR_FILT_BESSEL | Bessel |

A value of “None” disables the IIR filters. Note, however, that the FIR filter is always enabled and is not user-configurable.

The **cutoffFreq** parameter defines the cutoff (-3 dB) frequency for the low-pass filter indicated above. The EX1629 will locate this parameter in the range [$f_s$/1000, $f_c$ max] (see Table B-1), where $f_s$ is the current sampling frequency. Note that this value can change if the sampling frequency is altered. The actual value can be queried with the vtex1629_get_IIR_filter_configuration command.

The **transform** parameter provides for two modes of transformation:

| Decimal Value | Hex Value | #define Symbol | transform Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_TRANSFORM_BILINEAR | Bilinear |
| 1 | 0x01 | VTEX1629_TRANSFORM_MATCHEDZ | Matched-Z |

The **filterOrder** parameter defines the desired order of the filter. When the **filterType** is set to Butterworth, there is an additional option of 0. This corresponds to an automatic option, whereby the EX1629 will assign an order based on an analog prototype Butterworth design given the sampling frequency, cutoff frequency, and a -200 dB attenuation at the Nyquist frequency. The actual order can be determined using the command.

The **groupDelay** parameter is a measure of rate of change of the total phase shift with respect to angular frequency. When IIR filters are enabled, this number is specific to the dc portion of the signal. When IIR filters are disabled, this parameter reflects the group delay of the FIR filters only which are constant across all frequencies.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 ftype, fxfrm, forder
ViReal64 ffreq, grpdly;
…
status = vtex1629_get_IIR_filter_configuration(instrumentHandle,
                                               0,
                                               &ftype,
                                               &ffreq,
                                               &fxfrm,
                                               &forder,
                                               &grpdly);
```

# vtex1629_get_input_multiplexer

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_input_multiplexer (ViSession **vi**, ViInt32 _VI_FAR **channels[]**, ViInt32 **numberOfChannels**, ViPInt32 **muxInValue**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel[]** = an integer input array that specifies the channel for which the configuration will be returned. Valid input values: 0 to 47.
**numberOfChannels** = a return integer value indicating the number of channels currently included in the scan list. Valid return values: 1 to 48.
**muxInValue** = indicates the input multiplexer source. Valid return values: 0 to 4

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function returns the input multiplexer source of the channel(s) indicated by the **channel[]** parameter. Possible values for **muxInValue** are**:**

| Decimal Value | Hex Value | #define Symbol | muxInValue Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_INPUTMUX_BRIDGE_TYPE_FULL | Full Bridge |
| 1 | 0x01 | VTEX1629_INPUTMUX_BRIDGE_TYPE_HALF | Half Bridge |
| 2 | 0x02 | VTEX1629_INPUTMUX_BRIDGE_TYPE_QUARTER | Quarter Bridge |
| 3 | 0x03 | VTEX1629_INPUTMUX_BRIDGE_TYPE_CAL | Cal |
| 4 | 0x04 | VTEX1629_INPUTMUX_BRIDGE_TYPE_GND | Gnd |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 mux;
…
status = vtex1629_get_input_multiplexer(instrumentHandle, 0, &mux);
```

# vtex1629_get_instrument_serial_number

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_instrument_serial_number (ViSession **vi**, ViChar _VI_FAR **serialNumber[]**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**serialNumber[]** = indicates the instruments serial number. The client should allocate an array of 64 bytes for the serial number.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function returns the instrument's serial number.

## EXAMPLE

```
ViStatus status;
ViSession instrumentHandle;
ViChar serialNumber[64];
…
…
status = vtex1629_get_instrument_serial_number (instrumentHandle, serialNumber);
if (status < VI_SUCCESS)
{
    <inform the user the API call failed>
}
```

# vtex1629_get_lead_wire_resistance

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_lead_wire_resistance (ViSession **vi**, ViInt32 **channel**, ViPReal64 **resistance**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the IIR filter configuration will be returned. Valid input values: 0 to 47.
**resistance** = returned resistance value for the specified channel. Valid return values are numbers greater than 0.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the currently defined lead wire resistance value.

## EXAMPLE

```
ViSession instrumentHandle;
ViInt32 channel = 5;
ViReal64 resistance = 0;

status = vtex1629_get_lead_wire_resistance(instrumentHandle,
                                           channel,
                                           &resistance);
```

# vtex1629_get_linearscaling_configuration

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_linearscaling_configuration (ViSession **vi**, ViInt32 **channel**, ViPReal64 **m**, ViPReal64 **b**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the linear scaling coefficients will be returned. Valid input values: 0 to 47.
**m** = a real return value indicating the gain factor *m* in the linear equation y = *m*x + *b*.
**b** = a real return value indicating the offset factor *b* in the linear equation y = *m*x + *b*.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function returns the slope (**m**) and intercept (**b**) parameters for a channel when configured for linear EU conversion (*x* being in volts).

| NOTE | The **m** parameter (slope) is stored in the same location as the gage factor and the b parameter (intercept) is stored in the same location as the unstrained voltage. Since the linear scaling EU conversion and the strain EU conversions are mutually exclusive, this is never an issue in practice. |
|---|---|

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViReal64 m, b;
…
status = vtex1629_get_linearscaling_configuration(instrumentHandle, 24, &m, &b);
```

# vtex1629_get_lxibus_configuration

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_lxibus_configuration (ViSession **vi**, ViInt32 **lxiLine**, ViPInt32 **inOut**, ViPInt32 **transmissionScope**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**lxiLine** = an integer input value that indicates the LXI Trigger Bus channel for which the function will return configuration values. Valid values: 0 to 7.
**inOut** = an integer return value that indicates whether the specified LXI Trigger Bus channel is configured for input or output. Valid return values: 0 or 1.
**transmissionScope** = an integer return value indicating whether transmissions on the specified channel are configured to be input from and output to the external LXI bus or will be kept internal to the EX1629. Valid return values: 0 or 1.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns information pertaining to a specified LXI Trigger Bus channel. It determines how the channel's direction and scope of transmissions are currently configured.

| Decimal Value | Hex Value | #define Symbol | lxiLine Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_LXI_LINE_ZERO | LXI LINE 0 |
| 1 | 0x01 | VTEX1629_LXI_LINE_ONE | LXI LINE 1 |
| 2 | 0x02 | VTEX1629_LXI_LINE_TWO | LXI LINE 2 |
| 3 | 0x03 | VTEX1629_LXI_LINE_THREE | LXI LINE 3 |
| 4 | 0x04 | VTEX1629_LXI_LINE_FOUR | LXI LINE 4 |
| 5 | 0x05 | VTEX1629_LXI_LINE_FIVE | LXI LINE 5 |
| 6 | 0x06 | VTEX1629_LXI_LINE_SIX | LXI LINE 6 |
| 7 | 0x07 | VTEX1629_LXI_LINE_SEVEN | LXI LINE 7 |

The **inOut** parameter indicates whether the channel is configured as an input or an output. The return values indicate the following:

| Decimal Value | Hex Value | #define Symbol | inOut Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_LXI_INPUT | Input |
| 1 | 0x01 | VTEX1629_LXI_OUTPUT | Output |

The **transmissionScope** parameter indicates whether the specified LXI channel is configured to communicate with other devices on the external LXI bus or simply configured to communicate on the internal LXI bus. In the case of an output, the **transmissionScope** indicates whether the output will be driven out onto the external LXI bus in addition to being driven on the internal LXI bus. In the case of an input, the **transmissionScope** determines if the input is read from the external bus or read from the internal bus. The return value indicates the following:

| Decimal Value | Hex Value | #define Symbol | transmissionScope Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_LXI_INTERNAL | LXI bus signal routed internally |
| 1 | 0x01 | VTEX1629_LXI_INTERNAL_EXTERNAL | LXI bus signal routed internally and externally |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 inout, scope;
…
status = vtex1629_get_lxibus_configuration(instrumentHandle, 0, &inout, &scope);
```

# vtex1629_get_lxibus_input

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_lxibus_input (ViSession **vi**, ViPInt32 **input**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**input** = an integer return value, that indicates the current input state of each LXI Trigger Bus channel. Valid return values: 0 to 255.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the input state of each of the channels on the LXI Trigger Bus. The **input** parameter is an 8-bit integer where the least significant bit of the integer corresponds to LXI Trigger Bus channel zero and the most significant bit corresponds to LXI Trigger Bus channel seven.

If, for example, the value 129 (0x81) is returned, this corresponds to the 8-bit number 1000 0001. This indicates that LXI Trigger Bus channels zero and seven are currently inputting high signals while the other channels are inputting low signals.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 ins;
…
status = vtex1629_get_lxibus_input(instrumentHandle, &ins);
```

# vtex1629_get_lxibus_output

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_lxibus_output (ViSession **vi**, ViPInt32 **output**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**output** = an integer return value that indicates the current output state of each LXI Trigger Bus channel. Valid return values: 0 to 255.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the output state of each of the channels on the LXI Trigger Bus. The **output** parameter is an 8-bit integer where the least significant bit of the integer corresponds to LXI Trigger Bus channel zero, and the most significant bit corresponds to LXI Trigger Bus channel seven. If an LXI channel is configured as an input, the output state of that channel has no effect.

If, for example, the value 129 (0x81) is returned, this corresponds to the 8-bit number 1000 0001b, which indicates that LXI Trigger Bus channels zero and seven are configured to output high signals, while the rest will output low signals.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 outs;
…
status = vtex1629_get_lxibus_output(instrumentHandle, &outs);
```

# vtex1629_get_pattern_arm_configuration

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_pattern_arm_configuration (ViSession **vi**, ViInt16 _VI_FAR **lxiTrigLines[]**,ViInt16 _VI_FAR **dioLines[]**,ViPBoolean **timer**, ViPInt32 **lxiOutput**, ViPInt32 **lxiInput**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**lxiTrigLines[]** = an integer input array specifying the LXI Trigger Bus channel states that will be accepted as arm events. This includes both levels (high and low) and edges (rising and falling). Valid input values: 0 to 255.
**dioLines** = an integer input array specifying the digital I/O channels that will be accepted as arm events. This includes both levels (high and low) and edges (rising and falling). Valid input values: 0 to 255.
**timer** = a Boolean input value that indicates whether the EX1629 will generate arm events based on the internal timer. When this parameter is VI_TRUE(1), periodic arm events will be generated. Valid return values: VI_FALSE(0) or VI_TRUE(1).
**lxiOutput** = this parameter specifies which LXI Trigger Bus line will be used to output the arm event signals. Valid input values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN.
**lxiInput** = this parameter specifies which LXI Trigger Bus line will be used to input the arm event signals. Valid input values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the EX1629's current configuration for pattern arm mode operation. This mode allows the EX1629 to accept arm events from multiple sources. While in pattern mode, the instrument can accept arm events from the LXI Trigger Bus, digital I/O bus, internal timer, and the software arm function. There is no need to explicitly enable the software arm source. It is always available for use while in pattern arm mode. All of the conditions specified must be met for an arm event to be generated. If multiple conditions are specified for the same LXI or DIO line, any of the conditions for that line can be met to satisfy the pattern requirements for that line.

The **lxiTrigLines[]** parameter is an array of four elements with each array element being an unsigned 8-bit integer. Each bit of this integer corresponds to an LXI Trigger Bus channel. Specifically, the least significant bit corresponds to LXI Trigger Bus channel zero and the most significant bit corresponds to LXI Trigger Bus channel seven. Each element specifies which signals the EX1629 will accept for arm events on the LXI Trigger Bus for different edges or states. If a user wants to specify a channel for arm events, the corresponding bit should be set to "1". The individual array elements specify the following:

0 = lxiTrigLines (Positive Edge)
1 = lxiTrigLines (Negative Edge)
2 = lxiTrigLines (Positive Level)
3 = lxiTrigLines (Negative Level)

For example, if a user wishes to arm the EX1629 on a negative edge signal coming into the LXI Trigger Bus on channel 0 and a positive level on channels 3 and 6, then: **lxiTrigLines[1]** = 0000 0001 = 1 (0x01) and **lxiTrigLines[2]** = 0100 1000 = 72 (0x48)**.**

The **dioLines** parameter is an array of four elements with each array element being an unsigned 16-bit integer. Each bit of this integer corresponds to a digital I/O channel. Specifically, the least significant bit corresponds to a digital I/O channel zero, and the most significant bit corresponds to digital I/O channel seven. Each element specifies which events the EX1629 will accept as arm events on the digital I/O bus for different clock edges or states. If a user wants to specify a channel for arm events, the corresponding bit should be set to "1". Specifically, the individual array elements specify the following:

0 = dioLines (Positive Edge)
1 = dioLines (Negative Edge)
2 = dioLines (Positive Level)
3 = dioLines (Negative Level)

For example, if a user wishes to arm the EX1629 on a negative edge signal coming into the digital I/O bus on channels 0 and 3, then: **dioLines[1]** = 0000 1001 = 9 (0x09).

With regard to the **lxiOutput** parameter, it is important to note that since the EX1629 can simultaneously accept arm events from multiple sources, it is necessary to reserve one of the LXI Trigger Bus lines to communicate these events within the device and to other devices in a multi-device configuration. If the EX1629 is a master driving arm events to peripheral slaves, the **lxiOutput** parameter indicates the LXI Trigger Bus line that will be used to communicate the arm event to the slave devices. It is also necessary to configure this LXI Trigger Bus line to be used as an output (see vtex1629_set_lxibus_configuration).

The **lxiInput** parameter specifies which trigger bus line the master device uses for its arm events. Although this parameter is often set to the same value as **lxiOutput**, there are cases where it might be set to a different value. For instance, when the device is configured as a master device in a star multi-box configuration, the master might output the arm event on LXI2 and input it back in on LXI6.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt16 lxilines[VTEX1629_MAX_LINES];
ViInt16 diolines[VTEX1629_MAX_LINES];
ViBoolean timer_enabled;
ViInt32 lxi_out, lxi_in;
…
status = vtex1629_get_pattern_arm_configuration(instrumentHandle,
                                                lxilines, diolines,
                                                &timer_enabled,
                                                &lix_out,
                                                &lxi_in);
```

# vtex1629_get_pattern_trig_configuration

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_pattern_trig_configuration (ViSession **vi**, ViInt16 _VI_FAR **lxiTrigLines[]**,ViInt16 _VI_FAR **dioLines[]**, ViPBoolean **timer**, ViPInt32 **lxiOutput**, ViPInt32 **lxiInput**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**lxiTrigLines[]** = an integer input array specifying 1) the LXI Trigger Bus channels that will be configured to "listen" for TRIG events and 2) the polarity of the incoming signal will cause the EX1629 to trigger. Valid return values: 0 to 255.
**dioLines** = an integer input array specifying 1) the digital I/O channels that will be configured for trigger events, and 2) the polarity of the incoming signal will cause the EX1629 to trigger. Valid return values: 0 to 255.
**timer** = a Boolean input value that indicates whether the EX1629 will generate trigger events based on the internal timer. When this parameter is VI_TRUE(1), periodic trigger events will be generated. Valid return values: VI_FALSE(0) or VI_TRUE(1).
**lxiOutput** = this parameter specifies which LXI Trigger Bus line will be used to output the trigger event signals. Valid return values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN.
**lxiInput** = this parameter specifies which LXI Trigger Bus line will be used to input trigger event signals. Valid return values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This queries and returns the EX1629's current configuration for the pattern trigger mode of operation. This mode allows the EX1629 to accept TRIG events from multiple sources. While in pattern mode, the instrument can accept TRIG events from the LXI Trigger Bus, digital I/O bus, internal timer, and the software trigger function. The re is no need to explicitly enable the software trigger source. It is always available for use while in pattern trigger mode. All of the conditions specified must be met for a trigger event to be generated. If multiple conditions are specified for the same LXI or DIO line, any of the conditions for that line can be met to satisfy the trigger pattern requirements for that line.

The **lxiTrigLines[]** parameter is an array of four elements with each array element being an unsigned 8-bit integer. Each bit of this integer corresponds to an LXI Trigger Bus channel. Specifically, the least significant bit corresponds to LXI Trigger Bus channel zero and the most significant bit corresponds to LXI Trigger Bus channel seven. Each element specifies which events the EX1629 will accept for trigger events on the LXI Trigger Bus for different clock edges or states. If a user wants to specify a channel for trigger events, the corresponding bit should be set to "1". The individual array elements specify the following:

0 = lxiTrigLines (Positive Edge)
1 = lxiTrigLines (Negative Edge)
2 = lxiTrigLines (Positive Level)
3 = lxiTrigLines (Negative Level)

For example, if a user wishes to trigger the EX1629 on a negative edge signal coming into the LXI Trigger Bus on channel 0 and a positive level on channels 3 and 6, then: **lxiTrigLines[1]** = 0000 0001 = 1 (0x01) and **lxiTrigLines[2]** = 0100 1000 = 72 (0x48).

The **dioLines** parameter is an array of four elements with each array element being an unsigned 16-bit integer. Each bit of this integer corresponds to a digital I/O channel. Specifically, the least significant bit corresponds to a digital I/O channel zero and the most significant bit corresponds to digital I/O channel seven. Each element specifies which events the EX1629 will accept as trigger events on the digital I/O bus for different clock edges or states. If a user wants to specify a channel for trigger events, the corresponding bit should be set to "1". Specifically, the individual array elements specify the following:

0 = dioLines (Positive Edge)
1 = dioLines (Negative Edge)
2 = dioLines (Positive Level)
3 = dioLines (Negative Level)

For example, if a user wishes to trigger the EX1629 on a negative edge signal coming into the digital I/O bus on channels 0 and 3, then: **dioLines[1]** = 0000 1001 = 9 (0x09).

With regard to the **lxiOutput** parameter, it is important to note that since the EX1629 can simultaneously accept trigger events from multiple sources, it is necessary to reserve one of the LXI Trigger Bus line to communicate these events within the device and to other devices in a multi-device configuration. If the EX1629 is a master driving trigger events to peripheral slaves, the **lxiOutput** parameter specifies the LXI Trigger Bus line that will be used to communicate the trigger event to the slave devices. It is also necessary to configure this LXI Trigger Bus line to be used as an output (see vtex1629_set_lxibus_configuration).

The **lxiInput** parameter specifies which trigger bus line the master device uses for its trigger events. Although this parameter is often set to the same value as **lxiOutput**, there are cases where it might be set to a different value. For instance, when the device is configured as a master device in a star multi-box configuration, the master might output the trigger event on LXI2 and input it back in on LXI6.

| Decimal Value | Hex Value | #define Symbol | inLine/outLine Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_LXI_LINE_ZERO | LXI LINE 0 |
| 1 | 0x01 | VTEX1629_LXI_LINE_ONE | LXI LINE 1 |
| 2 | 0x02 | VTEX1629_LXI_LINE_TWO | LXI LINE 2 |
| 3 | 0x03 | VTEX1629_LXI_LINE_THREE | LXI LINE 3 |
| 4 | 0x04 | VTEX1629_LXI_LINE_FOUR | LXI LINE 4 |
| 5 | 0x05 | VTEX1629_LXI_LINE_FIVE | LXI LINE 5 |
| 6 | 0x06 | VTEX1629_LXI_LINE_SIX | LXI LINE 6 |
| 7 | 0x07 | VTEX1629_LXI_LINE_SEVEN | LXI LINE 7 |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt16 lxilines[VTEX1629_MAX_LINES];
ViInt16 diolines[VTEX1629_MAX_LINES];
ViBoolean timer_enabled;
ViInt32 lxi_out, lxi_in;
…
status = vtex1629_get_pattern_trig_configuration(instrumentHandle,
                                                 lxilines,
                                                 diolines,
                                                 &timer_enabled,
                                                 &lix_out,
                                                 &lxi_in);
```

# vtex1629_get_poisson_ratio

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_poisson_ratio (ViSession **vi**, ViInt32 **channel**, ViPReal64 **poissonRatio**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the Poisson ratio will be returned. Valid input values: 0 to 47.
**poissonRatio** = a real return value that indicates the Poisson ratio of the specified channel.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the Poisson ratio for a specific channel. This is one of the parameters used in some strain gage EU conversion calculations.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViReal64 p_ratio;
…
status = vtex1629_get_poisson_ratio(instrumentHandle, 12, &p_ratio);
```

# vtex1629_get_sample_clock_source

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_sample_clock_source (ViSession **vi**, ViPInt32 **sampleClockMode**, ViPInt32 **inLine**, ViPInt32 **outLine**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**sampleClockMode** = an integer output value that indicates whether the EX1629 is operating as a master or slave. Valid return values: VTEX1629_SAMP_CLK_MODE_MASTER or VTEX1629_SAMP_CLK_MODE_SLAVE
**inLine** = an integer output value that indicates the trigger bus line configured to listen for sample clock events. Valid return values: VTEX1629_LXI_LINE_ZERO, VTEX1629_LXI_LINE_FOUR, or VTEX1629_LXI_LINE_NONE.
**outLine** = an integer output value that indicates the trigger bus line configured to output sample clock events. Valid return values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN, VTEX1629_LXI_LINE_NONE.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the configured sample clock source.

The **sampleClockMode** parameter indicates whether the EX1629 is configured as a master device that outputs a sample clock for itself and other devices or as a slave device that receives its sample clock from another device. When operating in standalone mode, **sampleClockMode** should be configured as a master.

The **inLine** parameter indicates the LXI line that should be used as the sample clock input. This value is applicable regardless of whether the device is configured as a master or a slave. When **inLine** is set to VTEX1629_LXI_LINE_NONE, the internal sample clock line is used.

The **outLine** parameter indicates the LXI line that should be used as the sample clock output. This value is only applicable when the device is configured as a master. When **outLine** is set to VTEX1629_LXI_LINE_NONE, the sample clock is output on the internal sample clock line.

| Decimal Value | Hex Value | #define Symbol | inLine/outLine Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_LXI_LINE_ZERO | LXI LINE 0 |
| 4 | 0x04 | VTEX1629_LXI_LINE_FOUR | LXI LINE 4 |
| 8 | 0x08 | VTEX1629_LXI_LINE_NONE | None |

When in master mode, the **inLine** and **outLine** parameters may be the same or they may be different. One case where they would be different is if the master is outputting the sample clock on one LXI line and receiving it back in from a LXI Trigger Bus hub on another line. When in standalone mode, **inLine** and **outLine** will always be the same.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 sample_clock_mode;
ViInt32 in_line, out_line;
…
status = vtex1629_get_sample_clock_source(instrumentHandle,
                                          &samp_clock_mode,
                                          &in_line,
                                          &out_line);
```

# vtex1629_get_sample_count

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_sample_count (ViSession **vi**, ViPInt32 **preTrigSampleCount**, ViPInt32 **postTrigSampleCount**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**preTrigSampleCount** = an integer return value indicating the currently configured pre-trigger sample count. Valid return values: 0 to 400000000.
**postTrigSampleCount** = an integer return value indicating the currently configured post-trigger sample count. Valid return values: 0 to 400000000.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns both the pre-trigger and the post-trigger sample count for the EX1629. Specifically, this is the number of samples that will be taken per TRIG event. If "0" is returned, the sample count is infinite.

| **NOTE** | Pre-trigger sampling is not currently supported. Setting the **preTrigSampleCount** to a value other than zero (0) will result in an error. |
|---|---|

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 pretrig, posttrig;
…
status = vtex1629_get_sample_count(instrumentHandle, &pretrig, &posttrig);
```

# vtex1629_get_sample_frequency

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_sample_frequency (ViSession **vi**, ViPReal64 **sampleFrequency**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**sampleFrequency** = a real return value indicating the currently configured sample frequency in hertz (Hz).

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the currently configured sample frequency, in hertz (Hz), for all channels. The EX1629 offers a discrete number of sample frequencies. Programmed values that fall between valid values will be rounded to the closest valid value. This function returns the actual sample frequency, which, due to this quantization, may be different from that which was programmed with the vtex1629_set_sample_frequency call.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViReal64 samp_freq;
…
status = vtex1629_get_sample_frequency(instrumentHandle, &samp_freq);
```

# vtex1629_get_scanlist

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_scanlist (ViSession **vi**, ViInt32 _VI_FAR **channels[]**,ViPInt32 **numberOfChannels**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an integer return array indicating which channels are included in the scan list. Valid return values: 0 to 47.
**numberOfChannels** = a return integer value indicating the number of channels currently included in the scan list. Valid return values: 1 to 48.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns a list of channels currently configured to be sampled in the data acquisition process. The **channels** array must be at least 48 elements long.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[VTEX1629_MAX_SCANLIST_LENGTH];
ViInt32 numberOfChannels;
…
status = vtex1629_get_scanlist(instrumentHandle, channels, &numberOfChannels);
```

# vtex1629_get_selfcal_status

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_selfcal_status (ViSession **vi**, ViInt32 **failedChannelArraySize**, ViInt32 _VI_FAR **failedChannelArray[]**,ViPInt32 **failedChannelArrayActualSize**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**failedChannelArraySize** = defines the size of the **failedChannelArray[]** parameter. Valid return values: 1 to 48.
**failedChannelArray[]** = a integer return array of the channels that failed self-calibration. Valid return values: 0 to 47.
**failedChannelArrayActualSize** = the actual size of the returned array **failedChannelArray[]** parameter. Valid return values: 0 to 48.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns self-calibration failure status for the selected channels. To ensure that an incomplete list is not returned, it is recommended that the **failedChannelArraySize** parameter be set to 48. Note that if the **failedChannelArrayActualSize** parameter returns a '0', all channels passed self-calibration.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status = VI_SUCCESS;
ViInt32 failedChannelArraySize = MAX_CHANNELS;
ViInt32 failedChannelArray[MAX_CHANNELS];
ViInt32 failedChannelArrayActualSize = 0;

status = vtex1629_get_selfcal_status(instrumentHandle,
                                     failedChannelArraySize,
                                     failedChannelArray,
                                     &failedChannelArrayActualSize);
```

# vtex1629_get_settling_time

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_settling_time (ViSession **vi**, ViInt32 **channel**, ViPReal64 **settling_time**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the settling time will be returned. Valid input values: 0 to 47.
**settling_time** = a real return value that indicates the settling time (in seconds) of the specified channel.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the current settling time for a particular channel. Settling time is defined as the amount of time, in seconds, taken by the EX1629 signal conditioning module to settle to the input vale ± 2% of the input value after a reset of the signal conditioning path. The signal conditioning path is reset on a sync event. The settling time is a function of the sampling frequency and the IIR filter settings. As such, disabling the IIR filters does not make sampling time "0".

| | |
|---|---|
| **NOTE** | The signal conditioning path is reset on a sync event, which means that acquisition data will not reflect the input signals until **settling_time** seconds have elapsed. Settling times will vary between channels, depending on the channel's filter configuration. The recommended technique is to configure the instrument completely, query each channel's settling time delay, and then delay for the maximum of these settling delays after issue a sync (vtex1629_soft_synch) prior to initiating acquisition. |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViReal64 settling_time;
…
status = vtex1629_get_settling_time(instrumentHandle, 0, &settling_time);
```

# vtex1629_get_shunt_enabled

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_shunt_enabled (ViSession **vi**, ViInt32 **channel**, ViPBoolean **enabled**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the enabled status of the shunt resistor will be returned. Valid input values: 0 to 47.
**enabled** = a Boolean return value indicating whether or not the shunt resistor is enabled for the given channel. A returned value of "1" indicates that the shunt is enabled for that channel.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the enabled status of a particular channel's shunt resistor. Unless the shunt source is enabled by using the vtex1629_set_shunt_enabled function, it will not be applied in hardware. The shunt source can be configured using the vtex1629_set_shunt_source function.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViBoolean shunt_enabled;
…
status = vtex1629_get_shunt_enabled(instrumentHandle, 0, & shunt_enabled);
```

# vtex1629_get_shunt_source

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_shunt_source (ViSession **vi**, ViInt32 **channel**, ViPInt32 **shuntSource**)

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the shunt source mode will be returned. Valid input values: 0 to 47.
**shuntSource** = an integer return value that indicates the shunt source mode of the given channel. Valid return values: 0 to 4.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the shunt source mode for a particular channel. The **shuntSource** parameter returns an integer value which correlates to the following sources:

| Decimal Value | Hex Value | #define Symbol | shuntSource Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_FRONT_PANEL_REMOTE | Front panel remote |
| 1 | 0x01 | VTEX1629_FRONT_PANEL_LOCAL | Front panel local |
| 2 | 0x02 | VTEX1629_INTERNAL_REMOTE | Internal remote |
| 3 | 0x03 | VTEX1629_INTERNAL_REMOTE | Internal local |
| 4 | 0x04 | VTEX1629_TEDS_REMOTE | TEDS remote |

Local and Remote refer to how the shunt resistor is connected to the bridge. For “Local”, the connection is made within the EX1629. For “Remote”, the connection is made externally, using the ±RCal signals.

Front Panel, Internal, and TEDS, refer to the three types of shunt sources supported. “Front Panel” refers to the shunt resistors that may be connected directly to the front panel of the EX1629, which are shared by 16 channels (0 through 15, 16 through 31, and 32 through 47). Only one channel may be connected to the Front Panel shunt at a time. “Internal” refers to the internal, per-channel shunt resistor. Since each channel has its own, all channels may be shunted simultaneously. “TEDS” refers to a special shunt resistor/TEDS (1-Wire) configuration. Only one channel may be shunted via the TEDS shunt at a time.

| **NOTE** | The configuration of the shunt source and its enabling are separate operations. Thus, the return of a shunt source using the vtex1629_get_shunt_source function does not guarantee that the shunt source is enabled. That must be queried using the vtex1629_get_shunt_enabled function. |
|---|---|

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 shunt_src;
…
status = vtex1629_get_shunt_source(instrumentHandle, 0, &shunt_src);
```

# vtex1629_get_shunt_value

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_shunt_value (ViSession **vi**, ViInt32 **channel**, ViInt32 **shuntSource**, ViPReal64 **shuntValue**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the shunt value will be returned. Valid input values: 0 to 47.
**shuntSource** = an integer input value indicating the source of the desired shunt value. Valid input values: 0 to 4.
**shuntValue** = a real return value indicating the shunt value for the given channel and shunt source.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns a shunt resistor value based on a given channel and shunt source. The **shuntSource** parameter returns an integer value, 0 through 4, which correlates to the following shunt sources:

| Decimal Value | Hex Value | #define Symbol | shuntSource Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_FRONT_PANEL_REMOTE | Front panel remote |
| 1 | 0x01 | VTEX1629_FRONT_PANEL_LOCAL | Front panel local |
| 2 | 0x02 | VTEX1629_INTERNAL_REMOTE | Internal remote |
| 3 | 0x03 | VTEX1629_INTERNAL_REMOTE | Internal local |
| 4 | 0x04 | VTEX1629_TEDS_REMOTE | TEDS remote |

Local and Remote refer to how the shunt resistor is connected to the bridge. For “Local”, the connection is made within the EX1629. For “Remote”, the connection is made externally, using the ±RCal signals.

Front Panel, Internal, and TEDS, refer to the three types of shunt sources supported. “Front Panel” refers to the shunt resistors that may be connected directly to the front panel of the EX1629, which are shared by 16 channels (0 through 15, 16 through 31, and 32 through 47). Only one channel may be connected to the Front Panel shunt at a time. “Internal” refers to the internal, per-channel shunt resistor. Since each channel has its own, all channels may be shunted simultaneously. “TEDS” refers to a special shunt resistor/TEDS (1-Wire) configuration. Only one channel may be shunted via the TEDS shunt at a time.

Any shunt source mode may be queried for its value, regardless of whether it is selected as the current mode or enabled.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 shunt_src;
ViReal64 shunt_value;
…
status = vtex1629_get_shunt_value(instrumentHandle, 0, &shunt_src, &shunt_value);
```

# vtex1629_get_stored_config_digest

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_stored_config_digest (ViSession **vi**, ViInt32 **digestArraySize**, ViInt8 _VI_FAR **digest[]**, ViPInt32 **digestActualSize**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**digestArraySize** = contains the size of the allocated digest array. For consistency, the client application should allocate VTEX1629_MAX_DIGEST_LENGTH bytes.
**digest[]** = the current configuration's digest.
**digestActualSize** = the actual configuration digest size.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function retrieves the digest of the instrument configuration saved in non-volatile memory. The digest is a digital signature representing the configuration data.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 digestActualSize;
ViInt8 digest[VTEX1629_MAX_DIGEST_LENGTH];
…
…
status = vtex1629_get_stored_config_digest (instrumentHandle,
                                            VTEX1629_MAX_DIGEST_LENGTH,
                                            &digestActualSize);
```

# vtex1629_get_strain_units

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_strain_units (ViSession **vi**, ViInt32 **channel**, ViPBoolean **microStrain**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the strain units will be returned. Valid input values: 0 to 47.
**microStrain** = a Boolean return value that indicates whether the EX1629 will return strain measurements in units of strain (ε) or microstrain (με). A value of "1" indicates that the EX1629 is configured to return microstrain units for the given channel, whereas a value of "0" indicates the EX1629 is configured to return strain units for the given channel.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the configured strain units for a given channel. Each channel can be configured to return strain measurements in strain or microstrain (1 strain (ε) = $1x10^6$ microstrain (με)). This setting has no effect for non-strain EU conversions.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViBoolean units_are_microstrain;
…
…
status = vtex1629_get_strain_units(instrumentHandle, 0, &units_are_microstrain);
```

# vtex1629_get_synch_source

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_synch_source (ViSession **vi**, ViPInt32 **synchMode**, ViPInt32 **inLine**, ViPInt32 **outLine**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**synchMode** = an integer output value that indicates whether the EX1629 is operating as a master or slave. Valid return values: VTEX1629_SYNC_MODE_MASTER or VTEX1629_SYNC_MODE_SLAVE.
**inLine** = an integer output value that indicates the trigger bus line configured to listen for sample clock events. Valid return values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN, VTEX1629_LXI_LINE_NONE.
**outLine** = an integer output value that indicates the trigger bus line configured to output sample clock events. Valid return values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN, VTEX1629_LXI_LINE_NONE.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the configured synchronization source.

The **synchMode** parameter indicates whether the EX1629 is configured as a master device that outputs a synch signal for itself and other devices or as a slave device that receives its synch signal from another device. When operating in standalone mode, **synchMode** should be configured as a master.

The **inLine** parameter indicates the LXI line that should be used as the synch input. This value is applicable regardless of whether the device is configured as a master or a slave. When **inLine** is set to VTEX1629_LXI_LINE_NONE, the internal synch line is used.

The **outLine** parameter indicates the LXI line that should be used as the synch output. This value is only applicable when the device is configured as a master. When **outLine** is set to VTEX1629_LXI_LINE_NONE, the synch signal is output on the internal synch line.

When in master mode, the **inLine** and **outLine** values may be the same or they may be different. One case where they would be different is if the master is outputting the synch signal on one LXI line and receiving it back in from a LXI Trigger Bus hub on another line. When in standalone mode, **inLine** and **outLine** will always be the same.

| Decimal Value | Hex Value | #define Symbol | inLine/outLine Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_LXI_LINE_ZERO | LXI LINE 0 |
| 1 | 0x01 | VTEX1629_LXI_LINE_ONE | LXI LINE 1 |
| 2 | 0x02 | VTEX1629_LXI_LINE_TWO | LXI LINE 2 |
| 3 | 0x03 | VTEX1629_LXI_LINE_THREE | LXI LINE 3 |
| 4 | 0x04 | VTEX1629_LXI_LINE_FOUR | LXI LINE 4 |
| 5 | 0x05 | VTEX1629_LXI_LINE_FIVE | LXI LINE 5 |
| 6 | 0x06 | VTEX1629_LXI_LINE_SIX | LXI LINE 6 |
| 7 | 0x07 | VTEX1629_LXI_LINE_SEVEN | LXI LINE 7 |
| 8 | 0x08 | VTEX1629_LXI_LINE_NONE | None |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 synchmode, inline, outline;
…
…
status = vtex1629_get_synch_source(instrumentHandle, &synchmode, &inline, &outline);
```

# vtex1629_get_tare

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_tare (ViSession **vi**, ViInt32 **channel**, ViPReal64 **tareValue**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the tare value will be returned. Valid input values: 0 to 47.
**tareValue** = a real return value that indicates the currently configured tare value for the given channel.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function reads the currently configured tare value for a specific channel. The tare value is subtracted from the output of the EU conversion for the channel. It should be specified in the proper units for the EU conversion (e.g., strain, volts, etc.) For strain measurements, it is also important to take into account whether a strain measurement is being output in strain (ε) or microstrain (με) and configure the tare value appropriately.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViReal64 tare_value;
…
…
status = vtex1629_get_tare(instrumentHandle, 16, &tare_value);
```

# vtex1629_get_teds_data

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_teds_data (ViSession **vi**, ViInt32 **channel**, ViInt16 _VI_FAR **tedsID[]**, ViInt32 **maxLength**, ViInt16 _VI_FAR **tedsInfo[]**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the tare value will be returned. Valid input values: 0 to 47.
**tedsID** = a return array that will contain the TEDS ID. Each element of the array corresponds to a byte of data from the ID register of the TEDS device. The size of this array is VTEX1629_TEDS_IDSIZE (8).
**maxLength** = an integer input value that specifies the maximum number of bytes to be retrieved from the TEDS device into **tedsInfo**. In general, this should be equal to VTEX1629_TEDS_DATASIZE (32).
**tedsInfo[]** = A return array that will contain the TEDS Info. Each element of the array corresponds to a byte of data from the ID register of the TEDS device.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the TEDS data for a given channel. It only supports the DS2430 EEPROM. For communicating with other TEDS EEPROM devices, the vtex1629_read_teds_URN, vtex1629_read_teds_MLAN, and vtex1629_write_teds_MLAN functions should be used.

The **tedsID** element is a unique, 64-bit (8-byte) serial number assigned by the manufacturer to the 1-Wire TEDS device. The **tedsInfo** element contains the data stored in the DS2430's 32-bytes of non-volatile memory.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt16 tedsID[VTEX1629_TEDS_IDSIZE];
ViInt32 maxlen;
ViInt16 tedsinfo[VTEX1629_TEDS_DATASIZE];
…
…
status = vtex1629_get_teds_data(instrumentHandle,
                                47,
                                tedsID,
                                VTEX1629_TEDS_DATASIZE,
                                tedsinfo);
```

# vtex1629_get_trigger_count

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_trigger_count (ViSession **vi**, ViPInt32 **trigCount**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**trigCount** = an integer output value that specifies the currently configured trigger count for the EX1629. Valid return values: 1 to ($2^{31}$-1).

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the currently configured trigger count for the EX1629. Specifically, this is the number of times the EX1629 will wait for triggers after being armed before it will abort acquisition and return to the arm layer of the trigger state machine.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 trig_count;
…
…
status = vtex1629_get_trigger_count(instrumentHandle, &trig_count);
```

# vtex1629_get_trigger_delay

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_trigger_delay (ViSession **vi**, ViPReal64 **delay**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**delay** = a real output value, in seconds, indicating the trigger delay. Valid return values: 0 s to 4294.967295 s.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the trigger delay for the EX1629. Specifically, this is the amount of time, in seconds, that the EX1629 will wait after receiving a TRIG event before it begins to acquire data. Note that the value this function returns may not be identical to the value set by the vtex1629_set_trigger_delay function, as the actual delay time will vary with the set sample frequency (i.e., it is quantized, based on the sampling frequency).

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViReal64 trig_delay;
…
…
status = vtex1629_get_trigger_delay(instrumentHandle, &trig_delay);
```

# vtex1629_get_trigger_source

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_trigger_source (ViSession **vi**, ViPInt32 **triggerSource**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**triggerSource** = an integer return value that indicates the current source the EX1629 monitors for TRIG events. See the *Description* below for more information. Valid return values: 0 to 17.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the current setting for the trigger source. Possible values for the **triggerSource** parameter are:

| Decimal Value | Hex Value | #define | armSource Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_TRIG_SRC_IMMEDIATE | Immediate |
| 1 | 0x01 | VTEX1629_TRIG_SRC_PATTERN | Pattern |
| 2 | 0x02 | VTEX1629_TRIG_SRC_LXI0_POS | LXI 0 Positive Edge |
| 3 | 0x03 | VTEX1629_TRIG_SRC_LXI1_POS | LXI 1 Positive Edge |
| 4 | 0x04 | VTEX1629_TRIG_SRC_LXI2_POS | LXI 2 Positive Edge |
| 5 | 0x05 | VTEX1629_TRIG_SRC_LXI3_POS | LXI 3 Positive Edge |
| 6 | 0x06 | VTEX1629_TRIG_SRC_LXI4_POS | LXI 4 Positive Edge |
| 7 | 0x07 | VTEX1629_TRIG_SRC_LXI5_POS | LXI 5 Positive Edge |
| 8 | 0x08 | VTEX1629_TRIG_SRC_LXI6_POS | LXI 6 Positive Edge |
| 9 | 0x09 | VTEX1629_TRIG_SRC_LXI7_POS | LXI 7 Positive Edge |
| 10 | 0x0A | VTEX1629_TRIG_SRC_LXI0_NEG | LXI 0 Negative Edge |
| 11 | 0x0B | VTEX1629_TRIG_SRC_LXI1_NEG | LXI 1 Negative Edge |
| 12 | 0x0C | VTEX1629_TRIG_SRC_LXI2_NEG | LXI 2 Negative Edge |
| 13 | 0x0D | VTEX1629_TRIG_SRC_LXI3_NEG | LXI 3 Negative Edge |
| 14 | 0x0E | VTEX1629_TRIG_SRC_LXI4_NEG | LXI 4 Negative Edge |
| 15 | 0x0F | VTEX1629_TRIG_SRC_LXI5_NEG | LXI 5 Negative Edge |
| 16 | 0x10 | VTEX1629_TRIG_SRC_LXI6_NEG | LXI 6 Negative Edge |
| 17 | 0x11 | VTEX1629_TRIG_SRC_LXI7_NEG | LXI 7 Negative Edge |

**Immediate (0)**: an immediate TRIG source. After receiving the ARM event, the trigger state machine will bypass the TRIG layer and will automatically begin to acquire data.

**Pattern (1)**: this trigger source allows the EX1629 to "listen" for TRIG events on multiple sources. The EX1629 can be configured to "listen" for TRIG events on LXI Trigger Bus channels, digital I/O channels, a timer, and software triggers. The EX1629 can simultaneously "listen" for any combination of these events. The specific pattern is queried with the vtex1629_get_pattern_trig_configuration call.

**LXI *n* Positive Edge (2 – 9)**: these trigger sources refer to TRIG events coming from the LXI Trigger Bus and will cause the EX1629 to trigger on the positive edge of signals coming into the LXI Trigger Bus.

**LXI *n* Negative Edge (10 – 17)**: these trigger sources refer to TRIG events coming from the LXI Trigger Bus and will cause the EX1629 to trigger on the negative edge of signals coming into the LXI Trigger Bus.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 trigsource;
…
status = vtex1629_get_trigger_source(instrumentHandle, &trigsource);
```

# vtex1629_get_trigger_timer

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_trigger_timer (ViSession **vi**, ViPReal64 **timer**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**timer** = a real output value, in seconds, indicating the trigger system timer. Valid return values: 0 s to 4294.967295 s.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries and returns the trigger system timer for the EX1629. This is the amount of time, in seconds, that the EX1629 will wait before generating successive timer events, which can be used as an arm source and/or a trigger source.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViReal64 timer;
…
status = vtex1629_get_trigger_timer(instrumentHandle, &timer);
```

# vtex1629_get_unstrained_voltage

## FUNCTION PROTOTYPE

ViStatus vtex1629_get_unstrained_voltage (ViSession **vi**, ViInt32 **channel**, ViPReal64 **unstrainedVoltage**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the unstrained voltage will be returned. Valid input values: 0 to 47.
**unstrainedVoltage** = a real return value that indicates the currently configured unstrained voltage for the given channel.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

The function queries and returns the unstrained voltage currently configured for a given channel. This is one parameter in the EU calculations and represents the quiescent voltage across the bridge (i.e., with no load applied).

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViReal64 unstrained_voltage;
…
status = vtex1629_get_unstrained_voltage(instrumentHandle, 0, &unstrained_voltage);
```

# vtex1629_identify_sensor

## FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex1629_identify_sensor (ViSession **vi**, ViInt32 **channel**, ViBoolean **LEDOn**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an integer input value that specifies the channel for which the unstrained voltage will be returned. Valid input values: 0 to 47.
**LEDOn** = a real return value that indicates the currently configured unstrained voltage for the given channel. Valid input values: VI_TRUE = LED is on, VI_FALSE = LED is off.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

The function provides the user the ability to control and LED co-located to a sensor to ease the sensor identification process. Two LEDs cannot be illuminated on the same analog board (i.e. channels 0 through 15 are on one analog board, channels 16 through 31 are on another, and channels 32 through 47 are on another board) if front panel shunt, TEDS remote shunt, or internal shunt (remove) are active. If these conditions occur, an error will be generated.

## EXAMPLE

```
ViSession instrumentHandle;
ViInt32 channel = 10;
ViBoolean LED = VI_TRUE;

status = vtex1629_identify_sensor(instrumentHandle, channel, LED);
```

# vtex1629_init

## FUNCTION PROTOTYPE

ViStatus vtex1629_init (ViRsrc **resourceName**, ViBoolean **IDQuery**, ViBoolean **resetEX1629**, ViPSession **vi**);

## FUNCTION PARAMETERS

**resourceName** = this parameter must contain a unique descriptor for the EX1629 to which a session is to be opened. Part of this descriptor is the IP address of the instrument to which the user will connect. See *Description* below for more information.
**IDQuery** = specifies if an identification query will be sent to the instrument. Valid input values: VI_FALSE or VI_TRUE.
**resetEX1629** = determines whether a reset command will be issued to the instrument upon initialization. Valid input values: VI_FALSE or VI_TRUE.
**vi** = this output parameter holds the session handle to the instrument described by the input parameter **resourceName**. If vtex1629_init fails, the location pointed to by this parameter will contain a value of zero.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function opens a session with the instrument and returns a session handle. It can optionally perform an identification query and/or reset the instrument to its default state. If VI_TRUE is passed to the **IDQuery** parameter, an identification query will be made upon initialization. This is done as a protective measure to ensure that the instrument specified by the provided IP address is actually an EX1629. If a VI_TRUE is passed to the **resetEX1629** parameter, the instrument will be reset upon connection, putting it into a known, default configuration. If a VI_FALSE is passed for either parameter, the respective operation will not be performed upon initialization.

The format for the **resourceName** parameter is as follows:

'TCPIP::*y*::INSTR'

where *y* is the IP address or hostname of the instrument where a connection is desired (e.g. 'TCPIP::10.1.1.216::INSTR' specifies an instrument connected at the IP address of 10.1.1.216).

## EXAMPLE

```
ViStatus status;
ViSession instrumentHandle;
ViRsrc instrumentName = "TCPIP::169.128.1.2::INSTR";
…
status = vtex1629_init (instrumentName, VI_TRUE, VI_TRUE, &instrumentHandle);
```

# vtex1629_load_stored_config

## FUNCTION PROTOTYPE

ViStatus vtex1629_load_stored_config (ViSession **vi**, ViInt32 **digestArraySize**, ViInt8 _VI_FAR **digest[]**, ViPInt32 **digestActualSize**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**digestArraySize** = contains the size of the allocated digest array. For consistency, the client application should allocate VTEX1629_MAX_DIGEST_LENGTH bytes.
**digest[]** = the stored configuration's digest.
**digestActualSize** = the actual configuration digest size.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function loads the stored configuration from the instrument's non-volatile memory. It also returns a copy of the stored configuration digest, which is a digital signature representing the actual configuration data.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 digestActualSize;
ViInt8 digest[VTEX1629_MAX_DIGEST_LENGTH];
…
status = vtex1629_load_stored_config(instrumentHandle,
                                     VTEX1629_MAX_DIGEST_LENGTH,
                                     &digestActualSize);
```

# vtex1629_lock

## FUNCTION PROTOTYPE

ViStatus vtex1629_lock (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function attempts to acquire a lock on the instrument. When locked, the EX1629 will only accept calls from the instrument session that successfully acquired the lock. When no client has a lock, calls are accepted from all clients. A lock can only be acquired if the instrument is not already locked by another user.

By design, the locking mechanism is able to be overridden by a secondary host that issues a vtex1629_break_lock call. Thus, the lock provides a warning to other users that the unit is in a protected operation state, but not absolute security.

The lock status of the instrument is unaffected by the vtex1629_reset call. The instrument cannot be reset if the user is not the owner of the lock.

Self-calibration requires the acquisition of a lock prior to its initiation.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_lock(instrumentHandle);
```

# vtex1629_measure_confidence

## FUNCTION PROTOTYPE

ViStatus vtex1629_measure_confidence (ViSession **vi**, ViInt32 **confValue**, ViInt32 **numberOfChannels**, ViInt32 _VI_FAR **channels[]**,ViInt32 **sampleCount**, ViReal64 _VI_FAR **returnedValues[]**, ViPInt32 **numReturnedValues**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**confValue** = defines the confidence source for which to query. Valid input range: 0, 1, or 2.
**numberOfChannels** = a return integer value indicating the number of channels currently included in the scan list. Valid return values: 1 to 48.
**channel[]** = an integer input array that specifies the channel for which the configuration will be returned. Valid input values: 0 to 47.
**sampleCount** = an integer input value indicating the number of measurements to average.
**returnedValues[]**=a real return array of the measured values.
**numReturnedValues** = an integer return value that indicates the number of measured values returned in the **returnedValues** array.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function measures the indicated bridge parameter to indicate measurement confidence. The **confValue** parameter can assume the following values:

| Decimal Value | Hex Value | #define | confValue Description |
|---|---|---|---|
| 0 | 0x00 | CONFIDENCE_BUFFERED_INPUT | Buffered input |
| 1 | 0x01 | CONFIDENCE_EXCITATION_CURRENT | Excitation current |
| 2 | 0x02 | CONFIDENCE_COMMON_MODE_VOLTAGE | Common mode voltage |

The “Buffered Input” confidence source measures the Main ADC input with GAIN = 1. This is useful to verify the Main ADC value and the gain settings. The “Excitation Current” measures the current from the excitation source. Finally, the “Common Mode Voltage” measures the common mode voltage appearing across the inputs of the differential amplifier in the main ADC.

The **returnedValues[]** array returns the values measured. These values are placed in the array according to the **channels[]** array passed to the function. In other words, for every channel number in **channels[]**, there will be a corresponding measurement in **returnedValues**.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status = VI_SUCCESS;

ViInt32 numOfChannels = MAX_CHANNELS;
ViInt32 channels[MAX_CHANNELS];
ViInt32 sampleCount = 100;
ViReal64 returnedVals[MAX_CHANNELS];
ViInt32 numOfReturnedValues = 0;

int i = 0;

for(i = 0; i < numOfChannels; i++) {
    channels[i] = i;
}
```

```
memset(returnedVals, 0x00, sizeof(returnedVals));

status = vtex1629_measure_confidence(instrumentHandle,
                                     CONFIDENCE_EXCITATION_CURRENT,
                                     numOfChannels,
                                     channels,
                                     sampleCount,
                                     returnedVals,
                                     &numOfReturnedValues);
```

# vtex1629_measure_excitation_voltage

## FUNCTION PROTOTYPE

ViStatus vtex1629_measure_excitation_voltage (ViSession **vi**, ViInt32 _VI_FAR **channels[]**, ViInt32 **numberOfChannels**, ViInt32 **excitationSource**, ViInt32 **sampleCount**, ViReal64 _VI_FAR **measuredValues[]**,ViPInt32 **numMeasuredValues**, ViBoolean **euConversion**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the excitation will be measured. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**excitationSource** = an integer input value indicating the excitation sense lines to be measured. Valid input values: 0 or 1.
**sampleCount** = an integer input value indicating the number of measurements to average. See the *Description* section below for valid input values.
**measuredValues** = a real return array of the measured values. The measured values are placed in the array according to the **channels** array passed to the function. In other words, for every channel number in **channels**, there will be a corresponding excitation voltage measurement in **measuredValues**.
**numMeasuredValues** = an integer return value that indicates the number of measured values returned in the **measuredValues** array.
**euConversion** = a Boolean input value indicating whether the measured values should be used for future EU conversions.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function measures the total excitation voltage [(positive excitation voltage) – (negative excitation voltage)] for a list of channels. Setting the **euConversion** parameter to VI_TRUE (1) indicates that these values are to be used for future EU conversions. Setting it to VI_FALSE (0) causes this data to be discarded after being read. Using VI_TRUE is a short cut that eliminates the need for invoking vtex1629_set_euconv_excitation for each channel. Measuring the excitation voltage improves the accuracy of the strain gage EU Conversions.

The **excitationSource** parameter indicates the source to be used for measuring the excitation voltage. The parameter values are defined as follows:

VTEX1629_EXCITE_SRC_LOCAL (0) = local sense
VTEX1629_EXCITE_SRC_REMOTE (1) = remote sense

If the remote sense lines are not connected to the external strain bridge, such as in quarter-bridge configuration, either setting can be used. The values in either case are the same. However, if the remote sense lines are connected to the bridge, as they ideally should be in half- or full-bridge configuration, the remote sense lines should be measured, as they represent the true source output seen by the bridge.

| **NOTE** | This measurement is done with a sampling rate of 500 Hz. This requires that the instrument configuration be modified during the execution of this function, and, thus, requires a sync (see vtex1629_soft_synch) event to be generated before any other acquisitions are performed. |
|---|---|

The **sampleCount** parameter typically can be set to values between 1 and approximately 3000 samples if sampling is performed at 500 Hz and filtering turned off. Since the maximum number of samples that can be stored is dependent on available memory and other settings this value can vary.

## EXAMPLE

```
ViSession instrumentHandle;
ViInt32 channels[5] = {0, 1, 2, 3, 4};
ViInt32 numberOfChannels = 5, numValues = 0;
ViReal64 data[5];
ViStatus status;
…
status = vtex1629_measure_excitation_voltage( instrumentHandle,
                                              channels,
                                              numberOfChannels,
                                              VTEX1629_EXCITE_SRC_REMOTE,
                                              100,
                                              data,
                                              &numValues,
                                              VI_TRUE );
```

# vtex1629_measure_lead_wire_resistance

## FUNCTION PROTOTYPE

ViStatus vtex1629_measure_lead_wire_resistance (ViSession **vi**, ViInt32 **channelsArraySize**, ViInt32 _VI_FAR **channels[]**,ViReal64 _VI_FAR **resistance[]**,ViInt32 **sampleCount**, ViInt32 **setEuconv**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channelsArraySize** = the size of the **channel[]** array. Valid input values: VTEX1629_MIN_SCANLIST_LENGTH (1) to VTEX1629_MAX_SCANLIST_LENGTH (48).
**channel[]** = an integer input array that specifies the channel for which the configuration will be returned. Valid input values: 0 to 47.
**resistance[]** = a real return array of the measured values.
**sampleCount** = an integer input value indicating the number of confidence values to average.
**setEuconv** = a Boolean input value indicating whether or not the measured values should be used for future EU conversions. Valid input values: VI_TRUE or VI_FALSE. See *Description* below for more details.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function measures the lead wire resistance that exists in a strain gage set up. Specifically, there are confidence inputs that represent the absolute voltage of the -Sense node (-BUFFERED_IN) and the -Excitation node (-EXCITEOUT_BUFF). In a three-wire quarter bridge configuration, these points represent both sides of the lead wire resistance that exists between the gage and the -Excitation front panel connection. Moreover, the EX1629 has the capability, again through the confidence system, to determine the current that is flowing through the lead wire resistance. Having the voltage drop and the current flow, the lead wire resistance is then a simple calculation.

It should be noted that there is an implicit assumption that the two lead wire resistances connecting the strain gage to the EX1629 are essentially equal. That is, while this process measures the lead resistance that exists between the gage and the -Excitation front panel connection, it is actually the lead resistance that exists between the gage and the +Excitation front panel connection that causes the desensitization error. However, given that the lead wires should be made from the same material and inherently be equal in length, this is a valid assumption.

| **NOTE** | Early EX1629s do not have hardware which supports this functionality. Implementation of direct lead wire measurement is not possible on first generation units. |
|---|---|

For the **setEuconv** parameter, a value of VI_TRUE (1) instructs the EX1629 to use the measured values for future EU conversions. If set to VI_FALSE (0), the values are stored in the **resistance[]** array, but are not used in measurements. In order for these values to be used, the user must read the array and then manually enter the values using the vtex1629_set_lead_wire_resistance function.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status = VI_SUCCESS;

ViInt32 numberOfChannels = MAX_NUMBER_OF_CHANNELS;
ViInt32 channels[MAX_NUMBER_OF_CHANNELS];
ViReal64 resistance[MAX_NUMBER_OF_CHANNELS];
ViInt32 sampleCount = 100;

int i = 0;

for(i = 0; i < numberOfChannels; i++) {
    channels[i] = i;
}
```

```
memset(resistance, 0x00, sizeof(resistance));

status = vtex1629_measure_lead_wire_resistance(instrumentHandle,
                                               numberOfChannels,
                                               channels,
                                               resistance,
                                               sampleCount,
                                               VI_TRUE);
```

# vtex1629_measure_unstrained_voltage

## FUNCTION PROTOTYPE

ViStatus vtex1629_measure_unstrained_voltage (ViSession **vi**, ViInt32 _VI_FAR **channels[]**,ViInt32 **numberOfChannels**, ViInt32 **sampleCount**, ViReal64 _VI_FAR **measuredValues[]**, ViPInt32 **numMeasuredValues**, ViBoolean **setEuconv**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the unstrained voltage will be measured. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**sampleCount** = an integer input value that indicates the number of samples to average. See the *Description* section below for valid input values.
**measuredValues** = a real return array of the measured values. The measured values are placed in the array according to the **channels** array passed to the function. In other words, for every channel number in **channels**, there will be a corresponding unstrained voltage measurement in **measuredValues**.
**numMeasuredValues** = an integer return value that indicates the number of measured samples actually returned in the **measuredValues** array.
**setEuconv** = a Boolean input value indicating whether the measured values will be used for future EU conversions.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function measures the unstrained voltage for a particular list of channels. The unstrained voltage is one of the variables in the strain EU conversion functions (see the *Basic Operation* section). A precise reading of the unstrained voltage is necessary for accurate strain gage measurements. The measured values are returned in the user-provided **measuredValues** array. Setting the **setEuconv** parameter to VI_TRUE (1) indicates that these values are to be used for future EU conversions. Setting it to VI_FALSE (0) causes this data to be discarded after being read. Using VI_TRUE is a short cut that eliminates the need for invoking vtex1629_set_unstrained_voltage for each channel.

The **sampleCount** parameter typically can be set to values between 1 and approximately 9000 samples if sampling is performed at 500 Hz and filtering turned off. Since the maximum number of samples that can be stored is dependent on available memory and other settings this value can vary. If filters are turned on and/or sampling frequency is reduced, for example, this would decreases the maximum value.

## EXAMPLE

```
ViSession instrumentHandle;
ViInt32 channels[5] = {0, 1, 2, 3, 4};
ViInt32 numberOfChannels = 5, numValues = 0;
ViReal64 data[5];
ViStatus status;
…
status = vtex1629_measure_unstrained_voltage( instrumentHandle,
                                              channels,
                                              numberOfChannels,
                                              100,
                                              data,
                                              &numValues,
                                              VI_TRUE );
```

# vtex1629_read_fifo

## FUNCTION PROTOTYPE

ViStatus vtex1629_read_fifo (ViSession **vi**, ViInt32 **maxscans**, ViReal64 _VI_FAR **seconds[]**, ViReal64 _VI_FAR **nanoSeconds[]**, ViPInt32 **numscans**, ViInt32 **maxdata**, ViReal64 _VI_FAR **data[]**, ViPInt32 **numdata**, ViInt32 **to_secs**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**maxscans** = an integer input value indicating the maximum number of scans that the function should return from the EX1629. Valid input values: 1 to 10,000.
**seconds** = a returned array of real numbers. Each element in the array contains a time stamp, in seconds (s), corresponding to a single scan taken by the instrument.
**nanoSeconds** = a returned array of real numbers. Each element in the array contains a time stamp, in nanoseconds (ns), corresponding to a single scan taken by the instrument.
**numscans** = a returned integer value indicating the number of scans actually returned by the function call.
**maxdata** = an integer input value indicating the maximum length of the **data** array.
**data** = a returned array of real numbers that contains data sampled from the main inputs.
**numdata** = an integer return value indicating the number of data elements actually returned in the **data** array.
**to_secs** = an integer input value indicating the time, in seconds, that the function will spend retrieving data from the EX1629 before timing out and returning (timeout). If a zero is passed for this parameter, the timeout period is infinite.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function retrieves data from the instrument's FIFO (First In, First Out) buffer. Once data is retrieved, the data is permanently erased from the EX1629. The following data is returned by this function:

- Data timestamps in seconds (s) and nanoseconds (ns)
- Sampled data from the main inputs
- Number of scans returned
- Number of data points returned

Data items from the **data** parameter are organized in the array according to the scan list set at the time of data acquisition. For example, if channels 0 - 15 are the scan list channels at the time of acquisition, then after a vtex1629_read_fifo function call is made, the first sixteen elements of the **data** array will correspond to the data from the first scan for channels 0 through 15 (if there is a second scan of data, the next sixteen elements would contain the second scan, etc.).

For each scan of data returned, there will be one timestamp (a seconds and a nanoseconds value) and a number of data values equal to the length of the scanlist. Assuming that the timestamp and data arrays are sized properly, the number of data elements returned (**numdata**) will be equal to **numscans** multiplied by the scanlist length.

vtex1629_read_fifo will return as soon as either the **maxscans** scans of data have been retrieved, or the timeout (**to_secs**) expires, whichever happens first.

There are other mechanisms for retrieving data from the EX1629. Please see *Retrieving Data (Read FIFO and Streaming Data)* for more information.

## EXAMPLE

```
#define NUM_SCANS 10
#define NUM_CHANNELS 48
#define MAX_NUM_SAMPLES (NUM_SCANS * NUM_CHANNELS)
#define TIMEOUT_SECS 5

ViSession instrumentHandle;
ViReal64  seconds[NUM_SCANS];
ViReal64  fractseconds[NUM_SCANS];
ViReal64  acqdata[MAX_NUM_SAMPLES];
ViInt32   numdata, numscans;

result = vtex1629_read_fifo(instrumentHandle,
                            NUM_SCANS,
                            seconds,
                            fractseconds,
                            &numscans,
                            MAX_NUM_SAMPLES,
                            acqdata,
                            &numdata,
                            TIMEOUT_SECS);
```

# vtex1629_read_fifoEx

## FUNCTION PROTOTYPE

ViStatus vtex1629_read_fifoEx (ViSession **vi**, ViInt32 **maxscans**, ViReal64 _VI_FAR **seconds[]**,ViReal64 _VI_FAR **nanoSeconds[]**,ViPInt32 **numscans**, ViInt32 **maxdata**, ViReal64 _VI_FAR **data[]**, ViPInt32 **numdata**, ViInt32 **maxConfData**, ViReal64 _VI_FAR **confData[]**, ViPInt32 **numConfData**, ViInt32 **to_secs**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**maxscans** = an integer input value indicating the maximum number of scans that the function should return from the EX1629.
**seconds** = a returned array of real numbers. Each element in the array contains a time stamp, in seconds (s), corresponding to a single scan taken by the instrument.
**nanoSeconds** = a returned array of real numbers. Each element in the array contains a time stamp, in nanoseconds (ns), corresponding to a single scan taken by the instrument.
**numscans** = a returned integer value indicating the number of scans actually returned by the function call.
**maxdata** = an integer input value indicating the maximum length of the **data** array.
**data** = a returned array of real numbers containing data sampled from the main inputs.
**numdata** = an integer return value indicating the number of data elements actually returned in the **data** array.
**maxConfData** = an integer input value indicating the maximum length of the **confData** array.
**confData** = a returned array of real numbers containing sampled confidence data. See *Description* below for more information.
**numConfdata** = an integer return value indicating the number of elements actually returned in the **confData** array.
**to_secs** = an integer input value indicating the time, in seconds, that the function will spend retrieving data from the EX1629 before timing out and returning. If a zero is passed for this parameter, the timeout period is infinite.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function retrieves data from the instrument. Once data is retrieved, the data is permanently erased from the EX1629. In contrast to vtex1629_read_fifo, vtex1629_read_fifoEx provides access to additional acquisition data. Even more data is available via the streaming data interface (please see *Retrieving Data (Read FIFO and Streaming Data)* for more information).

The following data is returned from this function:

- Data timestamps in seconds (s) and nanoseconds (ns)
- Sampled data from the main inputs
- Confidence data
- Number of scans returned
- Number of data points returned
- Number of confidence data elements returned

Data items from the **data** parameter are organized in the array according to the scan list set at the time of data acquisition. For example, if channels 0 through 15 are the scan list channels at the time of acquisition, then after a vtex1629_read_fifo function call is made, the first sixteen elements of the **data** array will correspond to the data from the first scan for channels 0 through 15(if there is a second scan of data, the next sixteen elements would contain the second scan, etc.).

For each scan of data returned, there will be one timestamp (a seconds and a nanoseconds value) and a number of data values equal to the length of the scanlist. Assuming that the timestamp and data arrays are sized properly, the number of data elements returned (numdata) will be equal to numscans multiplied by the scanlist length.

The **confData** parameter is very similar to the **data** parameter. Where the **data** parameter returns a one-to-one channel-to-data ratio based on the scan list configuration, the **confData** has an additional scan list of its own (configured via vtex1629_set_conf_scanlist) that indicates how much confidence data will be acquired. If the

confidence scan list is set to five elements, five real values of confidence data will be acquired for each channel in the regular channel scan list. The next five data elements in the **confData[]** array will be confidence data for the next channel in the scan list and so on. If a scan list has *x* channels and the **confidence scanlist has a length of** *y*, then there will be *x•y* data items placed in the **confData** array each scan.

vtex1629_read_fifo will return as soon as either the **maxscans** scans of data have been retrieved, or the timeout (**to_secs**) expires, whichever happens first.

There are other mechanisms for retrieving data from the EX1629. Please see *Retrieving Data (Read FIFO and Streaming Data)* for more information.

**EXAMPLE**

```
#define NUM_SCANS 10
#define NUM_CHANNELS 48
#define MAX_NUM_SAMPLES (NUM_SCANS * NUM_CHANNELS)
#define CONF_LENGTH 2
#define MAX_CONF_NUM_SAMPLES (MAX_NUM_SAMPLES * CONF_LENGHT)
#define TIMEOUT_SECS 5

ViSession instrumentHandle;
ViReal64  seconds[NUM_SCANS];
ViReal64  fractseconds[NUM_SCANS];
ViReal64  acqdata[MAX_NUM_SAMPLES];
ViReal64  confdata[MAX_CONF_NUM_SAMPLES];
ViInt32   numdata, numscans, numconfdata;

result = vtex1629_read_fifoEx(instrumentHandle,
                              NUM_SCANS,
                              seconds,
                              fractseconds,
                              &numscans,
                              MAX_NUM_SAMPLES,
                              acqdata,
                              &numdata,
                              MAX_CONF_NUM_SAMPLES,
                              confdata,
                              &numconfdata,
                              TIMEOUT_SECS);
```

# vtex1629_read_teds_MLAN

## FUNCTION PROTOTYPE

ViStatus vtex1629_read_teds_MLAN (ViSession **vi**, ViInt32 **channel**, ViInt32 **bufferArraySize**, ViInt8 _VI_FAR **buffer[]**,ViPInt32 **bufferActualSize**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an input integer value that specifies the channel number for the TEDS device from which the MicroLAN (MLAN) buffer should be read. Valid input values: 0 to 47.
**bufferArraySize** = an input integer indicating the size of the array that holds the MLAN buffer data. Its value should be less than VTEX1629_MAX_MLAN_DATA_LEN.
**buffer[]** = an output array that will contain the TEDS MLAN buffer data. Its size should be equal to VTEX1629_MAX_MLAN_DATA_LEN.
**bufferActualSize** = an output integer indicating the number of bytes actually written to the array holding the MLAN buffer data.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function reads the MicroLAN (MLAN) response buffer from a TEDS device. This response buffer is the response from the TEDS device that corresponds to a series of MLAN commands that were issued to the device using a vtex1629_write_teds_MLAN function. Refer to the vtex1629_write_teds_MLAN function for more information, as well as the *MicroLAN (MLAN) Primer* appendix.

**NOTES**

**1)** Details of the MLAN specification can be found at *http://www.maxim-ic.com/products/ibutton/applications/* and other sites.
**2)** The bytes returned in 'buffer' need to be interpreted by the application in accordance with the MLAN specification.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 bufferActualSize;
ViInt8 mlanData[VTEX1629_MAX_MLAN_DATA_LEN];
…
status = vtex1629_read_teds_MLAN(instrumentHandle,
                                 15,
                                 VTEX1629_MAX_MLAN_DATA_LEN,
                                 mlanData,
                                 &bufferActualSize);
If (status >= VI_SUCCESS)
{
    <interpret the data structure in mlanData>
} else {
    <inform the user the API call failed>
}
```

# vtex1629_read_teds_URN

## FUNCTION PROTOTYPE

ViStatus vtex1629_read_teds_URN (ViSession **vi**, ViInt32 **channel**, ViInt32 **teds_urnArraySize**, ViInt8 _VI_FAR **teds_urn[]**, ViPInt32 **teds_urnActualSize**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channel** = an input integer value that specifies the channel number for the TEDS device from which the Unique Registration Number (URN) should be read. Valid input values: 0 to 47.
**teds_urnArraySize** = an integer indicating the size of the array that holds the URN. Its value should be equal to VTEX1629_MAX_MLAN_URN_SIZE.
**teds_urn[]** = an output array that will contain the TEDS URN. Its size should be equal to VTEX1629_MAX_MLAN_URN_SIZE.
**teds_urnActualSize** = Output integer indicating the number of bytes actually written to the array that holds the URN.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function reads the Unique Registration Number (URN) from a TEDS device. This is a 64-bit value which encodes a device family and a unique serial number.

**NOTES**

**1)** Details of the MLAN specification can be found at *http://www.maxim-ic.com/products/ibutton/applications/* and other sites.
**2)** The bytes returned in 'buffer' need to be interpreted by the application in accordance with the MLAN specification.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 urnActualSize;
ViInt8 urnData[VTEX1629_MAX_MLAN_URN_SIZE];
…
status = vtex1629_read_teds_URN
    (instrumentHandle, 15, VTEX1629_MAX_MLAN_URN_SIZE, urnData, &urnActualSize);
If (status >= VI_SUCCESS)
{
    <do something with the device's URN>
} else {
    <inform the user the API call failed>
}
```

# vtex1629_reset

## FUNCTION PROTOTYPE

ViStatus vtex1629_reset (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function commands the instrument to assume the default settings, as defined in Table 6-1.

| **NOTE** | This function will not release a lock on the EX1629 nor will it affect self-calibration data. |
|---|---|

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
…
status = vtex1629_reset(instrumentHandle);
```

# vtex1629_reset_fifo

## FUNCTION PROTOTYPE

ViStatus vtex1629_reset_fifo (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function clears all the currently stored data from the FIFO memory.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_reset_fifo(instrumentHandle);
```

# vtex1629_reset_tare

## FUNCTION PROTOTYPE

ViStatus vtex1629_reset_tare (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function resets the tare values for all channels to zero.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_reset_tare(instrumentHandle);
```

# vtex1629_reset_trigger_arm

## FUNCTION PROTOTYPE

ViStatus vtex1629_reset_trigger_arm (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function resets all trigger system configuration settings (Trigger and Arm) to their reset values, as defined in Table 6-1. When reconfiguring the trigger system, it is often easiest to reset to the default configuration and then apply the desired configuration.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_reset_trigger_arm(instrumentHandle);
```

# vtex1629_revision_query

## FUNCTION PROTOTYPE

ViStatus vtex1629_revision_query (ViSession **vi**, ViChar _VI_FAR **driverRev[]**, ViChar _VI_FAR **instrRev[]**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**driverRev** = a return string indicating the driver revision.
**instrRev** = a return string indicating the firmware revision.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function returns the revision of the driver as well as the instrument's firmware revision.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViChar driverRevision[256], instrumentRevision[256];
…
status = vtex1629_revision_query (instrumentHandle,
                                  driverRevision,
                                  instrumentRevision);
```

# vtex1629_self_cal_clear

## FUNCTION PROTOTYPE

ViStatus vtex1629_self_cal_clear (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function clears the current self-calibration data. This operation clears the volatile data, but does not affect any self-calibration data that is stored in nonvolatile memory. After this function, the effective calibration will be the full calibration (factory calibration, also known as the annual calibration) plus, if it exists, the nonvolatile self-calibration. If a self-calibration is performed, and non-volatile self-calibration data exists, executing this function effectively reverts to the non-volatile self-calibration data. With no non-volatile self-calibration data, this effectively reverts to full (factory/annual) calibration data.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_self_cal_clear (instrumentHandle);
```

# vtex1629_self_cal_clear_stored

## FUNCTION PROTOTYPE

ViStatus vtex1629_self_cal_clear_stored (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function clears self-calibration data from nonvolatile memory. It does not, however, affect the current self-calibration data, regardless of whether it came from a recent self-calibration or a loading from nonvolatile memory upon power-up initialization. Thus, the effective calibration will not be changed by this call. To return to an operating state where only the full calibration is used, the vtex1629_self_cal_clear function can be called, or the system can be rebooted.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_self_cal_clear_stored (instrumentHandle);
```

# vtex1629_self_cal_get_status

## FUNCTION PROTOTYPE

ViStatus vtex1629_self_cal_get_status (ViSession **vi**, ViPInt32 **percentComplete**, ViPInt32 **calStatus**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**percentComplete** = an integer return value, from 0 to 100, indicating a percentage of completion for the self-calibration process.
**calStatus** = an integer return value indicating the status of the currently running self-calibration process, or the last self-calibration process that was run, and completed. Expected data return values: 0, 1, or 2.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function returns the status of the self-calibration process. This will be the status of the currently running self-calibration routine, or the last self-calibration routine that was completed.

The **calStatus** parameter indicates the following:

0 = Self-calibration in progress
1 = Self-calibration complete
2 = Self-calibration failed.

| | |
|---|---|
| **NOTE** | Additional instrument driver calls should not be performed until the result of **calStatus** is equal to VTEX1629_SELF_CAL_COMPLETE. |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 percent;
ViInt32 calstatus;
…
status = vtex1629_self_cal_get_status (instrumentHandle, &percent, &calstatus);
```

# vtex1629_self_cal_init

## FUNCTION PROTOTYPE

ViStatus vtex1629_self_cal_init (ViSession **vi**, ViPInt32 **overRide**, ViPInt32 **recommendedUpTime**, ViPInt32 **currentUpTime**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**overRide** = a returned integer value indicating the override value.
**recommendedUpTime** = an integer return value indicating the number of seconds recommended for having the box powered on before attempting a self-calibration.
**currentUpTime** = an integer return value indicating the number of seconds that the unit has been powered up.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function initializes the self-calibration routine on the EX1629. Note that the device should be left on and undisturbed throughout the entire calibration process. To check the progress of the calibration routine, refer to the vtex1629_self_cal_get_status function.

| | |
|---|---|
| **NOTE** | The self-calibration uptime requirement is in place to protect the measurement integrity of the instrument. Overriding the requirement must only be done when the operating conditions allow it. An example of this is where the unit has actually been warmed up, but has simply been subjected to a quick power cycle or reboot. In order to ensure that the override is intentional, it is strongly recommended that user intervention be required in the software application to employ it. |

The **overRide** parameter is used in the case where a self-calibration is attempted on a unit that has not been powered on for a sufficient amount of time. In this special case, the first call to this function will return an error, but will also populate this override variable with a unique integer value. The vtex1629_self_cal_init function can then be called a second time using this unique value. The second call to the function will successfully initiate a self-calibration on a unit that has not been powered up for the recommended duration of time.

Because it modifies the system configuration, an instrument synch (vtex1629_soft_synch) will be required before continuing with other operations.

| | |
|---|---|
| **NOTE** | In order to perform a self-calibration, a lock on the instrument must first be acquired. Attempting to self-calibrate without the acquisition of a lock will generate an error that is not able to be overridden. See the vtex1629_lock function. |

| | |
|---|---|
| **NOTE** | Additional instrument driver calls, other than vtex1629_self_cal_get_status, should not be performed until the result of **calStatus** in the vtex1629_self_cal_get_status call is equal to VTEX1629_SELF_CAL_COMPLETE |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 ovrride, rec_uptime, act_uptime;
ViBoolean okay_to_override = VI_FALSE;
…
status = vtex1629_self_cal_init (instrumentHandle,
                                 &override,
                                 &rec_uptime,
                                 &act_uptime);

if (status >= VI_SUCCESS) {
```

```
   if( status == VTEX1629_ERROR_INSUFFICIENT_UPTIME_FOR_CAL ){
      <prompt user to verify that it is okay to override the uptime requirement>
      if( okay_to_override ) {
         status = vtex1629_self_cal_init (instrumentHandle,
                                          &override,
                                          &rec_uptime,
                                          &act_uptime);
         if( status < VI_SUCCESS ) {
            <inform the user the API call failed>
         }
      } else {
         <inform the user the API call failed>
      }
   }
}
```

# vtex1629_self_cal_is_running

## FUNCTION PROTOTYPE

ViStatus vtex1629_self_cal_is_running (ViSession **vi**, ViPBoolean **isRunning**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**isRunning** = a Boolean return value indicating whether self-calibration is running. A return value of "1" indicates that self-calibration is running, whereas "0" indicates that it is not.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function queries the status of self-calibration to determine if self-calibration is currently running.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViBoolean isrunning;
ViInt32 count = 0;
…
<start self calibration with vtex1629_self_cal_init>
…
while(count < 30) {
    status = vtex1629_self_cal_is_running (instrumentHandle, &isrunning);
    if (status >= VI_SUCCESS) {
      if( isrunning ) {
         sleep(5);
      } else {
         break;
      }
    } else {
      <inform the user the API call failed>
    }
    count += 1;
}
```

# vtex1629_self_cal_is_stored

## FUNCTION PROTOTYPE

ViStatus vtex1629_self_cal_is_stored (ViSession **vi**, ViPBoolean **stored**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**stored** = a Boolean return value indicating the presence of a previously saved self-calibration file within nonvolatile memory. A return value of VI_TRUE (1) indicates the existence of a file, whereas VI_FALSE (0) indicates that no file exists.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This functions queries the existence of a previously saved self-calibration file within non-volatile memory. Non-volatile self-calibration data is automatically loaded and used upon an instrument power cycle or reboot. It is stored with the vtex1629_self_cal_store function.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViBoolean isstored;
…
status = vtex1629_self_cal_is_stored (instrumentHandle, &isstored);
```

# vtex1629_self_cal_load

## FUNCTION PROTOTYPE

ViStatus vtex1629_self_cal_load (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function takes a currently stored self-calibration file from nonvolatile memory and loads it as the current self-calibration file to be used in data acquisition. If current (volatile) self-calibration data previously existed, it is simply overwritten and need not be cleared in advance.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_self_cal_load(instrumentHandle);
```

# vtex1629_self_cal_store

## FUNCTION PROTOTYPE

ViStatus vtex1629_self_cal_store (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function takes the current self-calibration image and stores it to nonvolatile memory, enabling it to be loaded upon instrument power cycle and reboot. Any other previously stored self-calibration data will be lost. Since the existence of nonvolatile self-calibration data represents a permanent (although revocable) change from the full calibration settings, its presence is able to be queried. See the vtex1629_self_cal_is_stored command.

| | |
|---|---|
| **NOTE** | Performing a self-calibration does not automatically store the determined calibration constants in nonvolatile memory. However, it does turn the determined constants into the current self-calibration data. A call to this function will store that image permanently in non-volatile storage. |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_self_cal_store(instrumentHandle);
```

# vtex1629_self_test

## FUNCTION PROTOTYPE

ViStatus vtex1629_self_test (ViSession **instrumentHandle**, ViPInt16 **selfTestResult**, ViChar _VI_FAR **testMessage[]**);

## FUNCTION PARAMETERS

**instrumentHandle** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**selfTestResult** = pointer to an INT16. Upon return from the function, it will contain a numerical value indicating the result of the self test.
**testMessage[]** = string where the driver places the textual representation of the self test result.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function causes the instrument to perform a self-test. It waits for the instrument to complete the test and then queries the instrument for the results of the self-test and returns the results to the user. If the instrument passes the self-test, this function returns the constant VTEX1629_SELF_TEST_PASSED in the **TestResult** parameter and "Self test passed" in the **TestMessage** parameter.

The **TestMessage** parameter must be able to store up to 256 characters.

The EX1629 self-test consists of a set of operations that are identical to those performed during self-calibration. No calibration data is modified by this function. Because it modifies the system configuration, however, an instrument synch (vtex1629_soft_synch) will be required before continuing with other operations.

There are two different interfaces to execute the self-test procedure. The vtex1629_self_test function is the simplest – it executes the self-test and returns when the test is complete. When working with a large number of instruments, however, running the self-test sequentially on all instruments can take quite a while. Instead of using vtex1629_self_test, the vtex1629_self_test_init, and vtex1629_self_test_get_status functions can be used, allowing the self-tests on all instruments to be executed in parallel.

## EXAMPLE

```
ViStatus status;
ViSession instrumentHandle;
ViInt16 selfTestResult;
ViChar testMessage[256];
…
status = vtex1629_self_test (instrumentHandle, &selfTestResult, testMessage);
if (status < VI_SUCCESS)
{
    <inform the user the API call failed>
}
if ((status >= VI_SUCCESS) && (selfTestResult != VTEX1629_SELF_TEST_PASSED))
{
    <inform the user the self test failed>
}
if ((status >= VI_SUCCESS) && (selfTestResult == VTEX1629_SELF_TEST_PASSED))
{
    <continue normal processing>
}
```

# vtex1629_self_test_get_status

## FUNCTION PROTOTYPE

ViStatus vtex1629_self_test_get_status (ViSession **vi**, ViPInt32 **selfTestStatus**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**selfTestStatus** = upon return from the function call, it will contain one of the following constants:

| | |
|---|---|
| VTEX1629_SELF_TEST_NO_STATUS | (self-test not started) |
| VTEX1629_SELF_TEST_RUNNING | (self-test started) |
| VTEX1629_SELF_TEST_PASSED | (self-test passed) |
| VTEX1629_SELF_TEST_FAILED | (self-test failed) |

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function returns the status of the self-test initiated via the vtex1629_self_test_init function. It should not be used (and is not necessary) with the vtex1629_self_test function, as that function does not return until the self-test has completed. The self-test takes approximately 3 minutes to execute.

Because the self-test modifies the system configuration, an instrument synch (vtex1629_soft_synch) will be required before continuing with other operations.

## EXAMPLE

```
ViStatus status;
ViSession instrumentHandle;
ViInt16 selfTestResult;
…
status = vtex1629_self_test_get_status (instrumentHandle, &selfTestResult);
if (status < VI_SUCCESS)
{
    <inform the user the API call failed>
}
if ((status >= VI_SUCCESS) && (selfTestResult != VTEX1629_SELF_TEST_PASSED))
{
    <inform the user the self test failed>
}
if ((status >= VI_SUCCESS) && (selfTestResult == VTEX1629_SELF_TEST_PASSED))
{
    <continue normal processing>

}
```

# vtex1629_self_test_init

## FUNCTION PROTOTYPE

ViStatus vtex1629_self_test_init (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function initiates a self-test. It returns immediately, without waiting for the self-test to complete. The vtex1629_self_test_get_status is used to monitor the self-test progress. The vtex1629_self_test function initiates the self-test and waits until the test has completed before returning.

Because it modifies the system configuration, an instrument synch (vtex1629_soft_synch) will be required before continuing with other operations.

## EXAMPLE

```
ViStatus status;
ViSession instrumentHandle;
…
status = vtex1629_self_test_init (instrumentHandle);
if (status < VI_SUCCESS)
{
    <inform the user the API call failed>
}
```

# vtex1629_send_dio_pulse

## FUNCTION PROTOTYPE

ViStatus vtex1629_send_dio_pulse (ViSession **vi**, ViInt32 **dioPulse**);

## FUNCTION PARAMETERS

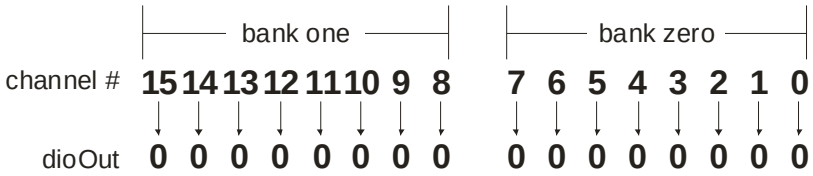
**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**dioPulse** = a bit mask which instructs the instrument to send a pulse on the specified DIO channels. The low 16 bits map to the 16 DIO channels. The upper 16 bits are ignored.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function sends a pulse out on the selected DIO channels.




The upper 16-bits will always be zero.

For example, to issue pulses on DIO lines 0 and 8 (assuming both backs are configured for output:

**dioPulse** = 49164 → 0x00000101 → 00000001 00000001b

Pulse widths are 1 microsecond (µs).

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 pulse_mask = 0x00000101; // pulse output bits 0 and 8
…
status = vtex1629_send_dio_pulse(instrumentHandle, pulse_mask);
```

# vtex1629_send_lxibus_pulse

## FUNCTION PROTOTYPE

ViStatus vtex1629_send_lxibus_pulse (ViSession **vi**, ViInt32 **pulseLines**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**pulseLines** = an integer input value that determines which channels on the LXI Trigger Bus will generate a pulse. Valid input values: 0 to 255.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function sends a pulse out on the desired LXI Trigger Bus channels. The **pulseLines** parameter is an 8-bit integer where the least significant bit of the integer corresponds to LXI Trigger Bus channel zero, and the most significant bit corresponds to LXI Trigger Bus channel seven. For example, if a user wants to send a pulse out on LXI Trigger Bus channels zero and seven, then: **pulseLines** = 10000001b (0x0000 0081), or 129.

Pulse widths are 1 microsecond (µs).

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 pulse_mask = 0x00000081; // pulse bits 7 and 0
…
status = vtex1629_send_lxibus_pulse(instrumentHandle, pulse_mask);
```

# vtex1629_set_arm_count

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_arm_count (ViSession **vi**, ViInt32 **armCount**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**armCount** = an integer input value that specifies the desired arm count for the EX1629. Valid input values: 0 to ($2^{31}$-1). Setting this parameter to "0" makes the **armCount** infinite.

## DATA ITEM RESET VALUE

**armCount** = 1

## DESCRIPTION

This function sets the arm count for the EX1629. This count represents the number of times the EX1629 will wait for arm events to occur after the trigger state machine leaves the INIT layer. Trigger counts should be kept in mind when considering this trigger state machine. If the state machine is configured with both arm and trigger counts greater than one, then, after an arm event is received, the state machine will go through all trigger counts before returning to the arm layer to wait for the next arm event.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_arm_count(instrumentHandle, 2);
```

# vtex1629_set_arm_delay

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_arm_delay (ViSession **vi**, ViReal64 **delay**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**delay** = a real input value, in seconds, indicating the desired arm delay. Valid input values: 0 s to 4294.967295 s.

## DATA ITEM RESET VALUE

**delay** = 0.000000000

## DESCRIPTION

This function sets the arm delay for the EX1629. This indicates the amount of time, in seconds, that the EX1629 will wait after receiving an ARM event before it transitions the trigger state machine from the ARM layer into the TRIG layer.

The actual delay exhibited by the EX1629 is dependent on the sample frequency, set by calling the vtex1629_set_sample_frequency function. The actual delay will be a multiple of the sample time. For example, if the sample frequency is 1 kHz, the sample time is 1 ms. If the arm delay is set to a value less than 0.5 ms, the EX1629 will experience no delay. If the arm delay is set to a value between 0.5 ms and 1.49 ms, the delay exhibited will be 1 ms.

As a result, it is best practice to perform a vtex1629_get_arm_delay call after a vtex1629_set_arm_delay or a vtex1629_set_sample_frequency call is performed to determine the actual delay.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_arm_delay(instrumentHandle, 0.01);
```

# vtex1629_set_arm_source

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_arm_source (ViSession **vi**, ViInt32 **armSource**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**armSource** = an integer input value that indicates the desired source to be monitored for ARM events. See the *Description* section below for more information. Valid input values: 0 to 17.

## DATA ITEM RESET VALUE

**armSource** = 0 (immediate)

## DESCRIPTION

This function sets the arm source on the EX1629. Possible values for the **armSource** parameter are:

| Decimal Value | Hex Value | #define | armSource Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_TRIG_SRC_IMMEDIATE | Immediate |
| 1 | 0x01 | VTEX1629_TRIG_SRC_PATTERN | Pattern |
| 2 | 0x02 | VTEX1629_TRIG_SRC_LXI0_POS | LXI 0 Positive Edge |
| 3 | 0x03 | VTEX1629_TRIG_SRC_LXI1_POS | LXI 1 Positive Edge |
| 4 | 0x04 | VTEX1629_TRIG_SRC_LXI2_POS | LXI 2 Positive Edge |
| 5 | 0x05 | VTEX1629_TRIG_SRC_LXI3_POS | LXI 3 Positive Edge |
| 6 | 0x06 | VTEX1629_TRIG_SRC_LXI4_POS | LXI 4 Positive Edge |
| 7 | 0x07 | VTEX1629_TRIG_SRC_LXI5_POS | LXI 5 Positive Edge |
| 8 | 0x08 | VTEX1629_TRIG_SRC_LXI6_POS | LXI 6 Positive Edge |
| 9 | 0x09 | VTEX1629_TRIG_SRC_LXI7_POS | LXI 7 Positive Edge |
| 10 | 0x0A | VTEX1629_TRIG_SRC_LXI0_NEG | LXI 0 Negative Edge |
| 11 | 0x0B | VTEX1629_TRIG_SRC_LXI1_NEG | LXI 1 Negative Edge |
| 12 | 0x0C | VTEX1629_TRIG_SRC_LXI2_NEG | LXI 2 Negative Edge |
| 13 | 0x0D | VTEX1629_TRIG_SRC_LXI3_NEG | LXI 3 Negative Edge |
| 14 | 0x0E | VTEX1629_TRIG_SRC_LXI4_NEG | LXI 4 Negative Edge |
| 15 | 0x0F | VTEX1629_TRIG_SRC_LXI5_NEG | LXI 5 Negative Edge |
| 16 | 0x10 | VTEX1629_TRIG_SRC_LXI6_NEG | LXI 6 Negative Edge |
| 17 | 0x11 | VTEX1629_TRIG_SRC_LXI7_NEG | LXI 7 Negative Edge |

**Immediate (0)**: an immediate ARM source. After initialization of the trigger system, the trigger state machine will bypass the ARM layer and will automatically transition into the TRIG layer.

**Pattern (1)**: this arm source allows the EX1629 to accept ARM events on multiple sources. Specifically, the EX1629 can be configured to accept ARM events on any LXI Trigger Bus channel, any digital I/O channel, on a timer, or for software arms. The instrument can be configured to accept any combination of these events simultaneously. The specific pattern is set with the vtex1629_set_pattern_arm_configuration command.

**LXI *n* Positive Edge (2 – 9)**: these arm sources refer to ARM events coming from the LXI Trigger Bus. More specifically, these arm sources will cause the EX1629 to arm on the positive edge of signals coming into the LXI Trigger Bus.

**LXI *n* Negative Edge (10 – 17)**: these arm sources are referring to ARM events coming from the LXI Trigger Bus. More specifically, these arm sources will cause the EX1629 to arm on the negative edge of signals coming into the LXI Trigger Bus.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_arm_source(instrumentHandle, VTEX1629_TRIG_SRC_PATTERN);
```

# vtex1629_set_bridge_limit

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_bridge_limit (ViSession **vi**, ViInt32 **numberOfChannels**, ViInt32 _VI_FAR **channels[]**, ViReal64 _VI_FAR **min[]**, ViReal64 _VI_FAR **max[]**);

## FUNCTION PARAMETERS

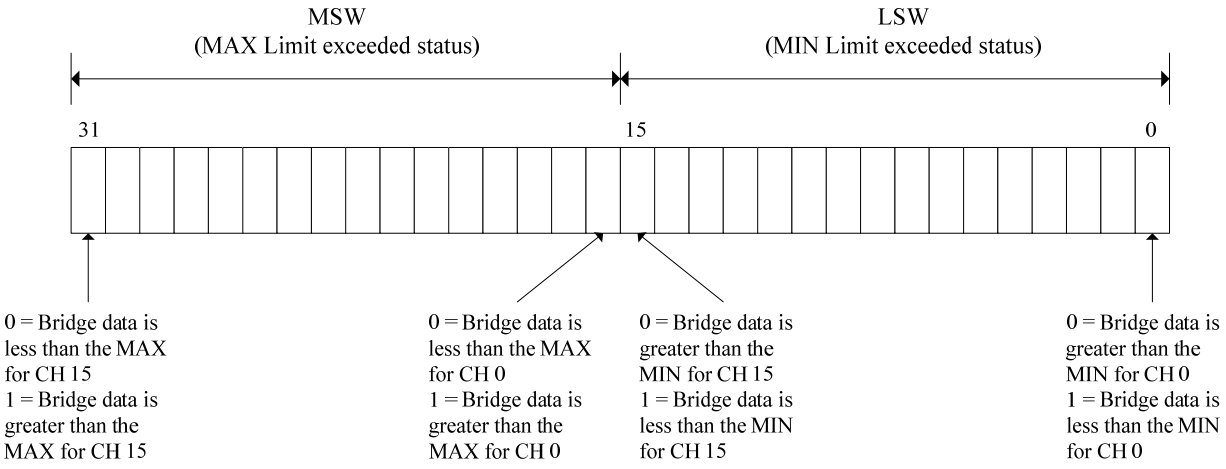
**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**channels[]** = an input integer array containing a list of channel numbers for which the completion resistor configuration will be set. Valid input values: 0 to 47.
**min[]** = an array of minimum bridge limit value.
**max[]** = an array of minimum bridge limit value.

## DATA ITEM RESET VALUE

**min[]** = -infinity (in floating point) **max[]** = +infinity (in floating point)

## DESCRIPTION

This function sets the minimum and maximum bridge limit values for an array of channels. The limit check data is part of the data page along with the bridge data. If the bridge data exceeds the maximum or minimum limit values set for any channel, the corresponding flags are set in the limit check result field in a data page.




The “limits” field is a bit-field. This UINT32 has two bits per channel (16-channels per analog board), one to represent MAX limit exceeded and one to represent MIN limit exceeded. The MSW (upper 16-bits) represent the MAX Limit Exceeded status for each of the 16-channels, and the LSW (lower 16-bits) represent the MIN Limit Exceeded status for each of the 16-channels. Bit 0 represents the MIN Limit Exceeded status for channel 0 (channels 0, 16, 32). Bit 16 represents the MAX Limit Exceeded status for Channel 0 (channels 0, 16, 32). Bit 15 represents the MIN Limit Exceeded status for channel 15 (channels 15, 31, and 47). Bit 31 represents the MAX Limit Exceeded status for channel 15 (channels 15, 31, and 47). The rest of the channels follow the same pattern.

| **NOTE** | The channel-to-bit mapping is constant, regardless of scanlist configuration. For example, whether or not channels 0 and 1 are enabled in the scanlist, for instance, channel 2’s MIN Limit Exceeded Bit and MAX Limit Exceeded Bit are always bits 2 and 18, respectively. |
|---|---|

This mode is valid for main bridge sampling frequencies of 1 kHz or less. If the sampling frequency exceeds 1 kHz a value of 0x0 is reported. Also, the bit fields corresponding to inactive channels in the scanlist will be 0.

Limit checking is performed on the output of the EU conversion. So, if the specified EU conversion is in Strain (Quarter, Half, or Full Bridge) the limit values are in strain (or microstrain). If the specified EU conversion is volts, then the limit values are in volts.

**EXAMPLE**

```
ViSession instrumentHandle;
ViStatus status = VI_SUCCESS;

ViInt32 numChannels = MAX_NUMBER_OF_CHANNELS;
ViInt32 channels[MAX_NUMBER_OF_CHANNELS];
ViReal64 minArr[MAX_NUMBER_OF_CHANNELS];
ViReal64 maxArr[MAX_NUMBER_OF_CHANNELS];

int i = 0;

for(i = 0; i < MAX_NUMBER_OF_CHANNELS; i++) {
    channels[i] = i;
    minArr[i] = (-1.0);
    maxArr[i] = 2.0;
}

status = vtex1629_set_bridge_limit(instrumentHandle,
                                   numChannels,
                                   channels,
                                   minArr,
                                   maxArr);
```

# vtex1629_set_bridge_limit_enabled

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_bridge_limit_enabled (ViSession **vi**, ViBoolean **enabled**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**enabled** = a Boolean input value setting the enabled status of the excitation source. A value of VI_TRUE enables the excitation source. A value of VI_FALSE disables the excitation source.

## DATA ITEM RESET VALUE

**enabled** = VI_FALSE

## DESCRIPTION

This function sets the enabled status of the bridge limit function.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViBoolean enable = VI_TRUE;

status = vtex1629_set_bridge_limit_enabled(instrumentHandle, enable);
```

# vtex1629_set_cal_out

## FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex1629_set_cal_out (ViSession **vi**, ViInt32 **outMode**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**outMode** = an integer input value that indicates the desired output of the calibration source. See the *Description* section below for more information.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function sets the calibration input source to a specified voltage. Possible values for the **outMode** parameter are:

VTEX1629_CAL_OFF (turns the calibration source off)
VTEX1629_CAL_ON (turns the calibration source on)
VTEX1629_CAL_SHORT (shorts the calibration source jacks)
VTEX1629_CAL_VREF (outputs precision calibration voltage source)

The vtex1629_set_cal_source function is used to configure the precision calibration voltage source.

| **NOTE** | This function is intended for factory use only. |
|---|---|

# vtex1629_set_cal_source

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_cal_source (ViSession **vi**, ViInt32 **calSource**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**calSource** = an integer input value that indicates the desired voltage of the calibration source. Valid input values: VTEX1629_CALSRC_0 to VTEX1629_CALSRC_N_14_0. See the *Description* section below for allowed values.

## DATA ITEM RESET VALUE

**calSource** = 0 (VTEX1629_CALSRC_0)

## DESCRIPTION

This function sets the calibration input source to a specified voltage. Valid input values for the **calSource** parameter are:

| Decimal Value | Hex Value | #define Symbol | Nominal Voltage (V) |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_CALSRC_0 | 0 |
| 1 | 0x01 | VTEX1629_CALSRC_P_0_07 | +0.07 |
| 2 | 0x02 | VTEX1629_CALSRC_N_0_07 | -0.07 |
| 3 | 0x03 | VTEX1629_CALSRC_P_0_11 | +0.11 |
| 4 | 0x04 | VTEX1629_CALSRC_N_0_11 | -0.11 |
| 5 | 0x05 | VTEX1629_CALSRC_P_0_14 | +0.14 |
| 6 | 0x06 | VTEX1629_CALSRC_N_0_14 | -0.14 |
| 7 | 0x07 | VTEX1629_CALSRC_P_0_7 | +0.7 |
| 8 | 0x08 | VTEX1629_CALSRC_N_0_7 | -0.7 |
| 9 | 0x09 | VTEX1629_CALSRC_P_1_1 | +1.1 |
| 10 | 0x0A | VTEX1629_CALSRC_N_1_1 | -1.1 |
| 11 | 0x0B | VTEX1629_CALSRC_P_1_4 | +1.4 |
| 12 | 0x0C | VTEX1629_CALSRC_N_1_4 | -1.4 |
| 13 | 0x0D | VTEX1629_CALSRC_P_7_0 | +7.0 |
| 14 | 0x0E | VTEX1629_CALSRC_N_7_0 | -7.0 |
| 15 | 0x0F | VTEX1629_CALSRC_P_11_0 | +11.0 |
| 16 | 0x10 | VTEX1629_CALSRC_N_11_0 | -11.0 |
| 17 | 0x11 | VTEX1629_CALSRC_P_14_0 | +14.0 |
| 18 | 0x12 | VTEX1629_CALSRC_N_14_0 | -14.0 |

**NOTE** This function is intended for factory use only.

# vtex1629_set_completion_resistor

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_completion_resistor (ViSession **vi**, ViInt32 _VI_FAR **channels[]**,ViInt32 **numberOfChannels**, ViInt32 **completionResistorMode**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the completion resistor configuration will be set. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**completionResistorMode** = an integer input value indicating the desired completion resistor mode. See *Description* below for possible values. Valid input values: 0, 3, 4, 120, or 350.

## DATA ITEM RESET VALUE

**completionResistorMode** = 0 (Full)

## DESCRIPTION

This function sets the mode of the completion resistor for a list of channels. The acceptable values for the **completionResistorMode** parameter are as follows:

| Decimal Value | Hex Value | #define Symbol | Resistor Mode | calibratedValue |
|---|---|---|---|---|
| 0 | 0x00 | VTEX1629_COMPRES_FULL | Full | 0.0 (N/A) |
| 3 | 0x03 | VTEX1629_COMPRES_USER | User-Defined | Actual value installed, 0.0 (N/A) otherwise |
| 4 | 0x04 | VTEX1629_COMPRES_OFF | OFF | 0.0 (N/A) |
| 120 | 0x78 | VTEX1629_COMPRES_120 | 120 Ω | Actual value |
| 350 | 0x15E | VTEX1629_COMPRES_350 | 350 Ω | Actual value |

Referring to the “Full” completion resistor is a bit of a misnomer – it really represents a short in the leg of the bridge circuit that contains the completion resistor. It is used in Full- and Half-Bridge mode.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0, 1, 2, 3, 4, 5, 6, 7};
ViInt32 numberOfChannels = 8;
…
status = vtex1629_set_completion_resistor(instrumentHandle,
                                          channels,
                                          numberOfChannels,
                                          VTEX1629_COMPRES_350);
```

# vtex1629_set_conf_scanlist

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_conf_scanlist (ViSession **vi**, ViInt32 _VI_FAR **confElements[]**,ViInt32 **numConfElements**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**confElements** = an integer input array indicating which confidence data elements will be measured. Valid input values: 0 to 12.
**numConfElements** = the size of the **confElements** list. Valid input values: 0 to 12.

## DATA ITEM RESET VALUE

**confElements** = None

## DESCRIPTION

This function sets the list of confidence data elements that will be measured and returned along with the main bridge data. The confidence data elements are the following:

| Decimal Value | Hex Value | #define Symbol | confElements Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_CONFSRC_BRIDGE_POS | Bridge (+) |
| 1 | 0x01 | VTEX1629_CONFSRC_BRIDGE_COMM | Bridge (common mode) |
| 2 | 0x02 | VTEX1629_CONFSRC_BRIDGE_NEG | Bridge (-) |
| 3 | 0x03 | VTEX1629_CONFSRC_EXCITE_POS | Excite (+) |
| 4 | 0x04 | VTEX1629_CONFSRC_EXCITE_NEG | Excite (-) |
| 5 | 0x05 | VTEX1629_CONFSRC_EXCITE_NEG_SENSE | Excite Sense (-) |
| 6 | 0x06 | VTEX1629_CONFSRC_EXCITE_POS_SENSE | Excite Sense (+) |
| 7 | 0x07 | VTEX1629_CONFSRC_EXCITE_POS_CURR | Excite Current (+) |
| 8 | 0x08 | VTEX1629_CONFSRC_EXCITE_NEG_CURR | Excite Current (-) |
| 9 | 0x09 | VTEX1629_CONFSRC_POS_CAL | Calibration Bus (+) |
| 10 | 0x0A | VTEX1629_CONFSRC_NEG_CAL | Calibration Bus (-) |
| 11 | 0x0B | VTEX1629_CONFSRC_GND | Ground |
| 12 | 0x0C | VTEX1629_CONFSRC_EXCITEOUT_BUFF | Excite Out (Buffered) |

**NOTES**
**1)** Confidence elements 9 through 11 are for system diagnostic use only and should not be employed during normal operation.
**2)** Confidence element 12 can only be used on EX1629 with firmware version 1.0 or later.

In order to clear the confidence scan list, a value of 0 should be set for the **numConfElements** parameter. In this case, the value of the **confElements** parameter is arbitrary.

**NOTE** The confidence data is filtered by a transfer function represented by the following differential equation: $y(n) = 0.01x(n) + 0.99y(n-1)$, where $y(n)$ is the filtered confidence data and $x(n)$ is the measured confidence data. This function serves to reduce noise variance.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 confchannels[] = {3, 4, 5, 6};
ViInt32 numberOfChannels = 4;
…
status = vtex1629_set_conf_scanlist (instrumentHandle,
                                     confchannels,
                                     numberOfChannels);
```

# vtex1629_set_confidence_limit

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_confidence_limit (ViSession **vi**, ViInt32 **confSrcEnum**, ViInt32 **numberOfChannels**, ViInt32 _VI_FAR **channelsArray[]**, ViReal64 _VI_FAR **min[]**, ViReal64 _VI_FAR **max[]**);

## FUNCTION PARAMETERS

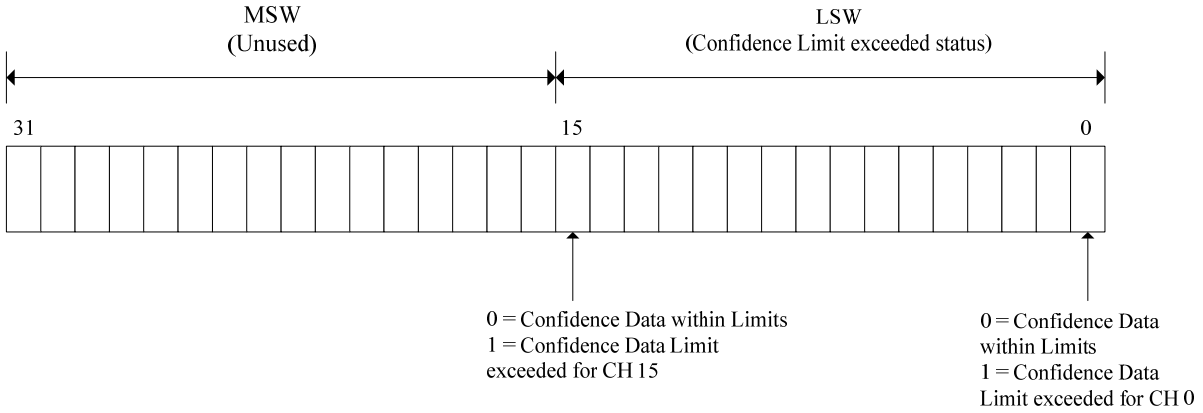
**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**confSrcEnum** = confidence source for which to set the minimum and maximum limits. Valid input values: 0 to 12.
**numberOfChannels** = a return integer value indicating the number of channels currently included in the scan list. Valid return values: 1 to 48.
**channelsArray[]** = the size of the **confElements** list. Valid input values: 0 to 12.
**min[]** = an array of minimum confidence limit values.
**max[]** = an array of minimum confidence limit values.

## DATA ITEM RESET VALUE

**min[]** = -infinity (in floating point) **max[]** = +infinity (in floating point)

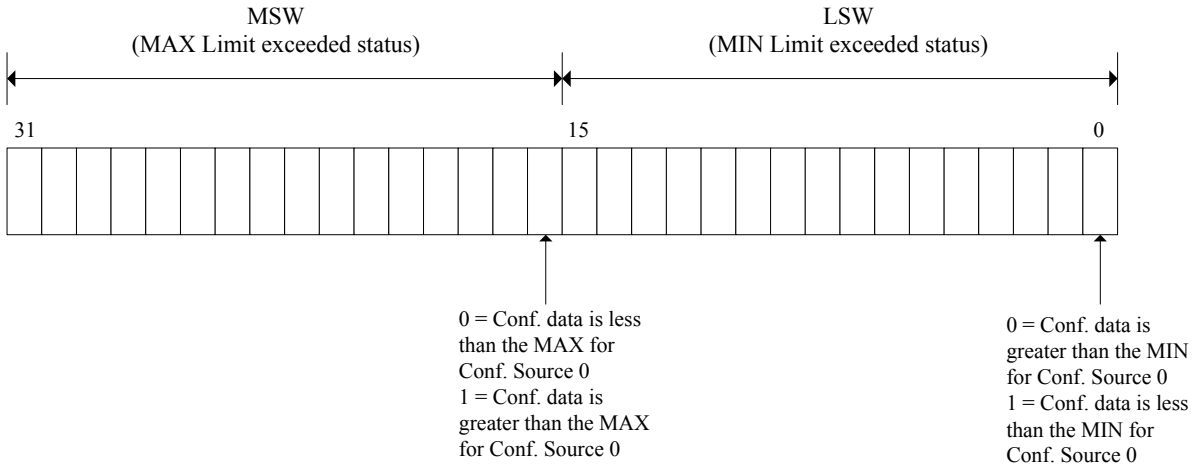
## DESCRIPTION

This function sets the minimum and maximum values for confidence data limit checking. Confidence limit checking mode is only valid for main bridge sampling frequencies less than 1 kHz. Returned values in the datapage correspond to the confidence channels for channels in the active scanlist. There exists a confidence limit check result summary field (shown in the diagram below) which indicates if any of the limits on all active confidence sources for a particular channel were exceeded or not. This is a 16-bit field, stored in the least-significant 16 bits of a UINT32 variable.




**Confidence Limit Check Result summary (Available per Analog Board)**

A detailed confidence limit check result (shown in the diagram below) is also available which returns two bits per channel per confidence source – that is, MAX Limit Exceeded and MIN Limit Exceeded, per channel, per confidence source. There is one UINT32 entry per bridge channel in the bridge scanlist. This UINT32 has two bits per confidence source (CONF_NUM_SRC sources per bridge channel), one to represent MAX limit exceeded and one to represent MIN limit exceeded. The MSW (lower CONF_NUM_SRC of the upper 16-bits) represent the MAX Limit Exceeded status for each of the CONF_NUM_SRC confidence sources, and the LSW (lower CONF_NUM_SRC bits of the lower 16-bits) represent the MIN Limit Exceeded status for each of the CONF_NUM_SRC confidence sources. Bit 0 represents the MIN Limit Exceeded status for source 0. Bit 16 represents the MAX Limit Exceeded status for source 0. Bit (CONF_NUM_SRC-1) represents the MIN Limit Exceeded status for source (CONF_NUM_SRC-1). Bit (16+CONF_NUM_SRC-1) represents the MAX Limit Exceeded status for source CONF_NUM_SRC. The rest of the sources follow the same pattern.




**Confidence Limit Check Detailed Result (Available per Bridge channel)**

| **NOTE** | The source-to-bit mapping is constant, regardless of confidence scanlist configuration. For example, whether or not sources 0 and 1 are enabled in the confidence scanlist, for instance, source 2's MIN Limit Exceeded Bit and MAX Limit Exceeded Bit are always bits 2 and 18, respectively. |
|---|---|

The confidence source mapping follows the same ordering as the source # define in vtex1629.h i.e. if sources 3, 8, and 10 are selected then they are reported in that order. Confidence sources that are not part of the confidence scanlist are not reported and will have their bit-fields set to 0.

Confidence values are reported at a maximum frequency of 500 Hz. This mode is supported up to 1 kHz sampling rate. Hence, at 1 kHz, every other packet will contain confidence information. The datapage size is 248 words when it has full confidence information i.e. confidence data and full limit check values, and is 24 words when it has no confidence information. Hence, the total data rate = ((248+24)/2)*4*8*1000 samples/second= 4.352 Mb/s.

**EXAMPLE**

```
ViSession instrumentHandle;
ViStatus status;

ViInt32 confSrcEnum = 0;
ViInt32 numChannels = MAX_NUMBER_OF_CHANNELS;
ViInt32 channels[MAX_NUMBER_OF_CHANNELS];
ViReal64 minArr[MAX_NUMBER_OF_CHANNELS];
ViReal64 maxArr[MAX_NUMBER_OF_CHANNELS];

int i = 0;

for(i = 0; i < MAX_NUMBER_OF_CHANNELS; i++) {
    channels[i] = i;
    minArr[i] = (-2.0);
    maxArr[i] = 4.0;
}

status = vtex1629_set_confidence_limit(instrumentHandle,
                                       confSrcEnum,
                                       numChannels,
                                       channels,
                                       minArr,
                                       maxArr);
```

# vtex1629_set_confidence_reporting_mode

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_confidence_reporting_mode (ViSession **vi**, ViInt32 **mode**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**mode** = sets the reporting mode for confidence limit checking. Valid input values: 0 through 2.

## DATA ITEM RESET VALUE

**mode** = VTEX1629_CONF_LIMIT_DISABLE_REPORT (0)

## DESCRIPTION

This function sets the reporting mode for confidence limit checking. Valid input values for the **mode** parameter are as follows:

| Decimal Value | Hex Value | #define Symbol | mode Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_CONF_LIMIT_DISABLE_REPORT | Reporting disabled |
| 1 | 0x01 | VTEX1629_CONF_LIMIT_SUMMARY_REPORT_ONLY | Summary report mode selected |
| 2 | 0x02 | VTEX1629_CONF_LIMIT_DETAILED_REPORT | Detailed report mode selected |

If set to VTEX1629_CONF_LIMIT_DISABLE_REPORT, the EX1629 will not collect confidence limit checking data. If set to VTEX1629_CONF_LIMIT_SUMMARY_REPORT_ONLY, an array will be created which indicates the channels that exceeded their limits. VTEX1629_CONF_LIMIT_DETAILED_REPORT, by contrast, provides an array that indicates if the minimum or maximum limit of a channel has been exceeded.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 mode = VTEX1629_CONF_LIMIT_SUMMARY_REPORT_ONLY;

status = vtex1629_set_confidence_reporting_mode(instrumentHandle, mode);
```

# vtex1629_set_dio_bank0_direction

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_dio_bank0_direction (ViSession **vi**, ViInt32 **direction**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**direction** = an integer input value indicating the desired direction of bank zero of the digital I/O. Valid input values: 0 or 1.

## DATA ITEM RESET VALUE

**direction** = 0 (input)

## DESCRIPTION

This function sets the direction of bank zero of the digital I/O as input or output. The **direction** parameter is defined as follows:

VTEX1629_DIO_DIRECTION_IN (0) = input VTEX1629_DIO_DIRECTION_OUT (1) = output

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_dio_bank0_direction(instrumentHandle,
                                          VTEX1629_DIO_DIRECTION_OUT);
```

# vtex1629_set_dio_bank0_pullup

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_dio_bank0_pullup (ViSession **vi**, ViInt32 **pullup**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**pullup** = an integer input value that indicates the desired pull-up mode for bank zero of the digital I/O. Valid input values: 0 or 1.

## DATA ITEM RESET VALUE

**pullup** = 0 (passive pull-up mode)

## DESCRIPTION

This function sets the pull-up mode for bank zero of the digital I/O to active or passive. The **pullup** parameter is defined as follows:

VTEX1629_PASIVE_PULLUP (0) = passive pull-up mode

VTEX1629_ACTIVE_PULLUP(1) = active pull-up mode

| **NOTE** | Active versus passive pullup applies only to banks that are in output mode. |
|---|---|

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_dio_bank0_pullup(instrumentHandle,
                                       VTEX1629_ACTIVE_PULLUP);
```

# vtex1629_set_dio_bank1_direction

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_dio_bank1_direction (ViSession **vi**, ViInt32 **direction**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**direction** = an integer input value indicating the desired direction of bank one of the digital I/O. Valid input values: 0 or 1.

## DATA ITEM RESET VALUE

**direction** = 0 (input)

## DESCRIPTION

This function sets the direction of bank one of the digital I/O as input or output. The **direction** parameter is defined as follows:

VTEX1629_DIO_DIRECTION_IN (0) = input VTEX1629_DIO_DIRECTION_OUT (1) = output

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_dio_bank1_direction(instrumentHandle,
                                          VTEX1629_DIO_DIRECTION_OUT);
```

# vtex1629_set_dio_bank1_pullup

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_dio_bank1_pullup (ViSession **vi**, ViInt32 **pullup**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**pullup** = an integer input value that indicates the desired pull-up mode for bank one of the digital I/O. Valid input values: 0 or 1.

## DATA ITEM RESET VALUE

**pullup** = 0 (passive pull-up mode)

## DESCRIPTION

This function sets the pull-up mode for bank one of the digital I/O to active or passive. The **pullup** parameter is defined as follows:

VTEX1629_PASIVE_PULLUP (0) = passive pull-up mode

VTEX1629_ACTIVE_PULLUP(1) = active pull-up mode

| **NOTE** | Active versus passive pullup applies only to banks that are in output mode. |
|---|---|

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_dio_bank1_pullup(instrumentHandle,
                                       VTEX1629_ACTIVE_PULLUP);
```

# vtex1629_set_dio_config_events

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_dio_config_events (ViSession **vi**, ViInt32 **inputLine**, ViInt32 **inputTrigType**, ViInt32 **numActions**, ViInt32 _VI_FAR **outputLineArr[]**, ViInt32 _VI_FAR **outputActionTypeArr[]**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**inputLine** = defines the DIO input line whose configuration is being queried. Valid input values: 0 through 7.
**inputTrigType** = defines the input trigger type. Valid input values: 0 (high-to-low transition) or 1 (low-to-high transition).
**numActions** = defines the size of the **outputLineArr[]** and **outputActionTypeArr[]** arrays. Valid input values: 0 through 8.
**outputLineArr[]** = an integer array containing a list of digital output lines that are affected by the **inputLine** and **inputTrigType** combination. Valid input values: 0 through 7.
**outputActionTypeArr[]** = an integer array containing a list of the output action that will occur based on the **inputLine** and **inputTrigType** parameters. Valid input values: 0 through 3.

## DATA ITEM RESET VALUE

**pullup** = 0 (passive pull-up mode)

## DESCRIPTION

This function sets the conditions under which DIO event transitions will occur.

The **numActions** parameter defines the size of both the **outputLineArr[]** and **outputActionTypeArr[]** arrays. Although any value 0 through 8 is acceptable, to avoid possible errors, it is recommended that this parameter be set to 8.

The **outputActionTypeArr[]** parameter is an array which contains a list of output actions that will occur based on events that occur on the specified **inputLine**. Note that each element of this array corresponds to the equivalent index in the **outputLineArr[]** parameter. For example, the action type at element *i* in this array corresponds to (i.e. will occur on) the line designated in element *i* of the **outputLineArr[]** array.

The DIO Event-Action items are saved in the configuration XML file as shown below:

```
<dioeventactions_0>
    <numActions> x </numActions>
    <inputLine> x </inputLine>
    <inputTriggerType> x </inputTriggerType>
    <outputLine_0> x </outputLine_0>
    :
    <outputLine_7> x </outputLine_7>
</dioeventactions_0>
:
<dioeventactions_15>
:
</dioeventactions_15>
```

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status = VI_SUCCESS;

ViInt32 inputLine = 0;
ViInt32 inputTrigType = 0;
ViInt32 numActions = MAX_DIO_CHANNELS;
```

```
ViInt32 outputLineArr[MAX_DIO_CHANNELS];
ViInt32 outputActionTypeArr[MAX_DIO_CHANNELS];

ViInt32 i = 0;

for(i = 0; i < MAX_DIO_CHANNELS; i++) {
    outputLineArr[i] = i;
    outputActionTypeArr[i] = 0;
}


status = vtex1629_set_dio_config_events(instrumentHandle,
                                        inputLine,
                                        inputTrigType,
                                        numActions,
                                        outputLineArr,
                                        outputActionTypeArr);
```

# vtex1629_set_dio_output

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_dio_output (ViSession **vi**, ViInt32 **dioOut**);

## FUNCTION PARAMETERS

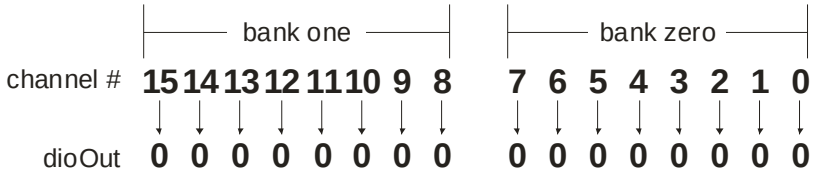
**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**dioOut** = an integer input value indicating the programmed output state of the digital I/O. See the *Description* below for more information concerning this parameter. Valid input values: 0 to 65535.

## DATA ITEM RESET VALUE

**dioOut** = 0

## DESCRIPTION

This function sets the programmed output state for both digital I/O banks. The **dioOut** parameter is an integer value that represents the desired state of the digital I/O. This binary value is constructed through the assignment of the eight most significant bits to the eight channels of bank one (channels 8-15) and the eight least significant bits to the eight channels of bank zero (channels 0-7). This is illustrated below.




For example, if a user wants to configure digital I/O channels 2 and 3 of bank zero and digital I/O channels 14 and 15 of bank one as high outputs, then **dioOut** should be set to the following:

**dioOut** = 11000000 00001100b → 0xC00C → 49164

| **NOTE** | The control of the digital I/O programmed output state and its direction are disjoint operations. Thus, the setting of a nonzero output state only affects its actual state if the direction of the appropriate bank is set to output. This is done with the vtex1629_set_dio_bank0_direction and vtex1629_set_dio_bank1_direction functions. |
|---|---|

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_dio_output(instrumentHandle, 0x0000C00C);
```

# vtex1629_set_EU_conversion

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_EU_conversion (ViSession **vi**, ViInt32 _VI_FAR **channels[]**, ViInt32 **numberOfChannels**, ViInt32 **EUConversionType**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the EU conversion type will be set. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**EUConversionType** = an integer input value indicating the desired type of EU conversion for the indicated channels. See *Description* below for more information. Valid input values: 0 to 10.

## DATA ITEM RESET VALUE

**EUConversionType** = 0 (voltage)

## DESCRIPTION

This function sets the EU conversion type for a given list of channels. The **EUConversionType** parameter values correspond to the following conversion types:

| Decimal Value | Hex Value | #define Symbol | EU ConversionType Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_EUCONV_VOLT_OUTPUT | Voltage |
| 1 | 0x01 | VTEX1629_EUCONV_QTR_BRIDGE_120 | Quarter-Bridge 120 |
| 2 | 0x02 | VTEX1629_EUCONV_QTR_BRIDGE_350 | Quarter-Bridge 350 |
| 3 | 0x03 | VTEX1629_EUCONV_QTR_BRIDGE_USER | Quarter-Bridge User |
| 4 | 0x04 | VTEX1629_EUCONV_HALF_BRIDGE_BEND | Half-Bridge Bending |
| 5 | 0x05 | VTEX1629_EUCONV_HALF_BRIDGE_POIS | Half-Bridge Poisson |
| 6 | 0x06 | VTEX1629_EUCONV_FULL_BRIDGE_BEND | Full-Bridge Bending |
| 7 | 0x07 | VTEX1629_EUCONV_FULL_BRIDGE_POIS | Full-Bridge Poisson |
| 8 | 0x08 | VTEX1629_EUCONV_FULL_BRIDGE_BPOIS | Full-Bridge Bending Poisson |
| 9 | 0x09 | VTEX1629_EUCONV_RATIOMETRIC | Ratiometric |
| 10 | 0x0A | VTEX1629_EUCONV_LINEAR | Linear |

Setting the EU Conversion for a channel automatically configures the completion resistor and input multiplexer for the most common usage of the specified EU Conversion. See the *Engineering Unit (EU)* Conversion section in Section 3 for more details.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0, 1, 2, 3, 4, 5, 6, 7};
ViInt32 numberOfChannels = 8;
…
/*
  configure first 8 channels for quarter bridge, 120 ohm mode
*/

status = vtex1629_set_EU_conversion(instrumentHandle,
                                    channels,
                                    numberOfChannels,
                                    VTEX1629_EUCONV_QTR_BRIDGE_120);
```

# vtex1629_set_euconv_dynamic_excitation_enabled

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_euconv_dynamic_excitation_enabled (ViSession **vi**, ViInt32 **channnelsArraySize**, ViInt32 _VI_FAR **channels[]**, ViBoolean **enabled**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channelsArraySize** = the size of the **channel[]** array. Valid input values: VTEX1629_MIN_SCANLIST_LENGTH (1) to VTEX1629_MAX_SCANLIST_LENGTH (48).
**channel[]** = an integer input array that specifies the channel for which the configuration will be returned. Valid input values: 0 to 47.
**enabled** = a Boolean input value indicating whether dynamic excitation EU is enabled on the listed channels. If enabled equals VI_TRUE, then dynamic excitation EU will be enabled on the channels listed in the **channels[]** array.

## DATA ITEM RESET VALUE

**enabled** = VI_TRUE (1)

## DESCRIPTION

This function sets the dynamic excitation EU conversion state. In this mode of operation, the EX1629 uses the excitation voltage measured by the confidence ADC (in real time) in its calculations. This mode is available for bridge sampling frequencies ($f_s$) less than 1 kHz. While in this mode, it is advised to give the confidence filters at least 1.5 s to settle, from the time the confidence source for excitation voltage is enabled or the excitation value is changed.

The vtex1629_measure_excitation_voltage interface uses the confidence subsystem to measure the excitation voltage. It returns these voltages to the calling function. Optionally, it can update the excitation voltage value used for the strain EU conversion. The dynamic excitation EU conversion is slightly different. It is a mode of operation that essentially does the same operations as vtex1629_measure_excitation_voltage, measuring the excitation voltage using the confidence subsystem and updating the excitation voltage value used in the EU conversion in real-time.

This is a Boolean mode of operation, selectable per channel. If the user enables this mode, the set excitation voltage EU function should return an error (users should not be able to manually set the excitation voltage EU value when in this automatic mode). If the user queries the excitation voltage EU value, the result is the latest, real-time value.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channelsArraySize = MAX_CHANNELS;
ViInt32 channels[MAX_CHANNELS];
ViBoolean set_enabled = VI_TRUE;
…
status = vtex1629_set_euconv_dynamic_excitation_enabled(vi,
                                                        channelsArraySize,
                                                        channels,
                                                        set_enabled);
```

# vtex1629_set_euconv_excitation

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_euconv_excitation (ViSession **vi**, ViInt32 _VI_FAR **channels[]**,ViInt32 **numberOfChannels**, ViReal64 **euConversionVoltage**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the EU conversion excitation voltage will be set. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**euConversionVoltage** = a real input value, in volts, that indicates the value that should be used in EU conversions for the excitation voltage. Valid input values: 0.00000 to +16.00000.

## DATA ITEM RESET VALUE

**euConversionVoltage** = 0.000000

## DESCRIPTION

This function sets the excitation voltage to be used in EU conversions for a particular list of channels.

| | |
|---|---|
| **NOTE** | The conventional method of providing a non-nominal value of the excitation voltage to the EU conversion is to conduct an excitation voltage measurement using the vtex1629_measure_excitation_voltage function. This function provides a manual method that is normally only used for system diagnostic purposes. |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0};
ViInt32 numberOfChannels = 1;
…
status = vtex1629_set_euconv_excitation(instrumentHandle,
                                        channels,
                                        numberOfChannels,
                                        2.0);
```

# vtex1629_set_excitation

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_excitation (ViSession **vi**, ViInt32 _VI_FAR **channels[]**, ViInt32 **numberOfChannels**, ViReal64 **positiveExcitationVoltage**, ViReal64 **negativeExcitationVoltage**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the programmed excitation voltage will be set. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**positiveExcitationVoltage** = a real input value, in volts, indicating the programmed positive excitation voltage. Valid input values: 0.000000 through 8.000000.
**negativeExcitationVoltage** = a real input value, in volts, indicating the programmed negative excitation voltage. Valid input values: -8.000000 through 0.000000.

## DATA ITEM RESET VALUE

**positiveExcitationVoltage** = 0.000000 **negativeExcitationVoltage** = 0.000000

## DESCRIPTION

This function sets the programmed excitation voltages for a given list of channels. The excitation value is quantized with a 14-bit DAC. Hence, the actual value can be queried with the vtex1629_get_excitation function. Additionally, the accuracy specifications (*EX1629 Specifications*) for the instrument require that the excitation voltage be measured (see vtex1629_measure_excitation_voltage) prior to taking strain gage readings.

Providing separate positive and negative excitation supply control permits the mid-point of each half of the bridge to be at a voltage other than 0. This is achieved by using asymmetric excitation voltages (e.g., +5.0 V and -1.0 V will produce 2.0 V at the mid-point of each half of the bridge.

**NOTES**

**1)** Due to hardware limitations, the setting of excitation voltages with a magnitude less than 100 mV may be imprecise. However, since the excitation voltage must be measured and the EU conversion variables updated prior to taking strain measurements, per the instrument specifications (*EX1629 Specifications*), this imprecision does not effect the accuracy of the strain gage readings.

**2)** The control of the excitation voltage values and their enabling are separate operations. Thus, setting a non-zero value for either parameter does not guarantee that the excitation source is enabled. That must be set with the vtex1629_set_excitation_enabled function.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0};
ViInt32 numberOfChannels = 1;
…
status = vtex1629_set excitation(instrumentHandle,
                                 channels,
                                 numberOfChannels,
                                 2.0,
                                 -2.0);
```

# vtex1629_set_excitation_enabled

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_excitation_enabled (ViSession **vi**, ViInt32 _VI_FAR **channels[]**, ViInt32 **numberOfChannels**, ViBoolean **enabled**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the excitation voltage will be enabled or disabled. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**enabled** = a Boolean input value which controls the enabling or disabling of the excitation voltage for a given list of channels. Valid input values: 0 or 1.

## DATA ITEM RESET VALUE

**enabled** = 0

## DESCRIPTION

This function enables or disables the excitation voltages for a list of channels. Setting the **enabled** parameter to VI_TRUE (1) enables excitation voltages, while setting it to VI_FALSE (0) disables excitation voltages. An excitation source that is not enabled will output 0 V, regardless of its programmed value (please see Note 1 of the vtex1629_set_excitation function for information regarding excitation precision).

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0};
ViInt32 numberOfChannels = 1;
…
status = vtex1629_set_excitation_enabled(instrumentHandle,
                                         channels,
                                         numberOfChannels,
                                         VI_TRUE);
```

# vtex1629_set_gain

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_gain (ViSession **vi**, ViInt32 _VI_FAR **channels[]**, ViInt32 **numberOfChannels**, ViReal64 **gain**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the gain will be set. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**gain** = a real input value indicating the desired gain value for the given set of channels. Valid input values: 1.00, 10.0, or 100.0.

## DATA ITEM RESET VALUE

**gain** = 1.0

## DESCRIPTION

This function sets the signal conditioning gain for a given list of channels.

| Decimal Value | Hex Value | #define Symbol | gain Description |
|---|---|---|---|
| 1 | 0x00 | VTEX1629_GAIN_ONE | 1 |
| 10 | 0x0A | VTEX1629_GAIN_TEN | 10 |
| 100 | 0x64 | VTEX1629_GAIN_HUNDRED | 100 |

| NOTE | While defined as a real parameter, **gain** has only three valid values, corresponding to discrete hardware gain configurations. The gain cannot be arbitrarily set. |
|---|---|

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0};
ViInt32 numberOfChannels = 1;
…
status = vtex1629_set_gain(instrumentHandle,
                           channels,
                           numberOfChannels,
                           VTEX1629_GAIN_HUNDRED);
```

# vtex1629_set_gauge_factor

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_gauge_factor (ViSession **vi**, ViInt32 _VI_FAR **channels[]**,ViInt32 **numberOfChannels**, ViReal64 **gageFactor**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the gage factor will be set. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**gageFactor** = a real input value indicating the desired gage factor.

## DATA ITEM RESET VALUE

**gageFactor** = 2.000000

## DESCRIPTION

This function sets the gage factor for a list of channels. This is one of the parameters used in EU conversion calculations.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0};
ViInt32 numberOfChannels = 1;
…
status = vtex1629_set_gauge_factor(instrumentHandle,
                                   channels,
                                   numberOfChannels,
                                   2.01);
…
```

# vtex1629_set_half_bridge_lead_wire_desensitization

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_half_bridge_lead_wire_desensitization (ViSession **vi**, ViInt32 **channelsArraySize**, ViInt32 _VI_FAR **channels[]**, ViReal64 _VI_FAR **factor[]**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channelsArraySize** = the size of the **channel[]** array. Valid input values: VTEX1629_MIN_SCANLIST_LENGTH (1) to VTEX1629_MAX_SCANLIST_LENGTH (48).
**channel[]** = an integer input array that specifies the channels to which the **factor** parameter will apply. Valid input values: 0 to 47.
**factor** = an input array of values which set the desensitization error for the channels indicated in the **channels[]** array. Valid input values are number greater than 1.

## DATA ITEM RESET VALUE

**factor** = 1

## DESCRIPTION

This function sets the lead wire desensitization factor for a given list of channels. The **factor** parameter is defined as follows:

$$factor = 1 + \frac{R_{lead}}{R_{gage}}$$

where $R_{lead}$ represents the resistance of the lead and $R_{gage}$ is the resistance of the strain gage.

| NOTE | Early EX1629s do not have hardware which supports this functionality. Implementation of direct lead wire measurement is not possible on first generation units. |
|---|---|

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channelsArraySize = MAX_CHANNELS;
ViInt32 channels[MAX_CHANNELS];
ViReal64 factor[MAX_CHANNELS];

ViInt32 i = 0;

for(i = 0; i < channelsArraySize; i++) {
    channels[i] = i;
}


status = vtex1629_set_half_bridge_lead_wire_desensitization(vi,
                                                            channelsArraySize,
                                                            channels,
                                                            factor);
```

# vtex1629_set_IIR_filter_configuration

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_IIR_filter_configuration (ViSession **vi**, ViInt32 _VI_FAR **channels[]**,ViInt32 **numberOfChannels**, ViInt32 **filterType**, ViReal64 **cutoffFreq**, ViInt32 **transform**, ViInt32 **filterOrder**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the IIR filter configuration will be set. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**filterType** = an integer input value indicating the type of filter to be used for the indicated channels. See *Description* below for more information. Valid input values: 0, 1, or 2.
**cutoffFreq** = a real input value indicating the filter's cutoff frequency in hertz (Hz), which is only relevant for Bessel and Butterworth filter types. See *Description* below for more information.
**transform** = an integer input value indicating the type of filter transform to be employed , which is only relevant for Bessel and Butterworth filter types. Valid input values: 0 or 1.
**filterOrder** = an integer input value indicating the order of this filter. Valid input values: 1 to 10 for the Bessel filter, 0 to 10 for the Butterworth filter.

## DATA ITEM RESET VALUE

**filterType** = 1 (Butterworth)
**transform** = 0 (bilinear)
**cutoffFreq** = 10
**filterOrder** = 6

## DESCRIPTION

This function configures the IIR filters for a given list of channels. The **filterType** parameter has three allowed values:

| Decimal Value | Hex Value | #define Symbol | filterType Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_IIR_FILT_NONE | None |
| 1 | 0x01 | VTEX1629_IIR_FILT_BUTTERWORTH | Butterworth |
| 2 | 0x02 | VTEX1629_IIR_FILT_BESSEL | Bessel |

The **cutoffFreq** parameter defines the cutoff (-3 dB) frequency for the low-pass filter indicated above. The EX1629 will locate this parameter in the range [$f_s$/1000, $f_c$ max] (see Table B-1), where $f_s$ is the current sampling frequency. Note that this value can change if the sampling frequency is altered. The actual value can be queried with the vtex1629_get_IIR_filter_configuration function.

The **transform** parameter provides for two modes of transformation:

| Decimal Value | Hex Value | #define Symbol | transform Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_TRANSFORM_BILINEAR | Bilinear |
| 1 | 0x01 | VTEX1629_TRANSFORM_MATCHEDZ | Matched-Z |

The **filterOrder** parameter defines the desired order of the filter. When the **filterType** is set to Butterworth, there is an additional option of 0. This corresponds to an automatic option, whereby the EX1629 will assign an order based on an analog prototype Butterworth design given the sampling frequency, cutoff frequency, and a -200 dB attenuation at the Nyquist frequency. The actual order can be determined using the vtex1629_get_IIR_filter_configuration function.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0};
ViInt32 numberOfChannels = 1;
…
/*
configure IIR filtering on channel 0, with a 5th order  Bessel filter with a cutoff
frequency of 50Hz, using the bilinear transform
*/
status = vtex1629_set_IIR_filter_configuration (instrumentHandle,
                                                channels,
                                                numberOfChannels,
                                                VTEX1629_IIR_FILT_BESSEL,
                                                50.0,
                                                VTEX1629_TRANSFORM_BILINEAR,
                                                5);

…
// disable IIR filtering on channel 0
status = vtex1629_set_IIR_filter_configuration (instrumentHandle,
                                                channels,
                                                numberOfChannels,
                                                VTEX1629_IIR_FILT_NONE,
                                                0.0,
                                                VTEX1629_TRANSFORM_BILINEAR,
                                                0);
```

# vtex1629_set_input_multiplexer

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_input_multiplexer (ViSession **vi**, ViInt32 _VI_FAR **channels[]**, ViInt32 **numberOfChannels**, ViInt32 **muxInValue**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels[]** = an input integer array containing a list of channel numbers for which the linear scaling coefficients will be set. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**muxInValue** = the input multiplexer source. Valid input values: 0 to 4

## DATA ITEM RESET VALUE

**muxInValue** = 0

## DESCRIPTION

This function sets the input multiplexer source. The **muxInValue** parameter can be set to the following values:

| Decimal Value | Hex Value | #define Symbol | muxInValue Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_INPUTMUX_BRIDGE_TYPE_FULL | Full Bridge |
| 1 | 0x01 | VTEX1629_INPUTMUX_BRIDGE_TYPE_HALF | Half Bridge |
| 2 | 0x02 | VTEX1629_INPUTMUX_BRIDGE_TYPE_QUARTER | Quarter Bridge |
| 3 | 0x03 | VTEX1629_INPUTMUX_BRIDGE_TYPE_CAL | Cal |
| 4 | 0x04 | VTEX1629_INPUTMUX_BRIDGE_TYPE_GND | Gnd |

| NOTE | This function provides a manual method that is normally only used for specialized cases like doing voltage measurements in quarter bridge mode. The vtex1629_set_EU_conversion command is the preferred way of setting up the input mux, completion resistors, and the EU conversion. |
|---|---|

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0};
ViInt32 numberOfChannels = 1;
…
/*
configure input multiplexer for channel 0 to be in quarter bridge mode
*/
status = vtex1629_set_input_multiplexer (instrumentHandle,
                                         channels,
                                         numberOfChannels,
                                         VTEX1629_INPUTMUX_BRIDGE_TYPE_QUARTER);
```

# vtex1629_set_lead_wire_resistance

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_lead_wire_resistance (ViSession **vi**, ViInt32 **channelsArraySize**, ViInt32 _VI_FAR **channels[]**, ViReal64 **resistance**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channelsArraySize** = the size of the **channel[]** array. Valid input values: VTEX1629_MIN_SCANLIST_LENGTH (1) to VTEX1629_MAX_SCANLIST_LENGTH (48).
**channel[]** = an integer input array that specifies the channels to which the **factor** parameter will apply. Valid input values: 0 to 47.
**resistance** = sets the lead wire resistance value. Valid input values are numbers greater than 0.

## DATA ITEM RESET VALUE

**resistance** = 0.

## DESCRIPTION

This function sets the resistance of the lead wire.

| | |
|---|---|
| **NOTE** | Early EX1629s do not have hardware which supports this functionality. Implementation of direct lead wire measurement is not possible on first generation units. |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channelsArraySize = MAX_CHANNELS;
ViInt32 channels[MAX_CHANNELS];
ViReal64 resistance;

ViInt32 i = 0;

for(i = 0; i < channelsArraySize; i++) {
    channels[i] = i;
}

memset(resistance, 0x00, sizeof(resistance));

status = vtex1629_set_lead_wire_resistance(vi,
                                           channelsArraySize,
                                           channels,
                                           resistance);
```

# vtex1629_set_linearscaling_configuration

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_linearscaling_configuration (ViSession **vi**, ViInt32 _VI_FAR **channels[]**,ViInt32 **numberOfChannels**, ViReal64 **m**, ViReal64 **b**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the linear scaling coefficients will be set. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**m** = a real input value indicating the gain factor, *m*, in the linear equation y = *m*x + *b*.
**b** = a real input value indicating the offset value, *b*, in the linear equation: y = *m*x + *b*.

## DATA ITEM RESET VALUE

**m** = 2.000000 **b** = 0.000000

## DESCRIPTION

This function sets the slope (**m**) and intercept (**b**) parameters for a channel when configured for linear EU conversion (*x* being in volts).

| | |
|---|---|
| **NOTE** | The **m** parameter (slope) is stored in the same location as the gage factor and the b parameter (intercept) is stored in the same location as the unstrained voltage. Since the linear scaling EU conversion and the strain EU conversions are mutually exclusive, this is never an issue in practice. |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {24};
…
status = vtex1629_set_linearscaling_configuration(instrumentHandle,
                                                  channels,
                                                  1,
                                                  1.0,
                                                  0.0);
```

# vtex1629_set_lxibus_configuration

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_lxibus_configuration (ViSession **vi**, ViInt32 **lxiLine**, ViInt32 **inOut**, ViInt32 **transmissionScope**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**lxiLine** = an integer input value that indicates which LXI Trigger Bus channel the function will configure. Valid input values: 0 to 7.
**inOut** = an integer input value that indicates whether the specified LXI Trigger Bus channel will be configured for input or output. Valid input values: 0 or 1.
**transmissionScope** = an integer input value indicating whether transmissions on the specified channel will be input from an output to the external LXI bus, or will be kept internal to the EX1629. Valid input values: 0 or 1.

## DATA ITEM RESET VALUE

**inOut** = 0 **transmissionScope** = 0

## DESCRIPTION

This function configures several characteristics of a specific LXI Trigger Bus channel. Specifically, it configures channel direction and the scope of transmissions.

| Decimal Value | Hex Value | #define Symbol | lxiLine Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_LXI_LINE_ZERO | LXI LINE 0 |
| 1 | 0x01 | VTEX1629_LXI_LINE_ONE | LXI LINE 1 |
| 2 | 0x02 | VTEX1629_LXI_LINE_TWO | LXI LINE 2 |
| 3 | 0x03 | VTEX1629_LXI_LINE_THREE | LXI LINE 3 |
| 4 | 0x04 | VTEX1629_LXI_LINE_FOUR | LXI LINE 4 |
| 5 | 0x05 | VTEX1629_LXI_LINE_FIVE | LXI LINE 5 |
| 6 | 0x06 | VTEX1629_LXI_LINE_SIX | LXI LINE 6 |
| 7 | 0x07 | VTEX1629_LXI_LINE_SEVEN | LXI LINE 7 |

The **inOut** parameter configures the direction of the LXI Trigger Bus channel. The permissible values indicate the following:

| Decimal Value | Hex Value | #define Symbol | inOut Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_LXI_INPUT | Input |
| 1 | 0x01 | VTEX1629_LXI_OUTPUT | Output |

The **transmissionScope** parameter indicates whether the specified LXI channel is configured to communicate with other devices on the external LXI bus or simply configured to communicate on the internal LXI bus. In the case of an output, the **transmissionScope** indicates whether the output will be driven out onto the external LXI bus in addition to being driven on the internal LXI bus. In the case of an input, the **transmissionScope** determines if the input is read from the external bus or read from the internal bus. The permissible values indicate the following:

| Decimal Value | Hex Value | #define Symbol | transmissionScope Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_LXI_INTERNAL | LXI bus signal routed internally |
| 1 | 0x01 | VTEX1629_LXI_INTERNAL_EXTERNAL | LXI bus signal routed internally and externally |

## EXAMPLE

```
ViSession master_instrumentHandle;
ViSession slave_instrumentHandle;
ViStatus status;
…
/*
  portion of configuring a master instrument for distribution of clock and sync
signals (LXI0, LXI1) for a master/slave configuration */

status = vtex1629_set_lxibus_configuration(master_instrumentHandle,
                                           VTEX1629_LXI_LINE_ZERO,
                                           VTEX1629_LXI_OUTPUT,
                                           VTEX1629_LXI_INTERNAL_EXTERNAL);

status = vtex1629_set_lxibus_configuration(master_instrumentHandle,
                                           VTEX1629_LXI_LINE_ONE,
                                           VTEX1629_LXI_OUTPUT,
                                           VTEX1629_LXI_INTERNAL_EXTERNAL);


/*  portion of configuring a slave instrument for distribution of clock and sync
signals (LXI0, LXI1) for a master/slave configuration */

status = vtex1629_set_lxibus_configuration(slave_instrumentHandle,
                                           VTEX1629_LXI_LINE_ZERO,
                                           VTEX1629_LXI_INPUT,
                                           VTEX1629_LXI_INTERNAL_EXTERNAL);

status = vtex1629_set_lxibus_configuration(slave_instrumentHandle,
                                           VTEX1629_LXI_LINE_ONE,
                                           VTEX1629_LXI_INPUT,
                                           VTEX1629_LXI_INTERNAL_EXTERNAL);
```

# vtex1629_set_lxibus_output

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_lxibus_output (ViSession **vi**, ViInt32 **output**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**output** = an integer input value that configures the output state of each LXI Trigger Bus channel. Valid input values: 0 to 255.

## DATA ITEM RESET VALUE

**output** = 0

## DESCRIPTION

This function configures the output state of each of the LXI Trigger Bus channels. This function only sets what will be output by the bus, but does not actually enable the outputs. See vtex1629_set_lxibus_configuration for information on enabling the output.

The **output** parameter is an 8-bit integer where the least significant bit of the integer corresponds to LXI Trigger Bus channel zero and the most significant bit corresponds to LXI Trigger Bus channel seven. For example, if a user wants to configure the LXI Trigger Bus to output high signals on channels zero and seven and low on all the rest of the LXI Trigger Bus channels, then this corresponds to the 8-bit number 10000001b → 0x81→ 129.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_lxibus_output(instrumentHandle, 0x81);
```

# vtex1629_set_pattern_arm_configuration

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_pattern_arm_configuration (ViSession **vi**, ViInt16 _VI_FAR **lxiTrigLines[]**, ViInt16 _VI_FAR **dioLines[]**, ViBoolean **timer**, ViInt32 **lxiOutput**, ViInt32 **lxiInput**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**lxiTrigLines[]** = an integer input array specifying the LXI Trigger Bus channel states that will be accepted as arm events. This includes both levels (high and low) and edges (rising and falling). Valid input values: 0 to 255.
**dioLines** = an integer input array specifying the digital I/O channels that will be accepted as arm events. This includes both levels (high and low) and edges (rising and falling). Valid input values: 0 to 255.
**timer** = a Boolean input value that indicates whether the EX1629 will generate ARM events based on the internal timer. Setting this parameter to VI_TRUE (1) will cause arm events to be generated. Valid input values: VI_FALSE (0) or VI_TRUE(1).
**lxiOutput** = this parameter specifies which LXI Trigger Bus line will be used to output the ARM event signals. Valid input values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN, LXI_NONE.
**lxiInput** = this parameter specifies which LXI Trigger Bus line will be used to input the ARM event signals. Valid input values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN, LXI_NONE.

## DATA ITEM RESET VALUE

**lxiTrigLines[]** = [0, 0, 0, 0]
**timer** = 0
**lxiInput** = 8 (LXI_NONE)
**dioLines** = [0, 0, 0, 0]
**lxiOutput**= 8 (LXI_NONE)

## DESCRIPTION

This function configures the EX1629's pattern arm operation mode. This mode allows the EX1629 to accept arm events from multiple sources. While in pattern mode, the instrument can accept arm events from the LXI Trigger Bus, digital I/O bus, internal timer, and the software ARM command. There is no need to explicitly enable the software arm source – it is always available for use while in pattern arm mode.

The **lxiTrigLines[]** parameter is an array of four elements with each array element being an unsigned 8-bit integer. Each bit of this integer corresponds to an LXI Trigger Bus channel. Specifically, the least significant bit corresponds to LXI Trigger Bus channel zero and the most significant bit corresponds to LXI Trigger Bus channel seven. Each element specifies which events the EX1629 will accept as arm events on the LXI Trigger Bus for different edges or states. If a user wants to specify a channel to for arm events, the corresponding bit should be set to "1". The individual array elements specify the following:

0 = lxiTrigLines (Positive Edge)
1 = lxiTrigLines (Negative Edge)
2 = lxiTrigLines (Positive Level)
3 = lxiTrigLines (Negative Level)

For example, if a user wishes to arm the EX1629 on a negative edge signal coming into the LXI Trigger Bus on channel 0 and a positive level on channels 3 and 6, then: **lxiTrigLines[1]** = 00000001b = 0x01 = 1 and **lxiTrigLines[2]** = 01001000b = 0x48 = 72.

The **dioLines** parameter is an array of four elements with each array element being an unsigned 16-bit integer. Each bit of this integer corresponds to a digital I/O channel. Specifically, the least significant bit corresponds to a digital I/O channel zero, and the most significant bit corresponds to digital I/O channel seven. Each element specifies which events the EX1629 will accept as arm events on the digital I/O bus for different clock edges or states. If a user wants to specify a channel to for arm events, the corresponding bit should be set to "1". Specifically, the individual array elements specify the following:

0 = dioLines[0] Positive Edge
1 = dioLines[1] Negative Edge
2 = dioLines[2] Positive Level
3 = dioLines[3] Negative Level

For example, if a user wishes to arm the EX1629 on a negative edge signal coming into the digital I/O bus on channels 0 and 3, then: **dioLines[1]** = 00001001b = 0x0009 = 9.

With regard to the **lxiOutput** parameter, it is important to note that since the EX1629 can simultaneously accept arm events from multiple sources, it is necessary to reserve one of the LXI Trigger Bus line to communicate these events within the device and to other devices in a multi-device configuration. If the EX1629 is a master driving arm to peripheral slaves, the **lxiOutput** parameter specifies the LXI Trigger Bus line that will be used to communicate the ARM event to the slave devices. It is also necessary to configure this LXI Trigger Bus line to be used as an output (see vtex1629_set_lxibus_configuration).

The **lxiInput** parameter specifies which trigger bus line the master device uses for its arm events. Although this parameter is often set to the same value as **lxiOutput**, there are cases where it might be set to a different value. For instance, when the device is configured as a master device in a star multi-box configuration, the master might output the ARM event on LXI2 and input it back in on LXI6.

| Decimal Value | Hex Value | #define Symbol | inLine/outLine Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_LXI_LINE_ZERO | LXI LINE 0 |
| 1 | 0x01 | VTEX1629_LXI_LINE_ONE | LXI LINE 1 |
| 2 | 0x02 | VTEX1629_LXI_LINE_TWO | LXI LINE 2 |
| 3 | 0x03 | VTEX1629_LXI_LINE_THREE | LXI LINE 3 |
| 4 | 0x04 | VTEX1629_LXI_LINE_FOUR | LXI LINE 4 |
| 5 | 0x05 | VTEX1629_LXI_LINE_FIVE | LXI LINE 5 |
| 6 | 0x06 | VTEX1629_LXI_LINE_SIX | LXI LINE 6 |
| 7 | 0x07 | VTEX1629_LXI_LINE_SEVEN | LXI LINE 7 |
| 8 | 0x08 | VTEX1629_LXI_NONE | None |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt16 triglines[VTEX1629_MAX_LINES];
ViInt16 diolines[VTEX1629_MAX_LINES];
…

triglines[0] = 0x00;
triglines[1] = 0x01;
triglines[2] = 0x48;
triglines[3] = 0x00;

diolines[0] = 0x0000;
diolines[1] = 0x0009;
diolines[2] = 0x0000;
diolines[3] = 0x0000;

status = vtex1629 vtex1629_set_pattern_arm_configuration (instrumentHandle,
                                                          triglines,
                                                          diolines,
                                                          VI_FALSE,
                                                          VTEX1629_LXI_LINE_TWO,
                                                          VTEX1629_LXI_LINE_TWO);
```

# vtex1629_set_pattern_trig_configuration

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_pattern_trig_configuration (ViSession **vi**, ViInt16 _VI_FAR **lxiTrigLines[]**, ViInt16 _VI_FAR **dioLines[]**, ViBoolean **timer**, ViInt32 **lxiOutput**, ViInt32 **lxiInput**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**lxiTrigLines[]** = an integer input array specifying the LXI Trigger Bus channel states that will be accepted as trigger events. This includes both levels (high and low) and edges (rising and falling). Valid input values: 0 to 255.
**dioLines** = an integer input array specifying the digital I/O channels that will be accepted as trigger events. This includes both levels (high and low) and edges (rising and falling). Valid input values: 0 to 255.
**timer** = a Boolean input value that indicates whether the EX1629 will generate TRIG events based on the internal timer. Setting this parameter to VI_TRUE(1) will cause TRIG events to be generated. Valid input values: VI_FALSE(0) or VI_TRUE(1).
**lxiOutput** = this parameter specifies which LXI Trigger Bus line will be used to output the trigger event signals. Valid input values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN, LXI_NONE.
**lxiInput** = this parameter specifies which LXI Trigger Bus line will be used to input the trigger event signals. Valid input values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN, LXI_NONE.

## DATA ITEM RESET VALUE

**lxiTrigLines[]** = [0, 0 ,0 ,0]
**dioLines** = [0, 0 ,0 ,0]
**timer** = 0
**lxiOutput** = LXI_NONE
**lxiInput** = LXI_NONE

## DESCRIPTION

This function configures the EX1629's pattern trigger mode of operation. This mode allows the EX1629 to accept trigger events from multiple sources. While in pattern mode, the instrument can accept trigger events from the LXI Trigger Bus, digital I/O bus, internal timer, and the software trigger command. There is no need to explicitly enable the software trigger source – it is always available for use while in pattern trigger mode. All of the conditions specified must be met for a trigger event to be generated. If multiple conditions are specified for the same LXI or DIO line, any of the conditions for that line can be met to satisfy the trigger pattern requirements for that line.

The **lxiTrigLines[]** parameter is an array of four elements with each array element being an unsigned 8-bit integer. Each bit of this integer corresponds to an LXI Trigger Bus channel. Specifically, the least significant bit corresponds to LXI Trigger Bus channel zero and the most significant bit corresponds to LXI Trigger Bus channel seven. Each element specifies which events the EX1629 will accept as trigger events on the LXI Trigger Bus for different edges or states. If a user wants to specify a channel to for trigger events, the corresponding bit should be set to "1". The individual array elements specify the following:

0 = lxiTrigLines (Positive Edge)
1 = lxiTrigLines (Negative Edge)
2 = lxiTrigLines (Positive Level)
3 = lxiTrigLines (Negative Level)

For example, if a user wishes to trigger the EX1629 on a negative edge signal coming into the LXI Trigger Bus on channel 0 and a positive level on channels 3 and 6, then: **lxiTrigLines[1]** = 00000001b = 0x01 = 1 and **lxiTrigLines[2]** = 01001000b = 0x48 = 72.

The **dioLines** parameter is an array of four elements with each array element being an unsigned 8-bit integer. Each bit of this integer corresponds to a digital I/O channel. Specifically, the least significant bit corresponds to a digital I/O channel zero, and the most significant bit corresponds to digital I/O channel seven. Each element specifies which events the EX1629 will accept as trigger events on the digital I/O bus for different clock edges or states. If a user wants to specify a channel for trigger events, the corresponding bit should be set to "1". Specifically, the individual array elements specify the following:

0 = dioLines[0] Positive Edge
1 = dioLines[1] Negative Edge
2 = dioLines[2] Positive Level
3 = dioLines[3] Negative Level

For example, if a user wishes to trigger the EX1629 on a negative edge signal coming into the digital I/O bus on channels 0 and 3, then: **dioLines[1]** = 00001001 = 0x09 = 9.

With regard to the **lxiOutput** parameter, it is important to note that since the EX1629 can simultaneously accept trigger events from multiple sources, it is necessary to reserve one of the LXI Trigger Bus line to communicate these events within the device and to other devices in a multi-device configuration. If the EX1629 is a master driving trigger events to peripheral slaves, the **lxiOutput** parameter specifies the LXI Trigger Bus line that will be used to communicate the trigger event to the slave devices. It is also necessary to configure this LXI Trigger Bus line to be used as an output (see vtex1629_set_lxibus_configuration).

The **lxiInput** parameter specifies which trigger bus line the master device uses for its trigger events. Although this parameter is often set to the same value as **lxiOutput**, there are cases where it might be set to a different value. For instance, when the device is configured as a master device in a star multi-box configuration, the master might output the trigger event on LXI2 and input it back in on LXI6.

| Decimal Value | Hex Value | #define Symbol | lxiOutput/lxiInput Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_LXI_LINE_ZERO | LXI LINE 0 |
| 1 | 0x01 | VTEX1629_LXI_LINE_ONE | LXI LINE 1 |
| 2 | 0x02 | VTEX1629_LXI_LINE_TWO | LXI LINE 2 |
| 3 | 0x03 | VTEX1629_LXI_LINE_THREE | LXI LINE 3 |
| 4 | 0x04 | VTEX1629_LXI_LINE_FOUR | LXI LINE 4 |
| 5 | 0x05 | VTEX1629_LXI_LINE_FIVE | LXI LINE 5 |
| 6 | 0x06 | VTEX1629_LXI_LINE_SIX | LXI LINE 6 |
| 7 | 0x07 | VTEX1629_LXI_LINE_SEVEN | LXI LINE 7 |
| 8 | 0x08 | VTEX1629_LXI_NONE | None |

**EXAMPLE**

```
ViSession instrumentHandle;
ViStatus status;
ViInt16 triglines[VTEX1629_MAX_LINES];
ViInt16 diolines[VTEX1629_MAX_LINES];
…
triglines[0] = 0x00;
triglines[1] = 0x01;
triglines[2] = 0x48;
triglines[3] = 0x00;

diolines[0] = 0x0000;
diolines[1] = 0x0009;
diolines[2] = 0x0000;
diolines[3] = 0x0000;

status = vtex1629 vtex1629_set_pattern_trigger_configuration (instrumentHandle,
                                                              triglines,
                                                              diolines,
                                                              VI_FALSE,
                                                              VTEX1629_LXI_LINE_THREE,
                                                              VTEX1629_LXI_LINE_THREE);
```

# vtex1629_set_poisson_ratio

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_poisson_ratio (ViSession **vi**, ViInt32 _VI_FAR **channels[]**, ViInt32 **numberOfChannels**, ViReal64 **poissonRatio**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the Poisson ratio will be set. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**poissonRatio** = a real input value indicating the Poisson ratio to be set for all the given channels.

## DATA ITEM RESET VALUE

**poissonRatio** = 0.300000

## DESCRIPTION

This function sets the Poisson ratio for a list of channels. This is one of the parameters used in some strain gage EU conversion calculations.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0,1,2,3};
…
status = vtex1629_set_poisson_ratio(instrumentHandle, channels, 4, 0.301);
```

# vtex1629_set_sample_clock_source

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_sample_clock_source (ViSession **vi**, ViInt32 **sampleClockMode**, ViInt32 **inLine**, ViInt32 **outLine**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**sampleClockMode** = an integer input value that determines whether the EX1629 is operating as a master or slave. Valid input values: VTEX1629_SAMP_CLK_MODE_MASTER or VTEX1629_SAMP_CLK_MODE_SLAVE.
**inLine** = an integer input value that determines the trigger bus line configured to use for sample clock input. Valid input values: VTEX1629_LXI_LINE_ZERO, VTEX1629_LXI_LINE_FOUR, or VTEX1629_LXI_LINE_NONE.
**outLine** = an integer input value that determines the trigger bus line configured to output sample clock events. Valid input values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN, VTEX1629_LXI_LINE_NONE.

## DATA ITEM RESET VALUE

**inLine** = VTEX1629_LXI_LINE_NONE **outLine** = VTEX1629_LXI_LINE_NONE
**sampleClockMode** = VTEX1629_SAMP_CLK_MODE_MASTER

## DESCRIPTION

This function sets the sample clock source for the device.

The **sampleClockMode** parameter indicates whether the EX1629 is configured as a master device that outputs a sample clock for itself and other devices or as a slave device that receives its sample clock from another device. When operating in standalone mode, **sampleClockMode** should be configured as a master.

The **inLine** parameter indicates the LXI line that should be used as the sample clock input. This value is applicable regardless of whether the device is configured as a master or a slave. When **inLine** is set to VTEX1629_LXI_LINE_NONE, a special, internal sample clock line is used.

The **outLine** parameter indicates the LXI line that should be used as the sample clock output. This value is only applicable when the device is configured as a master. When **outLine** is set to VTEX1629_LXI_LINE_NONE, the sample clock is output on a special, internal sample clock line. When in Slave mode, no output line can be used, so VTEX1629_LXI_LINE_NONE must be specified.

| Decimal Value | Hex Value | #define Symbol | inLine/outLine Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_LXI_LINE_ZERO | LXI LINE 0 |
| 4 | 0x04 | VTEX1629_LXI_LINE_FOUR | LXI LINE 4 |
| 8 | 0x08 | VTEX1629_LXI_LINE_NONE | None |

When in master mode, the **inLine** and **outLine** values may be the same or they may be different. One case where they would be different is if the master is outputting the sample clock on one LXI line and receiving it back in from an LXI Trigger Bus hub on another line. When in standalone mode, **inLine** and **outLine** will always be the same.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_sample_clock_source(instrumentHandle,
                                          VTEX1629_SAMP_CLK_MODE_MASTER,
                                          VTEX1629_LXI_LINE_ZERO,
                                          VTEX1629_LXI_LINE_ZERO);
```

# vtex1629_set_sample_count

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_sample_count (ViSession **vi**, ViInt32 **preTrigSampleCount**, ViInt32 **postTrigSampleCount**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**preTrigSampleCount** = an integer input value indicating the desired pre-trigger sample count. Valid input values: 0 to 400000000.
**postTrigSampleCount** = an integer input value indicating the desired post-trigger sample count. Valid input values: 0 to 400000000.

## DATA ITEM RESET VALUE

**preTrigSampleCount** = 0 **postTrigSampleCount** = 1000

## DESCRIPTION

This function sets both the pre-trigger and the post-trigger sample count for the EX1629. Specifically, this is the number of samples that will be taken per trigger event. If the **postTrigSampleCount** is set to "0" the sample count will be infinite.

| **NOTE** | Pre-trigger sampling is not currently supported. Setting the **preTrigSampleCount** to a value other than zero (0) will result in an error. |
|---|---|

## EXAMPLE

```
ViStatus status;
…
status = vtex1629_set_sample_count(instrumentHandle, 0, 10000);
```

# vtex1629_set_sample_frequency

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_sample_frequency (ViSession **vi**, ViReal64 **sampleFrequency**)

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**sampleFrequency** = a real input value that specifies, in hertz (Hz), the desired sampling frequency of the EX1629.

## DATA ITEM RESET VALUE

**sampleFrequency** = 1000.00000

## DESCRIPTION

This function sets the sampling frequency of all channels of the EX1629. The EX1629 offers a discrete number of sample frequencies. Programmed values that fall between valid values will be rounded to the closest valid value. See *Sampling Rate* in Section 3 for a list of the valid values. The actual sample frequency can be queried with the vtex1629_get_sample_frequency command.

As changing the **sampleFrequency** parameter effects the actual values of the arm and trigger delay calls, it is best practice to perform vtex1629_get_arm_delay and vtex1629_get_trigger_delay calls after making a vtex1629_set_sample_frequency call to determine if these delays have been effected.

## EXAMPLE

```
ViStatus status;
…
status = vtex1629_set_sample_frequency(instrumentHandle, 100.0);
```

# vtex1629_set_scanlist

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_scanlist (ViSession **vi**, ViInt32 _VI_FAR **channels[]**, ViInt32 **numberOfChannels**)

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an integer input array indicating which channels will be included in the scan list. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.

## DATA ITEM RESET VALUE

**channels** = 0 to 47

## DESCRIPTION

This function defines a list of channels which will be sampled in the data acquisition process. The term scanlist is used, although all channels are sampled synchronously, and in parallel (not "scanned" as in a multiplexed ADC system). The scanlist must include at least one channel and may not include any duplicate channels.

| | |
|---|---|
| **NOTE** | Regardless of the order of the elements in the array, when data is retrieved using vtex1629_read_fifo or vtex1629_read_fifoEx, the data will be organized in ascending order with respect to the elements in the scan list. |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0,1,2,3};
…
status = vtex1629_set_scanlist(instrumentHandle, channels, 4);
```

# vtex1629_set_shunt_enabled

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_shunt_enabled (ViSession **vi**, ViInt32 _VI_FAR **channels[]**,ViInt32 **numberOfChannels**, ViBoolean **enabled**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the shunt resistors will be enabled/disabled. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**enabled** = a Boolean input value indicating whether to enable the shunt resistors for the given list of channels. VI_TRUE(1) will enable the shunt resistors for the given list of channels and VI_FALSE(0) will disable the shunt resistors for the given list of channels..

## DATA ITEM RESET VALUE

**enabled** = 0 (disabled)

## DESCRIPTION

This function enables or disables the shunt resistors for a particular list of channels, based on the currently configured shunt source for each channel. A shunt source that is not enabled will not actually be applied in hardware.

| | |
|---|---|
| **NOTE** | The configuration of the shunt source and its enabling are separate operations. Thus, setting a shunt source using the **vtex1629_set_shunt_source** function does not guarantee that the shunt source is enabled. That must be set with the **vtex1629_set_shunt_enabled** command. |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0,1,2,3};
…
status = vtex1629_set_shunt_enabled(instrumentHandle, channels, 4, VI_TRUE);
```

# vtex1629_set_shunt_source

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_shunt_source (ViSession **vi**, ViInt32 _VI_FAR **channels[]**,ViInt32 **numberOfChannels**, ViInt32 **shuntSource**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the shunt source will be set. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**shuntSource** = an integer input value indicating the shunt source to be set for the given list of channels. Valid input values: 0 to 4.

## DATA ITEM RESET VALUE

**shuntSource** = VTEX1629_INTERNAL_REMOTE

## DESCRIPTION

This function sets the shunt source for a given list of channels. The following values are valid for the **shuntSource** parameter:

| Decimal Value | Hex Value | #define Symbol | shuntSource Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_FRONT_PANEL_REMOTE | Front panel remote |
| 1 | 0x01 | VTEX1629_FRONT_PANEL_LOCAL | Front panel local |
| 2 | 0x02 | VTEX1629_INTERNAL_REMOTE | Internal remote |
| 3 | 0x03 | VTEX1629_INTERNAL_LOCAL | Internal local |
| 4 | 0x04 | VTEX1629_TEDS_REMOTE | TEDS remote |

Local and Remote refer to how the shunt resistor is connected to the bridge. For “Local”, the connection is made within the EX1629. For “Remote”, the connection is made externally, using the ±RCal signals.

Front Panel, Internal, and TEDS, refer to the three types of shunt sources supported. “Front Panel” refers to the shunt resistors that may be connected directly to the front panel of the EX1629, which are shared by 16 channels (0 through 15, 16 through 31, and 32 through 47). Only one channel may be connected to the Front Panel shunt at a time. “Internal” refers to the internal, per-channel shunt resistor. Since each channel has its own, all channels may be shunted simultaneously when configured for Internal Remote or Internal Local. “TEDS” refers to a special shunt resistor/TEDS (1-Wire) configuration. Only one channel may be shunted via the TEDS shunt at a time.

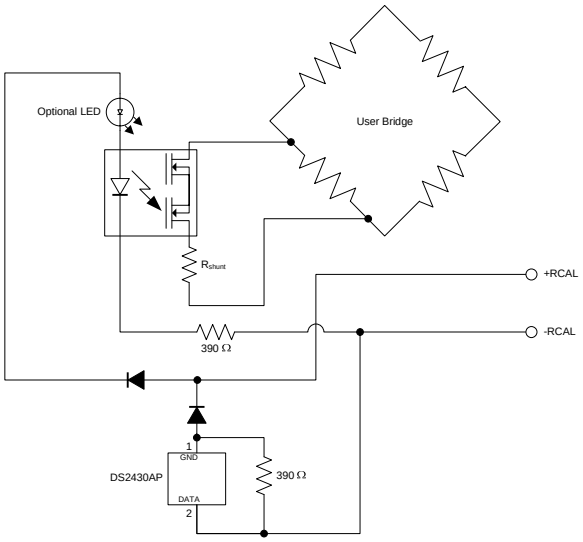
| Bridge Type | TEDS | FPS – Internal | FPS – External | IS – Internal | IS – External | TEDS/ Remote |
|---|---|---|---|---|---|---|
| Quarter | No | X | | X | | |
| | Yes | X | | X | | X |
| Half | No | | X | | X | |
| | Yes | | | | | X |
| Full | No | | X | | X | |
| | Yes | | | | | X |

**NOTE**: “X” denotes legal mode of operation.

| | |
|---|---|
| **NOTE** | The configuration of the shunt source and its enabling are separate operations. Thus, setting a shunt source using the vtex1629_set_shunt_source function does not guarantee that the shunt source is enabled. That must be set with the vtex1629_set_shunt_enabled command. |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0,1,2,3};
…
status = vtex1629_set_shunt_source (instrumentHandle,
                                    channels, 4,
                                    SHUNT_SOURCE_IS_LOCAL);
```




**TEDS Shunt Schematic**

# vtex1629_set_shunt_value

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_shunt_value (ViSession **vi**, ViInt32 _VI_FAR **channels[]**,ViInt32 **numberOfChannels**, ViInt32 **shuntSource**, ViReal64 **shuntValue**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the shunt value will be set. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**shuntSource** = a integer input value indicating the shunt source for which the value is to be set. Valid input values: 0 to 4.
**shuntValue** = a real input value that defines the value of the specified shunt.

## DATA ITEM RESET VALUE

**shuntValue** (source 0, 1, 4) = 0.00000

## DESCRIPTION

This function sets the value of a shunt resistor based on a given channel and shunt source. The following values are allowed for the **shuntSource** parameter:

| Decimal Value | Hex Value | #define Symbol | shuntSource Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_FRONT_PANEL_REMOTE | Front panel remote |
| 1 | 0x01 | VTEX1629_FRONT_PANEL_LOCAL | Front panel local |
| 2 | 0x02 | VTEX1629_INTERNAL_REMOTE | Internal remote |
| 3 | 0x03 | VTEX1629_INTERNAL_REMOTE | Internal local |
| 4 | 0x04 | VTEX1629_TEDS_REMOTE | TEDS remote |

Local and Remote refer to how the shunt resistor is connected to the bridge. For “Local”, the connection is made within the EX1629. For “Remote”, the connection is made externally, using the ±RCal signals.

Front Panel, Internal, and TEDS, refer to the three types of shunt sources supported. “Front Panel” refers to the shunt resistors that may be connected directly to the front panel of the EX1629, which are shared by 16 channels (0 through 15, 16 through 31, and 32 through 47). Only one channel may be connected to the Front Panel shunt at a time. “Internal” refers to the internal, per-channel shunt resistor. Since each channel has its own, all channels may be shunted simultaneously. “TEDS” refers to a special shunt resistor/TEDS (1-Wire) configuration. Only one channel may be shunted via the TEDS shunt at a time.

While the internal shunt sources are valid function parameters, their values are provided from the calibration file and are not user alterable.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0,1,2,3};
…
status = vtex1629_set_shunt_value(instrumentHandle,
                                  channels,
                                  4,
                                  SHUNT_SOURCE_FPS_LOCAL,
                                  50000.0);
```

# vtex1629_set_strain_units

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_strain_units (ViSession **vi**, ViInt32 _VI_FAR **channels[]**,ViInt32 **numberOfChannels**, ViBoolean **microStrain**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the strain units will be set. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**microStrain** = a Boolean input value indicating whether strain measurements will be returned in units of strain (ε) or microstrain (με). Passing a value of VI_TRUE (1) will configure the instrument to return microstrain while passing a value of VI_FALSE(0) will configure the instrument to return strain.

## DATA ITEM RESET VALUE

**microStrain** = VI_FALSE(0) (strain)

## DESCRIPTION

This function determines whether the EX1629 will return strain measurements in units of strain (ε) or microstrain (με) for a given list of channels. Each channel can be configured to return strain measurements in strain or microstrain (1 strain (ε) = $1x10^6$ microstrain (με)). This setting has no effect for non-strain EU conversions.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
…
status = vtex1629_set_strain_units(instrumentHandle, channels, 16, VI_TRUE);
```

# vtex1629_set_synch_source

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_synch_source (ViSession **vi**, ViInt32 **synchMode**, ViInt32 **inLine**, ViInt32 **outLine**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**synchMode** = an integer input value that indicates whether the EX1629 is operating as a master, slave, or as a standalone. Valid input values: VTEX1629_SYNC_MODE_MASTER or VTEX1629_SYNC_MODE_SLAVE.
**inLine** = an integer input value that indicates the trigger bus line to receive synch events. Valid input values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN, VTEX1629_LXI_LINE_NONE.
**outLine** = an integer input value that indicates the trigger bus line to output synch events. Valid input values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN, VTEX1629_LXI_LINE_NONE.

## DATA ITEM RESET VALUE

**synchMode** = VTEX1629_SYNC_MODE_MASTER **inLine** = VTEX1629_LXI_LINE_NONE
**outLine** = VTEX1629_LXI_LINE_NONE

## DESCRIPTION

This function sets the synchronization source.

The **synchMode** value indicates whether the EX1629 is configured as a master device that outputs a synch signal for itself and other devices or as a slave device that receives its synch signal from another device. When operating in standalone mode, **synchMode** should be configured as a master.

The **inLine** value indicates the LXI line that should be used as the synch input. This value is applicable regardless of whether the device is configured as a master or a slave. When **inLine** is set to VTEX1629_LXI_LINE_NONE, the special, internal synch line is used.

The **outLine** value indicates the LXI line that should be used as the synch output. This value is only applicable when the device is configured as a master. When **outLine** is set to VTEX1629_LXI_LINE_NONE, the synch signal is output on the special, internal synch line only. When in Slave mode, no output line can be used, so VTEX1629_LXI_LINE_NONE must be specified.

| Decimal Value | Hex Value | #define Symbol | inLine/outLine Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_LXI_LINE_ZERO | LXI LINE 0 |
| 1 | 0x01 | VTEX1629_LXI_LINE_ONE | LXI LINE 1 |
| 2 | 0x02 | VTEX1629_LXI_LINE_TWO | LXI LINE 2 |
| 3 | 0x03 | VTEX1629_LXI_LINE_THREE | LXI LINE 3 |
| 4 | 0x04 | VTEX1629_LXI_LINE_FOUR | LXI LINE 4 |
| 5 | 0x05 | VTEX1629_LXI_LINE_FIVE | LXI LINE 5 |
| 6 | 0x06 | VTEX1629_LXI_LINE_SIX | LXI LINE 6 |
| 7 | 0x07 | VTEX1629_LXI_LINE_SEVEN | LXI LINE 7 |
| 8 | 0x08 | VTEX1629_LXI_LINE_NONE | None |

When in master mode, the **inLine** and **outLine** values may be the same or they may be different. One case where they would be different is if the master is outputting the synch signal on one LXI line and receiving it back in from a LXI Trigger Bus hub on another line. When in standalone mode, **inLine** and **outLine** should always be VTEX1629_LXI_LINE_NONE.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_synch_source(instrumentHandle,
                                   VTEX1629_SYNC_MODE_MASTER,
                                   VTEX1629_LXI_LINE_ONE,
                                   VTEX1629_LXI_LINE_ONE);
```

# vtex1629_set_tare

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_tare (ViSession **vi**, ViInt32 _VI_FAR **channels[]**, ViInt32 **numberOfChannels**, ViReal64 **tareValue**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the tare value will be set. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**tareValue** = a real input value indicating the tare value to set for all the indicated channels.

## DATA ITEM RESET VALUE

**tareValue** = 0.00000

## DESCRIPTION

This function sets the tare values for a list of channels. The tare value is subtracted from the output of the EU conversion. It is used to provide a user-selectable offset value for a channel.

| | |
|---|---|
| **NOTE** | The tare value is not linked to the units setting. For correct results, the EU conversion and units (ε or με, if using a strain EU conversion) should be set first, followed by any tare value. |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0};
…
status = vtex1629_set_tare (instrumentHandle, channels, 1, 0.01);
```

# vtex1629_set_teds_data

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_teds_data (ViSession **vi**, ViInt32 **channel**, ViInt16 _VI_FAR **tedsID[]**, ViInt32 **maxLength**, ViInt16 _VI_FAR **tedsInfo[]**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the tare value will be set. Valid input values: 0 to 47.
**tedsID[]** = an integer input array specifying the unique id of the TEDS device to be set. Each element of the array represents a byte of data from the ID register of the TEDS device. The size of this array is equal to VTEX1629_TEDS_IDSIZE.
**maxLength** = an integer input value that specifies the maximum number of bytes that will be written to the TEDS device. Valid input values: 0 to 32.
**tedsInfo[]** = an integer input array specifying the TEDS data to be written. Each element of this array specifies a byte of TEDS data, and the array should not contain any more bytes than specified by the **maxLength** parameter. The maximum size for this array is VTEX1629_TEDS_DATASIZE.

## DATA ITEM RESET VALUE

Not applicable for this function.

## DESCRIPTION

This function writes data to the TEDS device indicated by the **channels** parameter. The only EEPROM supported is the DS2430A.

The **tedsID** parameter must be equal to the value in the component connected to the channel. This parameter can be queried by using the vtex1629_get_teds_data function.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt16 tedsID[VTEX1629_TEDS_IDSIZE];
ViInt32 maxlen;
ViInt16 tedsinfo[VTEX1629_TEDS_DATASIZE];
…
…
status = vtex1629_get_teds_data(instrumentHandle,
                                47,
                                tedsID,
                                VTEX1629_TEDS_DATASIZE,
                                tedsinfo);

memset(tedsinfo, 0x0, sizeof(tedsinfo)); // clear device memory

status = vtex1629_set_teds_data(instrumentHandle,
                                47,
                                tedsID,
                                VTEX1629_TEDS_DATASIZE,
                                tedsinfo);
```

# vtex1629_set_trigger_count

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_trigger_count (ViSession **vi**, ViInt32 **trigCount**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**trigCount** = an integer input value that specifies the trigger count for the EX1629. Valid input values: 1 to ($2^{31}$-1).

## DATA ITEM RESET VALUE

**trigCount** = 1

## DESCRIPTION

This function sets the trigger count for the EX1629. Specifically, this is the number of times the EX1629 will wait for triggers after being armed before it will return to the Arm layer of the trigger state machine.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_trigger_count(instrumentHandle, 5);
```

# vtex1629_set_trigger_delay

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_trigger_delay (ViSession **vi**, ViReal64 **delay**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**delay** = a real input value, in seconds, indicating the desired trigger delay. Valid input values: 0 s to 4294.967295 s.

## DATA ITEM RESET VALUE

**delay** = 0.000000000

## DESCRIPTION

This function sets the trigger delay for the EX1629. Specifically, this is the amount of time, in seconds, that the EX1629 will wait after receiving a TRIG event before it begins to acquire data.

The actual delay exhibited by the EX1629 is dependent on the sample frequency, set by calling the vtex1629_set_sample_frequency function. The actual delay will be a multiple of the sample time. For example, if the sample frequency is 1 kHz, the sample time is 1 ms. If the trigger delay is set to a value less than 0.5 ms, the EX1629 will experience no delay. If the trigger delay is set to a value between 0.5 ms and 1.49 ms, the delay exhibited will be 1 ms.

As a result, it is best practice to perform a vtex1629_get_arm_delay call after a vtex1629_set_trigger_delay or a vtex1629_set_sample_frequency call is performed to determine the actual delay.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_trigger_delay(instrumentHandle, 0.01);
```

# vtex1629_set_trigger_source

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_trigger_source (ViSession **vi**, ViInt32 **triggerSource**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**triggerSource** = an integer input value that indicates the desired source to be monitored for trigger events. Valid input values: 0 to 17.

## DATA ITEM RESET VALUE

**triggerSource** = 0 (immediate)

## DESCRIPTION

This function sets the trigger source on the EX1629. Possible values for the trigger source include:

| Decimal Value | Hex Value | #define | armSource Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_TRIG_SRC_IMMEDIATE | Immediate |
| 1 | 0x01 | VTEX1629_TRIG_SRC_PATTERN | Pattern |
| 2 | 0x02 | VTEX1629_TRIG_SRC_LXI0_POS | LXI 0 Positive Edge |
| 3 | 0x03 | VTEX1629_TRIG_SRC_LXI1_POS | LXI 1 Positive Edge |
| 4 | 0x04 | VTEX1629_TRIG_SRC_LXI2_POS | LXI 2 Positive Edge |
| 5 | 0x05 | VTEX1629_TRIG_SRC_LXI3_POS | LXI 3 Positive Edge |
| 6 | 0x06 | VTEX1629_TRIG_SRC_LXI4_POS | LXI 4 Positive Edge |
| 7 | 0x07 | VTEX1629_TRIG_SRC_LXI5_POS | LXI 5 Positive Edge |
| 8 | 0x08 | VTEX1629_TRIG_SRC_LXI6_POS | LXI 6 Positive Edge |
| 9 | 0x09 | VTEX1629_TRIG_SRC_LXI7_POS | LXI 7 Positive Edge |
| 10 | 0x0A | VTEX1629_TRIG_SRC_LXI0_NEG | LXI 0 Negative Edge |
| 11 | 0x0B | VTEX1629_TRIG_SRC_LXI1_NEG | LXI 1 Negative Edge |
| 12 | 0x0C | VTEX1629_TRIG_SRC_LXI2_NEG | LXI 2 Negative Edge |
| 13 | 0x0D | VTEX1629_TRIG_SRC_LXI3_NEG | LXI 3 Negative Edge |
| 14 | 0x0E | VTEX1629_TRIG_SRC_LXI4_NEG | LXI 4 Negative Edge |
| 15 | 0x0F | VTEX1629_TRIG_SRC_LXI5_NEG | LXI 5 Negative Edge |
| 16 | 0x10 | VTEX1629_TRIG_SRC_LXI6_NEG | LXI 6 Negative Edge |
| 17 | 0x11 | VTEX1629_TRIG_SRC_LXI7_NEG | LXI 7 Negative Edge |

**Immediate (0)**: an immediate trigger source. After receiving the arm event, the trigger state machine will bypass the trigger layer and will automatically begin to acquire data.

**Pattern (1)**: this trigger source allows the EX1629 to accept trigger events from multiple sources. The EX1629 can be configured to accept trigger events on LXI Trigger Bus channels, digital I/O channels, a timer, and software triggers. The EX1629 can simultaneously accept any combination of these events. The specific pattern is set with the vtex1629_set_pattern_trig_configuration command.

**LXI *n* Positive Edge (2 – 9)**: these trigger sources refer to trigger events coming from the LXI Trigger Bus and will cause the EX1629 to trigger on the positive edge of signals coming into the LXI Trigger Bus.

**LXI *n* Negative Edge (10 – 17)**: these trigger sources refer to trigger events coming from the LXI Trigger Bus and will cause the EX1629 to trigger on the negative edge of signals coming into the LXI Trigger Bus.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_trigger_source(instrumentHandle, VTEX1629_TRIG_SRC_LXI4_POS);
```

# vtex1629_set_trigger_source_timer

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_trigger_source_timer (ViSession **vi**, ViReal64 **timerPeriod**, ViInt32 **lxiOutput**, ViInt32 **lxiInput**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**timerPeriod** = a real input value indicating the time, in seconds, the EX1629 will wait in between generating TRIG events. Valid input values: 0 s to 167 s.
**lxiOutput** = this parameter specifies which LXI Trigger Bus line will be used to output the arm event signals. Valid input values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN, LXI_NONE.
**lxiInput** = this parameter specifies which LXI Trigger Bus line will be used to input the arm event signals. Valid input values: VTEX1629_LXI_LINE_ZERO to VTEX1629_LXI_LINE_SEVEN, LXI_NONE.

## DATA ITEM RESET VALUE

**timerPeriod** = 0
**lxiOutput** = 8 (LXI_NONE)
**lxiInput** = 8 (LXI_NONE)?

## DESCRIPTION

This is a convenience functions that configures the EX1629 to operate in pattern trigger mode using the trigger timer as the only source and configuring the trigger timer period.

With regard to the **lxiOutput** parameter, it is important to note that since the EX1629 can simultaneously accept trigger events from multiple sources, it is necessary to reserve one of the LXI Trigger Bus line to communicate these events within the device and to other devices in a multi-device configuration. If the EX1629 is a master driving trigger events to peripheral slaves, the **lxiOutput** parameter specifies the LXI Trigger Bus line that will be used to communicate the trigger event to the slave devices. It is also necessary to configure this LXI Trigger Bus line to be used as an output (see vtex1629_set_lxibus_configuration).

The **lxiInput** parameter specifies which trigger bus line the master device uses for its trigger events. Although this parameter is often set to the same value as **lxiOutput**, there are cases where it might be set to a different value. For instance, when the device is configured as a master device in a star multi-box configuration, the master might output the trigger event on LXI2 and input it back in on LXI6.

| Decimal Value | Hex Value | #define Symbol | lxiOutput/lxiInput Description |
|---|---|---|---|
| 0 | 0x00 | VTEX1629_LXI_LINE_ZERO | LXI LINE 0 |
| 1 | 0x01 | VTEX1629_LXI_LINE_ONE | LXI LINE 1 |
| 2 | 0x02 | VTEX1629_LXI_LINE_TWO | LXI LINE 2 |
| 3 | 0x03 | VTEX1629_LXI_LINE_THREE | LXI LINE 3 |
| 4 | 0x04 | VTEX1629_LXI_LINE_FOUR | LXI LINE 4 |
| 5 | 0x05 | VTEX1629_LXI_LINE_FIVE | LXI LINE 5 |
| 6 | 0x06 | VTEX1629_LXI_LINE_SIX | LXI LINE 6 |
| 7 | 0x07 | VTEX1629_LXI_LINE_SEVEN | LXI LINE 7 |
| 8 | 0x08 | VTEX1629_LXI_NONE | None |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_trigger_source_timer(instrumentHandle,
                                           10.0,
                                           VTEX1629_LXI_LINE_TWO,
                                           VTEX1629_LXI_LINE_TWO );
```

# vtex1629_set_trigger_timer

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_trigger_timer (ViSession **vi**, ViReal64 **timer**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**timer** = a real input value indicating the time, in seconds, to which to set the trigger system timer. Valid input values: 0 s to 4294.967295 s.

## DATA ITEM RESET VALUE

**timer** = 0

## DESCRIPTION

This function sets the trigger timer period for the EX1629. This is the amount of time, in seconds, that the EX1629 will wait before generating successive timer events, which can be used as an arm source or a trigger source, but not both.

The timer is specified as the arm or trigger source using the "pattern" source mechanism (see vtex1629_set_arm_source, vtex1629_set_pattern_arm_configuration, vtex1629_set_trigger_source, and vtex1629_set_pattern_trig_configuration).

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_set_trigger_timer(instrumentHandle, 1.0);
```

# vtex1629_set_unstrained_voltage

## FUNCTION PROTOTYPE

ViStatus vtex1629_set_unstrained_voltage (ViSession **vi**, ViInt32 _VI_FAR **channels[]**, ViInt32 **numberOfChannels**, ViReal64 **unstrainedVoltage**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer array containing a list of channel numbers for which the unstrained voltage will be set. Valid input values: 0 to 47.
**numberOfChannels** = the size of the **channels** list. Valid input values: 1 to 48.
**unstrainedVoltage** = a real input value that indicates the unstrained voltage to set for the given list of channels.

## DATA ITEM RESET VALUE

**unstrainedVoltage** = 0.000000

## DESCRIPTION

This function sets the unstrained voltage for a list of channels. This is one parameter in the EU calculations.

| | |
|---|---|
| **NOTE** | The conventional method of providing an unstrained voltage value to the EU conversion is to conduct an unstrained voltage measurement using the vtex1629_measure_unstrained_voltage function. This function provides a manual method that is normally only used for system diagnostic purposes. |

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 channels[] = {0};
…
status = vtex1629_set_unstrained_voltage(instrumentHandle, channels, 1, 0.01);
```

# vtex1629_soft_arm

## FUNCTION PROTOTYPE

ViStatus vtex1629_soft_arm (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function sends a software generated arm event to the EX1629. Note, a software arm is only legal when the arm source is configured as pattern (see vtex1629_set_arm_source).

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_soft_arm(instrumentHandle);
```

# vtex1629_soft_synch

## FUNCTION PROTOTYPE

ViStatus vtex1629_soft_synch (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function sends a software generated synchronization event to the device. This is only valid when the device is a synch master (see vtex1629_set_synch_source).

| **NOTE** | The signal conditioning path is reset on a sync event, which means that acquisition data will not reflect the input signals until the digital filters settled. See vtex1629_get_settling_time for more information. |
|---|---|

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_soft_synch(instrumentHandle);
```

# vtex1629_soft_trig

## FUNCTION PROTOTYPE

ViStatus vtex1629_soft_trig (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function sends a software-generated trigger event to the EX1629.

A software trigger is only legal when the trigger source is configured as pattern (see vtex1629_set_trigger_source).

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_soft_trig(instrumentHandle);
```

# vtex1629_store_current_config

## FUNCTION PROTOTYPE

ViStatus vtex1629_store_current_config (ViSession **vi**, ViInt32 **digestArraySize**, ViInt8 _VI_FAR **digest[]**, ViPInt32 **digestActualSize**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**digestArraySize** = contains the size of the allocated digest array. For consistency, the client application should allocate VTEX1629_MAX_DIGEST_LENGTH bytes.
**digest[]** = the stored configuration's digest.
**digestActualSize** = the actual size of the configuration digest.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function stores the current configuration of the instrument in the nonvolatile storage. A digest of the stored configuration is returned. The digest is a digital signature representing the actual configuration data.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 digestActualSize;
ViInt8 digest[VTEX1629_MAX_DIGEST_LENGTH];
…
status = vtex1629_store_current_config(instrumentHandle,
                                       VTEX1629_MAX_DIGEST_LENGTH,
                                       digest,
                                       &digestActualSize);
```

# vtex1629_trig_init

## FUNCTION PROTOTYPE

ViStatus vtex1629_trig_init (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function initiates the trigger system. Specifically, calling this function moves the trigger state machine from the Idle layer into the Init layer.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_trig_init(instrumentHandle);
```

# vtex1629_unlock

## FUNCTION PROTOTYPE

ViStatus vtex1629_unlock (ViSession **vi**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function unlocks the EX1629 instrument, if the user is the owner of the lock. The EX1629 can then be accessed by another user or another session.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
…
status = vtex1629_unlock(instrumentHandle);
```

# vtex1629_write_teds_MLAN

## FUNCTION PROTOTYPE

ViStatus vtex1629_write_teds_MLAN (ViSession **vi**, ViInt32 **channel**, ViInt32 **bufferArraySize**, ViInt8 _VI_FAR **buffer[]**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**channels** = an input integer value that specifies the channel number for the TEDS device to which the MLAN buffer should be written. Valid input values: 0 to 47.
**bufferArraySize** = an input integer indicating the size of the array that holds the MLAN command bytes. Its value should be less than VTEX1629_MAX_MLAN_DATA_LEN.
**buffer[]** = an input array that contains the TEDS MLAN data. Its size should be less than or equal to VTEX1629_MAX_MLAN_DATA_LEN.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function writes a variable sized block of MLAN commands and data to the TEDS EEPROM indicated by the **channel** parameter. The vtex1629_read_teds_MLAN command can be used to read the response back from the device. The commands and data in the buffer must comply with the MicroLAN specification in IEEE 1451.4-2004 Annex G.

**NOTES**

1) Details of the MLAN specification can be found at *http://www.maxim-ic.com/products/ibutton/applications/* and other sites.
2) The bytes returned in 'buffer' need to be interpreted by the application in accordance with the MLAN specification.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt8 mlanData[VTEX1629_MAX_MLAN_DATA_LEN];
…
    <prepare the command bytes in the mlanData array>
status = vtex1629_write_teds_MLAN (instrumentHandle,
                                   15,
                                   VTEX1629_MAX_MLAN_DATA_LEN,
                                   mlanData);

If (status < VI_SUCCESS)
{
    <inform the user the API call failed>
}
```

# vtex1629_zero_cal

## FUNCTION PROTOTYPE

ViStatus vtex1629_zero_cal (ViSession **vi**, ViPInt32 **overRide**, ViPInt32 **recommendedUpTime**, ViPInt32 **currentUpTime**);

## FUNCTION PARAMETERS

**vi** = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
**overRide** = a returned integer value indicating the over ride value.
**recommendedUpTime** = an integer return value indicating the recommended amount of time, in seconds, the EX1629 should be powered up attempting a zero calibration.
**currentUpTime** = an integer return value indicating the amount of time, in seconds, that the EX1629 has been powered up.

## DATA ITEM RESET VALUE

Not applicable to this function.

## DESCRIPTION

This function resets the offset values of the unit before a measurement is taken. It is designed to be a shorter calibration process than self-calibration.

The **overRide** parameter is used in the case where a zero calibration is attempted on a unit that has not been powered on for a sufficient amount of time. In this special case, the first call to this function will return an error, but will also populate this over ride variable with a unique integer value in memory. The vtex1629_self_cal_init function can then be called a second time using this unique value. The second call to the function will successfully initiate a self-calibration on a unit that has not been powered up for the recommended duration of time.

## EXAMPLE

```
ViSession instrumentHandle;
ViStatus status;
ViInt32 overRide = 0;
ViInt32 recommendedUpTime = 0;
ViInt32 currentUpTime = 0;

status = vtex1629_zero_cal(instrumentHandle,
                           &overRide,
                           &recommendedUpTime,
                           &currentUpTime);
```

## ERROR MESSAGES

Each function in this instrument driver returns a status code that either indicates success or describes an error or warning condition. Programs should examine the status code from each call to an instrument driver function to determine if an error occurred. The general meaning of the status code is as follows:

| Value | Meaning |
|---|---|
| VI_SUCCESS (0) | Success |
| Positive Values | Warnings |
| Negative Values | Errors |

The following table lists possible error codes and their meanings which may be returned by driver functions. The following error codes, in hexadecimal, may be encountered while operating the EX1629. The corresponding error messages' text can be obtained by using the vtex1629_error_message function.

| Error Code | Error Message | Meaning |
|---|---|---|
| FFBFFC09F0 | VTEX1629_ERROR_RUNTIME | Runtime error. |
| FFBFFC09F1 | VTEX1629_ERROR_MEM_ALLOC | Memory allocation error. |
| FFBFFC09F2 | VTEX1629_ERROR_RPC | RPC error. |
| FFBFFC09F3 | VTEX1629_ERROR_INVALID_RESOURCE | Not a valid LXI resource. |
| FFBFFC09F4 | VTEX1629_ERROR_ID_MISMATCH | ID mismatch. |
| FFBFFC09F5 | VTEX1629_ERROR_ID_QUERY | ID query. |
| FFBFFC09F6 | VTEX1629_ERROR_SYNCHRONIZATION | Synchronization error. |
| FFBFFC09F7 | VTEX1629_ERROR_MAX_INSTRUMENT | Max instrument exceeded. |
| FFBFFC09F8 | VTEX1629_ERROR_CLNT_CREATE | Could not create the client. Please ensure you have the latest firmware version. |
| FFBFFC09F9 | VTEX1629_ERROR_INSUFFICIENT_FIRMWARE_REV | Insufficient firmware revision |
| FFBFFC09FA | VTEX1629_WARNING_FUNC_SLF_TST_NOTSUPP | WARNING: Self-test function not yet supported. |
| FFBFFC09FB | VTEX1629_ERROR_VXI_11 | VXI-11 Error. |
| FFBFFC09FC | VTEX1629_ERROR_TRIGGER_SYSTEM_ERROR | Trigger system error. |
| FFBFFC09FD | VTEX1629_ERROR_OUT_OF_LINK_IDS | Link ID failure. |
| FFBFFC09FE | VTEX1629_ERROR_INVALID_LINK_ID | Invalid link ID. |
| FFBFFC09FF | VTEX1629_ERROR_UNKNOWN_FIRMWARE_ERROR | Unknown firmware error. |
| FFBFFC0A00 | VTEX1629_ERROR_INVALID_NUM_CHANNELS | Invalid number of channels. |
| FFBFFC0A01 | VTEX1629_ERROR_INVALID_CHANNEL | Invalid channel number. |
| FFBFFC0A02 | VTEX1629_ERROR_READ_FIFO_TIMEOUT | Timeout reading FIFO data. |
| FFBFFC0A03 | VTEX1629_ERROR_INVALID_EXITE_VOLT_RANGE | Excite voltage must be within an absolute range of 16.00 V. |
| FFBFFC0A04 | VTEX1629_ERROR_INVALID_EXITE_VOLT_NEG_RANGE | Negative excite voltage must be within the range of 0.0 V and -8.00 V. |
| FFBFFC0A05 | VTEX1629_ERROR_INVALID_EXITE_VOLT_POSS_RANGE | Positive excite voltage must be within the range of 0.0 V and 8.00 V. |
| FFBFFC0A06 | VTEX1629_ERROR_INVALID_GAIN_VALUE | Acceptable gain values are 1. |
| FFBFFC0A07 | VTEX1629_ERROR_INVALID_COMPRES_VALUE | Invalid value for completion resistor type. |
| FFBFFC0A08 | VTEX1629_ERROR_INVALID_INPUT_MUX_VALUE | Input multiplexer set value must be between 0 and 3. |
| FFBFFC0A09 | VTEX1629_ERROR_INVALID_INPUT_CAL_SOURCE | Cal source input value must be between the range of 0 and 12. |
| FFBFFC0A0A | VTEX1629_ERROR_INVALID_BOOLEAN | Invalid Boolean value in parameter. |
| FFBFFC0A0B | VTEX1629_ERROR_INVALID_CAL_OUT_MODE | Invalid value for routing the calibration source. |
| FFBFFC0A0C | VTEX1629_ERROR_INVALID_EU_CONVERSION | Invalid EU conversion type. |
| FFBFFC0A0D | VTEX1629_ERROR_INVALID_FILTER_TYPE | Invalid filter type. |
| FFBFFC0A0E | VTEX1629_ERROR_INVALID_FILTER_TRANSFORM | Invalid filter transform. |
| FFBFFC0A0F | VTEX1629_ERROR_INVALID_FILTER_ORDER | Invalid filter order. |

| Error Code | Error Message | Meaning |
|---|---|---|
| FFBFFC0A10 | VTEX1629_ERROR_STRMING_DATA_ENABLED | Streaming data already enabled. |
| FFBFFC0A11 | VTEX1629_ERROR_ENABLING_STRMING_DATA | Problem enabling streaming data. |
| FFBFFC0A12 | VTEX1629_ERROR_INVALID_BOARD | Hardware related ERROR: Front analog boards. |
| FFBFFC0A13 | VTEX1629_ERROR_INVALID_TRIGGER_SOURCE | Invalid trigger source parameter. |
| FFBFFC0A14 | VTEX1629_ERROR_INVALID_BIT_MASK_VALUE | Invalid bit mask value. |
| FFBFFC0A15 | VTEX1629_ERROR_INVALID_LXI_LINE | Invalid LXI line number. |
| FFBFFC0A16 | VTEX1629_ERROR_INVALID_SHUNT_SOURCE | Invalid shunt source value. |
| FFBFFC0A17 | VTEX1629_ERROR_INVALID_SAMPLE_CLOCK | Invalid sample clock source value. |
| FFBFFC0A18 | VTEX1629_ERROR_DISABLING_STRMING_DATA | Problem disabling streaming data. |
| FFBFFC0A19 | VTEX1629_ERROR_INVALID_LXI_LINE_DIRECTION | Invalid LXI line transmission direction. |
| FFBFFC0A1A | VTEX1629_ERROR_INVALID_LXI_TRANS_SCOPE | Invalid LXI transmission scope. |
| FFBFFC0A1B | VTEX1629_ERROR_INVALID_LXI_MASK_VALUE | Invalid LXI mask value. |
| FFBFFC0A1C | VTEX1629_ERROR_UNABLE_TO_OPEN_LOG | Unable to initialize logging utility. |
| FFBFFC0A1D | VTEX1629_ERROR_SYSTEM_FAILURE | System Failure. |
| FFBFFC0A1E | VTEX1629_ERROR_OUT_OF_MEMORY | Out of memory. |
| FFBFFC0A1F | VTEX1629_ERROR_CONNECT | Connection error. |
| FFBFFC0A20 | VTEX1629_ERROR_IO_ERROR | I/O failure. |
| FFBFFC0A21 | VTEX1629_ERROR_TIMEOUT | Device timeout. |
| FFBFFC0A22 | VTEX1629_ERROR_DEVICE_ERROR | Device error. |
| FFBFFC0A23 | VTEX1629_ERROR_DEVICE_LOCKED | Some operations cannot be executed on a locked device. |
| FFBFFC0A24 | VTEX1629_ERROR_DEVICE_NOT_LOCKED | Some operations cannot be executed on a device that is not locked. |
| FFBFFC0A25 | VTEX1629_ERROR_ACQUISITION_IN_PROGRESS | Some operations cannot be performed while there is an acquisition in progress. |
| FFBFFC0A26 | VTEX1629_ERROR_APP_UNKNOWN_STATUS | Unknown status code. |
| FFBFFC0A27 | VTEX1629_ERROR_APP_UNSUPPORTED_PROC | Unsupported remote procedure call |
| FFBFFC0A28 | VTEX1629_ERROR_DATA_FIFO_ERROR | Error storing data in the FIFO. |
| FFBFFC0A29 | VTEX1629_ERROR_SAMPLE_COUNT | Invalid sample Count. |
| FFBFFC0A2A | VTEX1629_ERROR_INCORRECT_TEDS_ID | Invalid TEDS ID. |
| FFBFFC0A2B | VTEX1629_ERROR_INVALID_ARG | Invalid function argument. |
| FFBFFC0A2C | VTEX1629_ERROR_INVALID_SAMPLING_RATE | Invalid sampling rate. |
| FFBFFC0A2D | VTEX1629_ERROR_INVALID_BRIDGE_CONFIG | Invalid bridge configuration. |
| FFBFFC0A2E | VTEX1629_ERROR_INVALID_SHUNT_SRC_FOR_BRIDGE_TYPE | Invalid shunt source for current bridge configuration. |
| FFBFFC0A2F | VTEX1629_ERROR_INVALID_CONFSRC_TYPE | Invalid confidence source type. |
| FFBFFC0A30 | VTEX1629_ERROR_INVALID_CONFGAIN_SETTING | Invalid confidence gain setting. |
| FFBFFC0A31 | VTEX1629_ERROR_BOX_ALREADY_LOCKED_U | This device has already been locked by you. |
| FFBFFC0A32 | VTEX1629_ERROR_BOX_ALREADY_LOCKED_NU | This device is already locked by another user. |
| FFBFFC0A33 | VTEX1629_ERROR_BOX_ALREADY_UNLOCKED | This device is not locked at this time. |
| FFBFFC0A34 | VTEX1629_ERROR_INVALID_NUM_CONF_ELEMENTS | Invalid number of confidence data elements in scanlist. |
| FFBFFC0A35 | VTEX1629_ERROR_INVALID_CONF_ELEMENT | Invalid confidence data element number in scanlist. |
| FFBFFC0A36 | VTEX1629_ERROR_BESSEL_REQUIRES_ORDER | Bessel filter type requires a filter order. |
| FFBFFC0A37 | VTEX1629_ERROR_IIR_ORDER_OUT_OF_RANGE | IIR filter order out of range. |
| FFBFFC0A38 | VTEX1629_ERROR_INVALID_TRANSFORM_TYPE | Invalid filter transform type. |
| FFBFFC0A39 | VTEX1629_ERROR_IIR_PARM_FAILURE | IIR filter parameter failure. |
| FFBFFC0A3A | VTEX1629_ERROR_INVALID_EU_BRIDGE_TYPE | Invalid bridge type. |
| FFBFFC0A3B | VTEX1629_ERROR_TEDS_DEVICE_NOT_PRESENT | TEDS device is not present. |
| FFBFFC0A3C | VTEX1629_ERROR_TEDS_CHKSUM_FAIL | TEDS check sum failure. |
| FFBFFC0A3D | VTEX1629_TEDS_WRITE_FAIL | Failure writing to TEDS device. |
| FFBFFC0A3E | VTEX1629_ERROR_XML_SAVE_FILE | Writing calibration file. |
| FFBFFC0A3F | VTEX1629_ERROR_XML_LOAD_FILE | Loading calibration file. |
| FFBFFC0A40 | VTEX1629_ERROR_XML_MALFORMED | Corrupted calibration file. |
| FFBFFC0A41 | VTEX1629_ERROR_XML_INVALID_PARM | Invalid parameter for calibration. |

| Error Code | Error Message | Meaning |
|---|---|---|
| FFBFFC0A42 | VTEX1629_ERROR_CAL_UNSET_VALUES | Missing calibration data. |
| FFBFFC0A43 | VTEX1629_ERROR_CAL_FILENAME_TOO_LONG | Calibration file name too long. |
| FFBFFC0A44 | VTEX1629_ERROR_CAL_NOMINAL_CAL_NOT_PRESENT | Calibration file missing. |
| FFBFFC0A45 | VTEX1629_ERROR_INVALID_CUTOFF_FREQ | Invalid filter cutoff frequency. |
| FFBFFC0A46 | VTEX1629_ERROR_SHUNT_FP_VAL_UNEQUAL | Front panel resistor value is not equal for blocks of 16 channels. |
| FFBFFC0A47 | VTEX1629_ERROR_LXI_INPUT_CONFIG_CONFLICT | LXI input configuration conflicts with an existing output. |
| FFBFFC0A48 | VTEX1629_ERROR_LXI_OUTPUT_CONFIG_CONFLICT | LXI output configuration conflicts with an existing input. |
| FFBFFC0A49 | VTEX1629_ERROR_LXI_INPUT_MISCONFIGURATION | The specified LXI input line is not configured as an input. |
| FFBFFC0A4A | VTEX1629_ERROR_LXI_OUTPUT_MISCONFIGURATION | The specified LXI output line is not configured as an output. |
| FFBFFC0A4B | VTEX1629_ERROR_LXI_OUTPUT_CONFLICT | The specified LXI output conflicts with another output. |
| FFBFFC0A4C | VTEX1629_ERROR_INVALID_MODE_FOR_SOFT_TRIGGERARM | Software arm and trigger is only available in trigger pattern mode. |
| FFBFFC0A4D | VTEX1629_ERROR_PRETRIGGER_NOT_SUPPORTED_YET | Pre-trigger samples are not yet supported. |
| FFBFFC0A4E | VTEX1629_ERROR_NO_SERIAL_NUMBER | No serial number programmed on device. |
| FFBFFC0A4F | VTEX1629_ERROR_FIFO_STREAMING_ENABLED | Read FIFO not valid. Streaming interface is enabled. |
| FFBFFC0A50 | VTEX1629_ERROR_INSUFFICIENT_UPTIME_FOR_CAL | Insufficient instrument uptime to perform calibration. Please see User's Manual for more details. |
| FFBFFC0A51 | VTEX1629_ERROR_CAL_SELF_CAL_NOT_PRESENT | Self-calibration file not present. |
| FFBFFC0A51 | VTEX1629_ERROR_CAL_SELF_CAL_NOT_PRESENT | Self-calibration data is not present. |
| FFBFFC0A52 | VTEX1629_ERROR_FPGA_TRIGINIT_NOT_IN_IDLE_STATE | Trigger state machine is not in the idle state. |
| FFBFFC0A53 | VTEX1629_ERROR_FPGA_TRIGINIT_IDLE_TO_ACQ_TIMEOUT | Time out in trigger system. |
| FFBFFC0A54 | VTEX1629_ERROR_NO_MAC_ADDR | No MAC Address programmed on device. |
| FFBFFC0A55 | VTEX1629_ERROR_CAL_IN_PROGRESS | Some operations cannot be performed while self-calibration is in progress. |
| FFBFFC0A56 | VTEX1629_ERROR_CAL_BLOCKS_PRESENT | Calibration blocks detected. |
| FFBFFC0A57 | VTEX1629_ERROR_XML_REQUIRE_TEDS | Required TEDS ID not available. |
| FFBFFC0A58 | VTEX1629_ERROR_TRIGINIT_FPGA_ADC_INTR_OFF | A soft sync is required before trigger initialization can be performed. |
| FFBFFC0A59 | VTEX1629_ERROR_SHUNT_FPS_ENABLED_ON_MULTIPLE_CHANNELS | Within sixteen channel blocks |
| FFBFFC0A5A | VTEX1629_ERROR_INVALID_TRIGGER_COUNT | Trigger/arm count must be greater than zero. |
| FFBFFC0A5B | VTEX1629_ERROR_SET_INT_SHUNT_VALUE | Internal shunt resistor is not user defined. |
| FFBFFC0A5C | VTEX1629_ERROR_INVALID_PULLUP_VALUE | Invalid pullup value. |
| FFBFFC0A5D | VTEX1629_ERROR_INVALID_DIO_DIRECTION | Invalid DIO direction. |
| FFBFFC0A5E | VTEX1629_ERROR_INVALID_URN_SIZE | Invalid MLAN URN size. |
| FFBFFC0A5F | VTEX1629_ERROR_INVALID_USER_DATA | Invalid user data. |
| FFBFFC0A60 | VTEX1629_ERROR_CONFIG_FILE_NOT_PRESENT | Specified configuration file not present. |
| FFBFFC0A61 | VTEX1629_ERROR_INCONSISTENT_CONF_WT | Confidence weight is not consistent across all channels. |
| FFBFFC0A62 | VTEX1629_ERROR_INCONSISTENT_TRIGGER_TIMESTAMP | Trigger timestamp is not consistent across all channels. |
| FFBFFC0A63 | VTEX1629_ERROR_INCONSISTENT_SYNC_TIMESTAMP | Sync timestamp is not consistent across all channels. |
| FFBFFC0A64 | VTEX1629_ERROR_MLAN_BUFFER_OVERFLOW | MicroLAN buffer overflow. |
| FFBFFC0A65 | VTEX1629_ERROR_MLAN_BUFFER_LEN_MISMATCH | MicroLAN buffer length mismatch. |
| FFBFFC0A66 | VTEX1629_ERROR_UNSUPPORTED_TEDS_DEVICE | Unsupported TEDS device. |

| Error Code | Error Message | Meaning |
|---|---|---|
| FFBFFC0A67 | VTEX1629_ERROR_INCORRECT_TEDS_ID_LEN | Incorrect TEDS device ID length. |
| FFBFFC0A68 | VTEX1629_ERROR_INCORRECT_TEDS_DATA_LEN | Incorrect TEDS data length. |
| FFBFFC0A69 | VTEX1629_ERROR_TEDS_DATA_READBACK_FAILURE | TEDS device readback failure. |
| FFBFFC0A6A | VTEX1629_ERROR_FIFO_OVERFLOW | Data FIFO overflowed. |
| FFBFFC0A6B | VTEX1629_ERROR_EXTERNAL_ADC_MASTER_CLOCK_LOST | External ADC master clock was lost. |
| FFBFFC0A6C | VTEX1629_ERROR_INVALID_CONF_FILTER_WT | Invalid confidence filter weight. |
| FFBFFC0A6D | VTEX1629_ERROR_IIR_NOT_SUPPORTED_FOR_CURR_FS | IIR filter is not supported for the current sample frequency. |
| FFBFFC0A6E | VTEX1629_ERROR_CAL_FACTORY_MODE_REQUIRED | The system must be in factory mode to perform this action. |
| FFBFFC0A6F | VTEX1629_ERROR_FS_NOT_SUPPORTED_WHEN_CONF_EN | This sample frequency is not supported when confidence sources are enabled. |
| FFBFFC0A70 | VTEX1629_ERROR_CONF_OPS_NOT_SUPPORTED_FOR_CURR_FS | Confidence operations are not supported at this sample frequency. |
| FFBFFC0A71 | VTEX1629_ERROR_INVALID_VI_SESSION | Invalid or null session to the instrument passed as parameter to function. |
| FFBFFC0A72 | VTEX1629_ERROR_RUNTIME_INVALID_PARAMETER | Invalid or null parameter passed to function. |
| FFBFFC0A73 | VTEX1629_ERROR_ILLEGAL_TRIGGER_SAMPLE_COUNT_SPECIFIED | Illegal trigger sample count. |
| FFBFFC0A74 | VTEX1629_ERROR_ILLEGAL_PRETRIGGER_SAMPLE_COUNT_SPECIFIED | Illegal pretrigger sample count. |
| FFBFFC0A75 | VTEX1629_ERROR_INVALID_ARM_SOURCE | Invalid arm source parameter. |
| FFBFFC0A76 | VTEX1629_ERROR_INVALID_ARM_COUNT | Invalid arm count parameter. |
| FFBFFC0A77 | VTEX1629_ERROR_INVALID_EXCITATION_SRC | Invalid excitation source. Use VTEX1629_EXCITE_SRC_LOCAL or VTEX1629_EXCITE_SRC_REMOTE. |
| FFBFFC0A78 | VTEX1629_ERROR_INVALID_SYNCH_SOURCE | Invalid synch source parameter. |
| FFBFFC0A79 | VTEX1629_ERROR_INVALID_TRIGGER_DELAY | Invalid trigger delay parameter. |
| FFBFFC0A7A | VTEX1629_ERROR_INVALID_TIMER_PERIOD | Invalid timer period parameter. |
| FFBFFC0A7B | VTEX1629_ERROR_INVALID_MLAN_DATA_LEN | Invalid MLAN data length parameter. |
| FFBFFC0A7C | VTEX1629_ERROR_CALCULATING_SETTLING_TIME | Error while calculating settling time. |
| FFBFFC0A7D | VTEX1629_ERROR_INVALID_ARM_DELAY | Invalid arm delay parameter. |
| FFBFFC0A7E | VTEX1629_ERROR_RESERVED_CODE_1220 | Reserved Error Code 0x1220. |
| FFBFFC0A7F | VTEX1629_ERROR_INCONSISTENT_SAMP_CLK_SRC | Clock source is not consistent across all channels and digital backend. |
| FFBFFC0A80 | VTEX1629_ERROR_HD_INVALID_STATE_TRANSITION | Improper sequence of hard disk commands. |
| FFBFFC0A81 | VTEX1629_ERROR_HD_MOUNT_FAIL | Error mounting hard disk. |
| FFBFFC0A82 | VTEX1629_ERROR_HD_UNMOUNT_FAIL | Error unmounting hard disk. |
| FFBFFC0A83 | VTEX1629_ERROR_HD_FILE_EXISTS | Hard disk file exists. |
| FFBFFC0A84 | VTEX1629_ERROR_HD_NO_DATA_FILE | No data file. |
| FFBFFC0A85 | VTEX1629_ERROR_HD_NO_INDEX_FILE | No index file. |
| FFBFFC0A86 | VTEX1629_ERROR_HD_FILE_OPEN_FAIL | Failure when attempting to open file. |
| FFBFFC0A87 | VTEX1629_ERROR_HD_NOT_SUPPORTED | Hard disk not supported. |
| FFBFFC0A88 | VTEX1629_ERROR_HD_HANG | Hard disk hang. |
| FFBFFC0A89 | VTEX1629_ERROR_NET_HANG | Network hang. |
| FFBFFC0A8A | VTEX1629_ERROR_INVALID_EUCONV_EXCITATION_VOLT_SRC | Illegal mode parameter. |
| FFBFFC0A8B | VTEX1629_ERROR_ARG_OUT_OF_RANGE | Argument out of range. |
| FFBFFC0A8C | VTEX1629_ERROR_INVALID_FS_FOR_THIS_SCANLIST | Maximum sample rate exceeded for current scanlist. Maximum is 12.5 kSa/s on 16 channels constrained to a bank of 1-15. |
| FFBFFC0A8D | VTEX1629_ERROR_INVALID_SCANLIST_FOR_THIS_FS | Maximum channels in scanlist exceeded for current sample rate. Maximum is 16 channels constrained to a bank of 1-15. |
| FFBFFC0A8E | VTEX1629_ERROR_DIO_NO_INPUT_BANK_CONFIGURED | Digital error. No input bank configured. |
| FFBFFC0A8F | VTEX1629_ERROR_DIO_NO_OUTPUT_BANK_CONFIGURED | Digital error. No output bank configured. |
| FFBFFC0A90 | VTEX1629_ERROR_DIO_CONFLICT_OUTPUT_IN_USE | Digital IO conflict because output is in use. |

| Error Code | Error Message | Meaning |
|---|---|---|
| FFBFFC0A91 | VTEX1629_ERROR_DIO_CONFLICT_DUPLICATE_ENTRY | Requested event configuration conflicts with existing configuration for the input event and/or output action specified. |
| FFBFFC0A92 | VTEX1629_ERROR_DIO_INVALID_INPUT_LINE | Invalid digital input line. |
| FFBFFC0A93 | VTEX1629_ERROR_DIO_INVALID_OUTPUT_LINE | Invalid digital output line. |
| FFBFFC0A94 | VTEX1629_ERROR_DIO_INVALID_INPUT_EVENT | Invalid input event. |
| FFBFFC0A95 | VTEX1629_ERROR_DIO_INVALID_OUTPUT_ACTION | Invalid output action. |
| FFBFFC0A96 | VTEX1629_ERROR_QTR_BRIDGE_NOT_SELECTED | Channel is not in quarter bridge configuration. |
| FFBFFC0A97 | VTEX1629_ERROR_TRIGINIT_NEED_SOFTSYNC | Triginit needs softsync first. |
| FFBFFC0A98 | VTEX1629_ERROR_FS_NOT_SUPPORTED_FOR_DYNAMIC_EXCITATION_EU | Sampling frequency greater than 1 kHz not supported while in current mode. |
| FFBFFC0A99 | VTEX1629_ERROR_DYNAMIC_EXCITATION_EU_NOT_SUPPORTED_AT_CURR_FS | This mode is not supported while sampling rate is greater than 1 kHz. |
| FFBFFC0A9A | VTEX1629_ERROR_DYNAMIC_EXCITATION_EU_CONF_EXCITEOUT_SRC_NOT_ENABLED | The confidence sources VTEX1629_CONFSRC_EXCITE_POS and VTEX1629_CONFSRC_EXCITE_NEG must be in the confidence scanlist in order to enable dynamic excitation EU. |
| FFBFFC0A9B | VTEX1629_ERROR_DYNAMIC_EXCITATION_EU_NEEDS_CONF_EXCITEOUT_SRC | Can not disable confidence excitation source while dynamic EU conversion is in progress. |
| FFBFFC0A9C | VTEX1629_ERROR_CONF_LIMIT_INVALID_REPORTING_TYPE | Invalid reporting type. |
| FFBFFC0A9D | VTEX1629_ERROR_BRIDGE_LIMIT_NOT_SUPPORTED_AT_CURR_FS | Can not enable bridge limit checking when the bridge sampling frequency is greater than 1 kHz. |
| FFBFFC0A9E | VTEX1629_ERROR_FS_NOT_SUPPORTED_FOR_BRIDGE_LIMIT_CHECKING | Can not set sampling frequency greater than 1 kHz while in the current mode. |
| FFBFFC0A9F | VTEX1629_ERROR_CONF_LIMIT_NOT_SUPPORTED_AT_CURR_FS | Summary reporting or detailed reporting not supported for sampling frequency greater than 1 kHz. |
| FFBFFC0AA0 | VTEX1629_ERROR_FS_NOT_SUPPORTED_FOR_CONF_LIMIT_CHECKING | Can not set sampling frequency greater than 1 kHz while in current mode. |
| FFBFFC0AA1 | VTEX1629_ERROR_HW_DOES_NOT_SUPPORT_CONF_EXCITEOUT_BUFF_SRC | Current hardware prohibits enabling EXCITEOUT_BUFF. This is only available on newer hardware revisions. |
| FFBFFC0AA2 | VTEX1629_ERROR_CAL_FILE_NOT_PRESENT | Calibration file not present. |
| FFBFFC0AA3 | VTEX1629_ERROR_CAL_FILE_TOO_LARGE | Calibration file too large. |
| FFBFFC0AA4 | VTEX1629_VXI11_ERROR_NO_ERROR | VXI11 No error. |
| FFBFFC0AA5 | VTEX1629_VXI11_ERROR_SYNTAX_ERROR | VXI11 Syntax error. |
| FFBFFC0AA6 | VTEX1629_VXI11_ERROR_DEVICE_NOT_ACCESSIBLE | VXI11 Device not accessible. |
| FFBFFC0AA7 | VTEX1629_VXI11_ERROR_INVALID_LINK_IDENTIFIER | VXI11 Invalid link identifier. |
| FFBFFC0AA8 | VTEX1629_VXI11_ERROR_PARAMETER_ERROR | VXI11 Parameter error. |
| FFBFFC0AA9 | VTEX1629_VXI11_ERROR_CHANNEL_NOT_ESTABLISHED | VXI11 Channel not established. |
| FFBFFC0AAA | VTEX1629_VXI11_ERROR_OPERATION_NOT_SUPPORTED | VXI11 Operation not supported. |
| FFBFFC0AAB | VTEX1629_VXI11_ERROR_OUT_OF_RESOURCES | VXI11 Out of resources. |
| FFBFFC0AAC | VTEX1629_VXI11_ERROR_DEVICE_LOCKED_BY_ANOTHER_LINK | VXI11 Device locked by another link. |
| FFBFFC0AAD | VTEX1629_VXI11_ERROR_NO_LOCK_HELD_BY_THIS_LINK | VXI11 No lock held by this link. |
| FFBFFC0AAE | VTEX1629_VXI11_ERROR_IO_TIMEOUT | VXI11 IO timeout. |
| FFBFFC0AAF | VTEX1629_VXI11_ERROR_IO_ERROR | VXI11 IO error. |
| FFBFFC0AB0 | VTEX1629_VXI11_ERROR_INVALID_ADDRESS | VXI11 Invalid address. |
| FFBFFC0AB1 | VTEX1629_VXI11_ERROR_ABORT | VXI11 Abort. |
| FFBFFC0AB2 | VTEX1629_VXI11_ERROR_CHANNEL_ALREADY_ESTABLISHED | VXI11 Channel already established. |
| FFBFFC0AB3 | VTEX1629_ERROR_BRIDGE_LIMIT_CHECKING_NOT_SUPPORTED_AT_CURR_FS | Bridge limit checking not supported at current sampling frequency. |
| FFBFFC0AB4 | VTEX1629_ERROR_FS_NOT_SUPPORTED_WHEN_BRIDGE_LIMIT_CHECKING_ENABLED | Sample frequency not supported when bridge limit checking is enabled. |

| Error Code | Error Message | Meaning |
|---|---|---|
| FFBFFC0AB5 | VTEX1629_ERROR_CONF_LIMIT_CHECKING_NOT_SUPPORTED_AT_CURR_FS | Confidence limit checking not supported at current sampling frequency. |
| FFBFFC0AB6 | VTEX1629_ERROR_FS_NOT_SUPPORTED_WHEN_CONF_LIMIT_CHECKING_ENABLED | Sample frequency not supported when confidence limit checking is enabled. |
| FFBFFC0AB7 | VTEX1629_ERROR_INVALID_COEFFICIENT_SELECTOR | Invalid coefficientSelector. |
| FFBFFC0AB8 | VTEX1629_ERROR_XML_PARSE_ERROR | Error parsing XML file. File structure might be incorrect. |
| FFBFFC0AB9 | VTEX1629_ERROR_FAILED_TO_LOAD_FACTORY_CAL | Unable to download nominal factory calibration file from instrument for use in reporting self-calibration coefficients. |
| FFBFFC0ABA | VTEX1629_ERROR_PASSED_IN_ARRAYS_NOT_LARGE_ENOUGH | The passed in array(s) are not large enough to store all the requested data from the instrument. |
| FFBFFC0ABB | VTEX1629_ERROR_SENSOR_LED_LIT_UP_ON_ANOTHER_CHANNEL | Only one sensor LED can be on at a time. |
| FFBFFC0ABC | VTEX1629_ERROR_TEDS_OPS_NOT_SUPPORTED_DURING_SENSOR_IDENTIFY | TEDS operations not allowed on the same analog board (channels 1-15). |
| FFBFFC0ABD | VTEX1629_ERROR_FRONT_PANEL_SHUNT_ACTIVE | Sensor identification not allowed on the same analog board (channels 1-15). |
| FFBFFC0ABE | VTEX1629_ERROR_TEDS_SHUNT_ACTIVE | Sensor identification not allowed on the same analog board (channels 1-15). |
| FFBFFC0ABF | VTEX1629_ERROR_INTERNAL_SHUNT_REMOTE_ACTIVE | Sensor identification not allowed on the same analog board (channels 1-15). |
| FFBFFC0AC0 | VTEX1629_ERROR_SENSOR_LED_ACTIVE | Operation not allowed when the sensor identification LED is on. |
| FFBFFC0AC1 | VTEX1629_ERROR_TEDS_SHUNT_ENABLED_ON_MULTIPLE_CHANNELS | TEDS shunt is enabled on multiple channels belonging to the same analog board (channels 1-15). |
| FFBFFC0AC2 | VTEX1629_ERROR_TEDS_OPS_NOT_SUPPORTED_WHEN_USING_TEDS_SHUNT | TEDS operation not supported when the TEDS shunt is enabled on the same analog board (channels 1-15). |
| FFBFFC0AC3 | VTEX1629_ERROR_SCANLIST_CONTAINS_BAD_CHANNEL | Scanlist contains channels that failed calibration. |
| FFBFFC0AC7 | VTEX1629_ERROR_INVALID_LID | Invalid link ID. |
| FFBFFC0AC8 | VTEX1629_ERROR_NOT_CONNECTED | Error: Not connected. |
| FFBFFC0AC9 | VTEX1629_ERROR_ALREADY_STREAMING | Error: Already streaming. |
| FFBFFC0ACA | VTEX1629_ERROR_APP_QUERY_RESPONSE_MISMATCH | Application query response mismatch. |
| FFBFFC0ACA | VTEX1629_ERROR_APP_QUERY_RESPONSE_MISMATCH | Application query response mismatch. |
| FFBFFC0ACB | VTEX1629_ERROR_DATA_READER_THREAD_ERROR | Data reader thread error. |
| FFBFFC0ACB | VTEX1629_ERROR_DATA_READER_THREAD_ERROR | Data reader thread error. |
| FFBFFC0ACC | VTEX1629_ERROR_INCONSISTENT_CONF_SCANLIST | Inconsistent confidence scanlist. |
| FFBFFC0ACC | VTEX1629_ERROR_INCONSISTENT_CONF_SCANLIST | Inconsistent confidence scanlist. |
| FFBFFC0ACD | VTEX1629_ERROR_INVALID_CHANNEL_LIST_LENGTH | Invalid channel list length. |
| FFBFFC0ACE | VTEX1629_ERROR_INVALID_GAIN | Invalid gain. |
| FFBFFC0ACF | VTEX1629_ERROR_INVALID_CONVERSION_CONFIG | Invalid conversion configuration. |
| FFBFFC0AD1 | VTEX1629_ERROR_INVALID_CONF_SCANLIST_LENGTH | Invalid confidence scanlist length. |
| FFBFFC0AD2 | VTEX1629_ERROR_INVALID_CONF_SCANLIST_ENTRY | Invalid confidence scanlist entry. |
| FFBFFC0AD3 | VTEX1629_ERROR_INVALID_PARAMETER_LIST_LENGTH | Invalid parameter list length. |
| FFBFFC0AD4 | VTEX1629_ERROR_INVALID_LEADWIRE_RESISTANCE_VALUE | Invalid lead wire resistance value. |
| FFBFFC0AD5 | VTEX1629_ERROR_INVALID_LEADWIRE_DESENSITIZATION_FACTOR | Invalid lead wire desensitization factor. |
| FFBFFC0AD6 | VTEX1629_ERROR_INVALID_EXCITE_POS_VOLT | Invalid positive excitation voltage. |
| FFBFFC0AD7 | VTEX1629_ERROR_INVALID_EXCITE_NEG_VOLT | Invalid negative excitation voltage. |
| FFBFFC0AD8 | VTEX1629_ERROR_INVALID_GAIN_SETTING | Invalid gain setting. |
| FFBFFC0AD9 | VTEX1629_ERROR_INVALID_COMPRES_TYPE | Invalid completion resistor type. |
| FFBFFC0ADA | VTEX1629_ERROR_INVALID_SHUNT_SRC | Invalid shunt source. |
| FFBFFC0ADB | VTEX1629_ERROR_CONF_TOO_MANY_ENTRIES | Too many conf entries. |
| FFBFFC0ADC | VTEX1629_ERROR_FPGA_TRIGABORT_ACQ_TO_IDLE_TIMEOUT | FPGA acquisition to idle timeout. |
| FFBFFC0ADD | VTEX1629_ERROR_TEDS_INFO_CRC_FAILURE | TEDS info CRC failure. |
| FFBFFC0ADE | VTEX1629_ERROR_TEDS_ID_CRC_FAILURE | TEDS ID CRC failure. |

| Error Code | Error Message | Meaning |
|---|---|---|
| FFBFFC0ADF | VTEX1629_ERROR_TEDS_WRITE_SCRATCHPAD_RDBK | TEDS write scratchpad readback error. |
| FFBFFC0AE0 | VTEX1629_ERROR_TEDS_WRITE_INFO_RDBK | TEDS write info readback error. |
| FFBFFC0AE1 | VTEX1629_ERROR_CAL_FACTORY_CAL_NOT_PRESENT | Factory calibration not present. |
| FFBFFC0AE2 | VTEX1629_ERROR_CAL_INCOMPLETE_CAL | Incomplete calibration. |
| FFBFFC0AE3 | VTEX1629_ERROR_INVALID_MODE_FOR_SOFT_TRIGGER | Invalid mode for soft trigger. |
| FFBFFC0AE4 | VTEX1629_ERROR_INVALID_MODE_FOR_SOFT_ARM | Invalid mode for soft arm. |
| FFBFFC0AE5 | VTEX1629_ERROR_INPUT_BRIDGE_NOT_COMPLETE | Input bridge not complete. |
| FFBFFC0AE6 | VTEX1629_WARNING_CAL_BUFFERSIZE_SMALLER_THAN_ACTUAL_SIZE | Warning: the passed in buffer / buffer size are not large enough to hold all of the cal data. Only buffer size amount was stored in the buffer. |
| FFBFFC0AED | VTEX1629_ERROR_PTP_NOT_RUNNING | Error: PTP not running. |
| FFBFFC0AEE | VTEX1629_ERROR_PROFILING_NOT_ENABLED | Error: Profiling not enabled. |
| FFBFFC0AEF | VTEX1629_ERROR_DYNAMIC_EXCITATION_EU_ENABLED | Can not set EU conversion on a channel where dynamic excitation EU conversion mode is enabled. |
| FFBFFC0AF0 | VTEX1629_TEMP_DIR_WRITE_PROTECTED | Error: The temporary directory C:\Temp either does not exist or is write-protected. |
| None | VI_WARN_NSUP_SELF_TEST | WARNING: Self-Test not supported. |
| None | VI_WARN_NSUP_ERROR_QUERY | WARNING: Error Query not supported. |

# APPENDIX A

# MULTI-INSTRUMENT OPERATION

## INTRODUCTION

While a single EX1629 provides a sophisticated strain gage instrument, the ability to connect multiple instruments together as an ensemble, creating a single, high sample-rate, high channel-count acquisition system provides enormous power. This section describes how configure such systems. Please refer to *Synchronizing Multiple Instruments* for a detailed example with software.

In a multi-instrument configuration, one device is selected as a master and one or more devices are slaves. The master device sources (outputs) its sample clock and synchronization signal. These signals allow the master and slave(s) to synchronize their ADCs so that sampled data is phase-aligned and coherently acquired. In addition, depending on the triggering needs, the master device may output an arm and/or a trigger signal. This section describes some common multi-instrument configurations.

### *Distributing Sample Clock and Synchronization Signals*

To guarantee that all instruments in a multi-instrument configuration acquire coherent, phase-aligned data, a single instrument (the "master") is chosen by the application designer to share its internal sample clock and synchronization signals with one or more slave instruments. These signals are distributed to the slaves via the LXI Trigger Bus.

| | |
|---|---|
| **NOTE** | The sample clock and synchronization signals may only be distributed via the LXI Trigger Bus. Further, only an EX1629 instrument can provide these signals – no external oscillators or synchronization signals are supported. |

Due to hardware constraints, only certain LXI Trigger Bus lines may be used to distribute the sample clock and synchronization signals (see vtex1629_set_sample_clock_source and vtex1629_set_synch_source for more detailed information). Each instrument has two LXI Trigger Bus Connectors that are connected together internally – they are functionally identical. When the LXI Trigger Bus is used, both connectors must be attached to either an LXI Trigger Bus Cable or a Trigger Bus Terminator. All LXI Trigger Bus configurations must involve two LXI Trigger Bus Terminators, one at each end of the bus.

| | |
|---|---|
| **NOTE** | When the LXI Trigger Bus is not being used (e.g., standalone mode) no LXI Trigger Bus Terminators are needed. |

Generally, there are two topologies for distributing these signals: Daisy-Chain and Star. In the daisy-chain topology, devices are connected serially, with two LXI Trigger Bus Cables connected to each instrument, except the two instruments at the ends of the daisy-chain, both of which have only a single LXI Trigger Bus Cable connected, with an LXI Trigger Bus Terminator on the other connector. The Daisy-Chain configuration is suitable for smaller ensembles of instruments, two to twenty.

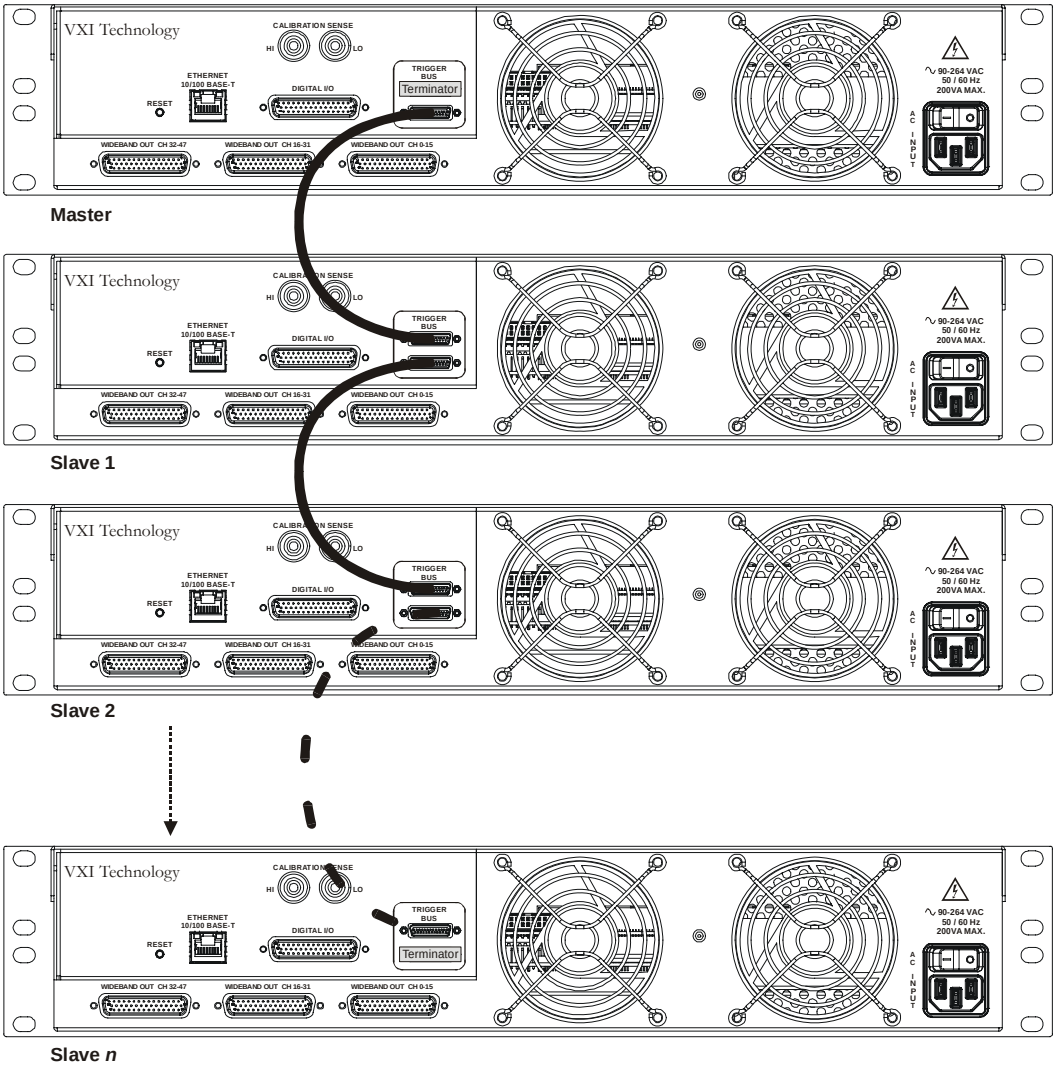



**FIGURE A-1: DAISY-CHAIN CONFIGURATION**

| **NOTE** | Due to signal integrity concerns, the maximum number of EX1629s that should be connected in a daisy-chain configuration is 20. |
|---|---|

The Star topology connects the Master and Slave(s) in parallel. In the Star topology, an LXI Trigger Bus Hub or Switch (e.g., EX2108) is used to distribute the LXI Trigger Bus signals. The Star topology has two potential advantages over the Daisy-Chain configuration: it can provide better cable-length matching (and thus signal propagation time matching), and more instruments can be connected together. In the Daisy-Chain topology, slave devices are connected to the master at various points along the shared bus, with differing amounts of cable between them and the master device. This causes signals to reach each of the slaves at slightly different times (a good rule of thumb is that electrical signals travel along cables at approximately 1 ft/ns).

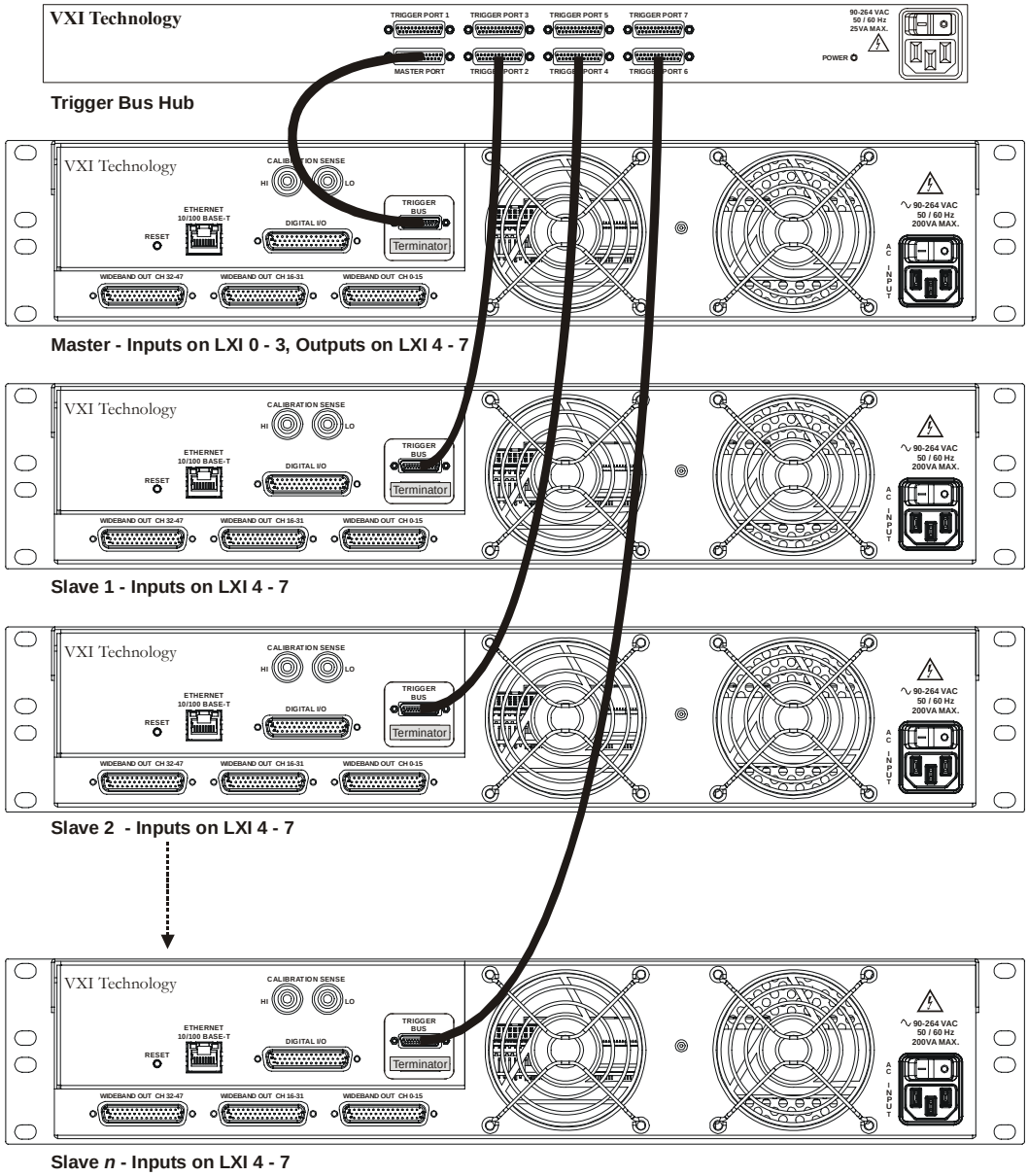



**FIGURE A-2: STAR CONFIGURATION**

For the tightest correlation between instruments, LXI Trigger Bus Cables can be length matched, and the slave devices connected via an LXI Trigger Bus Hub/Switch. By suitably configuring the LXI Trigger Bus Hub/Switch, and using the **lxiOutput** and **lxiInput** parameters of the functions used to configure the sample clock, synchronization, arm and trigger signals, even the propagation times as seen by the Master can be addressed. By configuring the Master device to output a signal (**lxiOutput**) on an LXI Trigger Bus line, and accept the signal (**lxiInput**) on a different line, the Master device can be made to see the same timing as the Slave devices. For example, the sample clock may be output on LXI Line 0 and input on LXI Line 4, with the LXI Trigger Bus Hub/Switch connecting LXI Line 0 to LXI Line 4. For more on configuration configuring the signals, see the vtex1629_set_sample_clock_source, vtex1629_set_synch_source, vtex1629_set_pattern_arm_configuration, and vtex1629_set_pattern_trig_configuration functions.

In addition, the Star topology may allow a larger number of instruments to be connected together, through the repeater capability of the LXI Trigger Bus Hub/Switch. Signals passing through the LXI Trigger Bus Hub/Switch are actively driven with transceivers, so the signal integrity issues that limit the number of devices that may be connected via the Daisy-Chain topology are mitigated.

In a "pure" Star topology, each instrument is directly connected to the LXI Trigger Bus Hub/Switch. In a "hybrid" Star (aka Star/Daisy-Chain) topology, an LXI Trigger Bus Hub/Switch is used to connect a number of Daisy-Chains together, allowing hundreds of EX1629 instruments to be connected together. It should be noted that in these Star/Daisy-Chain topologies, it is not possible to perfectly match propagation time, as in the Daisy-Chain configuration. For many applications, though, the small variations in propagation delays are not important. For very large acquisition systems, the sample rates are typically low (typically on the order of a few hundred samples per second), so the small propagation delay induced phase errors are negligible.

### *Triggering*

| **NOTE** | The discussion of triggering below applies equally to arming as well. |
|---|---|

There are several triggering modes that may be used in a multi-instrument configuration, depending on the application's needs. Fundamentally, all trigger and arm events are transmitted across the EX1629's internal LXI Trigger Bus to the actual acquisition hardware. The events may come from the external LXI Trigger Bus, or from other trigger domains, such as DIO, Timer, and software. Events from these other trigger domains may be transformed into LXI Trigger Bus events via the "pattern" trigger source. This "pattern" source is useful in several situations, including software triggering an ensemble of instruments.

The simplest triggering mode is to configure all instruments to accept an external LXI Trigger Bus event, such as a positive (rising) edge on LXI Line 3. The Master and the Slave(s) instruments are configured identically and some external device is used to generate an LXI Trigger Bus event.

The next simplest mode, which works well for a large number of data acquisition applications, is to trigger the Master device under software control, the Master device being configured to trigger the slaves, in turn. For this mode of operation, the Master device is configured with a trigger source of "pattern" (vtex1629_set_trigger_source). No DIO, LXI, or timer events are configured as part of the pattern; only the **lxiInput** and **lxiOutput** parameters need to be specified (vtex1629_set_pattern_trig_configuration). Since the Software Trigger is always valid while in pattern mode, it can be used to trigger the pattern. When the pattern is triggered, via the vtex1629_soft_trig function, an edge is generated on the **lxiOutput** line configured in the pattern. The Master device acquisition is triggered based on the **lxiInput** setting – it may be the same as **lxiOutput**, or different, depending on if an LXI Trigger Bus Hub/Switch is being used to route signals. Slave devices are configured with a non-pattern trigger source equal to the LXI line specified in the **lxiOutput** configuration of the Master. To trigger the ensemble of instruments in this configuration, a vtex1629_soft_trig function is used on the Master instrument. All instruments' Trigger subsystems must be initialized (vtex1629_trig_init) prior to the software trigger.

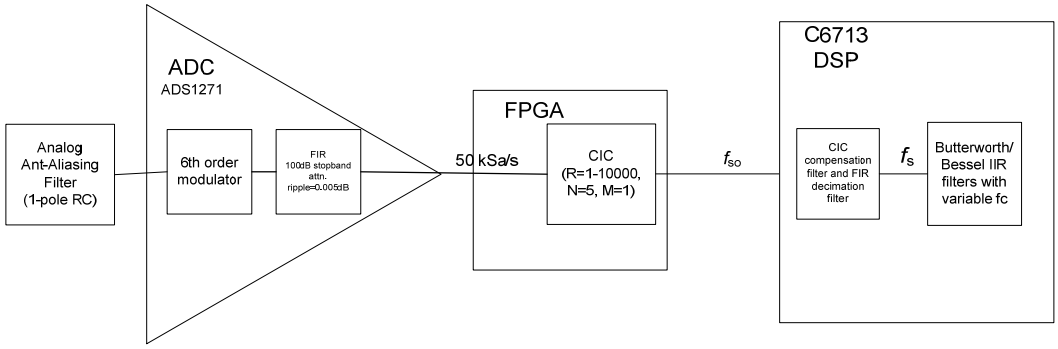
Another common configuration handles the situation when the trigger source is a TTL signal. For simplicity, this signal can be routed to the Master instrument and the Master instrument can be configured to generate an event on the LXI Trigger Bus to trigger itself and the Slave instruments. This configuration is similar to the above, software-triggered configuration, except that the Master's trigger pattern is configured with a DIO event or level appropriate to the external TTL event. The Master device, upon receiving the DIO event will generate a rising edge on the LXI line specified in the pattern's **lxiOutput** signal. Slaves are configured to trigger on a positive (rising) edge.

# APPENDIX B

# EX1629 FILTERING

## INTRODUCTION

The diagram below details the filter chain present in the EX1629 channels, which contains both analog and digital components. These components are discussed sequentially in the following sections.




where $f_{so}$ = input frequency to FIR decimator, $f_s$ = user sample rate

**FIGURE B-1: EX1629 FILTER CHAIN**

### *Analog Anti-Aliasing Filter*

The analog anti-aliasing filter consists of a single-pole RC filter with R = 5.62 kΩ and C = 470 pF, which has a -3 dB point of $f_c = 1/(2\pi*RC)$, or 60.254 kHz.

The rejection at 6.25 MHz (the ADC sigma delta modulator sampling rate) is:

$$f = \frac{1}{1+\left(\frac{f}{f_c^2}\right)} = \text{-40.52426 dB}$$

## DIGITAL FILTERS

The following table provides detailed information concerning the digital filters used by the ADC of the EX1629. For more information on digital filtering, *Discrete-Time Signal Processing (2nd Edition)* by Alan Oppenheim could be used as a reference.

| $f_s$ | $f_c$ max | CIC dec | FIR dec | 60 Hz rejection (dB) | GD for IIR N=4 (samples) | GD for IIR N=6 (samples) | GD for IIR N=8 (samples) | GD for IIR N=10 (samples) | Delay up to IIR (samples) |
|---|---|---|---|---|---|---|---|---|---|
| 50000 | 22650 | 1 | 1 | | | | | | 39.00000 |
| 25000 | 11500 | 1 | 2 | | | | | | 51.00000 |
| 12500 | 5255 | 1 | 4 | | | | | | 25.62000 |
| 10000 | 4005 | 1 | 5 | | | | | | 20.50000 |
| 6250 | 2130 | 1 | 8 | | | | | | 12.81200 |
| 5000 | 1505 | 1 | 10 | | | | | | 10.25000 |
| 3125 | 937.50 | 2 | 8 | | | | | | 9.71875 |
| 2500 | 970.00 | 4 | 5 | | | | | | 15.02500 |
| 2000 | 776.00 | 5 | 5 | | | | | | 14.66000 |
| 1250 | 485.00 | 8 | 5 | | | | | | 14.11250 |
| 1000 | 388.00 | 10 | 5 | | 0.949274 | 1.403572 | 1.862067 | 2.322195 | 13.93000 |
| 833.33 | 323.33 | 12 | 5 | | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.80833 |
| 625 | 242.50 | 16 | 5 | | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.65625 |
| 500 | 194.00 | 20 | 5 | | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.56500 |
| 400 | 155.20 | 25 | 5 | | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.49200 |
| 250 | 97.00 | 40 | 5 | | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.38250 |
| 200 | 77.60 | 50 | 5 | | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.34600 |
| 166.67 | 64.67 | 60 | 5 | | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.32167 |
| 125 | 48.50 | 80 | 5 | | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.29125 |
| 100 | 38.80 | 100 | 5 | | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.27300 |
| 80 | 31.04 | 125 | 5 | | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.25840 |
| 50 | 19.40 | 200 | 5 | | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.23650 |
| 40 | 15.52 | 250 | 5 | | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.22920 |
| 33.33 | 12.93 | 300 | 5 | | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.22433 |
| 25 | 9.70 | 400 | 5 | | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.21825 |
| 20 | 7.76 | 500 | 5 | >100 dB | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.21460 |
| 16 | 6.21 | 625 | 5 | >100 dB | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.21168 |
| 10 | 3.88 | 1000 | 5 | >100 dB | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.20730 |
| 8 | 3.10 | 1250 | 5 | >100 dB | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.20584 |
| 5 | 1.94 | 2000 | 5 | >100 dB | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.20365 |
| 4 | 1.55 | 2500 | 5 | Infinite | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.20292 |
| 2 | 0.78 | 5000 | 5 | Infinite | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.20146 |
| 1 | 0.39 | 10000 | 5 | Infinite | 0.479686 | 0.709252 | 0.940938 | 1.173450 | 13.20073 |

**TABLE B-1: DIGITAL FILTER DATA**

As shown in Figure B-1, the CIC (Cascaded Integrator-Comb) and FIR (Finite Impulse Response) filters are responsible for decimation of the ADC data. The sample rates available are shown in Table B-1. The corresponding CIC and FIR decimations are shown as well as the maximum passband frequencies.

Decimation has been divided into two stages in order to provide a "low" passband droop and a high alias rejection at frequencies close to the Nyquist rate. The CIC filter cannot achieve this alone. The scheme employed ensures that the passband droop is no more than ±0.01 dB and that the alias rejection is at least 100 dB.

If the user requires a smaller passband than the one passband indicated in the table for a particular $f_s$ or if they must reject specific frequencies, the user can make use of the IIR filters explained later in this section.

### *CIC Filter*

The CIC filter in the FPGA receives input from the anti-alias filter decimates the data by the factors shown in Figure B-2. The max_$f_c$ parameter, shown in Table B-1, is less than or equal to $0.05 \times f_{so}$. Stopband attenuation is greater than -110 dB.

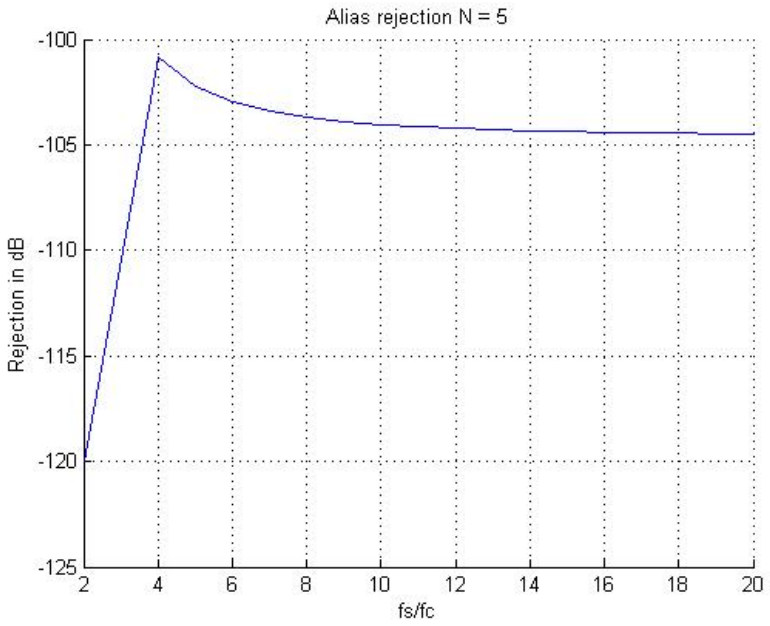



**FIGURE B-2: ALIAS REJECTION AS A FUNCTION OF DECIMATION**

### *DSP Filters*

#### *CIC compensation filter*

This filter compensates for the droop in the passband of the CIC. Six tap FIR filters are designed to compensate for the variable droop as a function of sampling rates. The need to have more than one filter arises due to the wide range of decimation factors. The CIC droop after compensation is bounded by ±0.0002. The group delay of this filter is 2.5 samples. Filtering operations done in floating point.

***FIR decimation filter***

There are up to 128 tap filters that decimate up to a factor of 10. These filters can operate in the linear range of the CIC response. Hence, the CIC filter is used to decimate frequencies up to max_$f_c$*20 or more, then the FIR filter is used to further decimate the data such that a passband close to the Nyquist frequency is achieved. The ripple of this filter for $f_s \geq 3125$ is 0.01 dB and for $f_s < 3125$ is 0.001 dB with an alias rejection of 100 dB.

FIR filtering is done in floating point.

***IIR filters***

The IIR filters are designed on-the-fly in the DSP. Given the following:

- filter type (i.e. Butterworth or Bessel)
- sampling frequency, $f_s$
- cutoff $f_c$ – -3 dB point can be, at most, max_$f_c$ to operate in the linear region of the decimating filters
- transform type (Bilinear/Matched Z)
- IIR filter order – supported ranges are 1-10. For the Butterworth filter type, if the filter order is set to 0, the DSP determines the order by assuming attenuation of 200 dB in stopband. The filter order MUST be specified for Bessel filters.
- the DSP designs a filter implemented as biquad sections. The maximum filter order is 10. The DSP also determines the risetime, overshoot, and group delay of the filter.

IIR filtering operations are done in double-precision floating point.

***Filtering Limitations***

- IIR filtering is done only at rates 10 kHz and below due to DSP processing constraints

***Group Delay***

***FIR filters***

The group delay ($\tau_g$) through the FIR filters is constant over frequency and are calculated as follows:

$\tau_{g\,FIR}$ = (*Ntaps*-1)/2, where *Ntaps* is the number of taps.

The settling time of a FIR filter is twice the group delay ($2\tau_{g\,FIR}$).

In the EX1629, the ADC and the CIC are FIR filters. Their group delays are:

ADC = 39 samples @ 50 kSa/s

$$CIC = \frac{N(RM-1)}{2} \text{ @ 50 kSa/s}$$

Where *R*=decimation
*N*=number of stages = 5
*M*=differential delay = 1

FIR filter: (*fir_Ntaps*-1)/2 @ CIC output rate, where *fir_ntaps* is the number of taps of the FIR filter.

Hence, the total delay through the FIRs at the ADC sample rate (50 kHz) is:

$$\tau_{g\,FIR} = \frac{39 + 5(cic\_dec - 1)}{2} + (fir\_Ntaps - 1)(cic\_dec) \times 20\mu s\,.$$

The settling time is double this time ($2\tau_{g\,FIR}$).

***IIR filters***

The group delay through the Bessel or Butterworth IIR filters is a function of cutoff frequency, sampling frequency, and the number of poles. The group delay for the dc component is calculated and reported to the digital board ($\tau_{g\,IIR}$).

The total delay through the IIR = $\tau_{g\,IIR}$ **cic_dec*fir_dec*20 µs*

The group delay will be provided to the end user. The samples will be group delay compensated.

## TRANSFORMATIONS

The EX1629 utilizes two types of transformations: bilinear and matched Z. Both transformations are discussed below.

***The Bilinear Transform***

The bilinear transformation (BLT) used in the EX1629 maps the entire *jw* axis in the *s* plane exactly once to the unit circle in the *z* plane:

$$s = c\frac{1 - z^{-1}}{1 + z^{-1}}$$

dc ($s = 0$) maps to dc ($z = 1$). As for the finite difference approximation (FDA), infinite frequency ($s = \infty$) maps to *half the sampling rate* ($z = -1$) instead of $z = 0$ for the FDA. As a result:

- damping characteristics are better preserved
- no aliasing (mapping is one-to-one)
- frequency axis remains warped away from dc

The real constant $c > 0$ allows *one* nonzero frequency (at $s = jw_\alpha$) to map exactly to any desired digital frequency (at $z = e^{jw_d T}$). All other frequencies are *warped*, or distorted in a non-linear fashion:

$$jw_\alpha = c\frac{1 - e^{-jw_d T}}{1 + e^{-jw_d T}} = jc\tan\left(\frac{w_d T}{2}\right)$$

***The Matched Z-transform***

**Under most circumstances, the BLT is superior to the matched Z-transform (MZT).** The principal disadvantage of the BLT is that the frequency scale is "warped" so that, for a low-pass filter:

| | |
|---|---|
| 0 Hz | In the analog domain maps to 0 Hz in the digital domain |
| $f_c$ | In the analog domain maps to $f_c$ in the digital domain |
| ∞ Hz | In the analog domain maps to $f_s/2$ in the digital domain |

where $f_c$ is the corner frequency and $f_s$ is the sampling frequency

Typically, this is what is desired of a low-pass or bandpass filter: a response which is exactly zero at the Nyquist frequency ($f_s/2$). For a high-pass filter, a response equal to unity at this frequency is also desirable.

The one exception to this is the Bessel filter. The advantage of a Bessel filter over a Butterworth or Chebyshev filter is that the phase response is nearly linear throughout the passband (the group

delay is almost constant throughout the passband). The “warping” inherent in the BLT method upsets this linearity. The MZT does not warp the frequency scale, so a digital Bessel filter designed by this method will have a near-linear phase characteristic.

In fact, a low-pass filter designed by MZT is impulse-invariant with the analog prototype. With the same impulse response, it behaves identically to the corresponding analog filter. This stands true only if aliasing is ignored. If the response of the digital low-pass filter is not negligible at $f_s/2$, the response in the vicinity of this frequency will be disturbed by the spurious response at frequencies above $f_s/2$, which are folded down into the band below $f_s/2$. This is why it is advisable to check the frequency response graphs carefully if this method is used.

A further advantage of the MZT is that the number of zeros is reduced, in comparison with the BLT. A low-pass filter designed by MZT has no $z$-plane zeros.

To summarize: matched Z-transform may be used if:

- a Bessel filter is required with an optimally linear phase response; or
- efficiency is more of a concern than the quality of the frequency response.

# APPENDIX C

# MICROLAN (MLAN) PRIMER

## INTRODUCTION

The MicroLAN specification details five major functions that are the centerpiece of any operations involving TEDS devices. These five functions are:

- Read serial/URN from device (GET_URN)
- Write to volatile memory scratchpad (READ_SCRATCHPAD)
- Read from volatile memory scratchpad (WRITE_SCRATCHPAD)
- Copy volatile scratchpad to nonvolatile memory (COPY_SCRATCHPAD)
- Read from non-volatile memory (READ_MEMORY)

The GET_URN command is a vital precursor to any of the other four operations. If the URN of the TEDS (1-Wire) device is not queried before the other operations are called, these operations will fail.

In general, there can be multiple 1-Wire devices per 1-Wire bus master (MLAN repeater) and the MLAN responder in the unit holds the state of the device being addressed. To interact with a 1-Wire device, the URN of the device is used as an address, allowing a single device to be selected. Each channel on the EX1629 has a 1-Wire bus master. Under typical operation on the EX1629, only a single 1-Wire device will be connected to each channel.

The GET_URN function is designed to identify a single 1-Wire TEDS device and return its unique 64-bit URN value. Sample code for how this function is implemented is provided. A variant of the GET_URN function can be used to search through multiple devices in order to select a specific device, but the example version of this function only supports one device per channel.

Once the 1-Wire device has been addressed, the MLAN responder will store its address, and it will not need to be referenced again until it is necessary to change devices. Any of the other commands can now be used to view or change data.

The READ_MEMORY function is probably the most commonly used MLAN command. It allows the user to query the non-volatile memory of any 1-Wire device and read back its contents. Currently, the maximum size of the EX1629 MLAN buffer is 48 bytes, but, because of packet overhead, this sample code limits data reads to 32 bytes. For some devices, notably the DS2431, the READ_MEMORY function may return multiple MLAN packets.

The “scratchpad” is a volatile buffer on a 1-Wire device where data is written before it is copied to memory. In order to write data to the main memory, it must first be written to the scratchpad. After writing to the scratchpad, the data can then be copied to the main memory.

| | |
|---|---|
| **WARNING** | ***The scratchpad will be erased if you unplug your MLAN device, power off the EX1629, or wait a significant amount of time between commands.*** VXI Technology recommends performing scratchpad operations in a production environment using the “write_and_copy” atomic command which is discussed in detail later. |

The WRITE_SCRATCHPAD command can be used to write arbitrary data to the scratchpad of a 1-Wire device. Unlike the GET_URN function, WRITE_SCRATCHPAD (and all other MLAN) functions are specific to the type of MLAN device being used. In the example code, write_scratchpad_2430() and write_scratchpad_2431() functions are used to represent the differences between the DS2430 and DS2431 devices. Be careful of the size differences between various devices' scratchpad buffers. In the case of the DS2431, 8 bytes of data, aligned on an 8-byte address, must be written together – that is, all memory writes involve 8 bytes. To modify a single byte of memory requires that the 8-byte block be read back from the device, the byte in question modified, and the resulting 8 bytes written back. The datasheet for the 1-Wire component in use should serve as the ultimate guide in programming the device.

The READ_SCRATCHPAD function can be used to read back a device's scratchpad. For example, it is always a good idea to do this after a WRITE_SCRATCHPAD call and before a COPY_SCRATCHPAD call to verify that the write was completed successfully and that the data was entered correctly before permanently overwriting main memory. As previously noted, using the individual functions for WRITE, READ, and COPY can cause data loss and the recommended method is to use the WRITE_AND_COPY command and read main memory.

COPY_SCRATCHPAD allows the user to transfer the scratchpad buffer on the MLAN device to the non-volatile memory of the device. This permanently overwrites the addressed non-volatile memory, so care should be exercised when doing so.

## PROGRAMMING MLAN

Below, we will discuss the example code for each function of the 2430 and 2431. First, however, there are some constants which should be discussed.

```
//MLAN commands
#define CMD_RESET 0x84
#define DATA_SEARCH_STATE 0x01
#define DATA_SEARCH_CMD 0x02
#define CMD_ML_DATA 0x0A
#define CMD_ML_RESET 0x80
#define CMD_ML_SEARCH 0x81
#define DATA_ID 0x00
#define CMD_GETBUF 0x85
#define CMD_ML_ACCESS 0x82
#define CMD_DELAY 0x0B
#define CMD_ML_BIT 0x09
#define DELAY_128 0x02
#define DELAY_MS 0x80
```

These #defines are commands that are sent to the MLAN controller. They do not modify the data on the device, but allow a device to be selected, tells the controller to return a buffer with the result, or sets up a delay on the MLAN line. They will be explained when they are used in the example code. These commands are defined by the MLAN specification. In general, these commands are targeted at the MLAN repeater (1-Wire bus master) itself, not the 1-Wire, TEDS devices.

```
//Functions that modify TEDS data
#define WRITE_SCRATCHPAD 0x0F
#define READ_SCRATCHPAD 0xAA
#define COPY_SCRATCHPAD 0x55
#define READ_MEMORY 0xF0
```

These are the actual opcodes for four of the five functions we outlined above. They are 1-Wire bus commands that are sent to the 1-Wire (TEDS) devices. The GET_URN function is not listed because it is not a single opcode that modifies data on the device or returns data except for the serial number/URN.

```
//Device-specific values
#define DS2430_SCRATCHPAD_LEN 32
#define DS2431_SCRATCHPAD_LEN 8
#define DS2430_MEMORY_LEN 32
#define DS2431_MEMORY_LEN 144
#define EX1629_MAX_TEDS_READ 32
```

These are the values that will be seen in the code, and are fairly self explanatory. The EX1629_MAX_TEDS_READ is an artifact of implementation – the minimum MLAN buffer is 48 bytes, which is what is supported by the EX1629. To avoid overruns, however, our example code will only read 32 bytes at a time, plus some MLAN overhead which will be explained later.

```
uint8_t SendPkt[256];
uint8_t RecPkt[256];
```

These are global buffers which will be used to store the sent and received packets.

Before beginning with the listed functions, a short example will be examined and described in detail.

```
int example_function(int channel)
{
  int sendLen, recLen = 0;
  sendLen = 1; // reserve first byte for length
  SendPkt[sendLen++] = CMD_RESET;
  SendPkt[sendLen++] = CMD_GETBUF;
  SendPkt[0] = sendLen - 1;
  MLanHostPacketSend(SendPkt, channel);
  recLen = MLanHostPacketReceive(RecPkt, channel, MLAN_PACKET_SIZE);
  return recLen;
}
```

This function performs an MLAN bus master reset, that is, it resets the MLAN repeater inside the EX1629. As can be seen, the first byte of the packet is reserved for the length of the packet. This not only defines a maximum size for an MLAN packet, but also tells the controller how much space to allocate for it. This is done for every packet sent. The controller also uses the first byte as the length of every packet received. Command and data bytes are appended to the byte array, with a post-increment of the index (sendLen).

The first command sent is CMD_RESET, or 0x84. This is the command that performs the bus master reset. The next command is CMD_GETBUF, or 0x84. This returns the response buffer from the repeater.

Here is the program's output, given just this function:

```
sent packet without errors
Packet length: 3
02 84 85
got a packet without errors on receive
Packet length: 3
02 84 00
```

The first line indicates that the driver call used to send the data was successful. The second indicates how long the packet sent was: 3 bytes, which is what was expected. The next line is the printout of the packet. The first byte in the packet, as previously stated, is the length of the packet.

In this case, 2 bytes (this length does not include the length byte). The other bytes are the CMD_RESET and CMD_GETBUF commands, in that order.

It is sometimes necessary to allocate space for a return buffer in the sent packet. All MLAN commands will send back at least two bytes: the command performed and the response to that command. Additional components may also be included, such as CRC bytes, data echo, or an error message, but two bytes is the minimum.

In this instance, three bytes were returned: the length of the data, the command performed, and the response to the command. As before, the length of the packet does not include the length byte. The two bytes returned are "84" and "00", where "84" is from the sent packet, the CMD_RESET command. This is the bus master echoing the command to confirm what was sent. The "00" byte is the response to that command, in this case ML_SUCCESS. The CMD_GETBUF is not echoed back in the response, nor is there an error code returned, as this information would be superfluous. This indicates that the MLAN repeater received the bus master reset and that it was successful. If an error was encountered during this process, it might look like this:

```
sent packet without errors
Packet length: 3
02 84 85
got a packet without errors on receive
Packet length: 3
03 84 86 02
```

This time, 3 bytes were received. The "84" for CMD_RESET, and then "86 02". From IEEE 1451.4, Annex G, "86" is the code for CMD_ERROR. This buffer was not processed successfully, and we have not reset the bus master. The "02" is also in Annex G, and means "RET_BUSY, previous buffer has not been processed yet." It will be necessary to wait until the last command completes before processing this one. Note that this error is purely hypothetical, but illustrates the typical format for an MLAN error.

With some basic programming completed, the more complex functions required to access a 1-Wire device can be examined.

The GET_URN function, below, is the simplest function in many ways, and the most generic.

```
int get_urn(int channel)
{
  int sendLen, recLen = 0;
  sendLen = 1; // reserve first byte for length
  // clear the search state so we find the 'first' device
  SendPkt[sendLen++] = CMD_RESET;
  // do a reset,search and then read results
  SendPkt[sendLen++] = CMD_ML_RESET;
  SendPkt[sendLen++] = CMD_ML_SEARCH;
  SendPkt[sendLen++] = DATA_ID;
  SendPkt[sendLen++] = 0;
  // request the result buffer as the last command
  SendPkt[sendLen++] = CMD_GETBUF;
  // set the length
  SendPkt[0] = sendLen - 1;
  // send and receive the frame
  MLanHostPacketSend(SendPkt, channel);
  recLen = MLanHostPacketReceive(RecPkt, channel, MLAN_PACKET_SIZE);
  return recLen;
}
```

As in the pervious example, the first byte is reserved for length and increments our index as commands are inserted into the packet.

First, an MLAN Bus Master Reset is performed. This allows ceases compunction with any other MLAN devices that were previously addressed, and uses the new one. The CMD_ML_SEARCH and DATA_ID commands tell the controller to find the next device and obtain its ID, respectively. If multiple CMD_ML_SEARCH and DATA_ID pairs were sent, it would be possible to determine how many devices were on this channel. CMD_ML_SEARCH returns 0x01 when no more devices are found.

The GET_URN() function, above, is equivalent to the vtex1629_read_teds_URN() function of the instrument driver. It is included for clarity and as one of the simpler examples of MLAN programming.

| **NOTE** | The EX1629 supports only one device per channel in 0.4.x and previous firmware revisions. |
|---|---|

Here is an example output from the GET_URN function using the example code:

```
sent packet without errors
Packet length: 7
06 84 80 81 00 00 85
got a packet without errors on receive
Packet length: 17
10 84 00 80 00 81 00 00 08 14 29 70 D3 01 00 00 60
```

The “14” in the response denotes the “family code” of the device, in this case, indicating that it is a DS2430. The “14” is the first byte of the unique serial number, and the “08” before it is the length of that serial number. The DS2431’s family code is “2D”.

## DS2430 COMMANDS

### *WRITE_SCRATCHPAD_2430*

The function used in writing data to an MLAN device is the WRITE_SCRATCHPAD function. It is important that the 1-Wire device be selected using the GET_URN function prior to using the remaining functions. The CMD_ML_ACCESS command to the MLAN repeater uses the address (URN) of the last selected device for all subsequent operations.

```
int write_scratchpad_2430(const char* data, int channel)
{
  int sendLen, recLen = 0;
  int i = 0;
  char byte[2];

  if(strlen(data) != ((DS2430_SCRATCHPAD_LEN * 2 ) +
(DS2430_SCRATCHPAD_LEN-1)))
  {
    printf("Data was not the right length (wanted 95, got %i)\n",
strlen(data)); //(SCRATCHPAD_LEN*2)+(SCRATCHPAD_LEN-1) = 95
    return -1;
  }
  sendLen = 1; // reserve first byte for length
  // access the current device with address in DATA_ID
  SendPkt[sendLen++] = CMD_ML_ACCESS;
  // construct a block of communication to MicroLAN
  SendPkt[sendLen++] = CMD_ML_DATA;
  SendPkt[sendLen++] = 3+DS2430_SCRATCHPAD_LEN; // block length
  SendPkt[sendLen++] = 2+DS2430_SCRATCHPAD_LEN; // data length
  // send the write scratchpad command
  SendPkt[sendLen++] = WRITE_SCRATCHPAD;
  // send the address byte
  SendPkt[sendLen++] = 0;
  // the bytes of data to write
```

```
  for (i = 0; i < ((2*DS2430_SCRATCHPAD_LEN) +
(DS2430_SCRATCHPAD_LEN-1)); i+=3)
  {
    strncpy(byte, &data[i],2);
    SendPkt[sendLen++] = (uint8_t)strtoul(byte, NULL, 16); //convert
to hex
  }
  // request the result buffer as the last command
  SendPkt[sendLen++] = CMD_GETBUF;
  // set the length
  SendPkt[0] = sendLen - 1;
  // send and receive the frame
  MLanHostPacketSend(SendPkt, channel);
  recLen = MLanHostPacketReceive(RecPkt, channel, MLAN_PACKET_SIZE);
  return recLen;
}
```

When the ASCII data format used in the example code is seen., the reason for the `(SCRATCHPAD_LEN * 2) + (SCRATCHPAD_LEN - 1)` code segment becomes more clear. For the DS2430, a data string for the example code might look like this:

```
"01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17
18 19 1A 1B 1C 1D 1F 20 21"
```

For each hex byte, there are two characters (`LEN * 2`), plus one space for all but the last one (`LEN-1`). This example code uses this format for easy parsing.

Once again, the first byte is reserved for the length of the packet, and then two very common MLAN commands are sent, CMD_ML_ACCESS and CMD_ML_DATA. The ML_ACCESS command allows the user to access a 1-Wire device which was previously searched for and discovered. ML_DATA tells the controller that data operations are going to begin on that device – that is, 1-Wire bus transactions will be performed.

The next command, `3+DS2430_SCRATCHPAD_LEN`, is actually the length of the command that is being sent to the device. Since the entire scratchpad length is being written, that must be included. Then, the `data length` block, the WRITE_SCRATCHPAD command itself, and the "0", which is the address in memory that the scratchpad will write to, must be added. Since data written to the scratchpad will ultimately end up in the non-volatile memory, the target address in the non-volatile memory must be provided when writing the data to the scratchpad. This provides a measure of error detection, when the address is later provided in the copy scratchpad operation, as well as allowing the device to return an error if the target memory is write-protected. The CMD_GETBUF is not part of this block, as it is a separate command.

Note that data length block is 2+DS2430_SCRATCHPAD_LEN. The reason for this is that, when writing to the scratchpad, the controller sends back what was written so that it can be verified. For any MLAN command, two must be added to the length of the data that is expected to be returned (for the command sent and command result to be returned), and the scratchpad data is the only data we expect to be returned.

On the DS2430, address 0 is always written to after the WRITE_SCRATCHPAD command, as the DS2430 scratchpad is the same size as the memory. Therefore, by writing a single scratchpad, the entire memory will be overwritten. This will not always be the case, and, in fact, is not on the DS2431, which will be seen later.

The "for" loop that is next in the code translates the ASCII text string "data", which was passed in (see the example data string above), into its hexadecimal equivalent to be sent to the controller.

The last command tells the controller to read back the result of the previous command. This is invaluable, as it allows error codes to be viewed, if errors are returned. The last steps are to send the completed packet and retrieve the response.

Here is how the output for a WRITE_SCRATCHPAD for the DS2430 looks when using the example code:

```
sent packet without errors
Packet length: 40
27 82 0A 23 22 0F 00 AA 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1F 20 21 85
got a packet without errors on receive
Packet length: 39
26 82 00 0A 22 0F 00 AA 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1F 20 21
```

As can be seen, WRITE_SCRATCHPAD echoes the data written to it as well, which is why space was allocated for it.

***READ_SCRATCHPAD_2430***

The READ_SCRATCHPAD command is nearly identical between the DS2430 and DS2431, but some differences exist. The DS2430 will be covered first.

```
int read_scratchpad_2430(int channel)
{
  int sendLen, recLen = 0;
  sendLen = 1; // reserve first byte for length
  // access the current device with address in DATA_ID
  SendPkt[sendLen++] = CMD_ML_ACCESS;
  // construct a block of communication to MicroLAN
  SendPkt[sendLen++] = CMD_ML_DATA;
  SendPkt[sendLen++] = 3; // block length
  SendPkt[sendLen++] = 2+DS2430_SCRATCHPAD_LEN; // data length of
read
  // send the read scratchpad command
  SendPkt[sendLen++] = READ_SCRATCHPAD;
  // send the address byte
  SendPkt[sendLen++] = 0;
  // request the result buffer as the last command
  SendPkt[sendLen++] = CMD_GETBUF;
  // set the length
  SendPkt[0] = sendLen - 1;
  // send and receive the frame
  MLanHostPacketSend(SendPkt, channel);
  recLen = MLanHostPacketReceive(RecPkt, channel, MLAN_PACKET_SIZE);
  return recLen;
}
```

The first three commands are the same as the WRITE_SCRATCHPAD command. The device must still be accessed and the controller must be put into data access mode. The block length, this time, is only 3, because the expected buffer size, the READ_SCRATCHPAD command, and the address to read from (always “0” for this example, but if less than the scratchpad length is read, this could be incremented and read multiple times) is all that must be sent.

The expected data length is `2+SCRATCHPAD_LEN`. See WRITE_SCRATCHPAD_2430 for why this length is used.

Here is some example output from READ_SCRATCHPAD using the example code:

```
sent packet without errors
Packet length: 8
07 82 0A 03 22 AA 00 85
got a packet without errors on receive
Packet length: 39
26 82 00 0A 22 AA 00 AA 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
11 12 13 14
 15 16 17 18 19 1A 1B 1C 1D 1F 20 21
```

This is identical to what was written with the WRITE_SCRATCHPAD command, so the data was written properly. Again, recall that the non-atomic operations cannot be guaranteed, as the 1-Wire devices are powered down between MLAN commands, which erases the scratchpads.

***COPY_SCRATCHPAD_2430***

The COPY_SCRATCHPAD command is the only method for writing data to the memory of a MLAN device; it copies the data in the scratchpad to the non-volatile memory. Like READ_SCRATCHPAD, the command is fairly simple, as the data already exists and only needs to be moved.

```
int copy_scratchpad_2430(int channel)
{
   int sendLen, recLen = 0;
   sendLen = 1; // reserve first byte for length
   // access the current device with address in DATA_ID
   SendPkt[sendLen++] = CMD_ML_ACCESS;
   // construct a block of communication to MicroLAN
   SendPkt[sendLen++] = CMD_ML_DATA;
   SendPkt[sendLen++] = 3; // block length
   SendPkt[sendLen++] = 2; // data length
   // send the copy scratchpad command
   SendPkt[sendLen++] = COPY_SCRATCHPAD;
   // send the validation key
   SendPkt[sendLen++] = 0xA5;
   // delay for 128ms
   SendPkt[sendLen++] = CMD_DELAY;
   SendPkt[sendLen++] = 1;
   SendPkt[sendLen++] = DELAY_128 | DELAY_MS;
   // request the result buffer as the last command
   SendPkt[sendLen++] = CMD_GETBUF;
   // set the length
   SendPkt[0] = sendLen - 1;
   // send and receive the frame
   MLanHostPacketSend(SendPkt, channel);
   recLen = MLanHostPacketReceive(RecPkt, channel, MLAN_PACKET_SIZE);
  return recLen;
}
```

Once again, the first three bytes are the same as WRITE_SCRATCHPAD and READ_SCRATCHPAD. Our block length is 3, as the expected response length, the COPY_SCRATCHPAD command, and the “validation key” that ensures we are not writing to the wrong device are sent. Each “family” of MLAN device has a different validation key. This will be seen with the DS2431. Note that the expected response length is the minimum 2 bytes. Since the COPY_SCRATCHPAD command does not actually return data to us, space does not have to be allocated in the return buffer.

Once the copy command is sent, a delay is sent to the MLAN controller to allow time for the copy to finish. The "1" sent after CMD_DATA is the length, in bytes, of the delay command which is on the next line. A 'bitwise or' function is used to combine the units of delay with the delay length – if a shorter or longer delay time is required, the 1-Wire specification defines several delay lengths and several different time units which can be used.

Here is an example output from the COPY_SCRATCHPAD command using the example code:

```
sent packet without errors
Packet length: 11
0A 82 0A 03 02 55 A5 0B 01 82 85
got a packet without errors on receive
Packet length: 7
06 82 00 0A 02 55 A5
```

As there is no user data returned, the reply to this command is short.

***READ_MEMORY_2430***

Although the READ_MEMORY command is probably the most useful of the MLAN command, it is discussed here as this is where the commands would logically appear in code.

```
int read_memory_2430(int channel)
{
  int sendLen, recLen = 0;
  sendLen = 1; // reserve first byte for length
  // access the current device with address in DATA_ID
  SendPkt[sendLen++] = CMD_ML_ACCESS;
  // construct a block of communication to MicroLAN
  SendPkt[sendLen++] = CMD_ML_DATA;
  SendPkt[sendLen++] = 3;  // block length
  SendPkt[sendLen++] = (2+DS2430_MEMORY_LEN); // data length with 32
bytes of reads
  // send the read memory command
  SendPkt[sendLen++] = READ_MEMORY;
  // send the address byte
  SendPkt[sendLen++] = 0;
  // request the result buffer as the last command
  SendPkt[sendLen++] = CMD_GETBUF;
  // set the length
  SendPkt[0] = sendLen - 1;
  // send and receive the frame
  MLanHostPacketSend(SendPkt, channel);
  recLen = MLanHostPacketReceive(RecPkt, channel, MLAN_PACKET_SIZE);
  return recLen;
}
```

Again, the first three are the same as those seen in previous examples. The block length is the return buffer size, the READ_MEMORY command, and the address offset. Since the whole address space can be read at once on the DS2430, the address offset is always zero. This in not always the case for the DS2431's, as the DS2431 has a much larger memory space. See the READ_MEMORY_2431command for an example of this.

Note that the return buffer size does have to be big enough to hold the whole address space, so the standard two bytes of MLAN data are added to the DS2430's address space size.

Here is an example output from the READ_MEMORY command using the example code:

```
sent packet without errors
Packet length: 8
07 82 0A 03 22 F0 00 85
got a packet without errors on receive
Packet length: 39
26 82 00 0A 22 F0 00 AA 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1F 20 21
```

This is the same string that was in the WRITE_SCRATCHPAD, now has been transferred into non-volatile memory.

***WRITE_AND_COPY_SCRATCHPAD_2430***

As noted above, the scratchpad is a volatile memory location. Between MLAN commands, the 1-Wire devices are powered down, and, hence, will lose all their scratchpad (volatile) data. In example code below, the commands occur fast enough that power is not lost to the 1-Wire devices and no data loss is experienced. This, however, is not recommended for a production environment.

As a consequence, the WRITE_AND_COPY command set is recommended. These commands are atomic operations – they perform the write and the copy in a single MLAN command. Because the read and write is accomplished in a single command, data integrity cannot be verified before it is copied to memory (e.g. a READ_SCRATCHPAD command cannot be performed in the middle of a WRITE_AND_COPY command set to verify the scratchpad write). However, the main memory can still be checked after the write, in a separate series of MLAN operations to ensure that what was written is correct.

```
int write_and_copy_scratchpad_2430(const char* data, int channel)
{
  int sendLen, recLen = 0;
  int i = 0;
  char byte[3];

  if(strlen(data) != ((DS2430_SCRATCHPAD_LEN * 2 ) +
(DS2430_SCRATCHPAD_LEN-1)))
  {
    printf("Data was not the right length (wanted 95, got %i)\n",
strlen(data));
    return -1;
  }
  sendLen = 1; // reserve first byte for length
  // access the current device with address in DATA_ID
  SendPkt[sendLen++] = CMD_ML_ACCESS;
  // construct a block of communication to MicroLAN
  SendPkt[sendLen++] = CMD_ML_DATA;
  SendPkt[sendLen++] = 3+DS2430_SCRATCHPAD_LEN; // block length
  SendPkt[sendLen++] = 4+DS2430_SCRATCHPAD_LEN; // data length
  // send the write scratchpad command
  SendPkt[sendLen++] = WRITE_SCRATCHPAD;
  // send the address byte
  SendPkt[sendLen++] = 0;
  // the bytes of data to write
  for (i = 0; i < ((2*DS2430_SCRATCHPAD_LEN) +
(DS2430_SCRATCHPAD_LEN-1)); i+=3)
  {
    strncpy(byte, &data[i],2);
    byte[2] = '\0';
    SendPkt[sendLen++] = (uint8_t)strtoul(byte, NULL, 16); //convert
to hex
  }
```

```
  //here is the copy:
  SendPkt[sendLen++] = CMD_ML_ACCESS;
  // construct a block of communication to MicroLAN
  SendPkt[sendLen++] = CMD_ML_DATA;
  SendPkt[sendLen++] = 3; // block length
  SendPkt[sendLen++] = 2; // data length
  // send the copy scratchpad command
  SendPkt[sendLen++] = COPY_SCRATCHPAD;
  // send the validation key
  SendPkt[sendLen++] = 0xA5;
  // delay for 128ms
  SendPkt[sendLen++] = CMD_DELAY;
  SendPkt[sendLen++] = 1;
  SendPkt[sendLen++] = DELAY_128 | DELAY_MS;
  // request the result buffer as the last command
  SendPkt[sendLen++] = CMD_GETBUF;
  // set the length
  SendPkt[0] = sendLen - 1;
  // send and receive the frame
  MLanHostPacketSend(SendPkt, channel);
  recLen = MLanHostPacketReceive(RecPkt, channel, MLAN_PACKET_SIZE);
  return recLen;
}
```

If one looks closely, it becomes obvious that the WRITE_AND_COPY command is simply a WRITE and a COPY command combined. There are multiple block length and data length bytes, multiple commands, and all operations are performed that each of these commands did individually. However, since they are issued in a single MLAN buffer, the device will not be powered down and will not have a chance to lose its volatile data.

The example output of the WRITE_AND_COPY command is fairly long, but, like the code that generates it, is very similar to an amalgamation of the WRITE_SCRATCHPAD and COPY_SCRATCHPAD commands.

```
sent packet without errors
Packet length: 49
30 82 0A 23 24 0F 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
11 12 13 14
 15 16 17 18 19 1A 1B 1C 1D 1F 20 21 82 0A 03 02 55 A5 0B 01 82 85
got a packet without errors on receive
Packet length: 47
2E 82 00 0A 24 0F 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
11 12 13 14
 15 16 17 18 19 1A 1B 1C 1D 1F 20 21 FF FF 82 00 0A 02 55 A5
```

## DS2431 COMMANDS

### *WRITE_SCRATCHPAD_2431*

The DS2431 has the same command set as the DS2430, but some of the commands are different, and some give more arguments to the MLAN controller. Some examples of these difference can be seen in the WRITE_SCRATCHPAD command.

```
int write_scratchpad_2431(const char* data, int channel, int address)
{
  int sendLen, recLen = 0;
  int i = 0;
  char byte[3];

  if(strlen(data) != ((DS2431_SCRATCHPAD_LEN * 2 ) +
(DS2431_SCRATCHPAD_LEN-1)))
```

```
  {
    printf("Data was not the right length (wanted 23, got %i)\n",
strlen(data));
    return -1;
  }
  sendLen = 1; // reserve first byte for length
  // access the current device with address in DATA_ID
  SendPkt[sendLen++] = CMD_ML_ACCESS;
  // construct a block of communication to MicroLAN
  SendPkt[sendLen++] = CMD_ML_DATA;
  SendPkt[sendLen++] = 4+DS2431_SCRATCHPAD_LEN; // block length
  SendPkt[sendLen++] = 5+DS2431_SCRATCHPAD_LEN; // data length
  // send the write scratchpad command
  SendPkt[sendLen++] = WRITE_SCRATCHPAD;
  // send the address byte
  if (address > (DS2431_MEMORY_LEN-DS2431_SCRATCHPAD_LEN) || address
< 0)
  {
    printf("Invalid scratchpad address, must be between 0 and 88
hex\n");
    return -1;
  }
  SendPkt[sendLen++] = address;
  SendPkt[sendLen++] = 0;
  // the 5 bytes of data to write
  for (i = 0; i < ((2*DS2431_SCRATCHPAD_LEN) +
(DS2431_SCRATCHPAD_LEN-1)); i+=3)
  {
    strncpy(byte, &data[i],2);
    byte[2]='\0';
    SendPkt[sendLen++] = (uint8_t)strtoul(byte, NULL, 16); //convert
to hex
  }
  // request the result buffer as the last command
  SendPkt[sendLen++] = CMD_GETBUF;
  // set the length
  SendPkt[0] = sendLen - 1;
  // send and receive the frame
  MLanHostPacketSend(SendPkt, channel);
  CRCcalc(SendPkt, 6, 11);
  recLen = MLanHostPacketReceive(RecPkt, channel, MLAN_PACKET_SIZE);
  return recLen;
}
```

Notice first that, in comparison to the DS2430 scratchpad write, the length of the data string that is checked is much smaller, although it uses the same formula,. This is because the DS2431 has only an 8-byte scratchpad, whereas the DS2430 has 32 bytes. The same format of data string is used to submit data, however. Here is an example of a data string for the DS2431:

```
"01 02 03 04 05 06 07 08"
```

Even on a different device, however, the basic format remains unchanged. The first byte received is for length and the next two are CMD_ML_ACCESS and CMD_ML_DATA, just like in the DS2430's command.

The block size is our first indicator that the DS2430 and DS2431, although similar, have slightly different programming requirements. This block size is one byte larger than the DS2430's block size, and the reason is readily apparent three commands down: there are two address registers instead of one. The first address register, T1, is the beginning offset, and T2 is the address. As can be seen in the example, T1 is used to offset the data within the main memory, allowing users to

write to any address between the bytes $0^h$ an 136e. From the $136^{th}$ byte, users can write all the way to the end of the memory.

The data length is also different. We add several bytes to the DS2430's command length, and two of these are the CRC16 value that the DS2431 device returns to us. The CRC uses the entire command, from WRITE_SCRATCHPAD all the way to the end of the data buffer, to generate its CRC. Notice, too, that a CRC is run on the packet, passing in the offset of the WRITE_SCRATCHPAD command in the packet and the length of the command plus the data. This function is explained in the *Additional Notes* section.

| **NOTE** | If the CRC returned is not the same as the calculated CRC given by this program, the data write was corrupted in transmission. Re-access the device and try the write again. |
|---|---|

The same code is used as with the DS2430 to turn the data string into hexadecimal and to tell the controller to send the response back with the CMD_GETBUF command.

The example output for the WRITE_SCRATCHPAD command for the DS2431 is very similar to the WRITE_SCRATCHPAD for the DS2430, with the exception that the data is shorter:

```
sent packet without errors
Packet length: 17
10 82 0A 0C 0D 0F 00 00 AA 02 03 04 05 06 07 08 85
CRC16: 74E4
got a packet without errors on receive
Packet length: 18
11 82 00 0A 0D 0F 00 00 AA 02 03 04 05 06 07 08 74 E4
```

This indicates that this write was successful, as the last two bytes returned are the same as the calculated CRC.

***READ_SCRATCHPAD_2431***

```
int read_scratchpad_2431(int channel)
{
  int sendLen, recLen = 0;
  sendLen = 1; // reserve first byte for length
  // access the current device with address in DATA_ID
  SendPkt[sendLen++] = CMD_ML_ACCESS;
  // construct a block of communication to MicroLAN
  SendPkt[sendLen++] = CMD_ML_DATA;
  SendPkt[sendLen++] = 3; // block length
  SendPkt[sendLen++] = 6+DS2431_SCRATCHPAD_LEN; // data length of
read
  // send the read scratchpad command
  SendPkt[sendLen++] = READ_SCRATCHPAD;
  // send the address byte
  SendPkt[sendLen++] = 0;
  // request the result buffer as the last command
  SendPkt[sendLen++] = CMD_GETBUF;
  // set the length
  SendPkt[0] = sendLen - 1;
  // send and receive the frame
  MLanHostPacketSend(SendPkt, channel);
  recLen = MLanHostPacketReceive(RecPkt, channel, MLAN_PACKET_SIZE);
  CRCcalc(RecPkt, 6, 12);
  return recLen;
}
```

Some differences from the DS2430 version are also seen here. Aside from using the DS2431_SCRATCHPAD_LEN instead of the DS2430_SCRATCHPAD_LEN, a much larger buffer is used for the data length. This is because the echo from the command being sent and a 2-byte CRC calculated on the returned data stream are expected in addition to the data in the scratchpad. It can also be seen that this is also passed into our the CRC calculating function so that the CRC can be verified. See the note in the WRITE_SCRATCHPAD_2431 section for what to do if the CRCs do not match.

Here is some example output from this command:

```
sent packet without errors
Packet length: 8
07 82 0A 03 0E AA 00 85
got a packet without errors on receive
Packet length: 19
12 82 00 0A 0E AA 00 00 07 AA 02 03 04 05 06 07 08 F9 19
CRC16: F919
The CRC again checks properly.
```

***COPY_SCRATCHPAD_2431***

```
int copy_scratchpad_2431(int channel, int address)
{
   int sendLen, recLen = 0;
   sendLen = 1; // reserve first byte for length
   // access the current device with address in DATA_ID
   SendPkt[sendLen++] = CMD_ML_ACCESS;
   // construct a block of communication to MicroLAN
   SendPkt[sendLen++] = CMD_ML_DATA;
   SendPkt[sendLen++] = 5; // block length
   SendPkt[sendLen++] = 4; // data length
   // send the copy scratchpad command
   SendPkt[sendLen++] = COPY_SCRATCHPAD;
   // send the validation key
   if (address > (DS2431_MEMORY_LEN-DS2431_SCRATCHPAD_LEN) || address
< 0)
   {
     printf("Invalid scratchpad address, must be between 0 and 88
hex\n");
     return -1;
   }
   SendPkt[sendLen++] = address;
   SendPkt[sendLen++] = 0;
   SendPkt[sendLen++] = 0x07;
   // delay for 128ms
   SendPkt[sendLen++] = CMD_DELAY;
   SendPkt[sendLen++] = 1;
   SendPkt[sendLen++] = DELAY_128 | DELAY_MS;
   // set the length
   SendPkt[0] = sendLen - 1;
   // send and receive the frame
   MLanHostPacketSend(SendPkt, channel);
   recLen = MLanHostPacketReceive(RecPkt, channel, MLAN_PACKET_SIZE);
  return recLen;
}
```

The COPY_SCRATCHPAD command for the DS2431 is more complicated than the DS2430 command. Here, the scratchpad address must be passed in or the device will assume it is receiving incorrect instructions and an error will occur. (The address should be the same address passed in to WRITE_SCRATCHPAD). The ES register must also be passed in, which should be “07”, identical to its value in the READ_SCRATCHPAD command (if the original write was

successful). If this value is not “07” on the device, or if a number other than 07 is passed in, it is assumed that the communication with this device was accidental and the copy will not occur. Please refer to the DS2431 data sheet for more information on the ES register.

Here is some more example MLAN output:

```
sent packet without errors
Packet length: 12
0B 82 0A 05 04 55 00 00 07 0B 01 82
got a packet without errors on receive
Packet length: 9
08 82 00 0A 04 55 00 00 07
```

***READ_MEMORY_2431***

Since the EX1629 can only read a buffer roughly corresponding to the size of the DS2430 memory and the DS2431 has almost five times that amount, reads of the DS2431 must be done in segments. The read is actually done by making four calls to Send and Receive. If a larger receive buffer were present, fewer calls could be made by increasing the returned data.

```
int read_memory_2431(int channel)
{
  int sendLen, recLen = 0;
  int totRecLen = 0;
  int i = 0;
  for (i=0; i< (((float)DS2431_MEMORY_LEN) /
((float)EX1629_MAX_TEDS_READ)) ; i++)
  {
    sendLen = 1; // reserve first byte for length
    // access the current device with address in DATA_ID
    SendPkt[sendLen++] = CMD_ML_ACCESS;
    // construct a block of communication to MicroLAN
    SendPkt[sendLen++] = CMD_ML_DATA;
    SendPkt[sendLen++] = 4;  // block length
    SendPkt[sendLen++] = (2+EX1629_MAX_TEDS_READ); // data length
with 32 bytes of reads
    // send the read memory command
    SendPkt[sendLen++] = READ_MEMORY;
    // send the address byte
    SendPkt[sendLen++] = (i * EX1629_MAX_TEDS_READ);
    SendPkt[sendLen++] = 0;
    // request the result buffer as the last command
    SendPkt[sendLen++] = CMD_GETBUF;
    // set the length
    SendPkt[0] = sendLen - 1;
    // send and receive the frame
    MLanHostPacketSend(SendPkt, channel);
    recLen = MLanHostPacketReceive(RecPkt, channel,
MLAN_PACKET_SIZE);
    totRecLen+=recLen;
  }
  return totRecLen;
}
```

Each command is similar to the READ_MEMORY command for the DS2430. The only real difference is the extra address field for offset, which is set to (i* EX1629_MAX_TEDS_READ). This creates a loop through that increases the offset by 32 each time, and continues until the entire memory is read. Once the end of the memory is reached, the device returns all “FF” bytes.

While it could be arranged for this example code to stop returning data once the end of the memory is reached, the code would be significantly harder to read, and, therefore, it is up to the user to determine where their memory ends. The example code is intended to be a starting point.

The READ_MEMORY output for this device is very long, as 144 bytes of data must be returned:

```
sent packet without errors
Packet length: 9
08 82 0A 04 22 F0 00 00 85
got a packet without errors on receive
Packet length: 39
26 82 00 0A 22 F0 00 00 AA 02 03 04 05 06 07 08 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
sent packet without errors
Packet length: 9
08 82 0A 04 22 F0 20 00 85
got a packet without errors on receive
Packet length: 39
26 82 00 0A 22 F0 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
sent packet without errors
Packet length: 9
08 82 0A 04 22 F0 40 00 85
got a packet without errors on receive
Packet length: 39
26 82 00 0A 22 F0 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
sent packet without errors
Packet length: 9
08 82 0A 04 22 F0 60 00 85
got a packet without errors on receive
Packet length: 39
26 82 00 0A 22 F0 60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
sent packet without errors
Packet length: 9
08 82 0A 04 22 F0 80 00 85
got a packet without errors on receive
Packet length: 39
26 82 00 0A 22 F0 80 00 00 00 00 00 00 55 00 00 00 00 00 00 00 00 02
0C FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

This device is empty except for the 8 bytes written in the example, so there are many empty segments. Notice the "55" near the end of the device – many of the registers near the end of memory control advanced uses of the device, such as write-protecting data. Please refer to the DS2431 data sheet for more information on the device features.

### *WRITE_AND_COPY_SCRATCHPAD_2431*

Like all 1-Wire devices, the DS2431 chips are powered down between MLAN commands on the EX1629. This can cause the volatile scratchpads to lose data, and, as such, the individual WRITE_SCRATCHPAD and COPY_SCRATCHPAD commands are only recommended for demonstration purposes.

The combined WRITE_AND_COPY command set issues a single MLAN buffer and has the advantage if being an atomic operation, meaning the device will not lose data in the middle. The disadvantage is that data verification is not possible until after the write has occurred.

```
int write_and_copy_scratchpad_2431(const char* data, int channel, int
address)
{
```

```
  int sendLen, recLen = 0;
  int i = 0;
  char byte[3];

  if(strlen(data) != ((DS2431_SCRATCHPAD_LEN * 2 ) +
(DS2431_SCRATCHPAD_LEN-1)))
  {
    printf("Data was not the right length (wanted 23, got %i)\n",
strlen(data));
    return -1;
  }
  sendLen = 1; // reserve first byte for length
  // access the current device with address in DATA_ID
  SendPkt[sendLen++] = CMD_ML_ACCESS;
  // construct a block of communication to MicroLAN
  SendPkt[sendLen++] = CMD_ML_DATA;
  SendPkt[sendLen++] = 4+DS2431_SCRATCHPAD_LEN; // block length
  SendPkt[sendLen++] = 5+DS2431_SCRATCHPAD_LEN; // data length
  // send the write scratchpad command
  SendPkt[sendLen++] = WRITE_SCRATCHPAD;
  // send the address byte
  if (address > (DS2431_MEMORY_LEN-DS2431_SCRATCHPAD_LEN) || address
< 0)
  {
    printf("Invalid scratchpad address, must be between 0 and 88
hex\n");
    return -1;
  }
  SendPkt[sendLen++] = address;
  SendPkt[sendLen++] = 0;
  // the 5 bytes of data to write
  for (i = 0; i < ((2*DS2431_SCRATCHPAD_LEN) +
(DS2431_SCRATCHPAD_LEN-1)); i+=3)
  {
    strncpy(byte, &data[i],2);
    byte[2]='\0';
    SendPkt[sendLen++] = (uint8_t)strtoul(byte, NULL, 16); //convert
to hex
  }
  //The copy command:
   // access the current device with address in DATA_ID
   SendPkt[sendLen++] = CMD_ML_ACCESS;
   // construct a block of communication to MicroLAN
   SendPkt[sendLen++] = CMD_ML_DATA;
   SendPkt[sendLen++] = 5; // block length
   SendPkt[sendLen++] = 4; // data length
   // send the copy scratchpad command
   SendPkt[sendLen++] = COPY_SCRATCHPAD;
   // send the validation key
   if (address > (DS2431_MEMORY_LEN-DS2431_SCRATCHPAD_LEN) || address
< 0)
   {
     printf("Invalid scratchpad address, must be between 0 and 88
hex\n");
     return -1;
   }
   SendPkt[sendLen++] = address;
   SendPkt[sendLen++] = 0;
   SendPkt[sendLen++] = 0x07;
   // delay for 128ms
   SendPkt[sendLen++] = CMD_DELAY;
   SendPkt[sendLen++] = 1;
   SendPkt[sendLen++] = DELAY_128 | DELAY_MS;
```

```
  // set the length
  SendPkt[0] = sendLen - 1;
  // send and receive the frame
  MLanHostPacketSend(SendPkt, channel);
  CRCcalc(SendPkt, 6, 11);
  recLen = MLanHostPacketReceive(RecPkt, channel, MLAN_PACKET_SIZE);
  return recLen;
}
```

This follows the same format as the DS2430's command as it is a concatenation of the WRITE_SCRATCHPAD and COPY_SCRATCHPAD commands. The CRC16 is calculated on the sent packet, and the returned result still contains the CRC bytes. However, as the results are already written to memory, a bad write will only be evident after the fact.

Here is some example output from this command:

```
sent packet without errors
Packet length: 27
1A 82 0A 0C 0D 0F 00 00 AA 02 03 04 05 06 07 08 82 0A 05 04 55 00 00
07 0B 01 82

CRC16: 74E4
got a packet without errors on receive
Packet length: 26
19 82 00 0A 0D 0F 00 00 AA 02 03 04 05 06 07 08 74 E4 82 00 0A 04 55
00 00 07
```

We can again see the CRC returned at the end of the WRITE_SCRATCHPAD portion of the WRITE_AND_COPY command, and that it matches what was calculated, indicating a successful write.

## ADDITIONAL NOTES

### *Checksums*

It should be noted that a "page" in the main memory of 1-Wire device consists of 32 bytes according to the IEEE 1451.4 specification. Each page of memory is supposed to have a one-byte checksum as the first bit, which when added to the other 31 bits in the page (dropping any carry) makes the result 0. In the example code (and in the EX1629 driver used to implement it) there are no checks made to ensure that the user inserts a checksum properly. If the user wishes to implement checksums (and, therefore, be fully compliant with the IEEE 1451.4 standard), a proper checksum should be written to the first page of memory, updated when memory is updated, and the checksums should be calculated and verified on reads.

### *Sending & Receiving*

The "`MLanHostPacketReceive`" and "`MLanHostPacketSend`" functions are not defined in the above examples. This is because the implementations of these functions are specific to the interface to the device. The example code does, however, provide definitions of these functions which work with the EX1629 instrument driver. These are below:

```
int MLanHostPacketReceive(uint8_t *receive_packet, int channel, int
len)
{
   int result = 0;
   int data = NULL;
   ViChar error[256];
   #define MLANBUFLEN 48
   // add platform/protocol specific code here
```

```
   if (len < MLANBUFLEN)
   {
     printf("Error, buffer size not large enough to hold MLAN
response");
     return -1;
   }

   result = vtex1629_read_teds_MLAN(vi, channel, MLANBUFLEN,
receive_packet, &data);
   if (result != 0)
   {
     vtex1629_error_message(vi, result, error);
    printf("error when receiving from driver: %s\n", error);
   }
   else
   {
    printf("got a packet without errors on receive\n");
    PrintPacket(receive_packet);
   }

   // return the length in bytes of the packet including length byte
   return receive_packet[0] + 1;
}

void MLanHostPacketSend(uint8_t *send_packet, int channel)
{
   int result = 0;
   ViChar error[256];

   // add platform/protocol specific code here

   result = vtex1629_write_teds_MLAN(vi, channel, (send_packet[0]+1),
send_packet);
   if(result != 0)
   {
     vtex1629_error_message(vi,result,error);
    printf("Error sending packet to driver: (%s)\n", error);
   }
   else
   {
    printf("sent packet without errors\n");
    PrintPacket(send_packet);
   }
   return;
}
```

These functions are fairly simple and only act as device-specific wrappers around the sending and receiving of commands.

### *Printing Packets*

The function that prints the packet is called from the sender & receiver, just to make it easier to determine which (sender vs. receiver) was doing the printing. Here is the code for the PrintPacket function.

```
void PrintPacket(uint8_t* pkt)
{
  int len = pkt[0]+1;
  int i = 0;

  printf("Packet length: %i\n", len);
  for (i=0 ; i< len ; i++)
```

```
  {
    printf("%02X ", pkt[i]);
  }
  printf("\n");
  return;
}
```

### *CRC Checking*

The CRC check function is called to calculate a CRC16 for the written data and the read data on the DS2431. Here is the code for, and a brief explanation of, this function:

```
unsigned int CRCcalc(uint8_t *pkt, int offset, int len)
{
   int i,j,k;
   uint8_t byte;
   unsigned int r = 0;

   for (k = (offset-1); k < ((offset + len)-1) ; k++) {
       //CRC16 CALCULATION
      byte = pkt[k];
      for(i=0;i!=8;byte>>=1,i++)
      {
          j=(byte^r)&1;
          r>>=1;

          if(j)
             r^=0xa001;
      }
   }
   r = ( ((r & 0x00FF) << 8) | ((r & 0xFF00) >> 8)); //bit-swapping
for endian-ness
   r = (uint16_t)~r;   //inverting to match MLAN
   printf("CRC16: %02X\n", r);
   return r;
}
```

The function first calculates a normal CRC16 and then does a bit-shift operation to swap the top and bottom halves of the CRC bytes (due to platform endian-ness) before inverting the CRC.

This is what the MLAN bus master will return to us on a little-endian host computer, like an x86 CPU (Intel or AMD): a byte-swapped, inverted CRC16. This way, the CRC16 value can be visually compared instead of having to do bitwise operations on what the MLAN device gives us.

### *Version Information*

The last function which has not yet been discussed is the REPEATER_TEST function. The purpose of this function is to query the MLAN command repeater inside the EX1629 and retrieve several items of data from it, including the version of the MLAN protocol it implements and the vendor identification string.

| **NOTE** | The version and vendor strings will both come back as null-terminated strings of hexadecimal digits, as the same PrintPacket function is used to print them as is used for the rest of the packets. |
|---|---|

For reference, the ML100 MLAN version string should appear as 4D 4C 31 30 30 00.

# INDEX


### *Numerics*

1-Wire ..... *See* MLAN

### *A*

acquisition data ..... 56
ADC clock ..... 57
ADC sample clock configuration ..... 70
ADC synchronization ..... 71
anti-alias filter ..... 301
ARM layer ..... 56
AutoIP ..... 34, 61

### *B*

bilinear transformation ..... 305
bridge configuration diagrams
  full-bridge ..... 31
  half-bridge ..... 30
  quarter-bridge ..... 29

### *C*

calibration
  self-calibration ..... 16, 24, 47
  shunt ..... 16, 19, 22, 26, 46
CIC filters ..... *See* digital filtering
clock ..... 57, 70, 72, 73, 107, 166, 177, 262
completion resistor ..... 40
  default settings ..... 41
confidence data ..... 57
confidence measurement system ..... 48, 57
configuration storage ..... 49
COPY_SCRATCHPAD
  DS2430 ..... 314
  DS2431 ..... 320

### *D*

daisy-chain configuration ..... 297
declaration of conformity ..... 11
default configuration
  completion resistor ..... 41
  input multiplexer ..... 41
DEVICE layer ..... 56
DHCP ..... 33, 61
digest ..... 49, 101, 133, 175
digital filters ..... 45
  CIC ..... 303
  FIR ..... 45, 303
  IIR ..... 45
digital I/O ..... 51
  pin assignments ..... 51
DIO ..... *See* digital I/O
  bank ..... 51, 73, 139, 140, 240
driver ..... 27, 33, 67
DSP filters
  CIC compensation filter ..... 303
  FIR decimation filter ..... 304
  IIR filters ..... 304

### *E*

engineering unit conversion ..... 35
engineering unit conversion calculations
  full-bridge bending Poisson strain ..... 39
  full-bridge bending strain ..... 38
  full-bridge Poisson strain ..... 38
  half-bridge bending strain ..... 37
  half-bridge Poisson strain ..... 37
  linear ..... 39
  nonstandard ..... 39
  quarter-bridge strain ..... 36
  ratiometric ..... 39
  voltage ..... 39
error messages ..... 289
EU ..... *See* engineering unit conversion
excitation source ..... 42
excitation source measurement ..... 43
explanation of specifications ..... 22

### *F*

factory default settings ..... 65
FIFO ..... 56
filtering
  analog anti-aliasing ..... 301
  IIR ..... 45, 151, 155, 249, 302
  limitations ..... 304
FIR filters ..... *See* digital filters
firmware upgrade ..... 64
full-bridge ..... 19, 22, 23, 31, 37, 38, 39, 42
function calls ..... 85
function return value ..... 85
function set ..... 90
function tree ..... 85

### *G*

gage factor ..... 41
  equation ..... 41
gain ..... 18, 42, 148, 246
gain error ..... 22, 23
gauge factor ..... *See* gage factor
GET_URN ..... 307, 310, 311
group delay ..... 304

### *H*

half-bridge ..... 30, 36, 37, 42

### *I*

IIR ..... 151, 155, 249, 302
Index web page ..... 59
infinite impulse response filtering ..... *See* IIR
initialize acquisition ..... 84
input multiplexer ..... 40
  default settings ..... 41
installation location ..... 27
instrument driver ..... 67
IP ..... *See* network configuration

## *L*


LAN Configuration Initialize ..... 33, 34, 54, 61
LCI ..... *See* LAN Configuration Initialize
lead wire ..... 22, 23, 25
locking ..... 48
LXI Trigger Bus ..... 52
  pin assignments ..... 52


## *M*


MAC address ..... 61
MAC address ..... 34
matched-Z transformation ..... 305
maximizing measurement performance ..... 24
measurement range ..... 42
MicroLAN ..... *See* MLAN
MLAN ..... 52, 307
  checksums ..... 324
  CRC checking ..... 326
  printing packets ..... 325
  receiving ..... 324
  sending ..... 324
  version information ..... 326
multi-instrument operation ..... 57, 297
  daisy-chain configuration ..... 297
  master ..... 15, 57, 70, 73, 166, 177, 262, 271, 298
  slave ..... 15, 57, 70, 73, 166, 177, 262, 271, 298
  star configuration ..... 298
  star/daisy-chain ..... 300
  synchronization ..... 297
  triggering ..... 300
multiplexer ..... *See* input multiplexer


## *N*


network configuration ..... 33
  resetting ..... *See* LAN configuration initialize
network configuration web page ..... 61
NTP ..... 63


## *P*


pattern
  arm ..... 72, 73, 116, 161, 223, 257
  trigger ..... 71, 163, 182, 259, 277
*plug&play* driver ..... 67
Poisson ratio ..... 41
  equation ..... 41


## *Q*


quarter-bridge ..... 23, 29, 35, 40


## *R*


READ_MEMORY
  DS2430 ..... 315
  DS2431 ..... 317, 321
READ_SCRATCHPAD ..... 307, 308
  DS2430 ..... 313
  DS2431 ..... 319
reboot web page ..... 61
reset
  network configuration ..... 54
reset web page ..... 60
retreiving data ..... 76
  asynchronous streaming data interface ..... 79
  read FIFO ..... 77


## *S*


sample clock ..... 70, 71, 166, 262
sample code
  closing a session ..... 67
  configuring the acquisition channels ..... 68, 69, 83
  multiple instrument configuration ..... 73
  opening a session ..... 67
  standalone configuration ..... 72
sample rates ..... 44
sampling rate ..... 18
scan list configuration ..... 44
scanlist
  confidence ..... *See* confidence scanlist
self-calibration ..... *See* calibration
self-test ..... 88, 216, 217, 218
shunt calibration ..... *See* calibration
shunt calibration configuration ..... 46
SNTP ..... 63
specifications
  bridge completion ..... 18
  bridge excitation ..... 18
  confidence measurements ..... 19
  confidence trigger bus ..... 21
  digital I/O ..... 21
  environmental ..... 21
  filtering ..... 20
  full-bridge strain measurements ..... 19
  general ..... 18
  input characteristics ..... 20
  mechanical ..... 21
  power requirements ..... 21
  quarter-bridge strain measurements ..... 19
  shunt calibration ..... 19
  voltage measurements ..... 20
  wideband output ..... 20
star configuration ..... 298
starting acquisition ..... 84
stopping acquisition ..... 84
strain ..... 35
  full-bridge ..... 31, 37, 38, 39
  function ..... 18
  half-bridge ..... 30, 36, 37
  quarter-bridge ..... 23, 29, 35, 40
  units ..... 176, 270
strain conversions units ..... 45
strain gage connector ..... 28
streaming data
  advanced ..... 83
  basic ..... 80
  callback function ..... 81
synchronization signal ..... 57


## *T*


tare ..... 45
TCP ..... *See* network configuration
TEDS ..... 52, *See* MLAN
  wiring schematic ..... 53
time configuration ..... 63
transducer electronic data sheets ..... *See* TEDS
transformations ..... 305
  bileniar ..... 305
  matched z ..... 305
TRIG event ..... *See* triggering
TRIG layer ..... 56
trigger initialize ..... 46, 56
trigger model ..... 55
trigger source programming ..... 71
triggering ..... 55, 300

## *U*

unstrained voltage measurement ..... 43

## *V*

voltage measurement configuration diagrams
floating input ..... 32
grounded input ..... 33
vtex1629_abort ..... 95
vtex1629_allow_all_channels ..... 96
vtex1629_break_lock ..... 97
vtex1629_check_lock ..... 98
vtex1629_clear_stored_config ..... 99
vtex1629_close ..... 100
vtex1629_compare_digests ..... 101
vtex1629_dio_clear_event ..... 102
vtex1629_dio_clear_events_all ..... 103
vtex1629_disable_logging ..... 104
vtex1629_disable_streaming_data ..... 105
vtex1629_enable_logging ..... 106
vtex1629_enable_streaming_data ..... 107
vtex1629_enable_streaming_dataEx ..... 109
vtex1629_erase_teds_data ..... 110
vtex1629_error_message ..... 111
vtex1629_error_query ..... 112
vtex1629_findinstr ..... 113
vtex1629_get_arm_count ..... 114
vtex1629_get_arm_delay ..... 115
vtex1629_get_arm_source ..... 116
vtex1629_get_bridge_limit ..... 118
vtex1629_get_bridge_limit_enabled ..... 120
vtex1629_get_cal_coefficients ..... 121
vtex1629_get_cal_file ..... 124
vtex1629_get_cal_file_size ..... 126
vtex1629_get_cal_source ..... 127
vtex1629_get_completion_resistor ..... 128
vtex1629_get_conf_scanlist ..... 129
vtex1629_get_confidence_limit ..... 130
vtex1629_get_confidence_reporting_mode ..... 132
vtex1629_get_current_config ..... 133
vtex1629_get_dio_ output ..... 140
vtex1629_get_dio_bank0_direction ..... 134
vtex1629_get_dio_bank0_pullup ..... 135
vtex1629_get_dio_bank1_direction ..... 136
vtex1629_get_dio_bank1_pullup ..... 137
vtex1629_get_dio_config_events ..... 138
vtex1629_get_dio_input ..... 139
vtex1629_get_dsp_version ..... 141
vtex1629_get_EU_conversion ..... 142
vtex1629_get_euconv_dynamic_excitation_enabled ..... 143
vtex1629_get_euconv_excitation ..... 144
vtex1629_get_excitation ..... 145
vtex1629_get_excitation_enabled ..... 146
vtex1629_get_fifo_count ..... 147
vtex1629_get_gain ..... 148
vtex1629_get_gauge_factor ..... 149
vtex1629_get_half_bridge_lead_wire_desensitization ..... 150
vtex1629_get_IIR_filter_configuration ..... 151
vtex1629_get_input_multiplexer ..... 153
vtex1629_get_instrument_serial_number ..... 154
vtex1629_get_lead_wire_resistance ..... 155
vtex1629_get_linearscaling_configuration ..... 156
vtex1629_get_lxibus_configuration ..... 157
vtex1629_get_lxibus_input ..... 159
vtex1629_get_lxibus_output ..... 160
vtex1629_get_pattern_arm_configuration ..... 161
vtex1629_get_pattern_trig_configuration ..... 163
vtex1629_get_poisson_ratio ..... 165
vtex1629_get_sample_clock_source ..... 166
vtex1629_get_sample_count ..... 167
vtex1629_get_sample_frequency ..... 168
vtex1629_get_scanlist ..... 169
vtex1629_get_selfcal_status ..... 170
vtex1629_get_shunt_enabled ..... 172
vtex1629_get_shunt_source ..... 173
vtex1629_get_shunt_value ..... 174
vtex1629_get_stored_config_digest ..... 175
vtex1629_get_strain_units ..... 176
vtex1629_get_synch_source ..... 177
vtex1629_get_tare ..... 178
vtex1629_get_teds_data ..... 179
vtex1629_get_trigger_count ..... 180
vtex1629_get_trigger_delay ..... 181
vtex1629_get_trigger_source ..... 182
vtex1629_get_trigger_timer ..... 183
vtex1629_get_unstrained_voltage ..... 184
vtex1629_identify_sensor ..... 185
vtex1629_init ..... 186
vtex1629_load_stored_config ..... 187
vtex1629_lock ..... 188
vtex1629_measure_confidence ..... 189
vtex1629_measure_excitation_voltage ..... 191
vtex1629_measure_lead_wire_resistance ..... 193
vtex1629_measure_unstrained_voltage ..... 195
vtex1629_read_fifo ..... 196
vtex1629_read_fifoEx ..... 198
vtex1629_read_teds_MLAN ..... 200
vtex1629_read_teds_URN ..... 201
vtex1629_reset ..... 202
vtex1629_reset_fifo ..... 203
vtex1629_reset_tare ..... 204
vtex1629_reset_trigger_arm ..... 205
vtex1629_revision_query ..... 206
vtex1629_self_cal_clear ..... 207
vtex1629_self_cal_clear_stored ..... 208
vtex1629_self_cal_get_status ..... 209
vtex1629_self_cal_init ..... 210
vtex1629_self_cal_is_running ..... 212
vtex1629_self_cal_is_stored ..... 213
vtex1629_self_cal_load ..... 214
vtex1629_self_cal_store ..... 215
vtex1629_self_test ..... 216
vtex1629_self_test_get_status ..... 217
vtex1629_self_test_init ..... 218
vtex1629_send_dio_pulse ..... 219
vtex1629_send_lxibus_pulse ..... 220
vtex1629_set_arm_count ..... 221
vtex1629_set_arm_delay ..... 222
vtex1629_set_arm_source ..... 223
vtex1629_set_bridge_limit ..... 224
vtex1629_set_bridge_limit_enabled ..... 226
vtex1629_set_cal_source ..... 227, 228
vtex1629_set_completion_resistor ..... 229
vtex1629_set_conf_scanlist ..... 230
vtex1629_set_confidence_limit ..... 231
vtex1629_set_confidence_reporting_mode ..... 233
vtex1629_set_dio_bank0_direction ..... 234
vtex1629_set_dio_bank0_pullup ..... 235
vtex1629_set_dio_bank1_direction ..... 236
vtex1629_set_dio_bank1_pullup ..... 237
vtex1629_set_dio_config_events ..... 238
vtex1629_set_dio_output ..... 240
vtex1629_set_EU_conversion ..... 241
vtex1629_set_euconv_dynamic_excitation_enabled ..... 242
vtex1629_set_euconv_excitation ..... 243
vtex1629_set_excitation ..... 244
vtex1629_set_excitation_enabled ..... 245

vtex1629_set_gain....246
vtex1629_set_gauge_factor....247
vtex1629_set_half_bridge_lead_wire_desensitization....248
vtex1629_set_IIR_filter_configuration....249
vtex1629_set_input_multiplexer....251
vtex1629_set_lead_wire_resistance....252
vtex1629_set_linearscaling_configuration....253
vtex1629_set_lxi_bus_output....256
vtex1629_set_lxibus_configuration....254
vtex1629_set_pattern_arm_configuration....257
vtex1629_set_pattern_trig_configuration....259
vtex1629_set_poisson_ratio....261
vtex1629_set_sample_clock_source....262
vtex1629_set_sample_count....263
vtex1629_set_sample_frequency....264
vtex1629_set_scanlist....265
vtex1629_set_shunt_enabled....266
vtex1629_set_shunt_source....267
vtex1629_set_shunt_value....269
vtex1629_set_strain_units....270
vtex1629_set_synch_source....271
vtex1629_set_tare....273
vtex1629_set_teds_data....274
vtex1629_set_trigger_count....275
vtex1629_set_trigger_delay....276
vtex1629_set_trigger_source....277
vtex1629_set_trigger_source_timer....278
vtex1629_set_trigger_timer....279
vtex1629_set_unstrained_voltage....280
vtex1629_soft_arm....281
vtex1629_soft_synch....282
vtex1629_soft_trig....283
vtex1629_store_current_config....284
vtex1629_trig_init....285
vtex1629_unlock....286
vtex1629_write_teds_MLAN....287
vtex1629_zero_cal....288
VXI-11 device discovery....62

## *W*

warm-up time....27
Web Page Operation....59
webpage password....60
WEEE....12
wideband output....49
  pin assignments....50
WRITE_AND_COPY_SCRATCHPAD
  DS2430....316
  DS2431....322
WRITE_SCRATCHPAD
  DS2430....311
  DS2431....317