EMX-4250 / EMX-4251

DCV/ACV and IEPE Input 16 / 8 Channel 204.8 kSa/s 24-bit Digitizer

EMX-4016B/ EMX-4016M

16 Channel Breakout Box w Buffer Out / w Bridge Signal Conditioning
For EMX-425x

EMX-4350

DCV/ACV and IEPE Input 4 Channel 625 kSa/s 24-bit Digitizer

EMX-4380

ACV, Charge, and IEPE Input 4 Channel 625 kSa/s 24-bit Digitizer

USER'S MANUAL

P/N: 82-0142-100
Released Sep 26, 2017

VTI Instruments Corp.

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

TABLE OF CONTENTS

CERTIFICATION

VTI Instruments Corp. (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. Note that the contents of this document are subject to change without notice.

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. Note that specifications are subject to change without notice.

For warranty service or repair, this product must be returned to a VTI Instruments 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.

VTI Instruments Corp. shall not be liable for injury to property other than the goods themselves. Other than the limited warranty stated above, VTI Instruments Corp. 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.

TRADEMARKS

Java Runtime Environment™ are trademarks or registered trademarks of Sun Microsystems, Inc. or its subsidiaries in the United States and other countries. LabVIEW™ and LabWindows/CVI™ are trademarks of National Instruments Corporation. Visual Basic®, Windows®, and Internet Explorer® are registered trademarks of the Microsoft Corporation or its subsidiaries. Linux® is a registered trademark of the Linux Foundation. IVI™ is a trademark of the IVI Foundation. Bonjour™ is a trademark of Apple, Inc.

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.

VTI Instruments Corp.
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	VTI Instruments Corporation
MANUFACTURER'S ADDRESS	2031 Main Street Irvine, California 92614-6509
PRODUCT NAME	Smart Dynamic Signal Analyzer
MODEL NUMBER(S)	EMX-4250, EMX-4350 and EX-4380 DSA Modules
PRODUCT OPTIONS	All
PRODUCT CONFIGURATIONS	All

VTI Instruments. 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	IEC 61010-1:2001 (2nd Ed.); EN 61010-1:2001 (2nd Ed.)
EMC	IEC 61326-1:2006 EN55011 Class A Group IEC 61000-4-2 IEC 61000-4-3 IEC 61000-4-4 IEC 61000-4-5 IEC 61000-4-6 IEC 61000-4-8 IEC 61000-4-11 CISPR 11 (2004) Class A ICES-001 (Issue 4) AS/NZS CISPR 11 (2nd ED) Class A FCC Part 15 Subpart B Class A

This product was tested in a typical configuration.

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.

July 2013

[illegible]

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. Note that this product contains no user serviceable parts or spare parts.

Service should only be performed by qualified personnel. Disconnect all power before servicing.

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 or in the manual:

⚠	**ATTENTION** - Important safety instructions
⚡	Indicates hazardous voltage.
⏚	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. The mains outlet that is used to power the equipment must be within 3 meters of the device and shall be easily accessible.
User Proper Fuse	To avoid fire hazard, only use the type and rating fuse specified for this product.
Power Consumption	Prior to using EMX series plug-in modules, it is imperative that the power consumption of all modules that will be installed in the mainframe be calculated for all power supply rails. The required information can be found in *Appendix B* of the *EMX Series User's Manual* (P/N: 82-0142-100). *Failure to do so may result in damaging the instrument and the chassis.*

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.
- When selecting the installation location, be certain that there is enough space around the power plug and the outlet so that they are readily accessible. Do not insert the power cord into an outlet where accessibility to the plug cord is poor.
- All unused slots should be closed with the dummy filler panels to ensure a proper air circulation. This is critical to avoid overheating of the cards.
- 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 VTI Instruments customer support centers.

VTI Instruments Corp.
World Headquarters

VTI Instruments Corp.
2031 Main Street
Irvine, CA 92614-6509

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

VTI Instruments
Cleveland Instrument Division

5425 Warner Road
Suite 13
Valley View, OH 44125

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

AMETEK Instruments India Pvt. Ltd.
VTI - Bangalore Instrument Division

Divyasree N R Enclave, Block A,
4th Floor, Site No 1, EPIP Industrial Area,
Whitefield, Bengaluru 560066, India

Phone: +91 (0) 80 6782 3200
Fax: +91 (0) 80 6782 3232

Technical Support

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

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

SECTION 1

INTRODUCTION

OVERVIEW

The EMX-4250, EMX-4251, EMX-4350 and EMX-4380 (referred to collectively as the "EMX-4250/4350/4380") are Smart Dynamic Signal Analyzers incorporate best-in-class analog design methodology to deliver industry leading measurement accuracy. This instrument is ideal for a wide range of applications including noise, vibration, and harshness (NVH); machine condition monitoring; rotational analysis; acoustic test; modal test; as well as general purpose high speed digitization and signal analysis.

FEATURES

Optimized Performance

Measurement performance is elevated to new levels with up to 625 k samples / second / channel data rates, true differential inputs with superior common mode performance (CMRR of -120 dB) reducing unwanted noise and interference, an industry leading spurious free dynamic range (SFDR of -125 dB) offering exceptional measurement fidelity, and uncompromised IEPE excitation flexibility, fully programmable from 2 mA to 20 mA, to maximize transducer performance and response. Industry-leading -125 dB spurious free dynamic range (SFDR) is a key measure of the superior measurement fidelity provided by this instrument

Self-Calibration

Runtime self-calibration ensures that instruments deliver the most accurate results possible by compensating for ambient temperature fluctuations, without the need to disconnect field wiring. This maximizes measurement accuracy across the entire measurement path using precision internal voltage sources to validate and adjust coefficients. This eliminates inaccuracies generated by internal circuitry temperature gradients. Embedded NIST traceable calibration eliminates lengthy test system down-time, simplifies calibration processes, and reduces spare equipment requirements, maximizing facility up-time and utilization with this completely automated embedded process. All internal calibration can be performed in-place without removing instrumentation.

There are two BNC connectors in the rear panel of EMX-4016B/EMX4016M BoB; one for trigger input and one for calibration output. The calibration output is used to calibrate the built-in calibration signals used by the EMX-425X module to perform self-calibration. The trigger input is used for trigger the digitizer by an external signal.

"Best-in-Class" Software

X-Modal is an experimental modal analysis software features intuitive task oriented user interfaces, extensive modal parameter estimation algorithms, parallel display capabilities, flexible data management, and unparalleled channel expandability. This software is MATLAB based open source.

EXLab is an easy to use, turn-key, data acquisition solution featuring intelligent configuration capabilities, automatic device discovery, extensive time and frequency domain data visualization, and post-acquisition display and analysis tools.

Flexible Application Programming Options

EMX-4250/4350/4380 module is delivered with an application programming interface (API) that conforms to the industry standard IVI™ specifications for its class. The IVI drivers port seamlessly into the most commonly used application development environments such as LabView™, LabWindows/CVI™, Matlab®, and Visual Studio®, among others. The intuitive APIs simplify programming and expose all available instrument functionality, eliminating the need for low-level coding. The EMX series driver is designed to support advanced functionality such as:

- Sophisticated arming and triggering options.
- Multiple FIFO models.
- Streaming API for high speed data transfer.
- LXISync API and IEEE1588 to synchronize with other LXI devices over Ethernet.

While IVI is intended for Windows®-based operating systems, VTI's innovative approach to driver development allows users to develop their applications using an IVI-like interface that can be imported into Linux® and other operating systems. This flexibility provides system developers with true OS independence without the need to sacrifice the convenience that instrument drivers deliver.

PREPARATION FOR USE

UNPACKING

When an EMX-4250/4350/4380 is unpacked from its shipping carton, the contents should include the following items:

- An EMX-4250, EMX-4350, or EMX-4380
- VTI Instruments Distribution CD

All components should be immediately inspected for damage upon receipt of the unit. ESD precautions should be observed while unpacking and installing the instrument into a PXI Express chassis. The part number, model number, and serial number can be found on the side cover of the card as shown below.

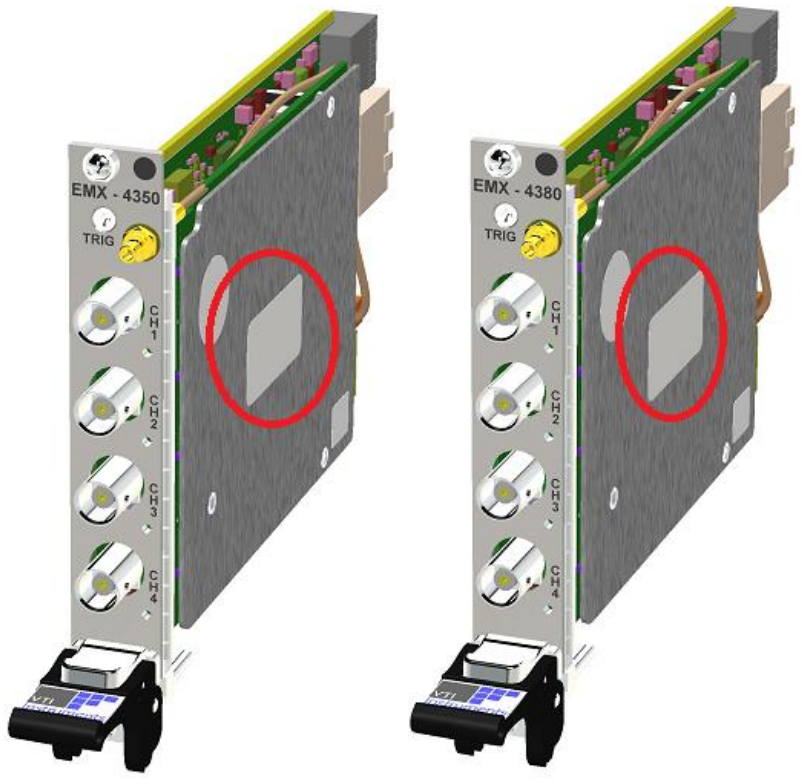


Figure 1-1: Digitizer Serial Number Location

INSPECTING THE EMX-4250/4350/4380

The EMX-4250/4350/4380 modules were carefully inspected both mechanically and electrically before shipment. They should be free of marks or scratches and they should meet their published specifications upon receipt. If the module was damaged in transit, do the following:

- Save all packing materials
- File a claim with the carrier
- Call a VTI Instruments sales and service office

INSTALLATION

INSTALLING EMX-4250/4350/4380

1) Set up the PXI Express chassis. See the chassis' installation guide for assistance.
2) Make sure the PXI Express chassis is powered off.
3) Select a PXIe slot or a hybrid slot in the PXI Express chassis for the EMX-4250/4350/4380 and insert it carefully.
4) After the EMX-4250/4350/4380 is inserted all the way, secure it with the screws at the top and bottom of the EMX-4250/4350/4380 front panel.

INSTALLING EMX-4016B/EMX-4016M

1) The EMX-4016B/ EMX-4016M BoB is supplied in a standard 19" rack mounting enclosure. For bench use, we recommend that the four self-adhesive rubber feet are attached to the bottom panel.
2) Connect the power supply adaptor, and ensure correct voltage level and necessary grounding
3) Connect the 25 pin Micro D-Sub cables from EMX-4250/51 to the BoB. Observe that J1 & J2 should not be interchanged, if you are using EMX-4250. If you are using EMX-4251, connect the cable only to J1 (Ch-1 to Ch-8)
4) Connect transducer cables to appropriate inputs, and buffered outputs to external equipment
5) The EMX-4016B/ EMX-4016M BoB does not use internal fans for forced air cooling. It does not require special cooling arrangements. Do not keep it near to heat sources, and ensure thermally stable operating environment (with ambient environment conditions as per specifications)

LED STATUS INDICATIONS:

1) Power LED (Glows when the power supply adaptor is connected)
2) Bi-colour Input signal health monitoring LED (LEDs will be GREEN during normal operation and will turn RED if an OPEN or SHORT is detected by the digitizer module when the input is set to IEPE)

DETERMINE SYSTEM POWER REQUIREMENTS

The power requirements of the PXIe module are provided in the *Power Consumption* section (page 62). As with any backplane-based system where modules share power, such as VXI and PXI, the possibility exists where certain plug-in module combinations can draw too much power from a power supply rail. As such, it is imperative that the chassis provide adequate power for the modules installed.

INSTALLING DRIVER SOFTWARE

After the hardware has been assembled, the next step in installing an EMX-4250/4350/4380 is to install the instrument driver. Refer to the *Getting Started with the Instrument* in *Section 2* to continue the installation process.

STORAGE AND SHIPMENT

STORING INSTRUMENTS

Store the module in a clean, dry, and static-free environment. For other requirements, see storage and transport restrictions in *Specification*.

TRANSPORTING INSTRUMENTS

Package the module using the original factory packaging or packaging identical to the factory packaging. Containers and materials identical to those used in factory packaging are available through VTI Instruments offices.

If returning the module to VTI Instruments for service, contact a VTI Service Center to set up an RMA. The following information will be required:

- Serial number
- Model number
- Type of service required
- Return address
- If applicable, a description of the problem that is being encountered which provides specific detail relating the instrument being returned.

In any correspondence, refer to the serial number and RMA number. Mark the container "FRAGILE" to ensure careful handling. If it is necessary to package the module in a container other than the original packaging, observe the following (although use of other packaging material is not recommended):

- Wrap the module in heavy paper or anti-static plastic
- Protect the front panel with cardboard
- Use a double-wall carton made of at least 350 lbs test material
- Cushion the module to prevent damage

CAUTION Do not use styrene pellets in any shape as packing material for the module. The pellets do not adequately cushion the module and do not prevent the module from shifting in the carton. In addition, the pellets create static electricity that can damage electronic components.

SPECIFICATIONS

DIGITIZER SPECIFICATIONS

	EMX-4250/4251	EMX-4350	EMX-4380
Input Specifications			
No. of Channels	4250: 16ch / 4251: 8ch	4ch	4ch
Input Connector	25-pin Micro-D Molex 83614-9012	BNC, shell floating from chassis ground Center Pin = HI; Shell = LO	
Input Type	Differential Pseudo-differential	Differential	Differential Single-Ended
Input Ground Isolation	None	None	ON or OFF ON: 50 Vdc, 100 MΩ
Input Coupling	AC, DC	AC, DC	AC
Common Mode Rejection (CMR) @ 100 Hz, AC or DC Coupling	<-80 dB	<-80 dB	<-80 dB
Input Over-Voltage Protection			
Differential	±40 V	±40 V	±40 V
Pseudo-Differential	HI: ±40V LO: ±10V	N/A	N/A
Input Impedance			
Differential Voltage	4 MΩ // 200 pF	2.4 MΩ // 140 pF	6 MΩ // 90 pF
Pseudo-Differential	HI: 2 MΩ // 200 pF LO: 100 Ω	N/A	N/A
Charge	N/A	N/A	500 MΩ
Input Ranges			
Voltage/IEPE (±V-pk)	0.1, 0.2, 0.5, 1, 2, 5, 10	0.1, 1, 10, 20	
Charge (±pC-pk)	N/A	N/A	100, 1k, 10k
IEPE			
IEPE Current (mA)	4.5, 10 mA	0-20 mA <0.1 mA resolution	4.5, 10 mA
IEPE Current Accuracy	±5%	±0.1 mA	±5%
IEPE Compliance Voltage @ 5 mA	>21 V		
IEPE Open/Short Fault Detection	Break-Out Box LED & Software	Front Panel LED & Software	
TEDS	IEEE 1451.4 / 1-wire		
Max. Input Shunt Capacitance for TEDS to work	2,600 pF	2,600 pF	2,600 pF
Measurement Accuracy			
Amplitude Accuracy @ 1 kHz			
Voltage & IEPE	Typ.: ±0.02 dB Max.: ±0.05 dB	Typ.: ±0.02 dB Max.: ±0.05 dB	Typ.: ±0.02 dB Max.: ±0.05 dB
Charge	N/A	N/A	Typ.: ±0.1 dB Max.: ±0.15 dB
Residual DC Offset			
Voltage & IEPE (AC or DC Coupling)	0.1 V, 0.2 V: <±0.2 mV Other Ranges: <±2 mV	0.1V, 1V: <±0.1 mV 10V, 20V: <±1 mV	
Charge	N/A	N/A	<1pC
Channel-to-Channel Matching			
Amplitude Match, @ 1 kHz	±0.005 dB	±0.01 dB	
Phase Match @ 1 kHz	±0.002°	±0.002°	
Phase Linearity	±0.05° up to 96 kHz	±0.05° up to 270 kHz	
Phase Accuracy vs. Trigger	<0.1° @ 1 kHz	<0.1° @ 1 kHz	<0.1° @ 1 kHz
Frequency Response Characteristics			
Slew Rate: 10% to 90% of Range	1.1 µs	0.5 µs	0.5 µs
AC Coupling -3 dB Corner (Hz)	IEPE/Volts: 0.24 Hz	IEPE/Volts: 0.5 Hz	IEPE/Volts: 0.20 Hz Charge: 0.32 Hz
Maximum Bandwidth			

	EMX-4250/4251	EMX-4350	EMX-4380
Volts (-0.2 dB), 625 kSa/s	N/A	270 kHz	270 kHz
Charge (-1 dB), 625 kSs/s, CSH=2 nF	N/A	N/A	270 kHz
Volts (-0.1 dB), 204.8 kSa/s	96 kHz	88 kHz	88 kHz
Alias Rejection, Typical, 10 V Range	-90 dB	-86 dB	
ADC			
ADC Resolution	24-bits	24-bits	24-bits
ADC Data Rate (Sa/s) (f_{DATA})	131,072: ÷1, ÷2, ÷4 204,800: ÷1, ÷2, ÷4	409,600: ÷1, ÷2, ÷4, ÷8 524,288: ÷1, ÷2, ÷4, ÷8 625,000: ÷1, ÷2, ÷4, ÷8	
ADC Post-Decimation	÷2^N, N=0, 1, .., 16 ÷5: Optional	÷2^N, N=0, 1, .., 16 ÷5: Optional	
ADC Pass-Band (0.0001 dB ripple)	0.454 * f_{DATA}	0.424 * f_{DATA}	
ADC Pass-Band (-0.1 dB)	0.469 * f_{DATA}	0.432 * f_{DATA}	
ADC Stop-Band (<-100 dB)	0.546 * f_{DATA}	0.576 * f_{DATA}	
Group Delay	39 ÷ f_{DATA}	28 ÷ f_{DATA}	
Dynamic Characteristics			
Noise @ frequencies >100 Hz	0.1 V: 30 nV/√Hz 0.2 V: 30 nV/√Hz 0.5 V: 40 nV/√Hz 1 V: 50 nV/√Hz 2 V: 86 nV/√Hz 5 V: 200 nV/√Hz 10 V: 400 nV/√Hz	0.1 V: 30 nV/√Hz 1 V: 40 nV/√Hz 10 V: 270 nV/√Hz 20 V: 500 nV/√Hz	
THD (dB) Typical @ 1 kHz	-86 dB	-100 dB	
SFDR (dB) Typical, @1 kHz	0.1 V: -92dB 0.2 V: -98 dB 0.5 V: -106 dB 1 V: -112 dB 2 V: -118 dB 5 V: -120 dB 10 V: -120 dB	0.1 V: -92 dB 1 V: -112 dB 10 V: -124 dB 20 V: -130 dB	
Channel-to-Channel Crosstalk, 1 kHz	-98 dB	0.1 V: -112 dB; Other Ranges: -122 dB	
Other			
Built-In Self-Calibration	Yes	Yes	Yes (Voltage only)
On-board Memory	128MBytes EMX-4250: 1.875 MSa/ch EMX-4251: 3.75 MSa/ch	128MBytes 7.5 MSa/ch	

Table 1.1 Digitizer Specifications

TRIGGER INPUT SPECIFICATIONS

	EMX-4250/4251	EMX-4350	EMX-4380
Input Connector	25-pin Micro-D Molex 83614-9012 Pins 5 (Trig) & 18 (Gnd)	SMB, shell connected to chassis ground	
Input Type	Single-Ended	Single-Ended	Single-Ended
Input Impedance	100 kΩ // 220pF	100 kΩ // 220 pF	100 kΩ // 220 pF
Input Voltage	0 V to 3.3 V	0 V to 3.3 V	0 V to 3.3 V
Input HIGH (V_{IH})	>1.6 V	>1.6 V	>1.6 V
Input LOW (V_{IL})	<0.6 V	<0.6 V	<0.6 V
Input Over-Voltage Protection	-1 V to +7 V	-1 V to +20 V	-1 V to +20 V
Input Hysteresis	0.76 V	0.76 V	0.76 V

Table 2.2 Digitizer Trigger Input Specifications

BREAK OUT BOX SPECIFICATIONS

	EMX-4016	EMX-4016B	EMX-4016M
Input Specifications			
No. of Channels	16 channels	16 channels	16 channels
Input Connector	BNC Isolated from Chassis	BNC Isolated from Chassis	RJ45 connector (8-pin), pinout compatible with EX1629
Input Type	Differential	Differential	Differential
Input Range	±24V	±24V	±10Vpk Min. Maximum Input Voltage: ±24V Custom models can be made with other voltage/gain ranges
Input Ground Isolation	No channel to channel Ground Isolation	No channel to channel Ground Isolation	No channel to channel Ground Isolation
Input Coupling	AC/ DC coupling	AC/ DC coupling	DC coupling only
Input Impedance (Buffered Output)	NA	10 MΩ Typical each input to ground, 20MΩ differential.	10 MΩ Typical each input to ground, 20MΩ differential.
Common Mode Rejection (Buffered Output)	NA	86dB Typical, 80dB Min. <10kHz (0°C to +55°C)	86dB Typical, 80dB Min. <10kHz (0°C to +55°C)
Channel-to-Channel Crosstalk	NA	-60dB Typical, <1kHz Overdriving 1 channel does not affect performance of other channels	-60dB Typical, <1kHz Overdriving 1 channel does not affect performance of other channels
Input Protection	ESD: Bidirectional TVS IEC61000-4-2, ±30kV Contact & Air	ESD: Bidirectional TVS IEC61000-4-2, ±30kV Contact & Air	ESD: Bidirectional TVS IEC61000-4-2, ±30kV Contact & Air
Bridge Zero/Balance	Not Supported	No zero/balance adjustment	No zero/balance adjustment
Bridge Configuration	Not Supported	Not Supported	Full Bridge only
Transducer support	IEPE transducers or piezoelectric transducers with a Remote Charge Convertor (RCC)	IEPE transducers or piezoelectric transducers with a Remote Charge Convertor (RCC)	Full Bridge type sensors
25pin DSUB Output Specifications			
No. of Channels	16 channels	16 channels	16 channels
Output Connector	25-pin Molex 83614-9012	25-pin Molex 83614-9012	25-pin Molex 83614-9012
Gain	Pass-through	Pass-through	Pass-through
Output Swing	±10Vpk	±10Vpk	±10Vpk
Bandwidth	-3dB: ≥ 100 kHz Typical	-3dB: ≥ 100 kHz Typical	-3dB: ≥ 100 kHz Typical
Output Ground Isolation	No channel to channel Ground Isolation	No channel to channel Ground Isolation	No channel to channel Ground Isolation
Buffered Output Specifications			
No. of Channels		16 channels	16 channels
Type		BNC Single-Ended	BNC Single-Ended
Gain		1 ±0.1%	1 ±0.05% Typical, ±0.1% Max. (0°C to +55°C)
Output Swing		±10Vpk	±10Vpk
Output Current Drive		10mA Minimum	10mA Minimum

	EMX-4016	EMX-4016B	EMX-4016M
Offset/Zero		< 10mV	<0.5mV Typical, <1mV Max. (0°C to +55°C) Typical Stability: ±20uV/°C
Output Impedance		< 50Ω	< 50Ω
Noise		30 µVRMS Typical; 60 µVRMS Max	30 µVRMS Typical; 60 µVRMS Max.
Bandwidth		-3dB: ≥ 200 kHz Typical	-3dB: ≥ 100 kHz Typical
Slew Rate		20V/us Typical	≥ 10 V/µs
Output Ground Isolation		No channel to channel Ground Isolation	No channel to channel Ground Isolation
Output Protection		ESD: ±32V Unidirectional TVS IEC61000-4-2, ±15kV Contact, ±30kV Air	ESD: Bidirectional TVS IEC61000-4-2, ±15kV Contact, ±30kV Air
Voltage Excitation specification (EMX-4016 M Only)			
Channel-to-Channel Isolation	All channels share the same ground Each channel provides separate excitation circuitry		
Voltage Excitation Protection	ESD: Bidirectional TVS IEC61000-4-2, ±30kV Contact & Air Crosstalk: A Short does not affect Excitation accuracy in other channels		
Voltage Excitation	Levels: +5VDC & +10VDC Set for all channels by one switch on front panel Accuracy: ±0.16% Typical; ±0.27% Max.; ±0.31% (0°C to +50°C) Typical Stability: ±10 ppm/°C ±7µV/°C ±100ppm/year Load regulation: <0.05% for load change < 32mA Crosstalk: <0.01% effect on other channels from load changes Current Limit: 48mA ±4mA; Output Impedance: <0.1Ω @ <60Hz Noise: 20 µVRMS Typical, 50kHz bandwidth Custom models can be made with other voltage excitation levels		
Voltage Excitation Monitoring	NO monitoring		
Excitation Sensing	None, Optional: Unit can be modified to provide remote excitation sensing		

Table 3.3 Break out box Output Specifications

ENVIRONMENTAL SPECIFICATIONS

	EMX-4250/4251	EMX-4350	EMX-4380	EMX-4016B / EMX-4016M
Operating Environment				
Operating Temperature	0°C to +55°C			
Relative Humidity, Non Condensing	10% to 90%			
Storage Environment				
Storage Temperature	-40°C to +70°C -40°C to +80°C			
Relative Humidity, Non Condensing	5% to 95%			
Altitude	3,000 m			
Pollution Degree	2			
Shock and Vibration				
Sinusoidal	5 Hz to 500 Hz, 0.3 grms. Tested per MIL-PRF-2880F Class 3			
Shock	30 g peak, half-sine, 11 ms pulse. Tested per MIL-PRF-2880F Class 3			

Table 4.4 Environmental Specifications

Notes

1) All specifications are typical unless otherwise stated as a minimum or maximum.
2) All specifications are for Input=Volts, Coupling=DC, unless otherwise noted.
3) All specifications subject to change without notice.
4) All specifications met within 24 hours and 5°C of self-calibration temperature unless otherwise specified

PHYSICAL CHARACTERISTICS

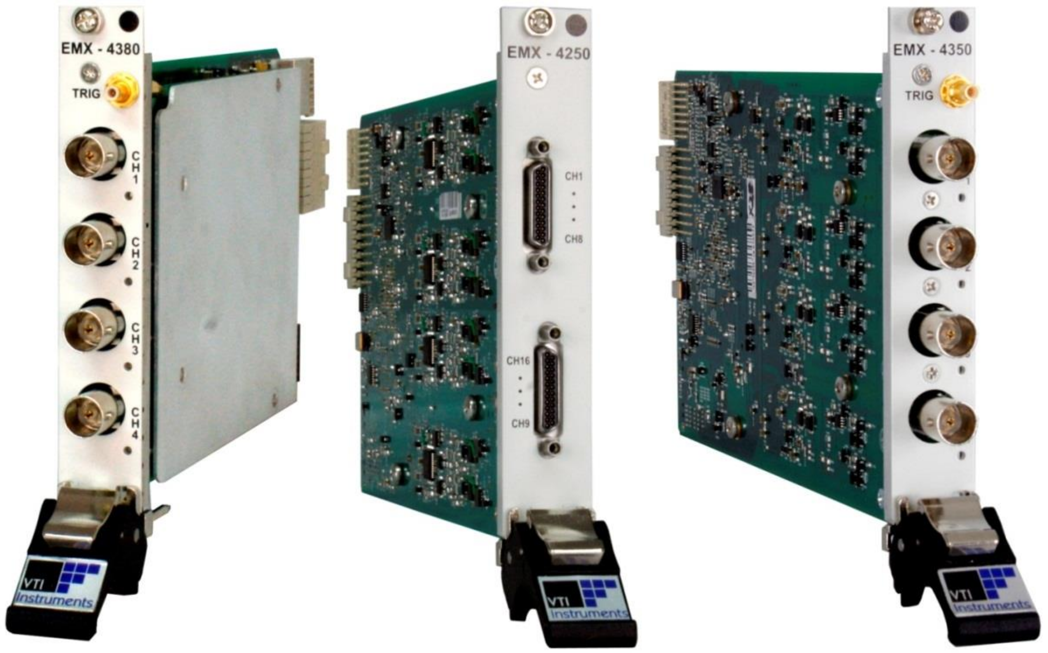


Figure 1-2: Photos: EMX-4380, EMX-4250, and EMX-4350

PHYSICAL SPECIFICATIONS

	EMX-4250/4251	EMX-4350	EMX-4380	EMS-4016B
Dimensions	16 cm x 10cm (6.3 in. x 3.9 in.)			19" x 5.8" x 1.75" (1U)
Weight	280 g / 220 g	300 g	300 g	3 pounds (~ 1.36 KG)
Digitizer Input Connector	25-pin Micro-D Molex 83614-9012	BNC, shell floating from chassis ground Center Pin = HI; Shell = LO		
Trigger Input Connector	Pins 5 (Trig) & 18 (Gnd)	SMB Male Jack Receptacle		

EMX-4250/4251, EMS-4016B/ EMX-4016M CONNECTOR PIN NUMBERING

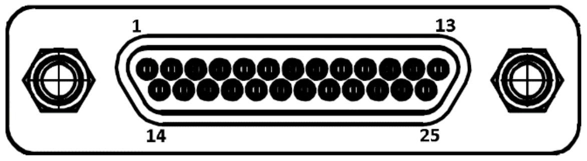


Figure 1-3: EMX-4250, EMS-4016B/ EMX-4016M Input Connector (Front View) Molex 83614-9012 Pin Numbering

EMX-4250/4251, EMX-4016B/EMX-4016M Connector Pin Assignments

The table below shows the connector pin assignments for the EMX-4250/4251, EMX-4016B/EMX-4016M. Signals identified in blue color are the signals referenced to DGND and used to control the fault LED's in the Break-Out-Boxes (BOB's).

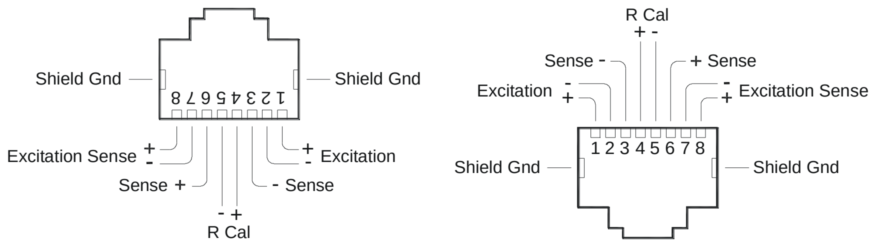
TRIG IN (in green) is referenced to digital ground.

VCALP & VCALN provide the built-in calibration signal out to a BNC in the BOB so it can be connected to a DMM and used during factory calibration.

BOB INT is used to let the EMX-4250/4251 know that a BOB has been connected to the digitizer.

CH #	Signal	Molex 83614-9012		Signal
		Pin #	Pin #	
1	CH1P	1	14	CH1N
2	CH2P	2	15	CH2N
3	CH3P	3	16	CH3N
4	CH4P	4	17	CH4N
	TRIG IN	5	18	DGND
	BOB INT	6	19	DGND
	AGND	7	20	3.3VDC
	VCALP	8	21	I2C CLK
	VCALN	9		I2C DAT
5	CH5P	10	22	CH5N
6	CH6P	11	23	CH6N
7	CH7P	12	24	CH7N
8	CH8P	13	25	CH8N

EMX-4250/4251, EMX-4016B/EMX-4016M DSUB Connector Pin Assignments



Color Coding per TIA/EIA-568-B

Pin 1 - Orange/White	🟧⬜	Pin 5 - Blue/White	🟦⬜
Pin 2 - Orange	🟧	Pin 6 - Green	🟩
Pin 3 - Green/White	🟩⬜	Pin 7 - Brown/White	🟫⬜
Pin 4 - Blue	🟦	Pin 8 - Brown	🟫

Figure 1-4: EMX-4016M RJ-45 Input Connector Pin Assignment

SECTION 2

GETTING STARTED WITH THE INSTRUMENT

INTRODUCTION

This section provides assistance in getting the EMX-4250/4350/4380 instruments running and making simple measurements. It shows how to install the instrument drivers and how to run some of the example programs that are included.

SYSTEM REQUIREMENTS

- A PC with either a supported Windows OS (XP, Vista 32, Windows 7 [32-bit and 64-bit], and Windows 8 [32-bit and 64-bit]) or Linux OS.
- An Ethernet port and a LAN cable to connect to EMX-2500 Ethernet controller.
- For Windows, any programming language that supports IVI-C, or IVI-COM, such as MS Visual Studio, Mathworks Matlab®, NI LabVIEW®. For Linux, a C++ compiler, such as GCC.

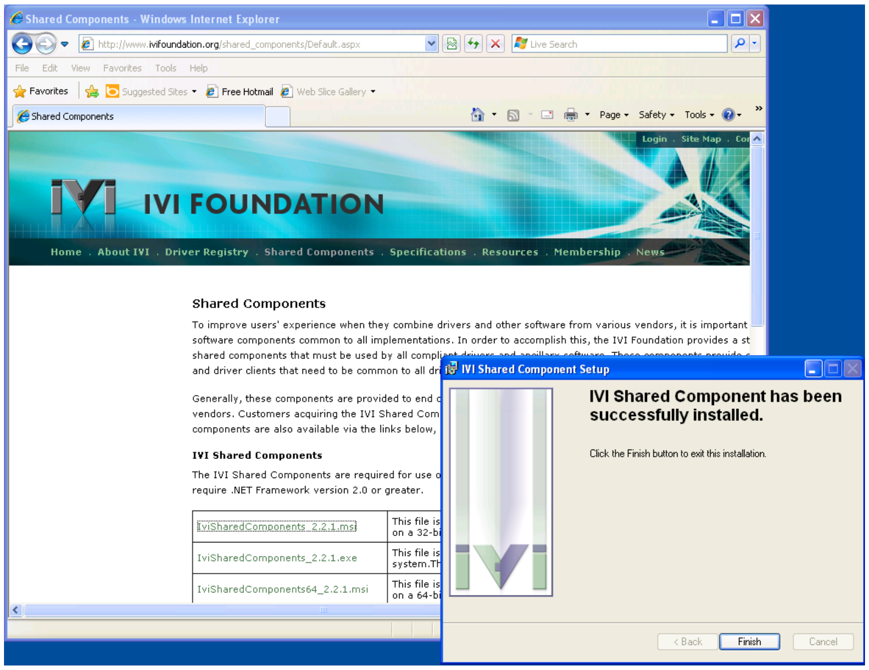
DRIVER INSTALLATION

To control the EMX series instruments programmatically (via a user generated program or through tools such as Agilent VEE®, NI LabVIEW®, Mathworks Matlab®, etc.), two additional components must be installed: the IVI Shared Components library (for Windows OS only) and the provided VTI Instruments driver. For 32-bit Windows OS, install the 32-bit driver. For Windows 7 (64-bit) and Windows 8 (64-bit), the 64-bit driver installer installs both 64-bit and 32-bit compatible drivers. The following sections describe installing the required software.

IVI Shared Components Installation (Windows Only)

If this component was installed during a previous LXI instrument installation, please proceed to *Instrument Driver Installation*. First, close all other open programs, leaving only Windows Explorer open. Navigate to the *<CD-ROM Drive>:\EX Platform Requisites* directory on the CD and run the **IVISharedComponentsX.X.X.exe** program. Next, follow the on-screen instructions. Do not proceed to the next step until this installation completes successfully. If instructed to reboot the PC, it will be necessary to do so at that time.

Alternatively, the latest IVI shared components can be downloaded and installed from IVI Foundation Web page as shown below.


Shared Components - Windows Internet Explorer
http://www.ivifoundation.org/shared_components/Default.aspx
Live Search
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Free Hotmail
Web Slice Gallery
Shared Components
Page
Safety
Tools
Login . Site Map . Co
IVI FOUNDATION
Home . About IVI . Driver Registry . Shared Components . Specifications . Resources . Membership . News
IVI Shared Component Setup
< Back
Finish
Cancel

Shared Components

To improve users' experience when they combine drivers and other software from various vendors, it is important software components common to all implementations. In order to accomplish this, the IVI Foundation provides a st shared components that must be used by all compli and driver clients that need to be common to all dri

Generally, these components are provided to end c vendors. Customers acquiring the IVI Shared Com components are also available via the links below,

IVI Shared Components

The IVI Shared Components are required for use o require .NET Framework version 2.0 or greater.

IviSharedComponents_2.2.1.msi	This file is on a 32-bi
IviSharedComponents_2.2.1.exe	This file is system.Th
IviSharedComponents64_2.2.1.msi	This file is on a 64-bi

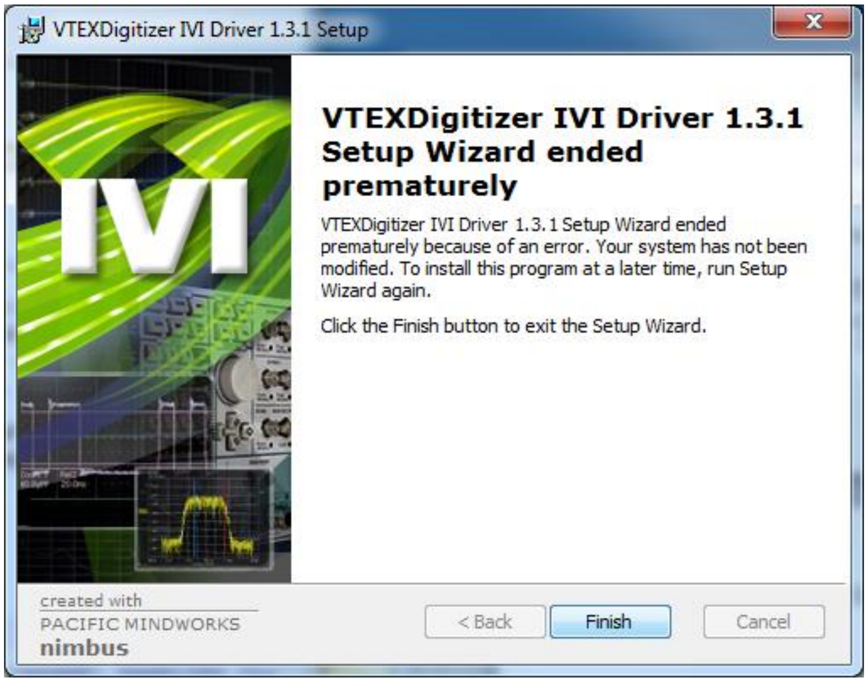
IVI Shared Component has been successfully installed.

Click the Finish button to exit this installation.

Figure 2-1: IVI Shared Component Installation

Instrument Driver Installation

If the VTEXDigitizer/DSA instrument driver was installed previously on the host PC, proceed to *Platform/LXISync Instrument Driver Installation*. To install the VTEXDigitizer/DSA instrument driver, navigate to *<CD-ROM Drive>:\Drivers\LXI Drivers\EMX Series*, on the CD, open the appropriate zip file in this directory, and then run the .msi installer.



VTEXDigitizer IVI Driver 1.3.1 Setup Wizard ended prematurely

VTEXDigitizer IVI Driver 1.3.1 Setup Wizard ended prematurely because of an error. Your system has not been modified. To install this program at a later time, run Setup Wizard again.

Click the Finish button to exit the Setup Wizard.

Figure 2-2: Instrument Driver Installation

The Linux driver (32-bit and 64-bit) is located under *<CD-ROM Drive>:*\Drivers\Linux Drivers\Linux EMX Series. Find out the version of your operating system and unzip the corresponding version into a folder.

Platform/LXISync Instrument Driver Installation

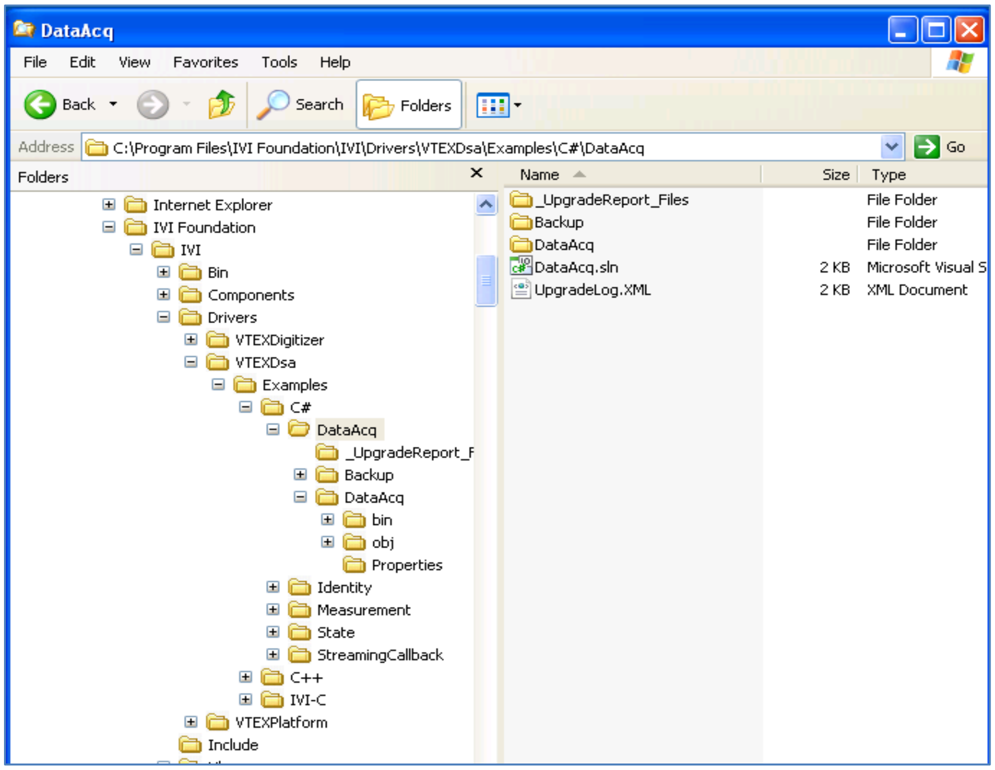
NOTE Complete this step only if the LXISync capabilities of the EMX platform are required. Also, if this driver was installed previously on the host PC, software installation is now complete.

To install the Platform/LXISync Instrument driver, navigate to *<CD-ROM Drive>:\LXI Drivers\EMX Platform Driver, IVI* on the CD and run the .msi installer located in this directory.

Please refer to *VTEX Digitizer/DSA Driver's online help file* for programming guidelines. Additional information about IVI drivers can be found on the web at http://ivifoundation.org. Information about the LXI standard and LXI technology can be found at http://www.lxistandard.org.

BUILDING AND RUNNING EXAMPLE PROGRAMS

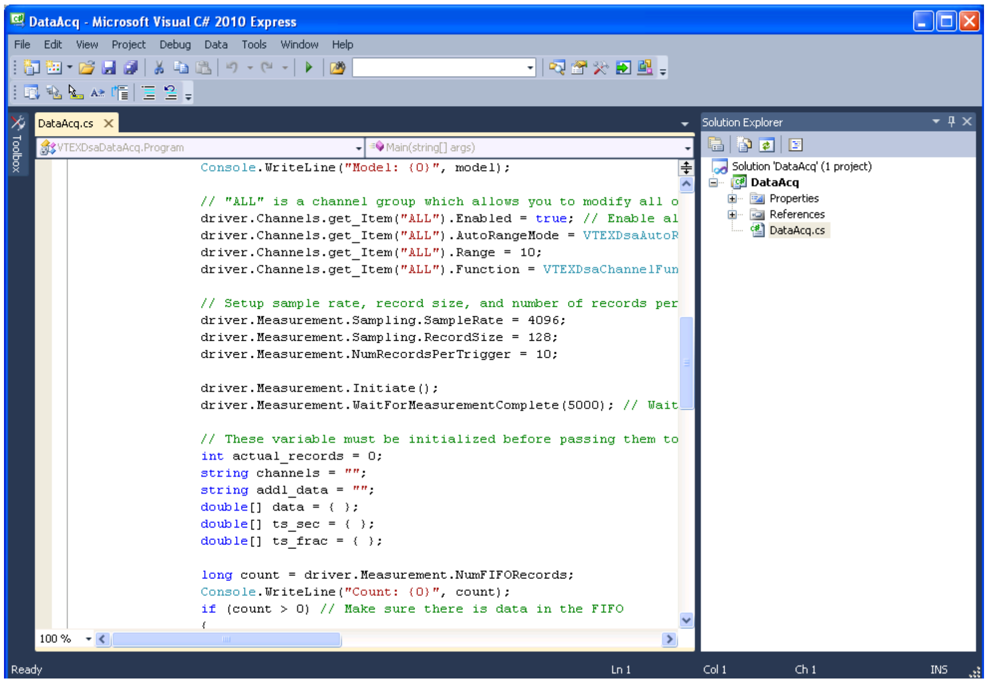
The instrument drivers come with example programs that the user can build and execute. Example programs are in C++ and C# (Windows only) programming language. They are installed in *Examples* sub-folder under standard IVI driver installation folder.



Address C:\Program Files\IVI Foundation\IVI\Drivers\VTEXDsa\Examples\C#\DataAcq

Name	Size	Type
_UpgradeReport_Files		File Folder
Backup		File Folder
DataAcq		File Folder
DataAcq.sln	2 KB	Microsoft Visual S
UpgradeLog.XML	2 KB	XML Document

Figure 2-3: Example Programs



```
Console.WriteLine("Model: {0}", model);

// "ALL" is a channel group which allows you to modify all o
driver.Channels.get_Item("ALL").Enabled = true; // Enable al
driver.Channels.get_Item("ALL").AutoRangeMode = VTEXDsaAutoR
driver.Channels.get_Item("ALL").Range = 10;
driver.Channels.get_Item("ALL").Function = VTEXDsaChannelFun

// Setup sample rate, record size, and number of records per
driver.Measurement.Sampling.SampleRate = 4096;
driver.Measurement.Sampling.RecordSize = 128;
driver.Measurement.NumRecordsPerTrigger = 10;

driver.Measurement.Initiate();
driver.Measurement.WaitForMeasurementComplete(5000); // Wait

// These variable must be initialized before passing them to
int actual_records = 0;
string channels = "";
string addl_data = "";
double[] data = { };
double[] ts_sec = { };
double[] ts_frac = { };

long count = driver.Measurement.NumFIFORecords;
Console.WriteLine("Count: {0}", count);
if (count > 0) // Make sure there is data in the FIFO
{
```

Figure 2-4: Building an Example

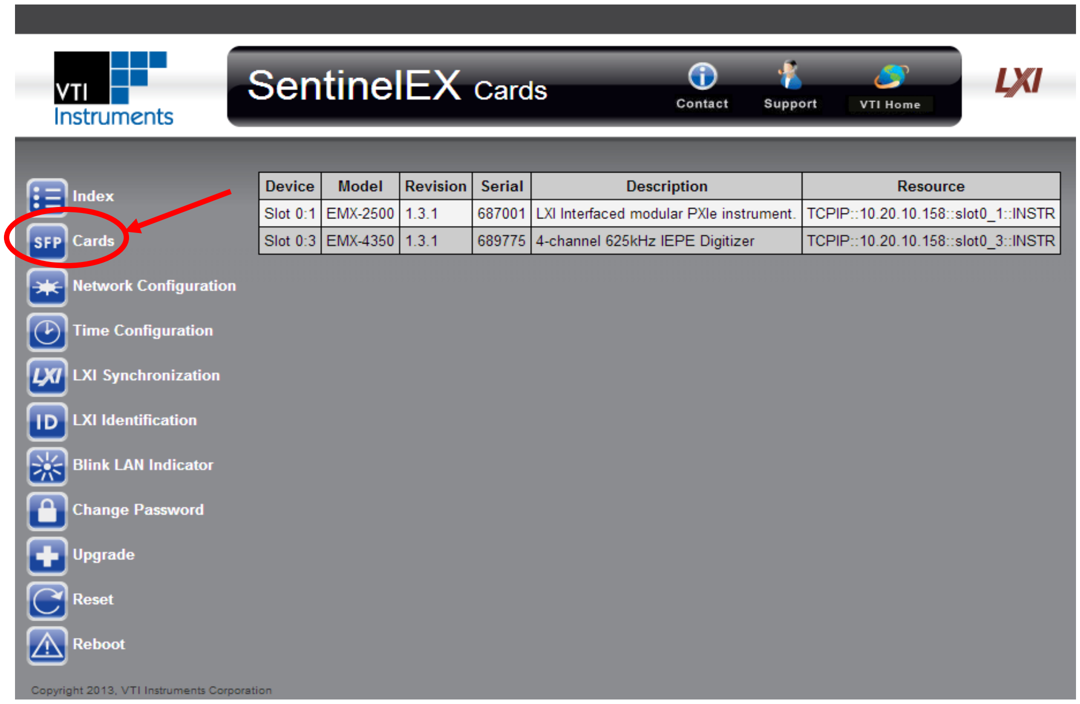
Linux Examples

Linux examples are stored at */opt/vti/share/doc/digitizer/examples/*. To build them, copy that folder to a writable location, change directory to the *examples* folder and run *make*.

```
~$ cd /tmp

/tmp$ cp -r /opt/vti/share/doc/digitizer/examples/ .

/tmp$ cd examples/

/tmp/examples$ make

g++ -o Initialization Initialization.cpp -I/opt/vti/include  -
L/opt/vti/lib -Wl,-rpath=/opt/vti/lib -lDigitizer

/tmp/examples$ ./Initialization 10.20.10.158
```

WEB INTERFACE

The SentinelEX family of products provide an embedded soft front panel which the user can access using a standard web browser. The user can open the EMX-2500 Ethernet controller's web page by specifying the EMX-2500's IP address in any web browser. If a CMX09 is used, the IP address can be retrieved from the display of the smart switch in the front panel of the chassis. The *Cards* page shows a list of instruments that are plugged in to the chassis along with their slot numbers, firmware revisions, and serial numbers.



Device	Model	Revision	Serial	Description	Resource
Slot 0:1	EMX-2500	1.3.1	687001	LXI Interfaced modular PXIe instrument.	TCPIP::10.20.10.158::slot0_1::INSTR
Slot 0:3	EMX-4350	1.3.1	689775	4-channel 625kHz IEPE Digitizer	TCPIP::10.20.10.158::slot0_3::INSTR

Figure 3-1: EMX-2500 Web Interface

CARD UPGRADE THROUGH WEB INTERFACE

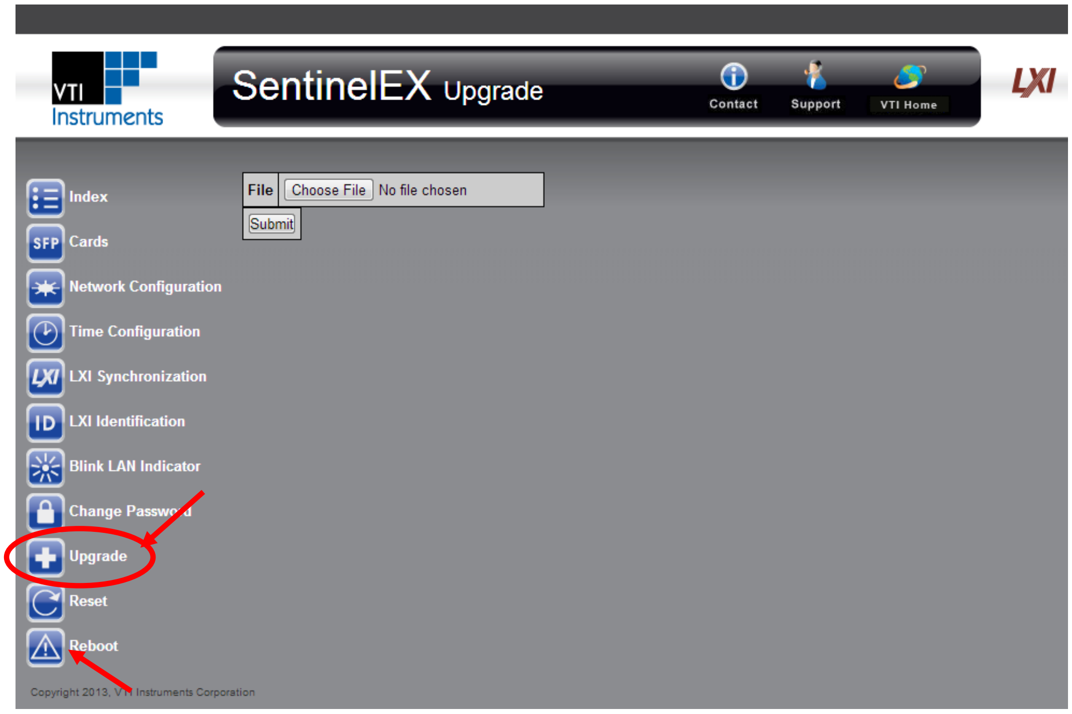
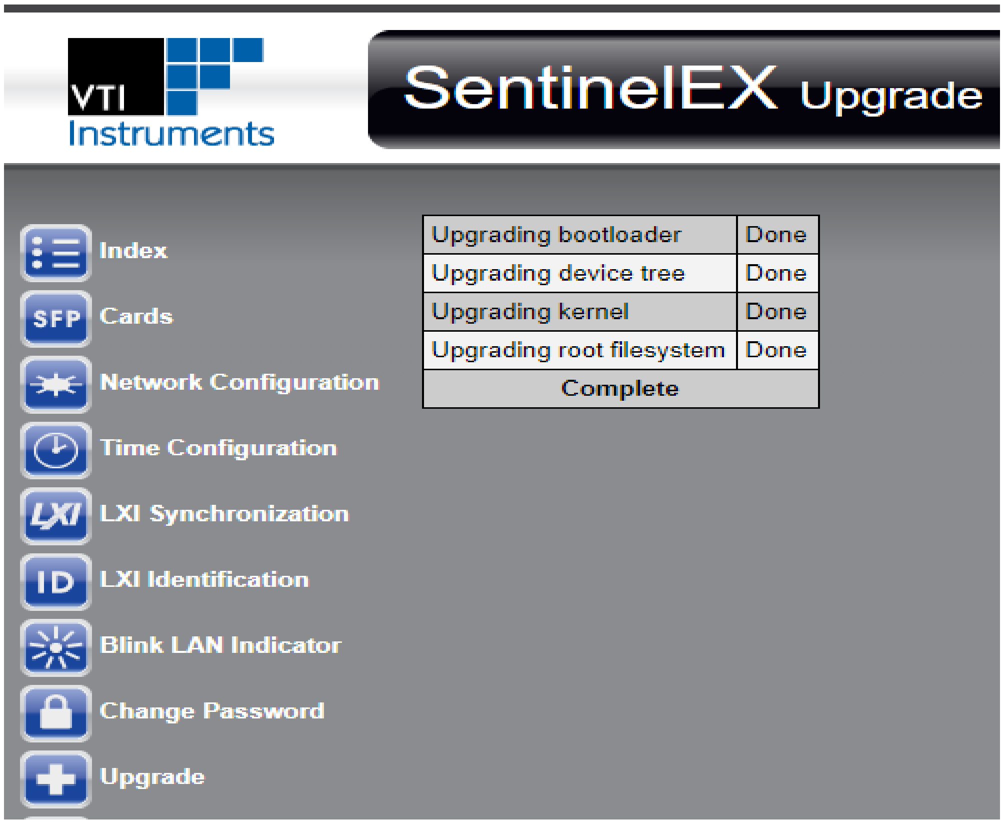


Figure 3-2: Firmware Update Using Web Interface

The firmware for the EMX-4250/4350/4380's can be updated from the *Upgrade* page as shown Figure 3-2. Simply select new firmware image file from the folder in which its kept and click submit button. All instruments in the chassis that use the same firmware will be automatically upgraded. See picture below.

Figure 3-3: Selecting Firmware Image to Update Using Web Interface



Upgrading bootloader	Done
Upgrading device tree	Done
Upgrading kernel	Done
Upgrading root filesystem	Done
Complete	

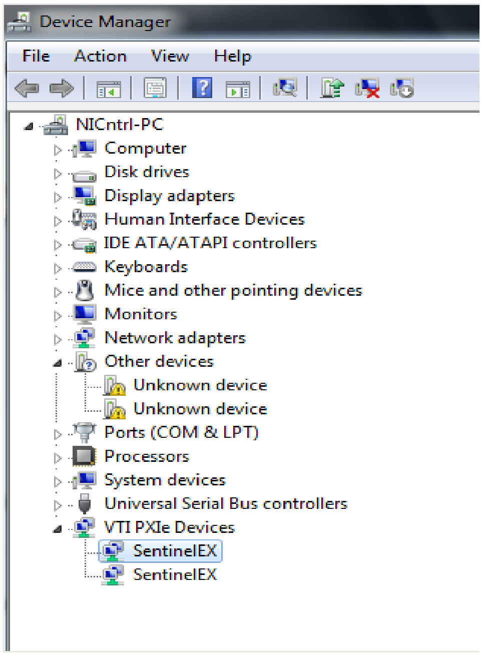
Figure 3-4: After Firmware Image is Updated Using Web Interface

After all cards are upgraded, the system must be rebooted for the new firmware to take effect. After system reboot, the details of the cards can be seen in the SFP web page. See below figure.

CONNECTING TO EMBEDDED SLOT-0 PXIE CONTROLLER

Before connecting to the Digitizer card from your Windows system, make sure you have downloaded and installed the "*PCIe System Software Package*" on your Windows system.
Refer to the ***Documents and Downloads*** section in the bottom of the page below to download the file.
http://www.vtiinstruments.com/Products-Services/EMX-Series/EMX-2401.aspx

Details of Sentinel series cards can also be observed under device manager. See picture below



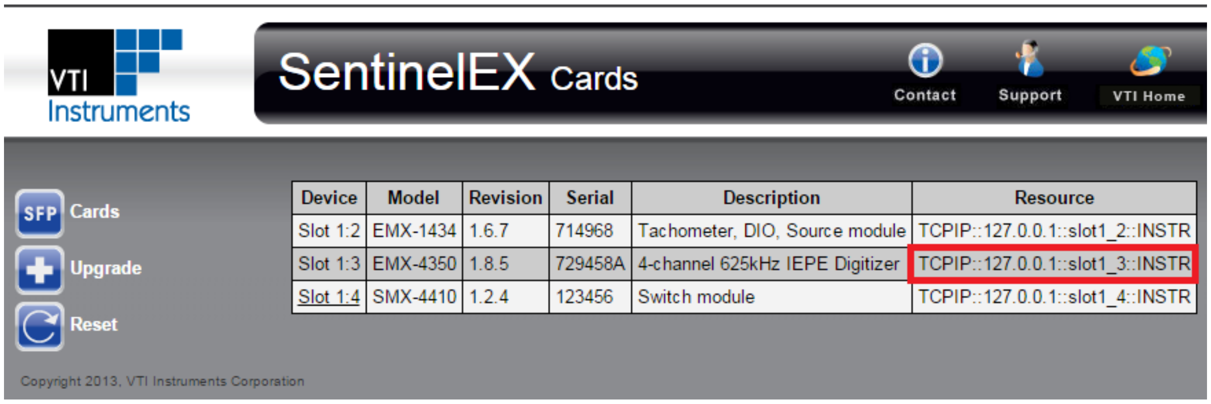
NICntrl-PC
- Computer
- Disk drives
- Display adapters
- Human Interface Devices
- IDE ATA/ATAPI controllers
- Keyboards
- Mice and other pointing devices
- Monitors
- Network adapters
- Other devices
  - Unknown device
  - Unknown device
- Ports (COM & LPT)
- Processors
- System devices
- Universal Serial Bus controllers
- VTI PXIe Devices
  - SentinelEX
  - SentinelEX

Figure 3-7: Sentiel Cards details in Device Manager

To connect to a digitizer card from the client source code, first declare its driver variable.

```
IVTEXDigitizerPtr digitizer(__uuidof(VTEXDigitizer));
```

Call the drivers Initialize function, which will open a connection to the card. This function expects 4 arguments. First is the connection string which is a combination of IP address and slot no in which the card is plugged in. The name of this Resource string can be taken from the Resource column in SFP page for that card. See the box in Red in the picture below. Here the Digitizer card is plugged in slot no 3 of system with IP 127.0.0.1 So our Resource string is "**TCPIP::127.0.0.1::slot1_3::INSTR**" (**Note : When an embedded slot-0 PXIe controller is used, the IP address by default becomes 127.0.0.1**). The 2nd and 3rd arguments are Query and Rest values which are Boolean values respectively. The last is an optional string in which we can set our own values if we don't want to use the default values for certain parameters like timeout, Query Instrument Status. By default always mention CACHE value as FALSE in this string.



Device	Model	Revision	Serial	Description	Resource
Slot 1:2	EMX-1434	1.6.7	714968	Tachometer, DIO, Source module	TCPIP::127.0.0.1::slot1_2::INSTR
Slot 1:3	EMX-4350	1.8.5	729458A	4-channel 625kHz IEPE Digitizer	TCPIP::127.0.0.1::slot1_3::INSTR
Slot 1:4	SMX-4410	1.2.4	123456	Switch module	TCPIP::127.0.0.1::slot1_4::INSTR

Figure 3-8: SFP page

```
_bstr_t option_string = "Cache=False, DriverSetup = preferpcieconnection = true";
_bstr_t resource = "TCPIP::127.0.0.1::slot1_3::INSTR";
digitizer->Initialize(resource,VARIANT_FALSE,VARIANT_TRUE,option_string);

//  The above call opens a connection to the card plugged in the chassis.
//  If Initialize call fails, a null value is returned to the driver.
//  Below is a sample code.

_bstr_t option_string = "Cache=False, DriverSetup = preferpcieconnection = true ";
_bstr_t resource = "TCPIP::127.0.0.1::slot1_3::INSTR";
IVTEXDigitizerPtr digitizer(__uuidof(VTEXDigitizer));
digitizer->Initialize(resource, VARIANT_FALSE, VARIANT_TRUE, option_string);

#ifdef ADLINK_TRIGGER_ROUTING
        using_2500 = false;
#endif

// Initialize the Digitizer driver
if(using_2500)
{
   digitizer->Initialize(resource, VARIANT_FALSE, VARIANT_TRUE, "");
   std::cout << "initialized" << std::endl;
}
else
{
   digitizer->Initialize(resource, VARIANT_FALSE, VARIANT_TRUE, option_string);
   std::cout << "initialized with direct PCIe" << std::endl;
   fprintf(fs, "initialized with direct PCIe\n");
}

if(using_2500)
{
   digitizer->ReferenceOscillator->Source = VTEXDigitizerReferenceOscillatorSourceSystem;
   digitizer->ReferenceOscillator->TimestampSource = \
                                   VTEXDigitizerReferenceOscillatorTimestampSourceSystem;
}

digitizer->Measurement->Sampling->ClockFrequency = clock_rate;
digitizer->Measurement->Sampling->SampleRate = clock_rate / (double)clock_divider;
digitizer->Measurement->Sampling->RecordSize = record_size;
double sample_rate = digitizer->Measurement->Sampling->SampleRate;

if(using_2500)
{
    digitizer->Sync->CoordinationLine = "LAN,PXI0";
}
else
{
    digitizer->Sync->CoordinationLine = "PXI0";
}

digitizer->Sync->Line = "PXI0";
digitizer->Measurement->NumRecordsPerTrigger = records_per_trigger;
digitizer->Trigger->TriggerCount = num_triggers;

if(digitizer->Initialized)
{
    digitizer->Measurement->Abort();
    digitizer->Close();
}

digitizer = NULL;
```

SECTION 4

USING THE EMX-4250/4350/4380

INTRODUCTION

This section describes how to use EMX-4250/4350/4380 using instrument driver. The instrument driver provides an API that allows the user to configure and control the EMX-4250/4350/4380 cards using high level commands.

Two types of drivers are available. One is an IVI driver based on the industry standard IVI driver architecture specification. The IVI driver exposes both COM and C language interface APIs for the Windows OSs. The other driver has C++ APIs for the Linux OS. Both Windows and Linux drivers have a consistent API design so that the application software developed for one can be easily migrated to the other. Our drivers are compatible with both the 32-bit and 64-bit operating system.

In general, the API descriptions in this document apply to both the Windows and Linux drivers unless otherwise specified.

INSTRUMENT DRIVERS

OVERVIEW

Three drivers, VTEXDigitizer, VTEXDSA, and VTEXPlatform (or libDigitizer, libDSA, and libPlatform, respectively, for Linux), are used to program the EMX-4250/4350/4380 instruments. The “Digitizer” drivers are common drivers used by all EMX digitizer cards provided by VTI Instruments. For a simple data acquisition application, the “Digitizer” driver may be the only driver the user needs to use. The “DSA” drivers are essentially a super set of digitizer driver. In addition to the data acquisition functionalities in “Digitizer” driver, the “DSA” driver also supports the EMX-1434’s signal output and tachometer input capabilities. The one feature available in “Digitizer” driver, but not in the “DSA” driver is support for IVI class compliant interfaces. In general, it is recommended to use the “DSA” driver instead of “Digitizer” driver unless the user needs to use the IviDigitizer or IviScope class compliant interfaces for instrument interchangeability. The “Platform” drivers are used to configure the EMX-2500 Ethernet controller card and the CMX09/18 PXIe chassis. Using the EMX-2500 in an CMX chassis makes EMX-4250/4350/4380-based data acquisition systems LXI devices so that they can synchronized using IEEE 1588 PTP and LAN events over Ethernet (see Figure 4-6).

IVI DRIVERS

The IVI Foundation defines IVI driver specifications. IVI specification information is available at the IVI Foundation website, www.ivifoundation.org. The IVI-3.2 Inherent Capability Specification defines a common set of basic functionality that all IVI driver must support. This ensures that users can perform basic operations and identify its capability for any IVI driver using the exact same API functions. The IVI drivers are implemented using a common code provided by the IVI Foundation in order to guarantee this consistent behavior. This common code is called IVI Shared Components. The IVI Shared Components must be installed separately prior to any IVI drivers from VTI. The shared components installer is available for download from the IVI Foundation website.

The IVI Foundation specifies that the IVI driver be based on Microsoft Component Object Model, called IVI-COM and an IVI driver using standard C language API, called IVI-C. For those who develop applications in Windows .NET languages, such as C#, VB.NET, or other Object Oriented Language, such as C++, IVI-COM gives APIs logically organized by interfaces. IVI-C gives more traditional C language functions.

While VTI’s IVI drivers are architected based on IVI-COM, the driver installer also installs a wrapper library that exposes IVI-C functions so that the user can use develop in both types of environment.

Header and Library Files

The IVI driver specification defines the install directory structure and software components to be installed. For 32-bit Windows systems, the root of install directory is C:\Program Files\IVI Foundation\IVI. For 64-bit Windows systems, the 32-bit driver is installed at C:\Program Files (x86)\IVI Foundation\IVI and the 64-bit driver is installed at C:\Program Files\IVI Foundation\IVI. Driver header files and library files are installed in several sub directories. The **Bin** subdirectory contains IVI-COM and IVI-C driver DLL files. The **Component** subdirectory contains IVI-COM and IVI-C shared components files. The **Drivers** subdirectory contains the driver specific sub directories. VTEXDigitizer, and VTEXDSA driver’s online help files and example programs are installed here. The **Include** subdirectory contains header files. The **Lib** subdirectory contains 32-bit import library files. The **Lib_x64** subdirectory contains 64-bit import library files.

DRIVER FOR LINUX OS

In addition to the IVI drivers for Windows OS, C++ libraries are provided for the Linux OS compatible with LSB (Linux Standard Base) 4.0 or later. The Linux drivers are supported on distributions running Linux kernel version 2.6.32 or later. In addition, the Linux drivers require GCC version 4.3 or later and glibc 2.11 or later. The driver for Linux organizes and names each C++ class and members consistent to the IVI-COM driver. The driver description in this manual applies to both IVI driver and the library for Linux OS.

The Linux drivers are supplied as RPM packages which are supported by a wide variety of distributions. In distributions which natively support RPM packages, such as Red Hat Enterprise Linux, Fedora Linux, or CentOS, the driver packages can be installed by running the command:

```
rpm –Uvh packagename.rpm
```

There are many other popular Linux distributions which do not natively support RPM packages, but instead use 3rd party tools to install them. Debian and Ubuntu Linux are both very popular, but do not support RPM packages out of the box. To use these packages on these systems, ‘alien’ is recommended which should be available in the package repository for these distributions. To install the drivers using alien, run the command:

```
alien –i packagename.rpm
```

Currently, there are both 32- and 64-bit driver packages for libDigitizer, libDsa, and libPlatform. There is also a package which installs common libraries and dependencies used by all drivers, libCommon. The package for libCommon is universal for both 32- and 64-bit systems, and should be installed before installing any of the other driver packages.

Header and Library Files

/opt/vti/include sub directory contains header files.

/opt/vti/lib sub directory contains driver shared object files.

/opt/vti/share/doc sub directory contains release notes, driver online help and example programs.

COMPATIBILITY

DRIVERS, INSTRUMENTS, AND OS

Driver	Instruments	Operating System
Digitizer	EMX-4250, EMX-4350, EMX-4380	Windows XP, Vista, 7 (32-bit and 64-bit), 8 (32-bit and 64-bit) Linux (32-bit and 64-bit)
DSA	EMX-4250, EMX-4350, EMX-4380, EMX-1434	Windows XP, Vista, 7 (32-bit and 64-bit), 8 (32-bit and 64-bit) Linux (32-bit and 64-bit)
Platform	EMX-2500, CMX09, CMX18	Windows XP, Vista, 7 (32-bit and 64-bit), 8(32-bit and 64-bit) Linux (32-bit and 64-bit)
Switch	SMX Series switch cards	Windows XP, Visa, 7 (32-bit and 64-bit), 8(32-bit and 64-bit) Linux (32-bit and 64-bit)

DRIVER AND FIRMWARE REVISIONS

The instrument driver and firmware have three revision fields: <Major>, <Minor>, and <Build>. For the firmware installed on the instrument to be compatible with the driver being used, the <Major> version number must be equal and the <Minor> version must be equal or newer than the driver. Otherwise, the firmware needs to be updated. It is recommended to use the same <Major> and <Minor> version pair, if possible.

DRIVER API AND INSTRUMENTS

Digitizer/DSA drivers are designed to work with many digitizer cards from VTI. Not all API functions defined in the driver applies to every card since each models is unique in the feature set. Calling unsupported API functions will result in an *unsupported* error.

Digitizer/DSA APIs	EMX-4250	EMX-4350	EMX-4380
Alarm	Not supported	Not supported	Not supported
Arm	Supported RPM(DSA), Pattern (DSA) not supported	Supported RPM(DSA), Pattern (DSA) not supported	Supported RPM(DSA), Pattern (DSA) not supported
Calibration	Supported	Supported	Supported
Channels	Supported AutoCal, AutoRange, Filter, Offset, Measurement, Strain, Temperature, Transducer, Weghting are not supported Function: Voltage, IEPE, Cal Coupling: AC, DC, Ground Mode: Differential, PseudoDifferential	Supported AutoCal, AutoRange, Filter, Offset, Measurement, Strain, Temperature, Transducer, Weghting are not supported Function: Voltage, IEPE, Cal Coupling: AC, DC, Ground Mode: Differential	Supported AutoCal, AutoRange, Filter, Offset, Measurement, Strain, Temperature, Transducer, Weghting are not supported Function: Voltage, IEPE, Charge, Cal Coupling: AC, Ground Mode:Differential, SignleEnded
Configuration	Supported	Supported	Supported

Digitizer/DSA APIs	**EMX-4250**	**EMX-4350**	**EMX-4380**
Events	Supported	Supported	Supported
Measurement	Supported Multipass, Oversample not supported	Supported Multipass, Oversample not supported	Supported Multipass, Oversample not supported
Platform	Supported	Supported	Supported
ReferenceOscillator	Supported OutputEnabled is not supported	Supported OutputEnabled is not supported	Supported OutputEnabled is not supported
Registers (Digitizer)	Internal use	Internal use	Internal use
Start	Supported	Supported	Supported
StreamingData	Supported FileCount, FileMaxSize, Filename, FileRotate not supported	Supported FileCount, FileMaxSize, Filename, FileRotate not supported	Supported FileCount, FileMaxSize, Filename, FileRotate not supported
Sync	Supported	Supported	Supported
Temperature	Supported	Supported	Supported
Time	Supported	Supported	Supported
Trigger	Supported MaxQueueSize, QueueEnabled not supported	Supported MaxQueueSize, QueueEnabled not supported	Supported MaxQueueSize, QueueEnabled not supported
Dac (DSA only)	Not supported	Not supported	Not supported
Dio (DSA only)	Not supported	Not supported	Not supported
Tach (DSA only)	Not supported	Not supported	Not supported

DRIVER STRUCTURE

MEASUREMENT

This section provides information related to configuring the basic measurement setup and control. The basic measurement configuration and control can be done through the driver's Measurement interface. The Measurement interface configures parameters that are global to entire system, rather than individual channels, or instrument modules when more than one module is included in the driver.

The parameters the user can set using the Measurement interface are:

- Sampling parameters, including ADC sampling rate, digital decimation filters span, and data record size
- Number of data acquisition records at each trigger event
- FIFO mode of operation
- Data format

The Measurement interface can be used to query the current measurement state information.

- Measurement state machine state
- Total number of records available in FIFO

Methods to control measurement, such as:

- Initiating measurement
- Aborting measurement
- Retrieving acquired data

CHANNELS AND CHANNEL GROUPS

This section provides information related to using channels and channel groups. For more detailed information, see the online help file provided with the Digitizer/DSA drivers.

The *Channel*s interface contains both channel objects and channel group objects in the same array. A channel object represents individual analog input channel. A channel group object represents one or more analog input channels as a group. When a driver is initialized, or reset, an array of all analog channels and one channel group object that represents all analog channels and one or more channel groups that represent all analog channels from each digitizer model. For example, when there are two EMX-4250s and one EMX-4350 in a single driver session, the *Channel*s interface contains an array totaling 39 channel objects. They are 32 EMX-4250 individual analog inputs, four EMX-4350 individual analog inputs, one channel group object that represents all 36 analog input channels, one channel group object for all 32 EMX-4250 inputs, and one channel group for all EMX-4350 inputs. These channel groups are named as "All", "EMX-4250", and "EMX-4350".

The channel array is created in the ascending slot order of the chassis. The first element in the array is the first analog input of the digitizer card that is installed at the lowest slot in a chassis. When there are more than one chassis are included in the total system, the order is determined by the resource string used in the driver's *Initialize* call.

The individual channel objects are used to configure or query individual input channel's configuration.

The channel group objects can be used to configure multiple channels to the same value. In general, the user can configure multiple channels faster using a channel group than setting channels individually. Querying the current setting through channel groups works only when all channels are set to the same value.

The channel name is defined as <slot no>!CH<channel no>. For example, 4!CH2 indicates the 2nd channel of a card installed at slot4 of a chassis. The predefined channel group names are "ALL" or an instrument model number such as "EMX-4350". Optionally, the user can create new custom channel groups using *AddChannelGroup* method. When more than one chassis are in the session, the channel name of the 2nd chassis adds 100 to the slot number, such as 104!CH2, and 200 for the 3rd chassis, 300 for the 4th, etc.

The *NumChannels* property gives total number of individual input channels, while the *Count* property is the total number of channel or channel group objects in an array.

START, ARM, TRIGGER, AND ALARM

EMX-4250/4350/4380 implements sophisticated Arm/Trigger model as shown in Figure 4-1. This trigger model conforms industry standard trigger models defined in the IVI Digitizer specification or LXI Sync specification with some additional features.

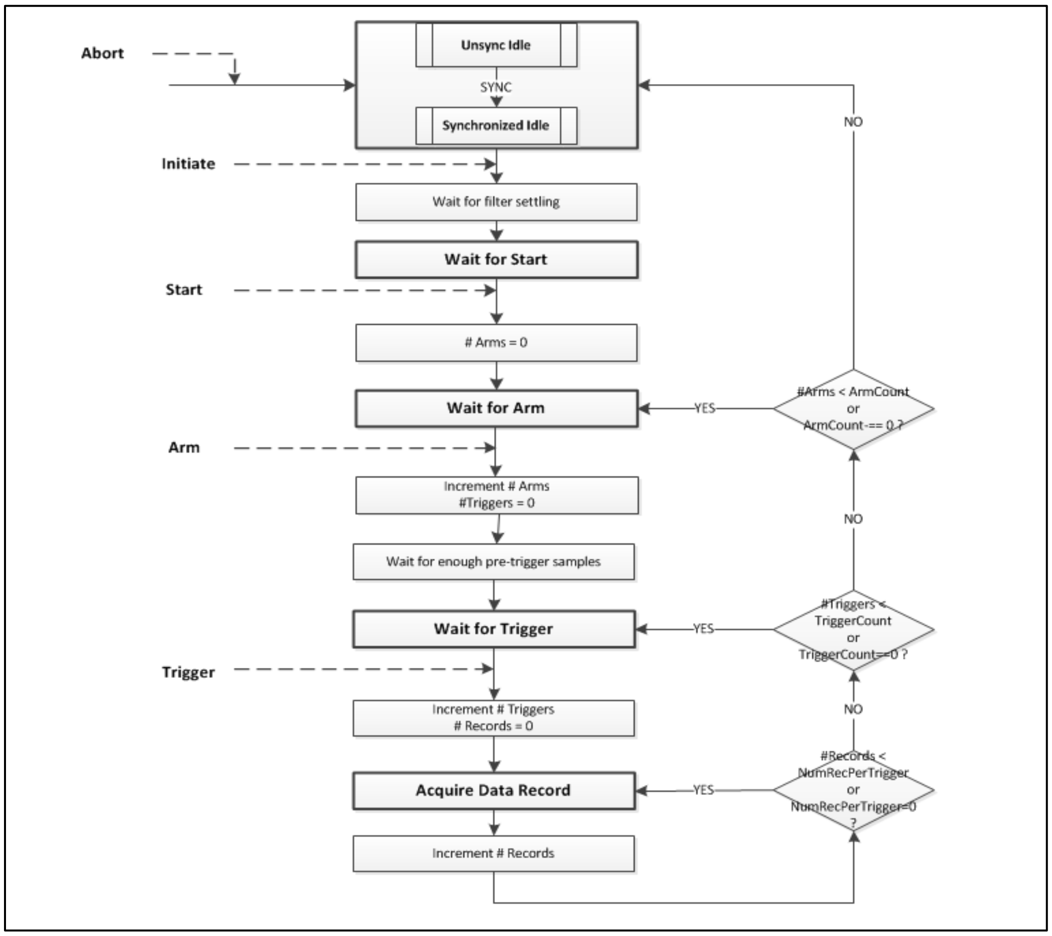


Figure 4-1: Trigger Model

Sync

Sync is an event that synchronizes the entire system. The *Sync* interface is used to configure the synchronization between cards. All cards are simultaneously started by the SYNC signal and then they synchronize the state machine transition with each other through a coordination signal. The *Line* and *CoordinationLine* properties define which PXI trigger lines to send these signals.

Start

Once the measurement is initiated by the Initiate command, instrument completes all preparation and becomes ready to start taking data immediately and then the state machine moves to the *Wait for Start* state. The *Start* interface provides methods and properties to advance to the next state. The amount of time it takes to reach to *Wait for Start* after measurement *Initiate* command varies depending on the measurement configuration. For example, It takes longer when the measurement sample rate (or measurement span) is low because the filter settling time is longer. The *Start* interface is useful when the user wants to have instrument complete all the preparation and hold in that state, so that it can start taking data immediately without wasting time.

Arm

The instruments must be armed before triggering data acquisition. The *Arm* interface is used to configure this arming condition. The *Sources* property in the *Arm* interface defines the arming event sources. The default arming source is *Immediate*, which means automatic arming. The *SourceOperator* property allows the user to define an arming condition by logically combining multiple arming sources. The *Delay* property defines amount of time the instrument waits before moving out from *Wait for Arm* state after the defined arming conditions are met. The *ArmCount* property defines how many times the measurement repeat arming and triggering before it completes. The default is once. Setting *ArmCount* to 0 forces the measurement to repeat arm and trigger indefinitely until it is aborted by *Abort* command.

Trigger

Data acquisition begins when a triggering condition is met. The triggering condition is configured using the *Trigger* interface. Similar to Arm interface, the *Sources* property in the *Trigger* interface define the triggering event sources. Trigger sources can be logically combined using *SourceOperator* in the *Trigger* interface. The *Delay* property defines the amount of time between the trigger event and the beginning of the data acquisition. The *Delay* value can be negative indicating pre-trigger data acquisition. In this case, the acquired data block starts earlier than the trigger event. This is achieved by buffering the data in the instruments FIFO priory to the trigger event. For more information, see FIFO section.

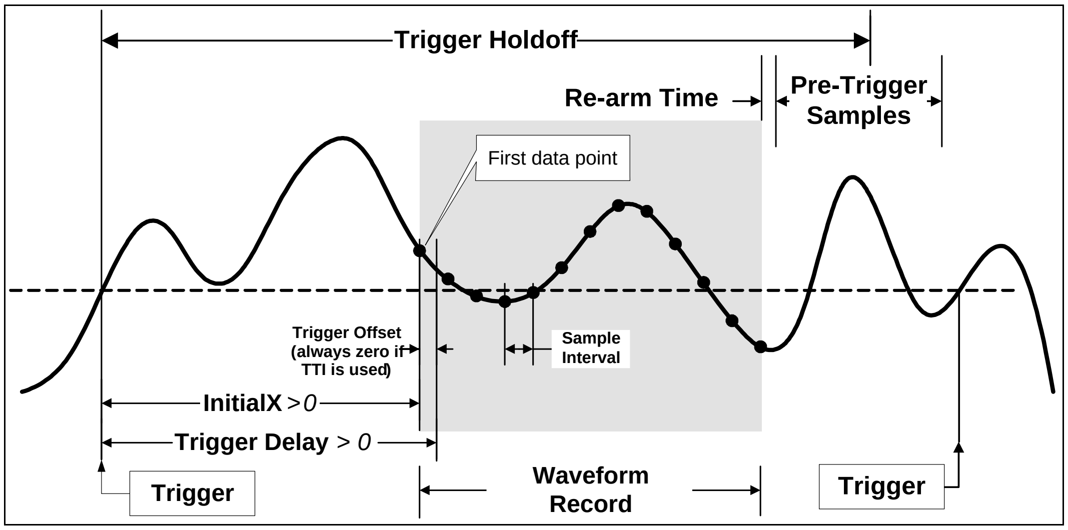


Figure 4-2: Positive Trigger Delay

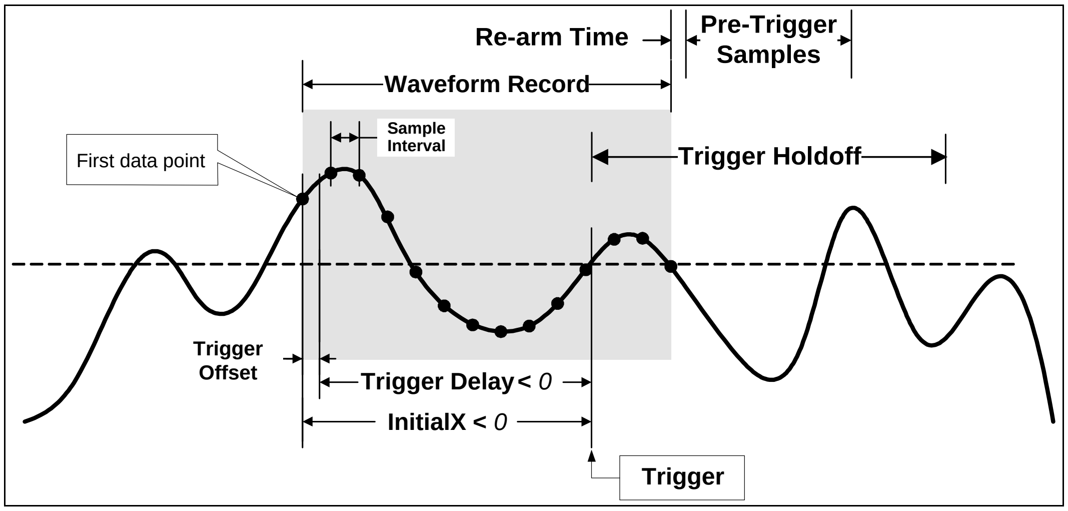


Figure 4-3: Negative Trigger Delay

The *TriggerCount* property defines how many times the trigger events are accepted and data blocks are acquired. After "*TriggerCount*" triggers are processed, the measurement waits for the next arming condition or finishes. The *HoldOff* time specifies the minimum amount of time the measurement has to wait before it can be triggered again once a trigger is detected. Any trigger events that occurred during the *HoldOff* time are ignored. When the specified *Holdoff* time is shorter than one data record time length, the two successive data records may be overlapped.

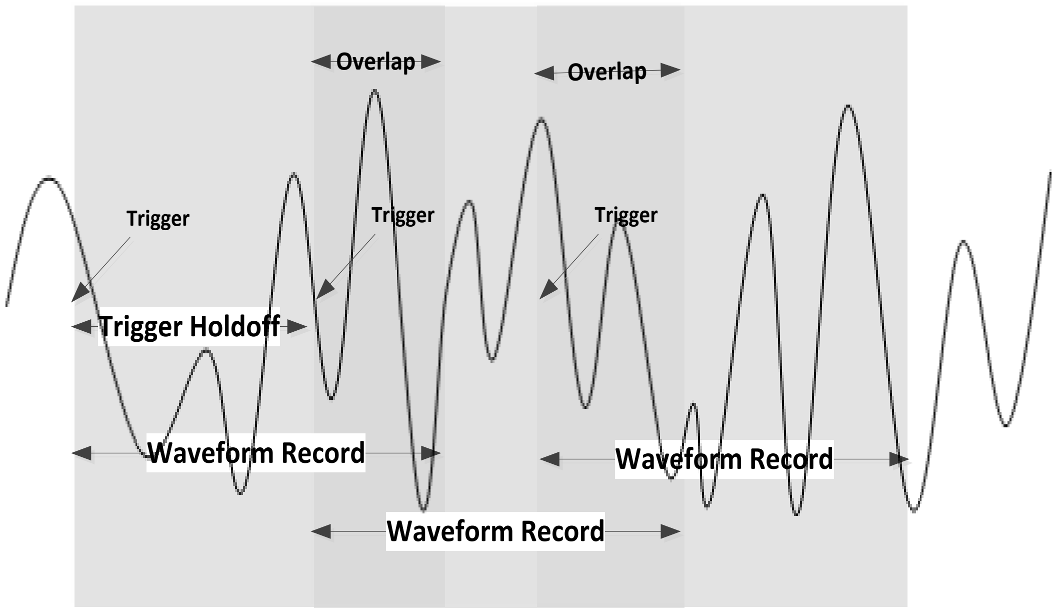


Figure 4-4: Overlapped Data Acquisition

Alarm

The Alarm is a mechanism that generates events at fixed time interval. The Alarm starts at the time specified by *TimeSeconds* and *TimeFraction* and repeats for *RepeatCount* times. The *Period* property defines the interval between the two successive alarm events. The Alarm can be used as an arm or a trigger event source. The Alarm is currently supported by EMX-2500 and Platform driver.

RETRIEVING DATA

The Digitizer/DSA driver provides two ways to retrieve data from EMX-4250/4350/4380. The acquired data is stored in the instrument’s FIFO buffer. The data in the FIFO can be read using standard *Read* method. The 2nd method is data streaming. The streaming mechanism makes it possible to transfer data faster than standard FIFO *Read* method with some tradeoffs.

FIFO Read

Once the measurement is triggered and data becomes available in the instrument’s FIFO buffer, data can be retrieved using the *Read* method in the *Measurement* interface. The *Read* method returns the specified number of data records from all enabled input channels in a channel order in the *Channels* array. The *NumFIFORecords* property in *Measurement* interface returns the number of data records currently available in the instrument’s FIFO buffer. This value decreases when the data is retrieved by the host, and increases when new trigger events are processed. FIFO buffer overflow may happen when the trigger events arrive faster than the host can retrieve data. See the *FIFO* description in the *Data Acquisition* section for more information.

Streaming

Streaming data is an alternative method for retrieving data from the EMX-4250/4350/4380. Unlike the FIFO read function, instruments send new data records to the host PC as soon as it becomes available when streaming data. The data is kept in the host memory buffer managed by Digitizer/DSA driver. The data in this memory buffer is then retrieved to the user’s application through the *MemoryRead* method. The streamed data can be directly written into disk files. In order to use this streaming mechanism, it must be enabled by *EnableStreaming* method in *StreamingData* interface.

The key advantage of the streaming method over the FIFO Read method data transfer speed. It provides the best data transfer performance. However, this method may be less convenient when data transfer speed is not important since the retrieved data records using streaming are not sorted to the channel order when more than one instruments are involved in the data acquisition. The streamed data is returned to the user or written to the host disk file in the order of arrival to the host. One way to avoid this channel order problem is to open a driver session for each card separately.

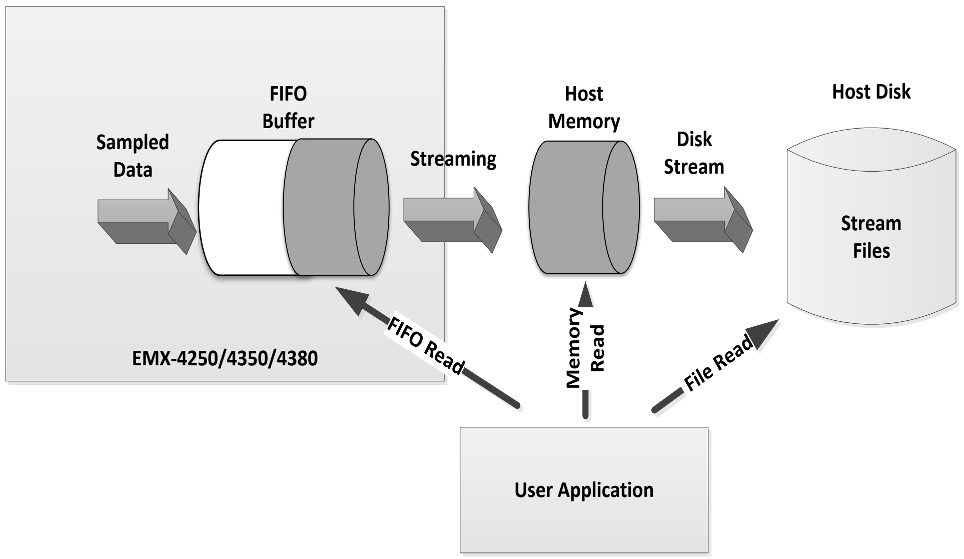


Figure 4-5: FIFO Read and Streaming

REFERENCE CLOCK AND TIME STAMP

The *Source* property and *TimestampSource* property in the *ReferenceOscillator* interface configure the reference oscillator to generate an ADC sampling clock and data time stamp clock. The PXI_CLK10 (10 MHz) or PXIe_CLK100 (100 MHz) reference signal is used by default. When the system is synchronized to the IEEE 1588 PTP grand master via Ethernet, the IEEE 1588 clock can be used as a reference oscillator.

SELF-CALIBRATION

The instrument's internal gain and offset can drift with the temperature change. It is recommended to perform frequent self-calibration prior to the tests. The self-calibration re-calibrates complete analog and digital signal paths using internal calibration signal source. The self-calibration, calculates new gain and offset values of all input channels for every ranges. Because of this, the self-calibration can be also used as a way to self-test the instrument. The self-calibration can be performed by *Initiate* method in *Calibration. Self* interface. See Appendix for more information on calibration.

LXI AND LAN EVENTS

When the EMX-4250/4350/4380 is used with EMX-2500 Ethernet controller (and optionally with Platform driver), it can act as an LXI device. These instruments can be synchronized to an IEEE 1588 PTP grand master clock. They can be armed or triggered by LAN events or they can generate LXI LAN events to synchronize with other LXI devices through LXI Sync interface defined by IVI standard.

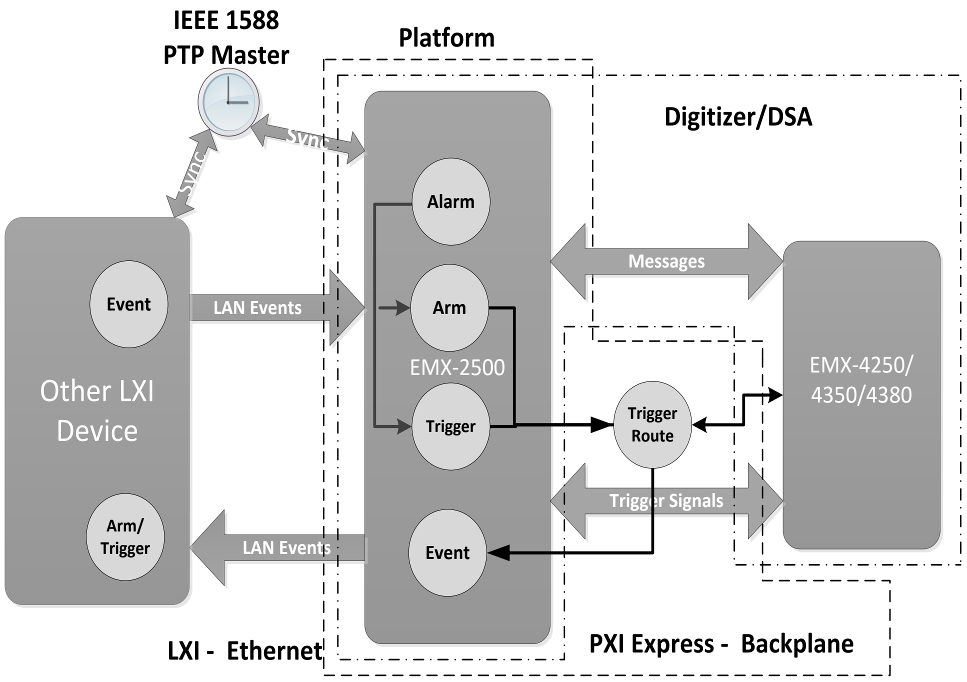


Figure 4-6: LXI and LAN Events

IVI CLASS COMPLIANT INTERFACES

In addition to the IVI-LXISync interface, the VTEXDigitizer driver exposes the IVI-Digitizer and IVI-Scope class compliant interfaces. These interfaces are exposed for the instruments interchangeability. However, these interfaces are currently not supported.

MULTIPLE CARDS, SEGMENTS, AND CHASSIS

INITIALIZING DRIVER WITH MULTIPLE CARDS

Unlike many typical instruments driver, both the Digitizer and DSA drivers treat multiple EMX-4250/4350/4380 cards as a single, collective instrument. The user can initialize a single driver for two EMX-4250 cards and treat them as a single, 32 channel instrument instead of creating two driver sessions with 16 input channels each.

A driver must be initialized first before starting to communicate to with the instruments. The resource name is passed as an argument to the *Initialize* method that specifies which instruments will communicate. The resource name has the following syntax:

<address 1>[::<slot 1>,<slot 2>,..,slot N>] | <address 2> [::<slot 1>,<slot 2>,…,<slot M>] | …

Where:

<address x> is the IP address or host name of EMX-2500

<slot x> is the slot number identifier of the instrument in a chassis. The slot number identifier is a string as “slot0_5” indicates 5th slot of the first chassis controlled by an EMX-2500. The slot1_6 indicates the 6th slot of the 2nd chassis extended by a bus extender, the extended chassis is not supported yet.

The slot numbers are optional. When no slot numbers are specified, all supported instruments within the chassis will be used.

Where there are more than one EMX-2500 and chassis are involved, they are concatenated with “|” character.

MULTIPLE CARD MEASUREMENTS / COORDINATION LINE

A PXI system can be built with more than one bus segment by using standard PCI-PCI bridge technology. VTI’s CMX18, a eighteen slot chassis, has three independent trigger bus segments, while the CMX09, a nine slot chassis, has only one PXI trigger bus segment. When all cards included in a measurement are in single chassis and in a single PXI trigger bus segment, they will be automatically synchronized using one of the PXI trigger lines on the chassis backplane. The PXI0 trigger line is used by default. Any other PXI trigger line can be specified by *CoordinationLine* property in the *Sync* interface if the PXI0 trigger line is used by other instruments in a same trigger bus segment. In multiple cards configuration, the card installed at the lowest slot number becomes a trigger master and the other cards synchronize to it using a trigger coordination engine implemented in each instrument. However, in this configuration, any card in the system can detect an arm or trigger event. For instance, all EMX-4250/4350/4380 cards can be triggered by analog signal at any one of input channels in the system.

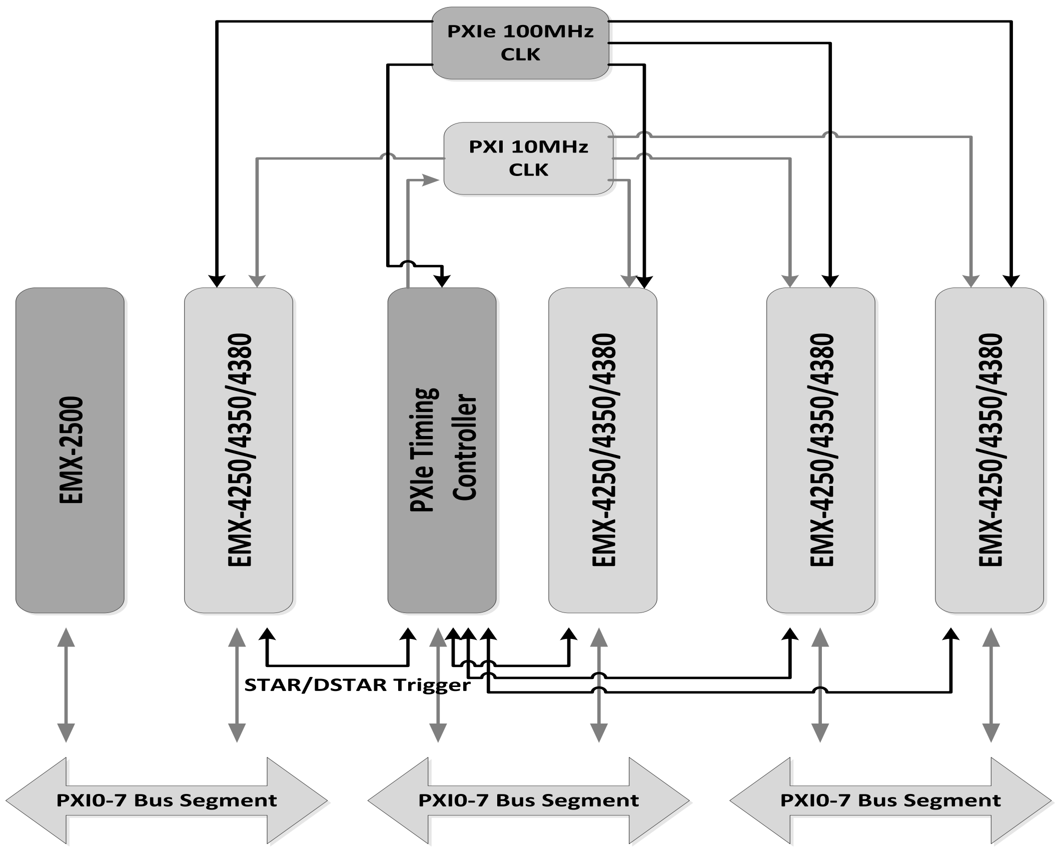


Figure 4-7: PXIe Chassis and Trigger Bus

MULTIPLE SEGMENT MEASUREMENT

In order to synchronize all cards installed on multiple trigger segments, trigger line between bus segments must be bridged together using the *Routes* interface in Platform driver. This is because PXI lines in each trigger segments are independent trigger buses. See the *EMX-2500 User's Manual* or Platform driver's online help for more detail.

When two trigger lines in different segments are bridged using the *Routes* interface, the signal only travels in one direction. Unlike the single segment installation, this restricts the arm or trigger source to the cards in the same segment as a trigger master card. The trigger master card is usually the card installed at the lowest slot in a chassis. The arm or trigger events cannot be initiated from any cards in the other segments.

MULTIPLE CHASSIS SYNCHRONIZATION WITH TRIGGER LINE

Multiple chassis synchronization is essentially the same as multiple segments synchronization within a single chassis. A coordination trigger line must be bridged between two chassis instead of two trigger bus segments. This can be achieved by physically connecting the PXI trigger lines using an external cable through the EMX-2500 trigger connector. The PXI trigger signal can be routed out from the master segment to the EMX-2500's TRIG connector. The signal is then distributed to all other chassis using external cables and routed into slave segments from the EMX-2500 trigger connector. The signal routing between PXI trigger line and EMX-2500 TRIG connector is configured using the Platform driver. In this configuration, only cards in the master segment in a chassis can detect an arm or trigger event. All other cards in slave chassis synchronize to it.

For the best sample-to-sample synchronization between chassis, coordinating trigger events may not be good enough. The sampling clock needs to be synchronized together as well. When VTI's

CMX09 or CMX18 chassis are used, the PXI 10 MHz reference clocks can be synchronized using BNC connectors on the back of each chassis.

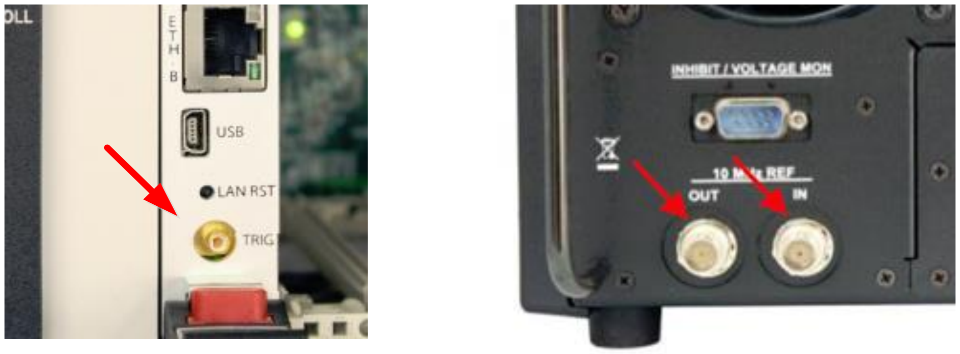


Figure 4-8: EMX-2500 Trigger and CMX09 10 MHz Reference

MULTIPLE CHASSIS SYNCHRONIZATION WITH LAN EVENTS

When chassis are geographically separated by a long distance, it may not be practical to connect a physical trigger and a reference clock line between them. The EMX-2500 makes EMX-4250/4350/4380s in each chassis an LXI device so they can be synchronized together using IEEE 1588 clock and LAN events.

When multiple chassis containing EMX-4250/4350/4380s and EMX-2500 controllers are initialized in a single Digitizer/DSA driver session, they can be automatically synchronized using LAN events specified by the LXI specification. This is equivalent to synchronizing chassis using a trigger signal, except a LAN message is used over Ethernet. In this configuration, sampling and timestamp clocks are synchronized to the IEEE 1588 PTP master chosen by the best master clock (BMC) algorithm. The PTP master can be an IEEE 1588-capable GPS unit or an EMX-2500s.

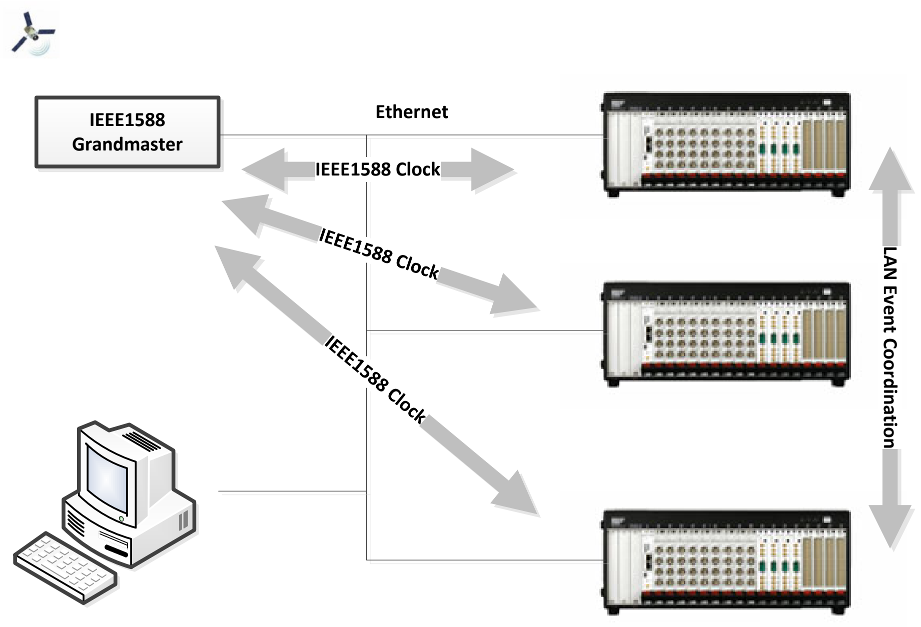
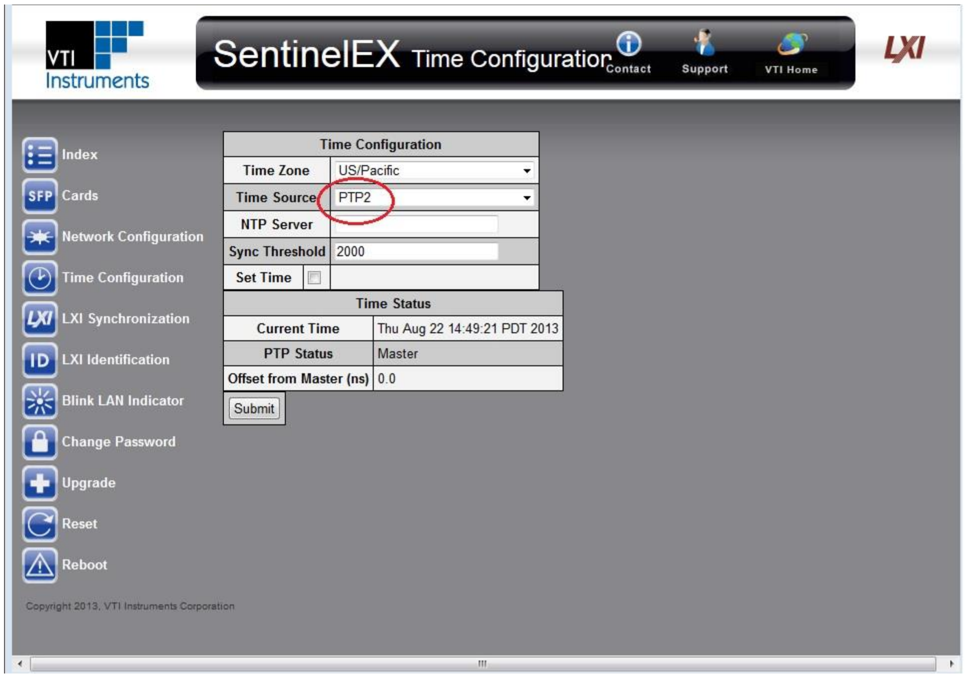


Figure 4-9: LAN Synchronization

The key advantage of LAN synchronization is that the user can achieve almost equivalent synchronization performance between chassis without running additional trigger or clock cables. Just like trigger line synchronization, the entire system can be triggered by an external trigger signal connected to one instrument, by sending a trigger command, by sending a LAN event trigger, or by an analog signal at one of the ADC channels crossing trigger threshold level.

When chassis are separated by a long distance or when LAN communication is slow, a certain restrictions can apply. For example, when one of analog channel's detects a trigger, it sends a LAN message to all chassis so that they trigger at the same time. A problem can occur if the another channel in the a different chassis also detect a trigger condition before it receives a LAN message trigger from the first chassis that detected a previous trigger condition. In order to avoid this confusion, the user may need to set restrictions so that only certain channels can detect trigger events.

In order to configure LAN synchronization, the EMX-2500's in each chassis must be configured to use PTP2 as its Time Source, as shown in Figure 4-10. This allows the chassis to synchronize with an IEEE 1588 Grandmaster clock on the system. Then, after opening a Digitizer/Dsa driver session, the *ReferenceOscillator. Source* and *TimestampSource* properties must be set to "System" (IEEE 1588). Once these steps have been completed, the EMX-4250/4350/4380's *Sync.CoordinationLine* property must be set to "LAN,PXI0". Once this has been performed, the EMX-4250/4350/4380's state transitions will be automatically coordinated by LAN events between chassis and PXI0 trigger line within chassis.



Time Configuration	
Time Zone	US/Pacific
Time Source	PTP2
NTP Server	
Sync Threshold	2000
Set Time	☐

Time Status	
Current Time	Thu Aug 22 14:49:21 PDT 2013
PTP Status	Master
Offset from Master (ns)	0.0

Figure 4-10: PTP Time Source

DATA ACQUISITION

DATA FLOW

The EMX-4250/4350/4380 instruments have a dedicated A/D converter at each analog input channel. The A/D converter samples analog data at fixed frequency specified by a multiple of the *ClockFrequency* (F_C). The output data rate of the ADC is F_C. The digitized data by the A/D converter is decimated and filtered by multiple stages of digital filters until the signal is band limited to the desired frequency span and the sample rate (F_S). These band-limited samples are constantly sent to the FIFO memory buffer located in DRAM.

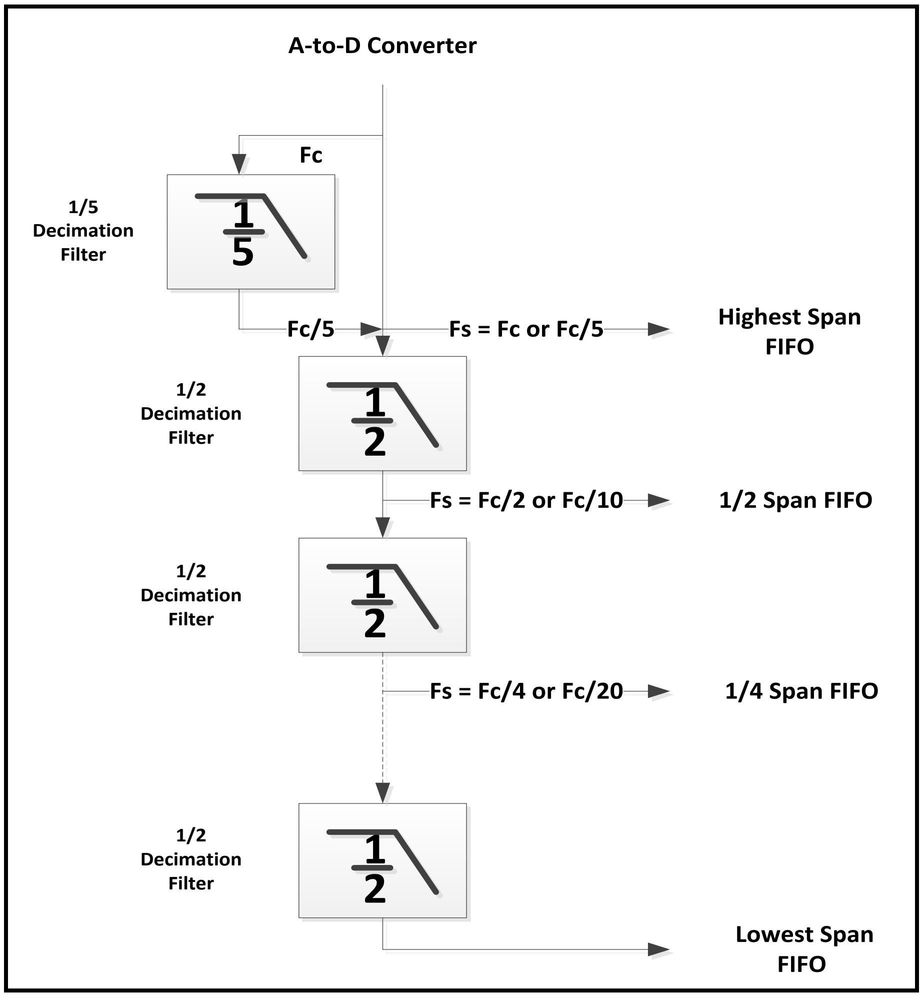


Figure 4-11: Decimation Filter and Measurement Span

FIFO

The decimated and band-limited samples by decimation filters are temporarily stored in a circular buffer (or FIFO) until the data is read out and transferred to the host.

Figure 4-12 illustrates the circular data buffer (FIFO). The figure shows three unread data records in the data buffer. The white *unused space* and *over-writable space* are the areas where new data can be written until this data buffer becomes full.

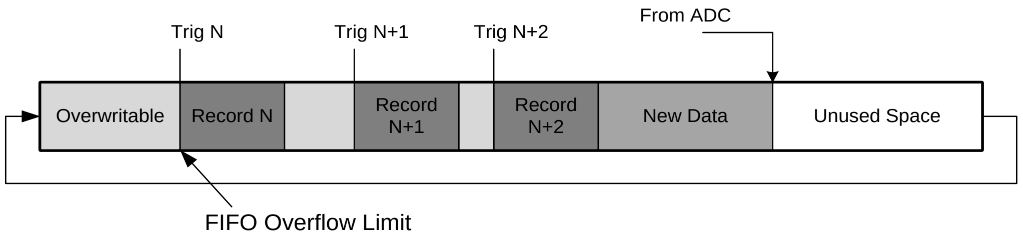


Figure 4-12: Circular FIFO

The EMX-4250/4350/4380 implements three different FIFO modes of operation in order to meet varieties of application needs. The FIFO mode defines the behavior when the FIFO buffer is full and there is no more space to write new data records.

Stop FIFO Mode

In *stop* FIFO mode, the instrument stops writing new data to the FIFO. The measurement is aborted after the last data record in the FIFO is read, and FIFO overflow error will be issued.

Overwrite FIFO Mode

In *overwrite* FIFO mode, the oldest data record is overwritten by the last data record. The overwritten data record will be permanently lost, but the measurement will not be aborted.

Wait FIFO Mode

In *wait* FIFO mode, the instrument will not write new data until oldest data record is read and enough data space becomes available. In this mode, all cards will stop accepting triggers until they are able to save their data without overwriting unread records. The measurement will not be aborted.

As a result of this circular buffer, when in FIFO *stop* mode, a FIFO overflow doesn't happen when enough records have been collected to fill the buffer, but when enough time has passed since the oldest unread record in the FIFO was saved. This amount of time is the size of the channel's circular buffer divided by its *SampleRate*. Each card can store 30 MSamples (31,457,280 samples) split among all enabled channels. If the sample rates the same on all channels, this time is ([31,457,280 Samples / number of channels] / *SampleRate*) seconds.

DATA STREAMING

The digitizer and DSA driver comes with an optimized data-reading interface for high speed and/or low-latency acquisitions. This interface relies on a separate thread which receives data asynchronously from the instrument as soon as it is available and acts on it immediately based on user-set preferences. There are two streaming modes, both can be enabled at the same time.

Memory Streaming

Memory streaming is the default mode for these instruments. When the user enables Streaming without any configuration, the data is streamed into a dynamically allocated buffer on the host PC and then retrieved via a Read-like interface. This is the fastest streaming interface, since memory is the fastest storage device on the PC. However, this mode is unsuited for long acquisitions because it can grow beyond the memory limits of the device and start paging to disk. The internal buffer grows as needed but does not shrink until streaming is disabled.

Additionally, this mode can be set to watch only select channels of interest. If the *MemoryChannelsList* property is set, the data for only the selected channels will be streamed to memory.

If another type of streaming is enabled while memory streaming is enabled, memory streaming is disabled unless the *MemoryChannelList* property is set.

Disk Streaming

Disk streaming is intended for high performance applications and/or long acquisitions. When enabled by setting a filename, the driver optimizes the data path by not converting the data from the native VRT data format, but, instead, delivering it as directly as possible from the network socket to the disk. A streaming file library has been made available so that post-processing can be done to turn this data from VRT format into any of several other formats.

Disk streaming has an optional maximum file size. When this is set, an error will occur and streaming will be disabled if the maximum file size is reached. This may be useful if the user has a disk quota or other limiting size they wish to avoid.

Another option provided is a maximum file count. When paired with the above option, the user-provided filename will be appended with a number ("Datalog" would turn into "Datalog0"), up to the maximum file count. This may be useful if the user can only process a file of a certain maximum size, but wishes to continue to take data after that size is reached.

A file rotation option has also been provided. When the *FileRotate* property is set to true and paired with the two options above, the driver will create *FileCount* files of *FileMaxSize* bytes, and then begin overwriting the earliest file created. This may be useful for infinite acquisitions, but assumes the user will be able to process the completed files (or copy them to another location) before the driver overwrites them.

SAMPLING RATES

The properties in the *Sampling* interface configure the A/D converter and decimation filters to specify the sampling rate and frequency span of the data to be acquired. Some properties in this interface are interrelated. Changing one property value can affect the other.

There is a *Sampling* property in the *Measurement* interface as well as the individual *Channel* interface. The global *Sampling* parameter that applies to all channels can be configured from the *Measurement* interface, while the *Channel* interface allows the user to configure each channel independently.

ClockFrequency

The *ClockFrequency* specifies the A/D conversion rate. This property determines the highest *SampleRate* of the data acquisition session. The highest rate can be achieved by bypassing all decimation filters. There are discrete sets of *ClockFrequency* values that the user can specify depending on the instrument model. See instrument specification for the list of *ClockFrequency* values. *ClockFrequency* is a global parameter. It cannot have different values for each channels. While it is possible to have different *ClockFrequency* values between cards in a single driver, it would be more convenient to have multiple driver instances for each *ClockFrequencies.*

Prescaler

This is the sampling rate divider before the ADC data is decimated by the ½ decimation filter stages. When a value of 5 is specified, the output from A/D converter is sent to 1/5 decimation filter first before going into ½ decimation filter stages. Otherwise, the *Prescaler* value is 1 and the data goes directly to the ½ decimation filters.

SampleRate

This is the effective data rate. The inverse of *SampleRate* specifies the interval between data samples that the user can obtain. The *ClockFrequency*, *Prescaler*, and *SampleRate* determines the number of ½ decimation filter stages. While the *ClockFrequency* is global, the *SampleRate* and *Span* value can be different in each channel. The value must be the one derived from a global *ClockFrequency*.

Span

The *Span* is the nominal frequency range of the acquired signal. The value of *Span* is determined by the decimation filter's cutoff frequency. In non-oversample mode, the *Span* = *SampleRate*/2.56 in. When *Oversample* is true, the Span = *SampleRate*/5.12.

RecordSize

The *RecordSize* specifies the number of data samples read back from the instrument at a time. The total number of data samples captured for each trigger is equal to the product of *RecordSize* and the *Measurement.NumRecordsPerTrigger* property -- that is, *NumRecordsPerTrigger* records are captured, and each is of size *RecordSize*. Figure 4-12 shows the case where *NumRecordsPerTrigger* = 4. As a special case, when *NumRecordsPerTrigger* is set to 0, an infinite number of data records are returned in *RecordSize* sample chunks after a trigger event until the measurement is aborted.

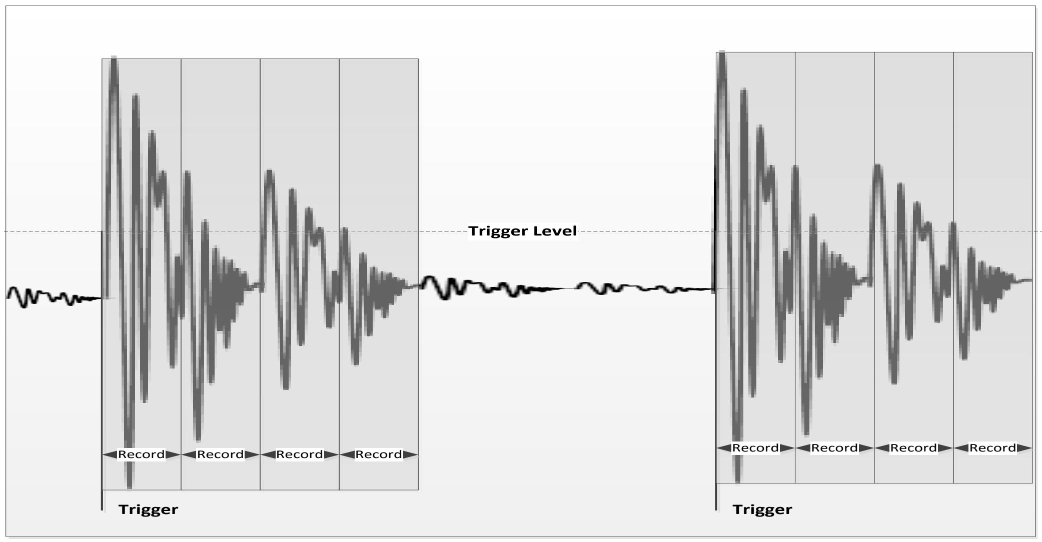


Figure 4-13: Record Size and *NumRecordsPerTrigger*

With EMX-4250/4350/4380, the minimum *RecordSize* is 1 and the maximum power of 2 *RecordSize* is 32768. Since each data packet transferred from the instrument to the host contains one data record from each ADC channel, in general, the user can achieve higher data transfer using a larger *RecordSize* value due to reduced overhead, but this affects the data update interval, as the amount of time to acquire one data record is: T = *RecordSize*/*SampleRate*.

Oversample

When the *Oversample* is set to true, the data is twice oversampled than Nyquist sample rate. The *SampleRate* becomes 5.12 times of *Span*. The *Oversample* mode is currently not supported.

Multipass

When the *Multipass* property is set to true, the data record contains data samples from multiple ½ decimation spans below the current decimation span specified by *Span* property. The *RecordSize*/2 samples is at *Span*, *RecordSize*/4 samples is at *Span*/2, *RecordSize*/8 samples is *Span*/4, etc., until it reaches less than 1 sample or the lowest decimation stage. *Multipass* is currently not supported.

OVERLOADS AND OPEN TRANSDUCER DETECTION

When the analog signal amplitude exceeds the input range, the digitized samples are truncated resulting in a distorted waveform. To avoid this, users can increase the input range of the instrument, attenuate the signal level, or discard overloaded data from processing. Apart from over-range, there are other fault conditions that can invalidate measurement data, such as open transducer. EMX-4250/4350/4380 is capable of detecting some of these common fault conditions and reporting it, allowing the user to act.

The fault status can be queried from the digitizer and DSA driver's API. The status can be indicated at the front panel LED. The information is also associated to the acquired data record and returned in *AdditionalData* string.

The digitizer and DSA drivers define each fault status as an *Overload Status* bit field. The user can configure which fault conditions to reported, if an LED indicator is shown, or it can latch so that the momentarily fault won't be overlooked.

Bit Field	Description	EMX-4250/51	EMX-4350	EMX-4380
0x00000000	Within the range	-	-	-
0x00000001	Signal level less than next lower input range	No	No	No
0x00000002	Exceed upper transducer limit	No	No	No
0x00000004	Exceed lower transducer limit.	No	No	No
0x00000008	Exceed upper user limit*	Yes	Yes	Yes
0x00000010	Exceed lower user limit*	Yes	Yes	Yes
0x00000020	Open Transducer	IEPE only	IEPE only**	IEPE only**
** User limit detection occurs right after ADC at the top span, before decimation filtering.* *** Open transducer is detected at around 100 ms - 200 ms interval with EMX-4350/4380.*				

Table 4-1: General Fault Status Bit Support

Bit Field	Description	EMX-4250/51	EMX-4350	EMX-4380
0x00010000	ADC overload (This includes differential and common mode overload)	Yes	Yes	Yes
0x00020000	Hardware common mode overload	No	Yes	Yes
0x00040000	IEPE transducer short	IEPE only	IEPE only*	IEPE only*
***** *IEPE short is detected at around 100 ms - 200 ms interval with EMX-4350/4380.*				

Table 4-2: Instrument-Specific Fault Status Bit Support

TIME STAMP

During data acquisition, the EMX-4250/4350/4380 returns timestamps along with digitized analog data. The timestamps are created based on the *TimestampSource* clock specified in *ReferenceOscillator* interface. When IEEE 1588 synchronized time stamps are desired, the *TimestampSource* property must be set to *ReferenceOscillatorTimestampSourceSystem*. Otherwise, the time returned will be the time elapsed since the SYNC signal was received.

When measurement data is retrieved, both a timestamp of the data record and a timestamp of trigger event are returned. The combination of *TimeSeconds* and *TimeFraction* parameters indicate the time of the first data sample in each retrieved data record. The time of trigger event is returned in the *AdditionalData* parameter.

Timestamp Source	Timestamp value	Resolution
System (IEEE 1588)	PTP or TAI (International Atomic Time)	20 ns
PXIe_CLK100	Elapsed time from last SYNC	20 ns
PXI_CLK10	Elapsed time from last SYNC	100 ns

Table 4-3: Timestamp Source and Resolution

ADDITIONAL DATA

The additional information associated with the data records are returned as a JSON (Java Script Object Notation) array of name/value pairs:

```
[[JSON array for channel 1's first record], [JSON array for channel
2's first record], ….]
```

The JSON object array for each channel is:

```
[{Object1}, {Object2}, …]
```

Each object is a list of name/value pairs:

```
{name1:value1,  name2: value2,…}
```

JSON objects can be parsed using the following defined name strings.

Object	Name String	Value
Over range	Over-Range	True when any overload status bit is set.
Trigger time	Trigger Timestamp Seconds	The trigger timestamp in seconds
	Trigger Timestamp Fraction	The fraction portion of trigger timestamp.
Dropped trigger	timestamp_sec	The timestamp of when the trigger would have occurred.
	timestamp_frac	The fraction portion.
	Trigger Dropped	The channel name of the lost trigger.

Table 4-4: Additional Data JSON Names and Values

MEASUREMENT PROCESS

MEASUREMENT SETUP

After the instruments are initialized by driver *Initialize* method, or the previous measurement is finished, the instrument is in *Idle* state. While in the *Idle* state, the user prepares for the next data acquisition by configuring the setup parameters.

Start, Arm, and Trigger

Properties in the *Start*, *Arm*, and *Trigger* interfaces configure the data acquisition gating condition and timing. *Start* determines when the data acquisition starts and when the instrument is ready for the next acquisition. By the time of *Start*, all hardware configurations and filters settling need to be complete in order to ensure valid data is acquired.

Sampling Parameters

The sampling parameters are specified by properties in *Sampling* interface. *Sampling* in the *Measuremen*t interface configures parameters common to entire system. Optionally, the user can configure the parameters of individual input channels using the *Channels* interface (with some restrictions).

Analog Front End

While some parameters can be changed during the data acquisition, most configuration options should be performed prior to data acquisition to avoid data glitches. Most of front end parameters can be configured independently for each input channel. The front end configuration includes input range, AC/DC coupling, IEPE current, voltage/charge input function, and single-ended or differential input mode. To ensure the most accurate measurements, it is highly recommended that *SelfCalibration* be performed prior acquiring data.

MEASUREMENT INITIATION

After the user has completed configuring the instrument, the user can initiate the data acquisition process by calling the *Initiate* method in the *Measurement* interface. During measurement initiation, the instrument starts to prepare for the actual data acquisition. If the ADC has not been synchronized, the SYNC signal is sent and ADC then begins digitization. When the ADC's reference oscillator has been changed, the ADC sampling clock must be re-locked with the PLL. Filter settling also occurs at this time. Once the filters have settled and the instrument is ready to acquire valid data, the measurement moves to the *Wait for Start* state. The source of the *Start* event is *Immediate* by default. In this case, the measurement starts automatically. Otherwise, it must be started by an event specified by the *Source* property of the *Start* interface.

During this period, the digitized signal from the A/D converter is continuously filtered and discarded until the filters have settled. Once settled, the filtered data samples are stored in the internal data buffer.

MEASUREMENT LOOP

Once the measurement begins, the state machine cycles through arming and triggering for the number of times specified by *ArmCount* and *TriggerCount*. The measurement stops when the specified number of arm and trigger loops are completed or when it is aborted by an *Abort* command or when the FIFO buffer became full and the FIFO mode is set to *Stop*.

ARMING

Arm is the gating condition to acquire data. In order to trigger data acquisition, the measurement must be armed first. There are several ways to arm a measurement. The default condition is to arm automatically (or *Immediate* arming).

Self-arming

When the *Immediate* arm source is enabled, the EMX-4250/4350/4380 arms by itself.

Arming by User's Command

The EMX-4250/4350/4380 arms by the *SendSoftwareArm* method when a *Software* arm source is enabled.

Arming at a Certain Time Interval

The EMX-2500 controller can be configured to assert a PXI trigger line (PXI0-7) at a specified time interval using the platform driver's *Alarm* interface. The EMX-4250/4350/4380 can be armed with these events.

Arming from Backplane Trigger Line by Other Instruments

The EMX-4250/4350/4380 can receive an arm event from the other instruments in the same PXI chassis by configuring one a PXI trigger line (PXI0-7) as an arm source.

Arming from LAN Events by Other Instruments

The user, or another LXI device, can send LAN events to arm the EMX-4250/4350/4380s using the digitizer/DSA driver. Alternatively, a LAN event can be sent to the EMX-2500 controller using the platform driver. Upon receiving a LAN arm event, the EMX-2500 can assert one of the PXI trigger lines to arm the EMX-4250/4350/4380 (or any other devices within a same chassis). The former method is more flexible, and is the recommended approach when using all SentinelEX plugins. It is only recommended to use the Platform method when synchronizing with non-SentinelEX cards in the same chassis.

TRIGGERING

When the measurement is armed and when there are enough digitized samples already collected in the FIFO for pre-trigger delay, the measurement becomes ready to receive an trigger event.

Self-triggering

When the *Immediate* trigger source is enabled, the EMX-4250/4350/4380 triggers automatically and acquires data records as soon as it is ready to receive a new trigger event.

Triggering by User's Command

The EMX-4250/4350/4380 triggers by the *SendSoftwareTrigger* method when a software trigger source is enabled.

Triggering by Analog Signal

The EMX-4250/4350/4380 is triggered when an analog signal at an input channel crosses the trigger threshold level. In order to trigger from the analog signal, the channel must be enabled as a trigger.

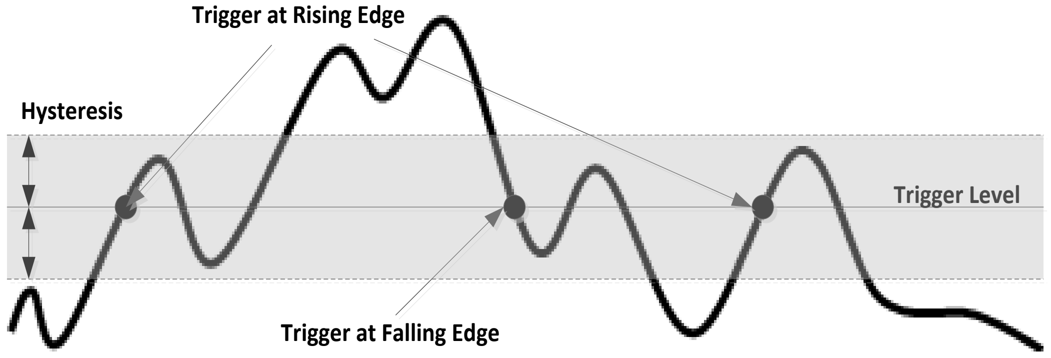


Figure 4-14: Triggering by Analog Signal

Triggering by External Pulse

The EMX-4250/4350/4380 can receive a trigger from the front panel trigger SMB connector by enabling the EXT trigger source.

Triggering from Backplane Trigger Line by Other Instruments

The EMX-4250/4350/4380 can receive a trigger from other instruments in the same PXI chassis by configuring one of PXI trigger lines (PXI0-7) as a trigger source.

Triggering from LAN Events by Other Instruments

The user, or other LXI device, can send a LAN event to trigger the EMX-4250/4350/4380s using the digitizer/DSA driver. Alternatively, the LAN event can be sent to the EMX-2500 controller using the platform driver. Upon receiving a LAN trigger event, the EMX-2500 can assert one of the PXI trigger lines to arm the EMX-4250/4350/4380 (or any other devices within a same chassis). The former method is more flexible, and is the recommended approach when using all SentinelEX plugins. It is only recommended to use the Platform method when synchronizing with non-SentinelEX cards in the same chassis.

Triggering by Stimulus Signal

When the EMX-4250/4350/4380 is used with the EMX-1434 in single DSA driver session, one of EMX-1434's DAC channels can be specified as a trigger source. In this configuration, every time the EMX-1434 generates a signal, it also triggers the EMX-4250/4350/4380 for data acquisition.

DATA RETRIEVAL

After a measurement is triggered and at least one record of data (the number of samples specified by *RecordSize* property in *Sampling*) is available in FIFO buffer, it can be retrieved using the *Read* method in *Measurement* interface or via the *Streaming* method described in the *Data Acquisition* section in this manual. At each trigger event, the number of contiguous records specified by *NumRecordsPerTrigger* are acquired. Multiple records can be read separately or all in once if they are available in the instrument's FIFO buffer. The *Read* method in *Measurement* returns at least one record from all enabled channels.

When the user wishes to acquire continuous samples indefinitely after a single trigger event, set the *NumRecordsPerTrigger* property to 0 (infinite) and set FIFO mode to *Stop*. The data acquisition stops when the user aborted using the *Abort* method or when the FIFO buffer becomes full (FIFO overflow). In this setup, the user must retrieve data faster than the ADC data filling into FIFO in order to avoid FIFO buffer from overflowing.

Pipeline Delay and Latency

For high speed data recording, data transfer speed is key. For applications that require real-time data monitoring or processing, on the other hand, the data update rate becomes more important. A real-time closed loop control is an example. It is necessary to understand that there are delays at almost every stage of the data acquisition process. Some are within the instrument while others occurs outside of the instrument, such as at the transducer or in the user's application.

The analog signal conditioning circuit introduces some delay before the signal reaches the A/D converter. This is further discussed in 0. Usually the delay in the analog section is small and negligible, but it must be noted that the AC coupling filter introduces a non-linear delay, especially in the low frequency range.

In the digital section, the digital filters in the A/D converter and decimation filters introduce a group delay. It is about 45 µs at 625,000 Hz and increases at lower frequency spans. Please refer to Table A-2 for more info. The filtered data is transferred to FIFO buffer through a "ping pong" buffer at a specified internal (see Table 4-5). Once a trigger is detected, the data in the FIFO can be read out to the user's application at the chunk of *RecordSize* samples.

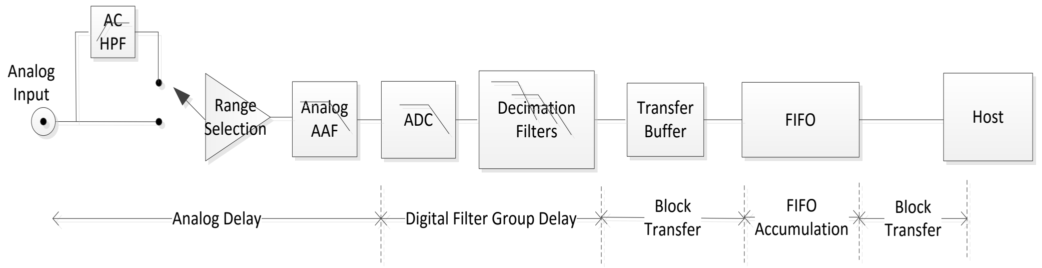


Figure 4-15: Delays in Data Path

Model	EMX-4350/4380			EMX-4250/4251	
Clock Freq. (F_{CLK})	409600 Hz	524288 Hz	625000 Hz	131072 Hz	204800 Hz
F_{CLK}	0.0025	0.001953125	0.0016384	0.0078125	0.005
1/2 F_{CLK}	0.005	0.00390625	0.0032768	0.015625	0.01
1/4 F_{CLK}	0.010	0.0078125	0.0065536	0.03125	0.02
1/8 F_{CLK}	0.020	0.015625	0.0131072	▨	▨

Table 4-5: FIFO Buffer Update Rate (Seconds)

EVENTS

Events are an optional feature allowing the EMX-4250/4350/4380 to send notifications to the user, or other instruments, when a specified event occurs. The notification can be via LAN message or a PXI trigger line. The user can specify an event when the measurement state is armed, triggered, or finished. See digitizer/DSA driver's online help for more information.

USING WITH EMX-1434

This section describes how the EMX-1434 can be used together with the EMX-4250/4350/4380 instruments. See *EMX-1434 User's Manual* for more detail information for this instrument.

Currently, EMX-1434 is configured and controlled by DSA driver. When both EMX-1434 and EMX-4250/4350/4380 modules are included in a single DSA driver session, they work synchronously. When the user wishes to control EMX-1434 independently from the data acquisition, a separate driver instance must be created.

STIMULUS SIGNAL GENERATION

The EMX-1434 has four DAC output channels. For stimulus-response measurement, the EMX-1434 can be used as a stimulus signal generator. Since EMX-1434 can generate four independent signals at the same time, it can be used for MIMO (Multiple-Input-Multiple-Output) system analysis. Using a single DSA driver session, the data acquisition and stimulus signal generation can be synchronized together by the common trigger, sampling clock, and a state machine.

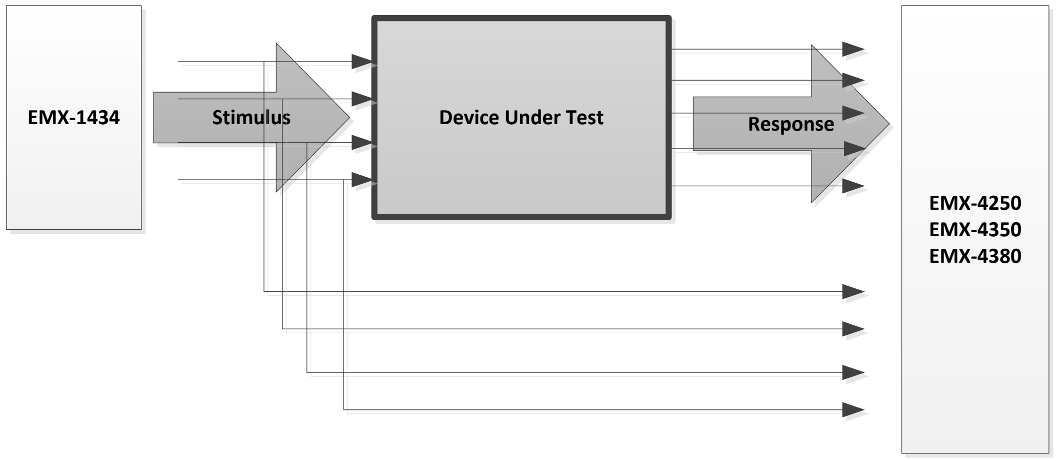


Figure 4-16: Stimulus and Response

TACHOMETER INPUTS

The EMX-1434 has two tachometer input channels. The main application for tachometer inputs is for rotating machinery testing. The EMX-1434 can measure instantaneous rotating speed in RPMs from tachometer pulses connected to one of tachometer channels. The RPM values can be associated with the EMX-4250/4350/4380 data records returned.

Some applications require acquired measurement data to be associated with a certain reference signal phase. This signal can also be a single pulse per revolution tachometer signal. For this application, the EMX-1434's tachometer channel can be used as a trigger source of the measurement.

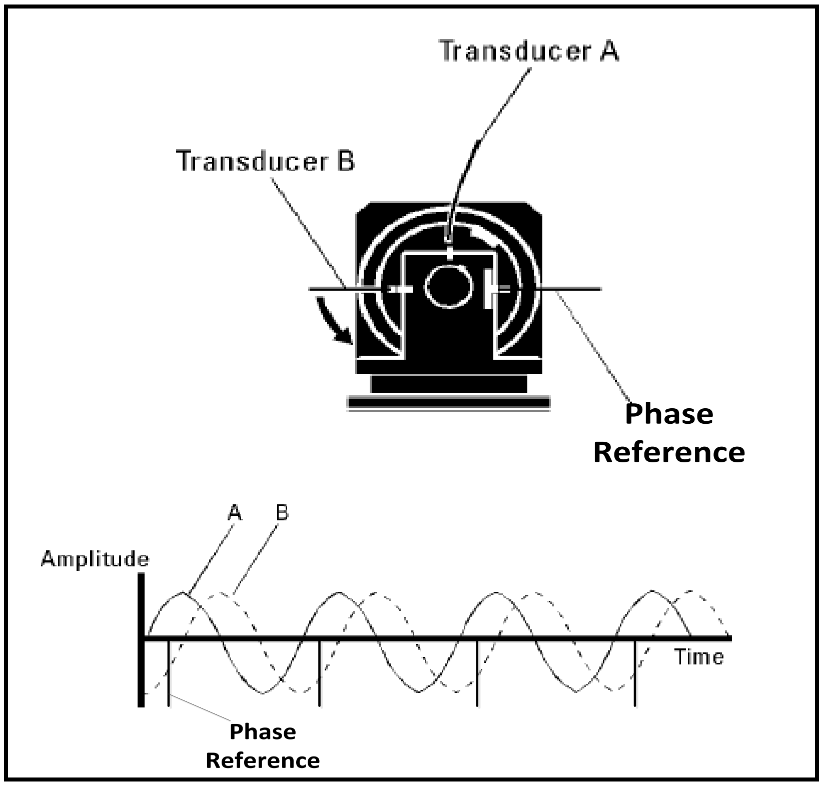


Figure 4-17: Tachometers

The EMX-1434 can be configured to continuously return time stamps when the tachometer signal is detected. The array of time stamps can be used to resample [1] the data from EMX-4250/4350/4380's for synchronous measurement.

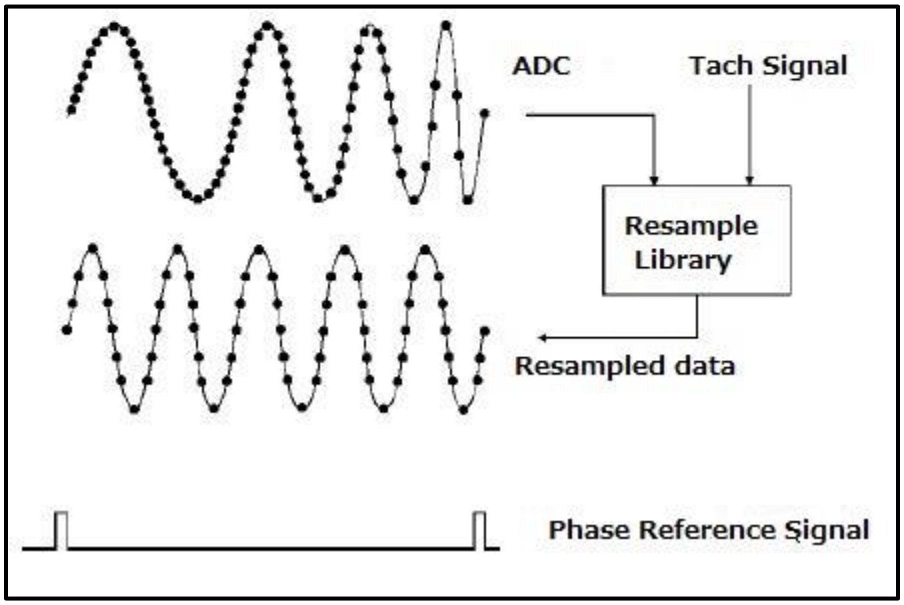


Figure 4-18: Resampling ADC data by tachometer pulses

[1] VTI provides a separate library to resample data acquired by EMX-4250/4350/4380 and other VTI digitizer.

DIGITAL INPUT AND OUTPUT

The EMX-1434 has 4-bit DIO channels. The digital signal can be associated to the EMX-4250/4350/4380 measurement data record. Optionally, DIO value can be used as an arming condition of the measurement.

WHERE TO FIND MORE INFORMATION

DRIVER API REFERENCE

The complete driver's API reference is available as online help. Each driver comes with .chm format help file.

Each drivers come with several useful example programs in C++ and C#.

OTHER MANUALS

User's Manuals are available for the EMX-2500, CMX09, CMX18, and EMX-1434.

SPECIFICATION INFORMATION

EMX-4250/4350/4380 conforms to many industry standards in both hardware and software architecture. Although products can be used without knowing these standards, some knowledge can be useful to take full advantage of VTI Instruments products.

- LXI specification is available from LXI consortium at www.lxistandard.org
- PXI specification is available from PXI System Alliance at www.pxisa.org
- IVI driver specification is available from IVI Foundation at www.ivifoundation.org
- IEEE 1451 Smart Transducer Interface Standard and IEEE 1588 Precision Synchronization Protocol Standard at www.nist.gov/el/isd/ieee
- ANSI/VITA 49.0 (VRT) is specified as a part of VITA specification at www.vita.com

SECTION 5

DIGITIZER MODULE DESCRIPTIONS

OVERVIEW

The Smart Dynamic Signal Analyzers family of products incorporates best-in-class analog design methodology to deliver industry leading measurement accuracy. These instruments are ideal for a wide range of applications including noise, vibration, and harshness (NVH); machine condition monitoring; rotational analysis; acoustic test; modal test; as well as general purpose high speed digitization and signal analysis.

DIGITIZER FEATURE COMPARISON

	EMX-4250/4251	EMX-4350	EMX-4380
No. of Channels	4250: 16ch / 4251: 8ch	4	4
IEPE Input Type	Pseudo-Differential	Differential	Differential or Single-Ended
Max. Sample Rate (samples/s)	204.8k	625k	625k
Voltage Input (V)	0.1, 0.2, 0.5, 1, 2, 5, 10	0.1, 1, 10, 20	0.1, 1, 10, 20
Charge Input (kpC)	N/A	N/A	0.1, 1, 10
IEPE Current (mA)	4.5 mA, 10 mA	0 mA - 20 mA	4.5 mA, 10 mA
Input Coupling	AC, DC	AC, DC	AC
AC Coupling -3 dB Corner (Hz)	IEPE/Volts: 0.24 Hz	IEPE/Volts: 0.5 Hz	IEPE/Volts: 0.20 Hz Charge: 0.32 Hz
Input Ground Isolation from Chassis	None	None	ON or OFF ON: 50 V dc, 100 MΩ

Table 5.1: Digitizer Feature

POWER CONSUMPTION

The table below shows the +3.3 Vdc and +12 Vdc current consumption drawn from the backplane for each module. The current drawn from the +12 Vdc rail is shown for three common IEPE current settings (all channels with the same setting): 0 mA, 4.5 mA, and 10 mA.

Model	**Current Requirement (A)**				**Max. Power (Watts)**
		12 V dc			
	3.3 V dc	**IEPE=0 mA**	**IEPE=10 mA**	**IEPE=4.5 mA**	
EMX-4350	1.32	0.46	0.56	0.51	12.3
EMX-4380	1.32	0.45	0.60	0.52	11.6
EMX-4250	1.64	0.60	0.96	0.76	16.9
EMX-4251	1.64	0.33	0.53	0.42	11.7

Model	**EMX-4016**	**EMX-4016B**	**EMX-4016M**
Power	NA	< 2W	12.5 Watt Typical, Vexc=10V, RL=350Ω 16.5 Watt Typical, Vexc=5V, RL=120Ω 20.0 Watts Max. Vexc shorted

Table 5.2: Power Consumption

TABLE 5.2: POWER CONSUMPTION

BLOCK DIAGRAMS

EMX-4350 Block Diagram

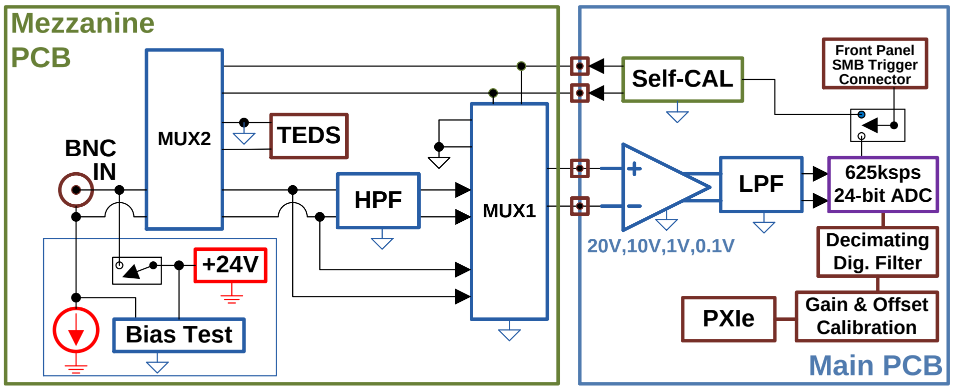


Figure 4-18: EMX-4350 Block Diagram

EMX-4380 Block Diagram

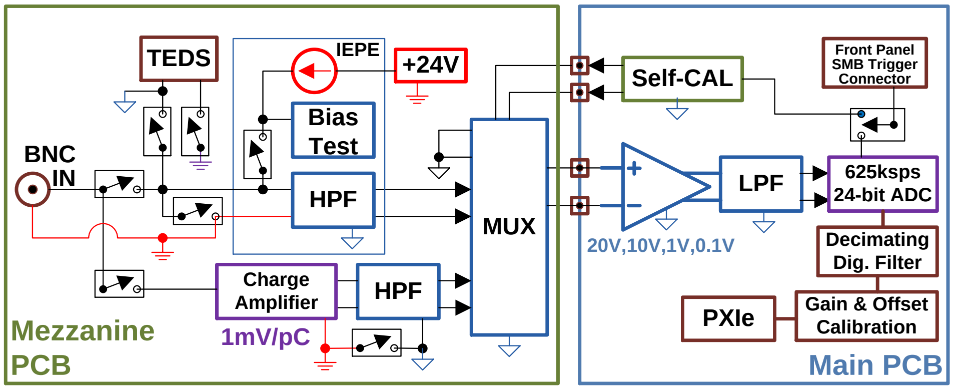


Figure 4-19: EMX-4380 Block Diagram

EMX-4250/4251 Block Diagram

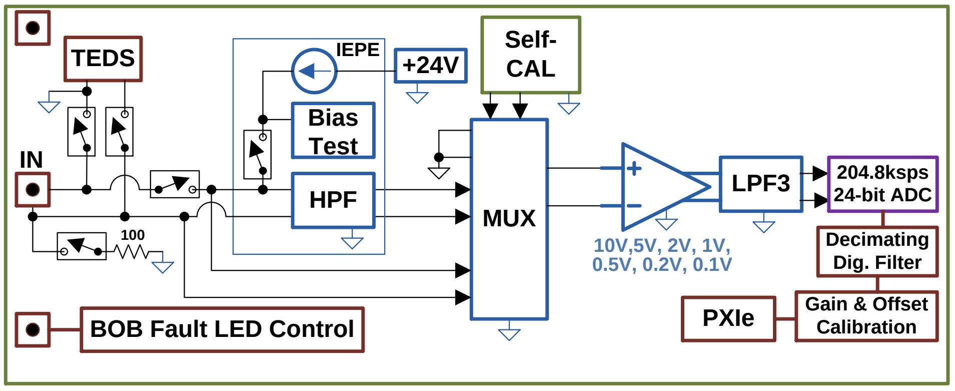


Figure 4-20: EMX-4250/4251 Block Diagram

EMX-4016B Block Diagram

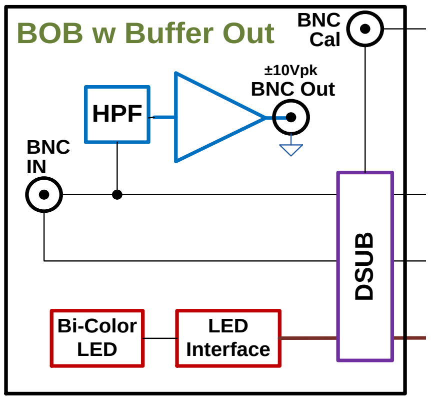


Figure 4-21: EMX-4016B Block Diagram

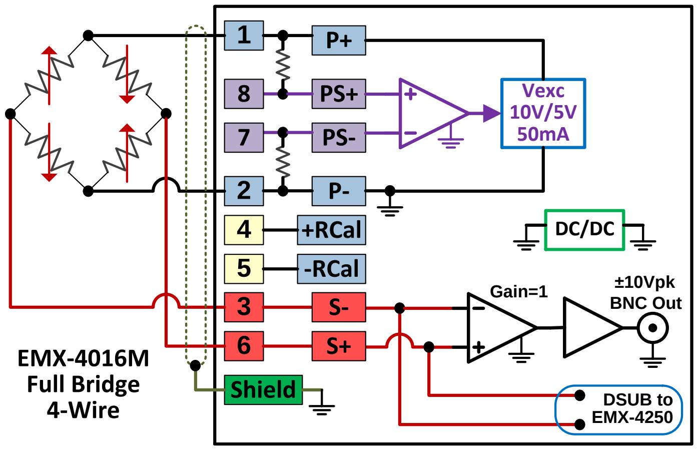


Figure 4-5: EMX-4016M Block Diagram

INPUT TYPES

Voltage

The EMX-4350 and EMX-4380 have BNC input connectors that are isolated from chassis ground. The center conductor connects to HI and the outer shell connects to LO. The maximum voltage applied to each input (differential plus common mode) cannot exceed ±20 V. A floating source needs to have a DC path to ground through some resistance to make sure the common mode voltage stays close to ground.

The EMX-4250/51 have 25pin microD connectors and voltage signals applied to each input channels using EMX-4016/EMX-4016B/EMX-4016M Break out box. These Break out boxes pass through the signals directly from the input BNC connector to the 25pin microD connected to the EMX-4250/51 without any signal conditioning.

IEPE

IEPE sensors operate by receiving a DC current (usually 4.5 mA) to power the internal electronics. A properly operating IEPE sensor will establish a DC bias voltage of 9 V dc to 15 V dc at its output. The AC voltage output is proportional to the physical measurement measured by the transducer. The voltage output from IEPE accelerometers can be converted back to Engineering Units (g's or m/sec^2) by dividing the measured volts by the sensitivity (mV/g or mV/m/sec^2) of the IEPE accelerometer. Note that other types of sensors besides accelerometer can be of IEPE type. There are IEPE microphones (mV/Pa) and IEPE force sensors (mV/lbf).

IEPE sensors are known by other registered trademark names depending on the manufacturer: ICP® (PCB Piezotronics), Isotron® (Endevco), Piezotron® (Kistler), DeltaTron® (Brüel & Kjær).

The IEPE current source needs to be set to the desired amplitude, usually > 4 mA and input coupling needs to be set to AC. It has a very high compliance voltage (>21 V) and high output impedance. The current source is polarized and must be operated with positive current flowing from the (+) terminal to the (-) terminal. The IEPE current source can be enabled individually per channel. Channels that do not have a current source enabled can be used for standard voltage measurements.

The EMX-4250's input is set to pseudo-differential when set to IEPE. The LO side of the input is shunted to amplifier signal ground through a 100 Ω resistor.

Each channel's current source in the EMX-4350 and EMX-4380 is isolated, allowing operation of all channels in IEPE mode to float relative to one another and maintaining the input of each channel as fully differential.

The frequency response of the IEPE sensor is affected by the cable capacitance, the amplitude of the IEPE current source, and the maximum voltage swing. The following equation can be used to calculate the approximate maximum frequency:

$$F_{max} = \frac{I - I_S}{C*L*V*K}$$

- I: amplitude of IEPE current source (amps)
- I_S: current required by the IEPE sensor to power its internal electronics (usually around 0.001 A)
- C: cable capacitance per foot (usually between 15 pF/ft to 30 pF/ft)
- L: cable length (ft) L can be in meters if C is given in pF/m
- V: maximum output peak voltage swing from sensor (volts)
- K: factor depending on amplitude accuracy: K = 2.9 for -0.5 dB; K=4.5 for -0.1 dB

Figure 4-6 below shows the frequency response for C = 20 pF, V = 5 V pk, K = 2.9 (-0.5 dB), I_S = 1 mA, I = 4.5 mA or 10 mA.

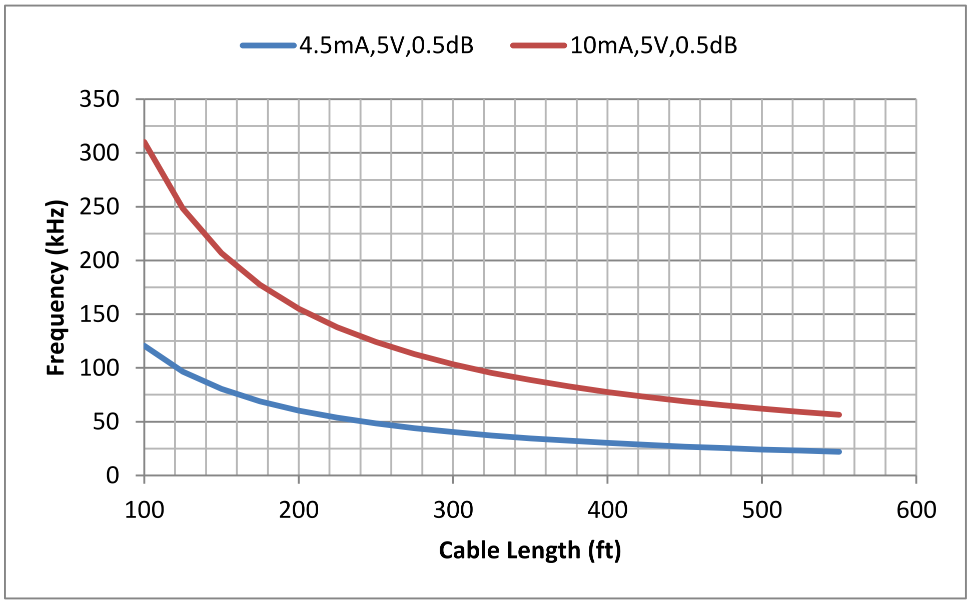


Figure 4-6 –Frequency Response of IEPE Sensor Affected by Cable Length

Charge (EMX-4380)

The charge amplifier is used to convert the charge output (in picocoulombs - pC) from Piezo-Electric (PE) transducers to volts. IEPE sensors are PE transducers with built-in charge amplifiers. PE sensors are used in high temperature environments where the built-in electronics cannot survive or operate poorly such as inside jet engines. The charge amplifier does not have DC response. The charge amplifier operates from an isolated power supply, so the BNC input shell is isolated from chassis ground.

The charge amplifier is tested by using a series capacitor Cin (see Figure 4-7) to convert volts to pC. A 1000 pF capacitor converts a 1 V pk sine-wave into 1000 pC pk sine wave. The gain of the charge amplifier is 1 mV/pC nominal (±2% over temperature), so the output of the charge amplifier produces 1 Vpk sine wave output. Accuracy of the charge amplifier gain calibration is dependent on how accurately Cin is known.

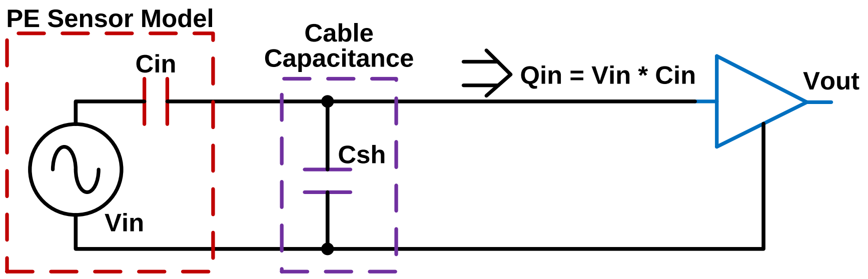


Figure 4-7: Voltage to Charge Conversion by Using Series Capacitor

The EMX-4380 can operate with input isolation turned ON or OFF. When isolation is ON, the input LO side is floating from the digitizer ground and the voltage output of the charge amplifier is injected into a differential input voltage amplifier. This makes the input look like a differential charge amplifier since all common mode signals are passed through the charge amplifier and rejected by the differential input voltage amplifier located in the main PCB. Both differential and single-ended charge output sensors can be connected to the EMX-4380 when isolation is ON. When isolation is OFF, the input of the charge amplifier operates in single-ended mode with LO side connected to digitizer ground.

The accuracy in the pass-band of the charge converter is not dependent on cable capacitance Csh. This would not be the case if the front-end is a voltage amplifier instead of a charge amplifier since Csh together with Cin would be an attenuation stage. Csh and Cin affect the high frequency response of the charge amplifier. Use the equation below to compute the high frequency corner for the charge amplifier.

$$F_{max} = \frac{1}{(C_{in}+C_{sh}) * 200\pi * K}$$

- C_{sh}: Cable capacitance
- C_{in}: PE transducer source capacitance
- K: factor depending on attenuation at F_{max}: K=3 for -0.5dB; K=6 for -0.1dB; K=1 for -3dB

Figure 4- shows the amplitude frequency response of the front-end charge amplifier stage for various values of source capacitance (Csh+Cin). This response needs to be combined with the corresponding charts shown in the Anti-Alias Filter section to get the overall response of the EMX-4380 digitizer when it is set to Charge input.

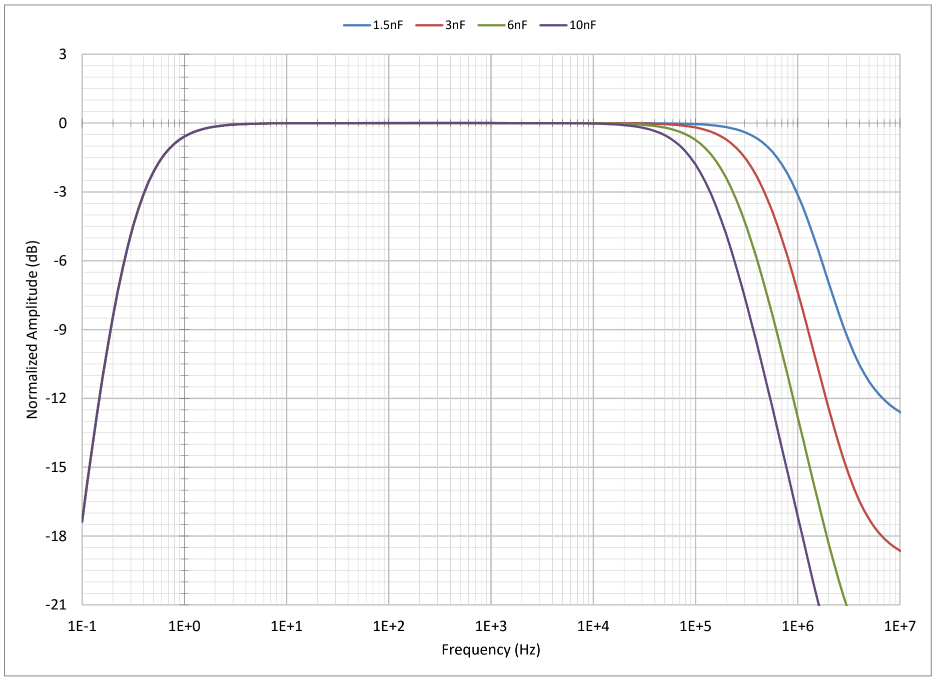


Figure 4-8: EMX-4380 Charge Amplifier Front-End Magnitude Frequency Response

The referred to input (RTI) noise of the charge amplifier is dependent on the source impedance at its input. Use the equation below to compute by how much the noise will increase based on the source resistance Rs of the transducer:

$$\text{Noise Multiplier} = 1 + \frac{500\text{M}\Omega}{\text{Rs}}$$

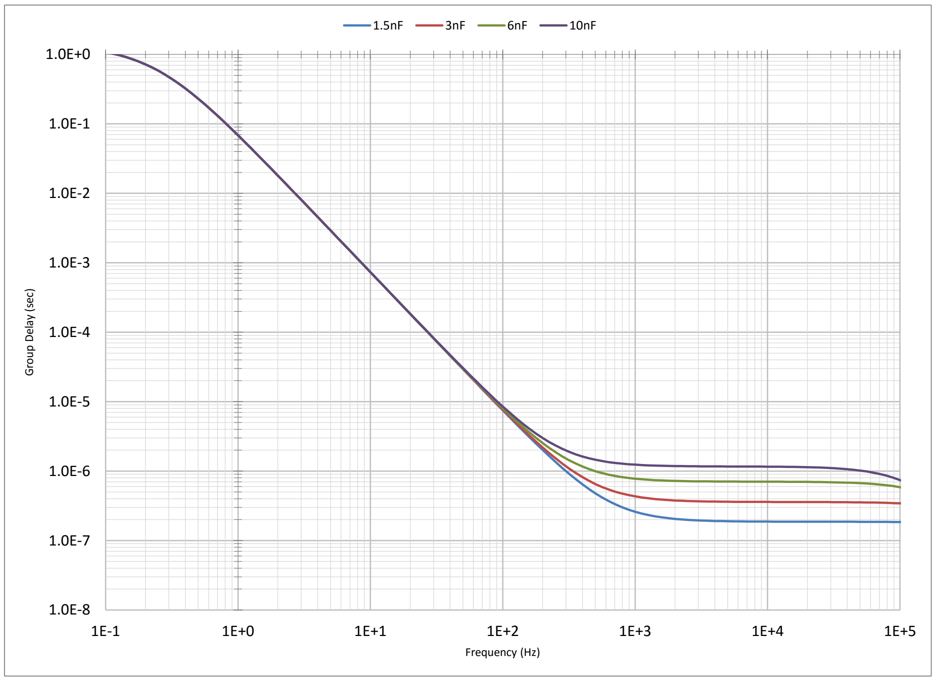


Figure 4-9: EMX-4380 Charge Amplifier Front-End Group Delay Frequency Response

BREAK OUT BOX OUTPUT

The EMX-4016B/EMX-4016M BoB provides only one output per channel, which is directly fed into EMX-4250/51 card. The EMX-4016B BoB is designed to provide, buffered analog output per each input channel, in addition to feed into EMX-4250/51 card. For some high reliability applications, IEPE signal conditioning Break-out-Box should have dual analog outputs, to support system level functions, such as redundant recording, real time monitoring, independent control, etc.

The EMX-4016B Break out Box is essentially a 16 Channel Signal Conditioner, which is designed for use with integral electronics piezoelectric (IEPE) transducers or piezoelectric transducers with a Remote Charge Convertor (RCC). It supplies power to the transducer from a two wire constant current source. This current source is controlled via EMX4250/51 card, and can be set to 4.5mA or 10mA as per software control.

The EMX-4016M Break out Box is essentially a 16 Channel Signal Conditioner, which is designed for use with full bridge type sensors. It provides voltage excitation to the transducer. This voltage excitation source can be set to 5V, 10mA and OFF as per the front panel switch.

Each channel has a Status LED Indicator to notify the user if the input conditions are healthy or input has a fault (open or short circuit condition). The buffered analog outputs are AC coupled though a 0.5Hz high pass filter, and has a fixed gain of x1. There are no user configurable jumpers available in this BoB, since it is fully controlled via EMX4250/51 digitizer card.

The input power for the operation of this BoB is supplied by an external DC power adaptor (Delta Electronics, Model # MDS-060AAS24 BA). This external power supply is compatible for use with wide range of AC input (90Vac to 275Vac), and is provides low earth leakage, certified EMC standards compliance according to EN 55011 for industrial, scientific and medical (ISM) radio-frequency equipment and EN 55022 for Information Technology Equipment (ITE) radio-frequency equipment.

The transducer input and buffered output connectors are electrically isolated BNC type sockets with side connected to circuit common (no electrical channel to channel ground isolation). The input connectors are on the front panel, along with LED indicators.

The communication, and signal output connectors for connecting to EMX-4250/51 card, are dual 25pin Micro D-Sub connectors, which are located at the rear side of BoB. The buffered output BNC connectors are also located on the rear panel.

NOTE: Buffer output in the 4016M will not work if the EMX-4250 is set to IEPE mode.

BRIDGE TRANSDUCER

Voltage Excitation (EMX-4016M)

Two voltage excitation levels can be selected by two (2) front panel switches. One switch sets the voltage for channels 1 to 8, and the other switch for channels 9 to 16. EMX-4016-001 can set the excitation level remotely by the EMX-4250. The front panel switch needs to be in the "OFF" position for the EMX-4250 to be able to remotely set the excitation level. Setting the front panel switch to select one of the voltage excitation levels overrides remote control. Each channel has 2 LED indicators to show which excitation level has been selected. The left or right LED will turn ON corresponding to the switch position.

AC INPUT COUPLING

The charts below shows the amplitude and group delay frequency response when coupling is set to AC for the EMX-4250/4251, EMX-4350, and EMX-4380. This responses need to be combined with the corresponding charts shown in the Anti-Alias Filter section to get the overall response of the digitizer when they are set to IEPE or Voltage with AC coupling.

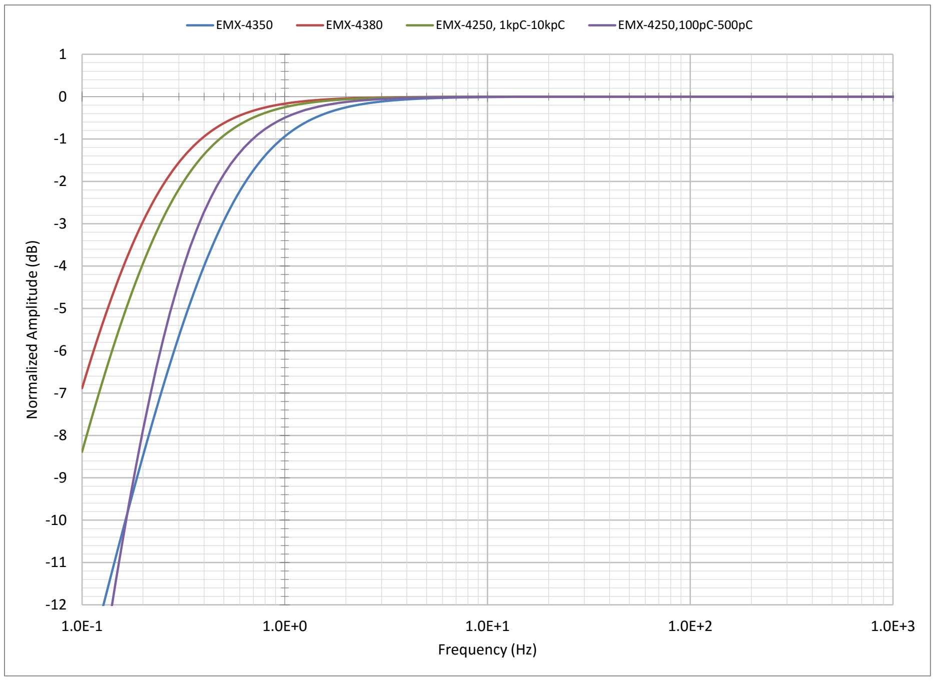


Figure 4-22: AC Coupling Voltage Magnitude Frequency Response

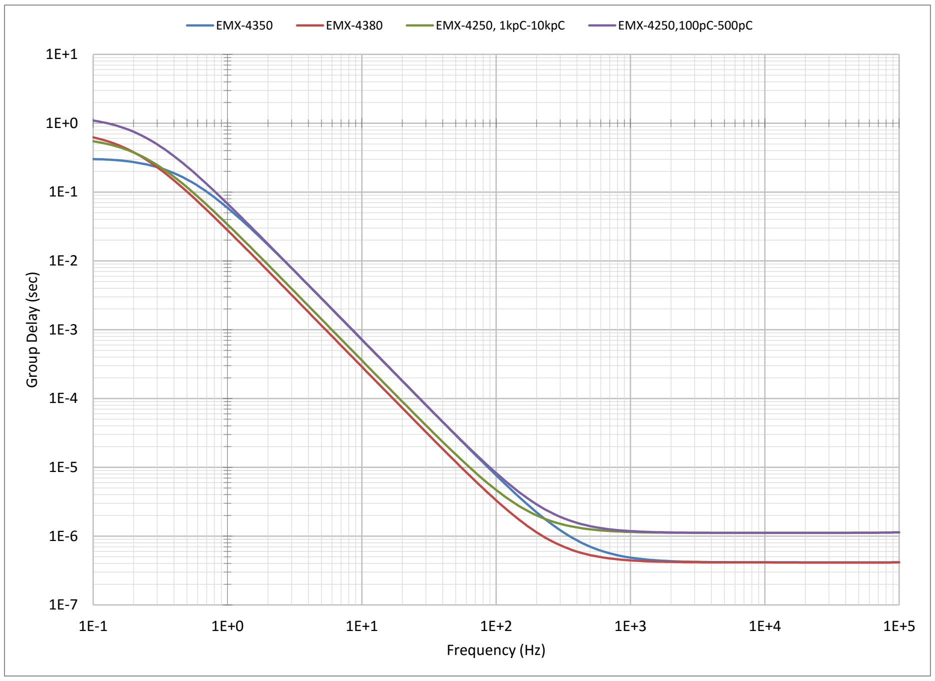


Figure 4-23: AC Coupling Voltage Group Delay Frequency Response

IEPE Input Fault Detection

IEPE input fault detection is accomplished in hardware by monitoring the DC bias of the input signal. IEPE sensors have a DC bias between 9 V dc and 15 V dc when they receive a DC current (usually between 4 mA dc and 10 mA dc). The AC signal riding on the DC bias is the signal proportional to the physical measurement. An open or short condition at the input is detected by monitoring the DC bias voltage at the input. A SHORT is declared when the DC bias is less than threshold voltage VTL and an OPEN is declared when the DC bias is greater than threshold voltage VTH. It is possible for the unit to declare a failure if the AC signal is low frequency (<0.5 Hz) and its peak amplitude (DC+AC) exceeds the threshold voltages VTL or VTH. The table below shows the typical threshold voltages VTL and VTH for each Model.

Model	VTL	VTH
EMX-4250/EMX-4251	1.30 V	21 V
EMX-4350/EMX-4380	0.95 V	20 V

Table 4-6: Typical VTL and VTH Threshold Voltages

The front panel fault LED in the EMX-4350 and EMX-4380 will turn RED if the input is either OPEN or SHORT and the module is set to IEPE and the fault can also be read through software.

The EMX-4250 & EMX-4251 can control the fault LED in the EMX-4016 16-channel break-out-box (BOB). This is done through the I2C serial communication lines available in the 25-pin input connector.

TEDS

Each channel has the ability to read and write TEDS data from IEEE 1451.4 TEDS (Transducer Electronic Data Sheet) enabled transducers. The figure below shows a connection diagram of an IEPE sensor with a 1-wire memory containing TEDS information. Schottky Diode D1 is used to block the IEPE current powering the IEPE sensor from damaging the memory chip. D1 needs to have a low (<0.5 V) forward voltage. Note that the memory chip is connected backwards: its ground is connected to HI and the IO pin is connected to the LO. TEDS cannot be read by the digitizer if LO is connected to chassis ground.

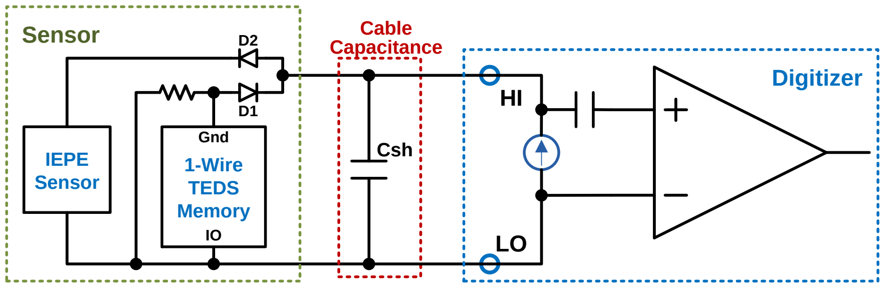


Figure 4-24: IEPE with 1-wire TEDS Memory

The forward voltage of diode D1 and cable capacitance (C_{sh}) and limit the maximum cable length at which TEDS can be read. The IEPE sensor output is temporarily disabled by diode D2 when reading TEDS because this involves turning OFF the IEPE current source and applying a positive bias to the LO terminal while grounding the HI terminal.

Be careful when connecting a 1-wire memory IC directly to the BNC input. Enabling the IEPE current source will damage the memory IC if diode D1 is not present.

Analog to Digital Converter (ADC)

The DSA digitizers use 24-bit delta-sigma analog-to-digital converters. The ADC runs at a fixed sampling rate so there is no need to change the corner of the analog anti-aliasing filter. Lower sample rates are obtained by a decimating digital filter. Decimation by 2 and 5 are supported.

The table below shows the available data rates at which the ADC runs for each model.

Model	**Oversampling Ratio (f_{CLK} / f_{DATA})**				**Oversampling Clock (f_{CLK})**
	32	**64**	**128**	**256**	
	Data Rate (f_{DATA}) - Samples per Second (SPS)				
EMX-4250	131,072	65,536	32,768	N/A	4,194,304 Hz
EMX-4251	204,800	102,400	51,200	N/A	6,553,600 Hz
EMX-4350 EMX-4380	409,600	204,800	102,400	51,200	13,107,200 Hz
	524,288	262,144	131,072	65,536	16,777,216 Hz
	625,000	312,500	156,250	78,125	20,000,000 Hz

Table 4-7: Available Data Rates

Anti-Alias Filter

The delta-sigma analog-to-digital converters have a built-in digital anti-aliasing low-pass filter that eliminates any signals with frequencies above the Nyquist frequency and assure that no aliasing occurs. The ADC digital low-pass filter is a linear phase digital filter exhibiting constant delay time vs. frequency (constant group delay). The table below shows the key characteristics of the ADC digital low-pass filter. Key characteristics of digital filters always track the selected data rate.

	EMX-4350 EMX-4380	**EMX-4250 EMX-4251**
Passband (<±0.001 dB ripple) (Hz)	0.424 x f_{DATA}	0.417 x f_{DATA}
-3dB Corner Frequency (Hz)	0.488 x f_{DATA}	0.424 x f_{DATA}
Stopband (<-90 dB attenuation) (Hz)	0.576 x f_{DATA}	0.583 x f_{DATA}
Group Delay (seconds)	28 / f_{DATA}	39 / f_{DATA}

Table 4-8: ADC Digital Low-pass Filter Characteristics

The digital low-pass filter is not effective at $f_{CLK} \pm 0.58*f_{DATA}$ so additional analog low-pass filtering is provided to eliminate signals with frequencies in these bands. The corner frequency of the analog anti-aliasing low-pass filter is fixed regardless of the selected data rate. The table below shows the critical frequencies for each data rate at which aliasing can occur and at which the analog low-pass filter needs to provide attenuation.

Model	Oversampling Ratio (f_{CLK} / f_{DATA})				Oversampling Clock (f_{CLK})
	32	64	128	256	
	Data Rate (f_{DATA}) - Samples per Second (Sa/s)				
EMX-4250 EMX-4251	4,118,282	4,156,293	4,175,299	▨	4,194,304 Hz
	6,434,816	6,494,208	6,523,904	▨	6,553,600 Hz
EMX-4350 EMX-4380	12,869,632	12,988,416	13,047,808	13,077,504	13,107,200 Hz
	16,473,129	16,625,172	16,701,194	16,739,205	16,777,216 Hz
	19,637,500	19,818,750	19,909,375	19,954,688	20,000,000 Hz

Table 4-9: Critical Frequencies for f_{DATA}

EMX-4350/4380 Anti-Aliasing

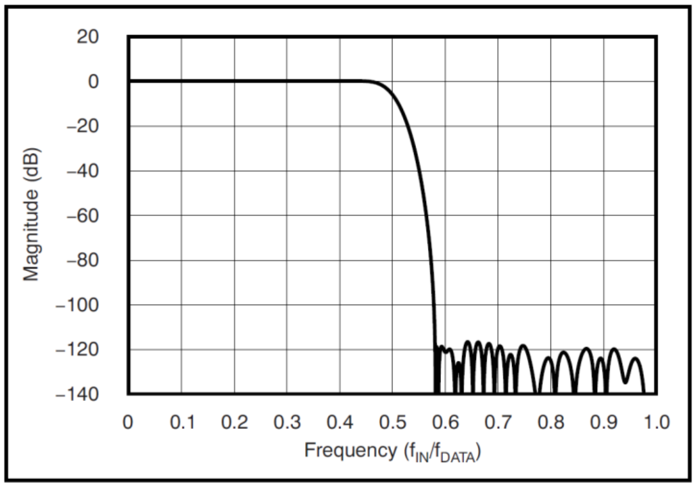


Figure 4-25: EMX-4350/4380 ADC Digital Filter Frequency Response

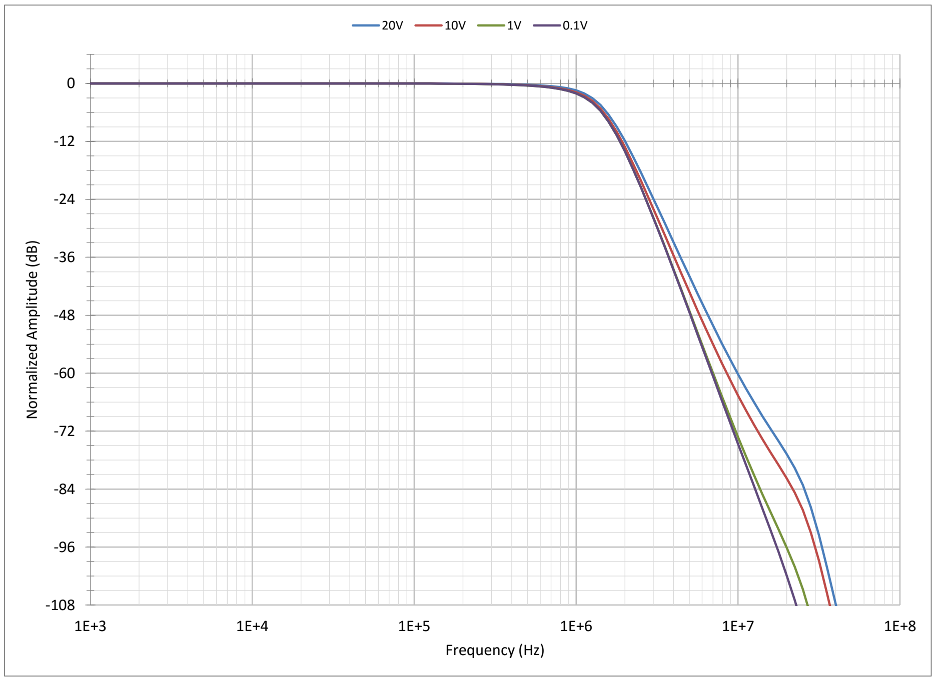


Figure 4-26: EMX-4350/4380 Voltage Amplitude Frequency Response

EMX-4250/4251 Anti-Aliasing

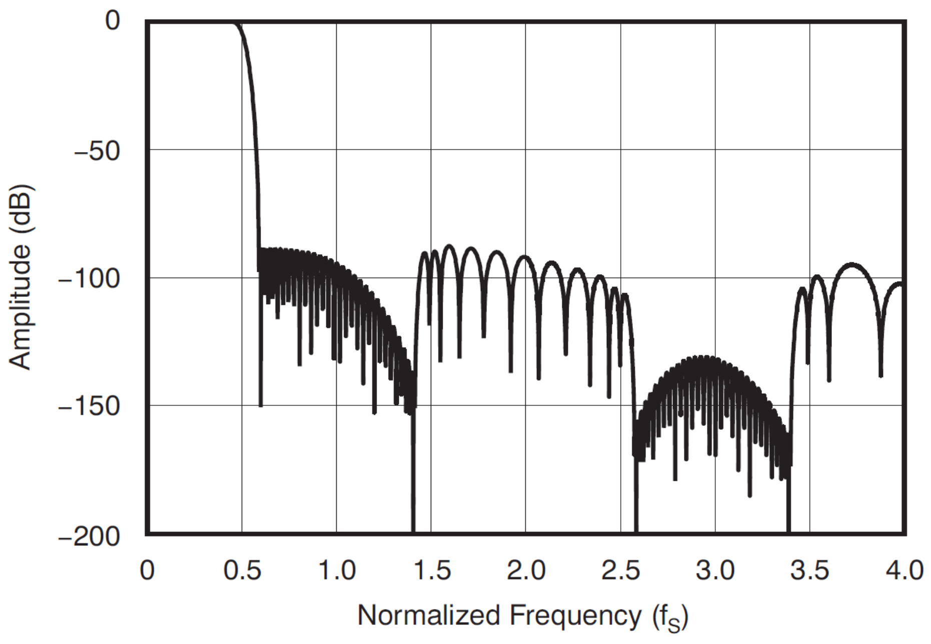


Figure 4-274: EMX-4250/4251 ADC Digital Filter Freq. Response for f_{DATA} ÷1

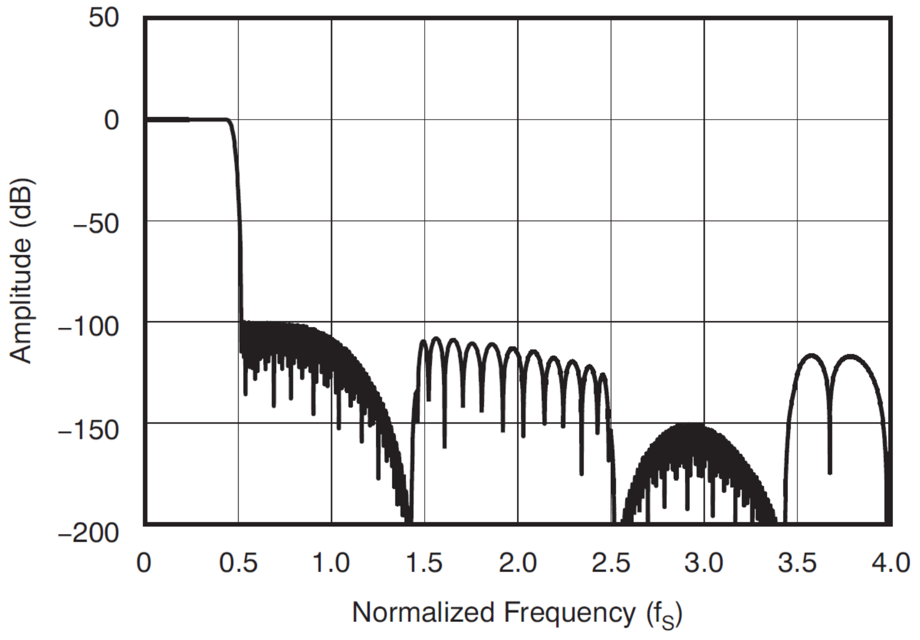


Figure 4-28: EMX-4250/4251 ADC Digital Filter Freq. Response for f_{DATA} ÷2 & ÷4

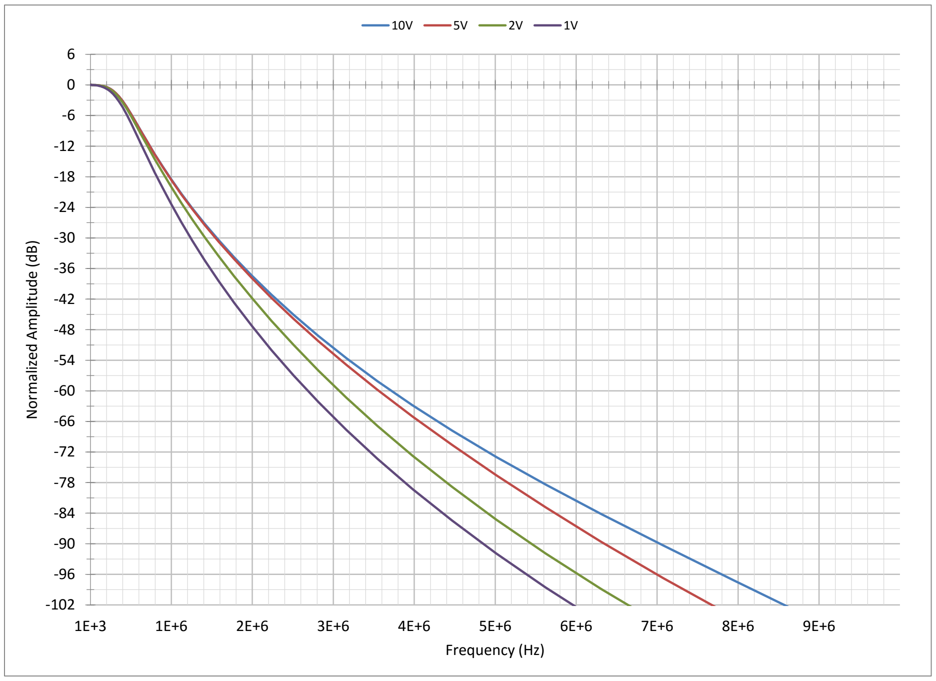


Figure 4-29: EMX-4250/4251 Voltage Amplitude Frequency Response

FLATNESS

EMX-4350/4380 Flatness Frequency Response

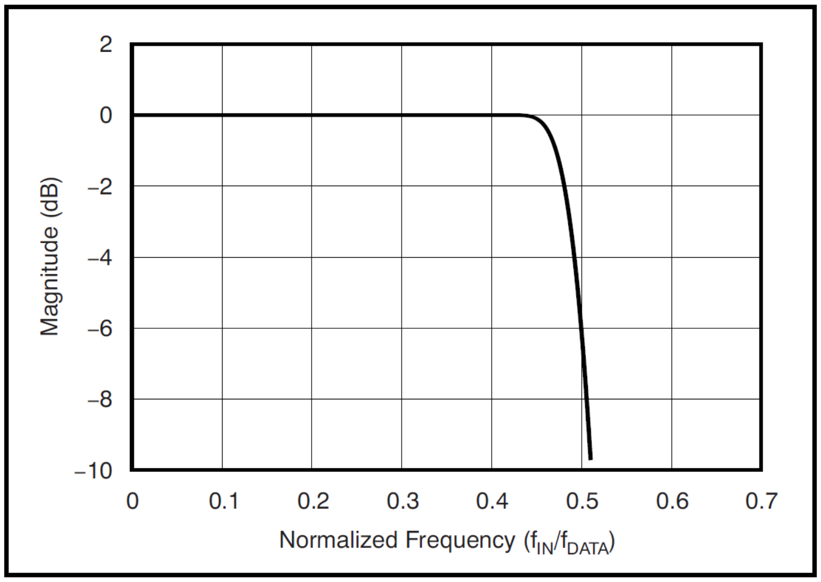


Figure 4-30: EMX-4350/4380 ADC Digital Filter Transition Band

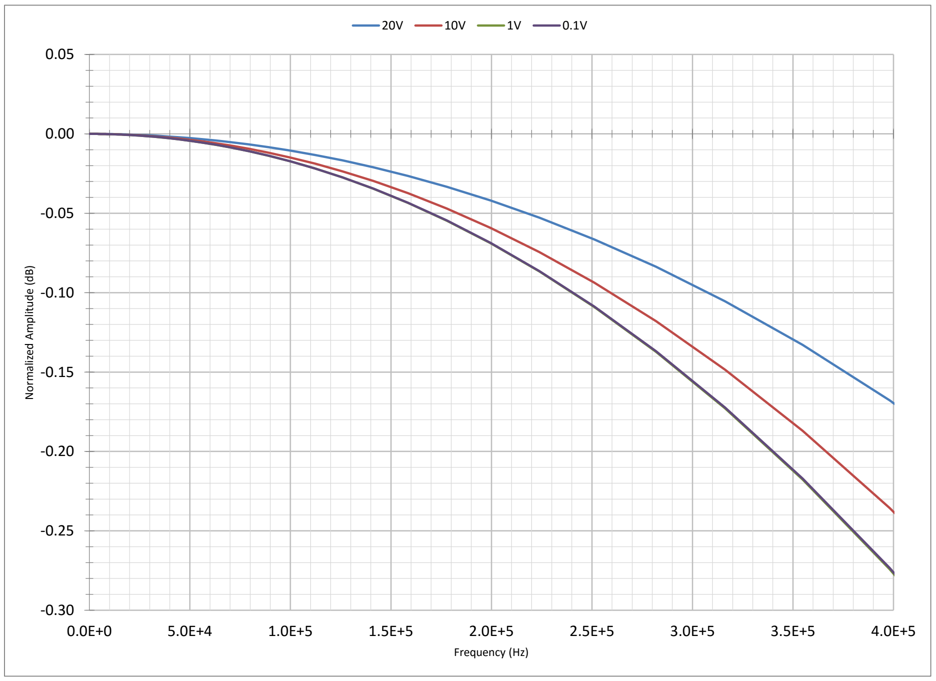


Figure 4-31: EMX-4350/4380 Voltage Flatness Frequency Response

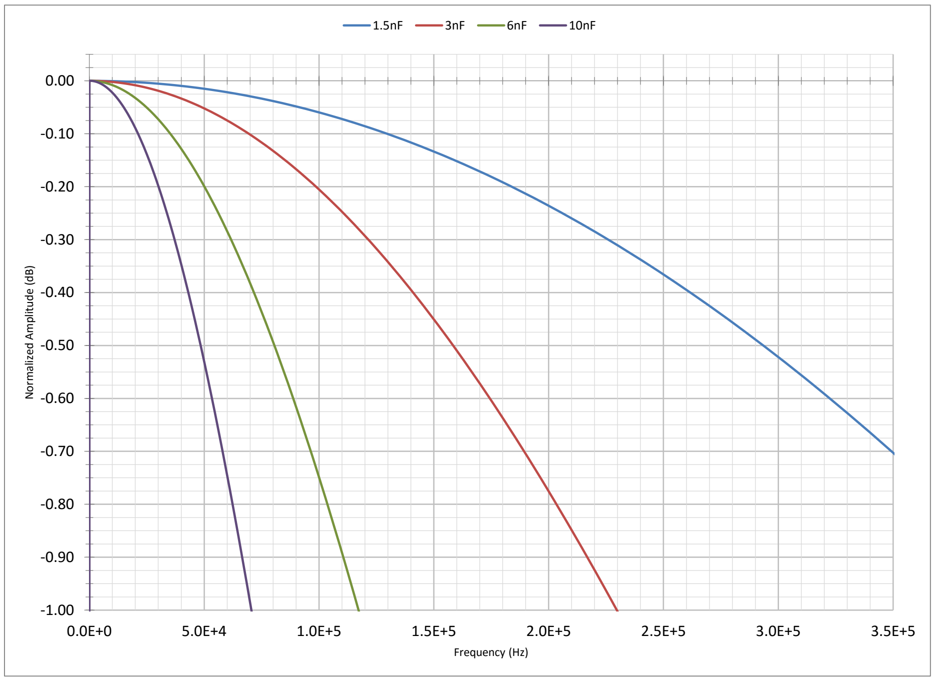


Figure 4-32: EMX-4380 Charge 10 kpC Range Flatness Frequency Response

EMX-4250/4251 Flatness Frequency Response

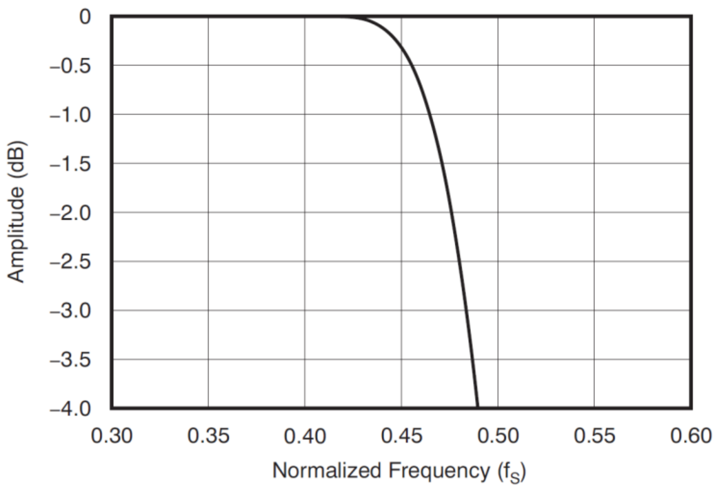


Figure 4-33: EMX-4250/4251 ADC Digital Filter Transition Band for f_{DATA} ÷2 and ÷4

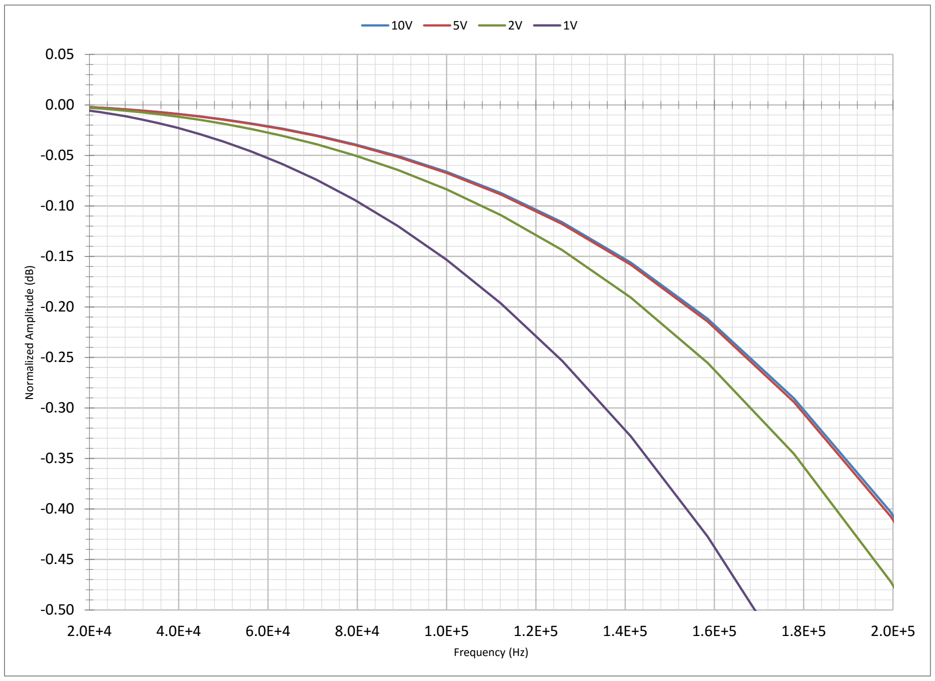


Figure 4-34: EMX-4250/4251 Voltage Flatness Frequency Response

GROUP DELAY

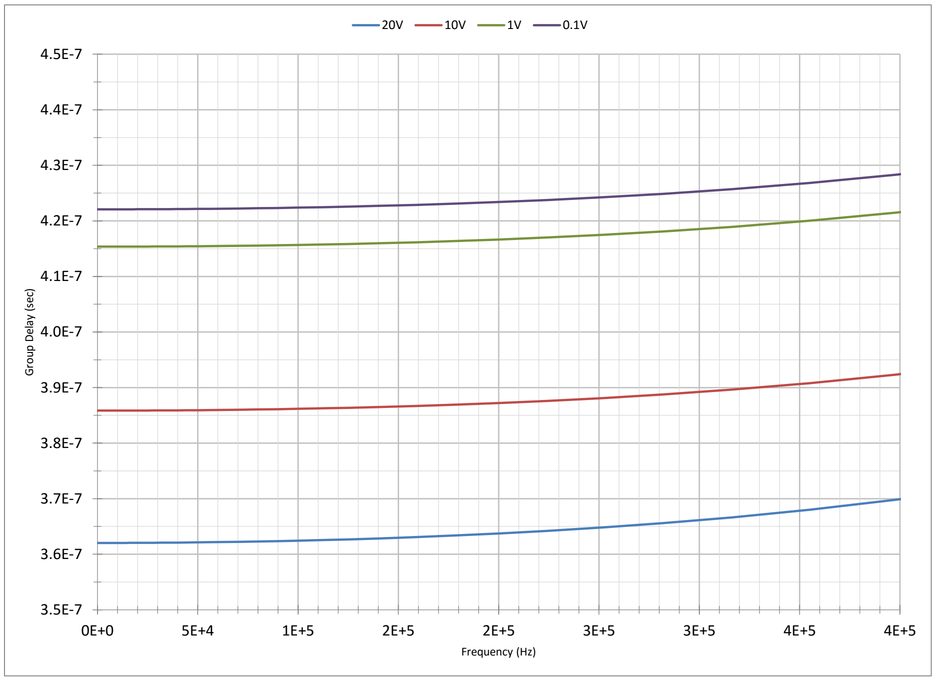


Figure 4-35: EMX-4350/4380 DC Coupling Voltage Group Delay Frequency Response

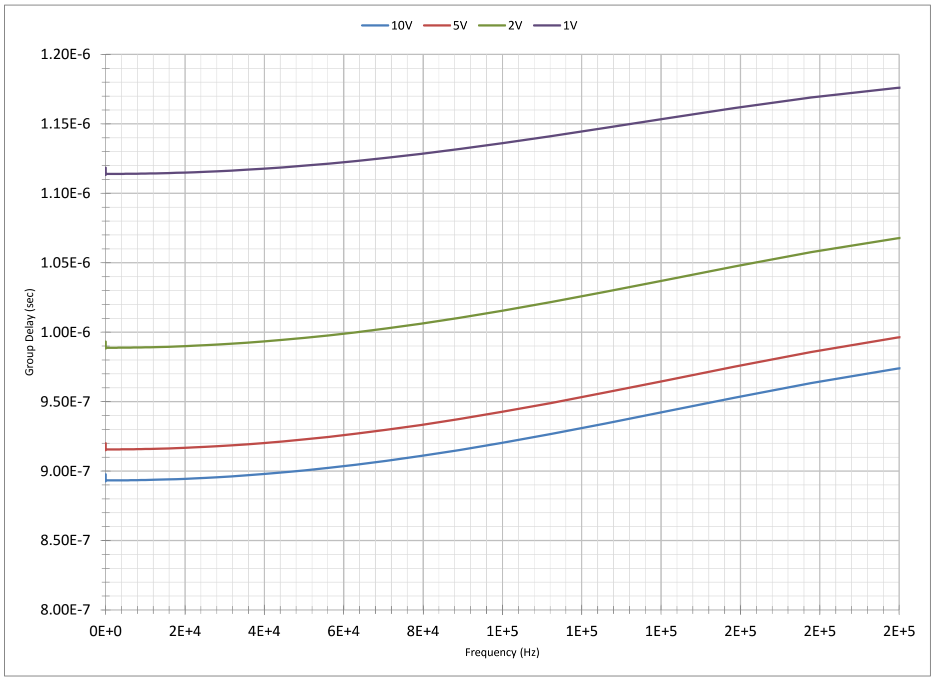


Figure 4-36: EMX-4250/4251 DC Coupling Voltage Group Delay Frequency Response

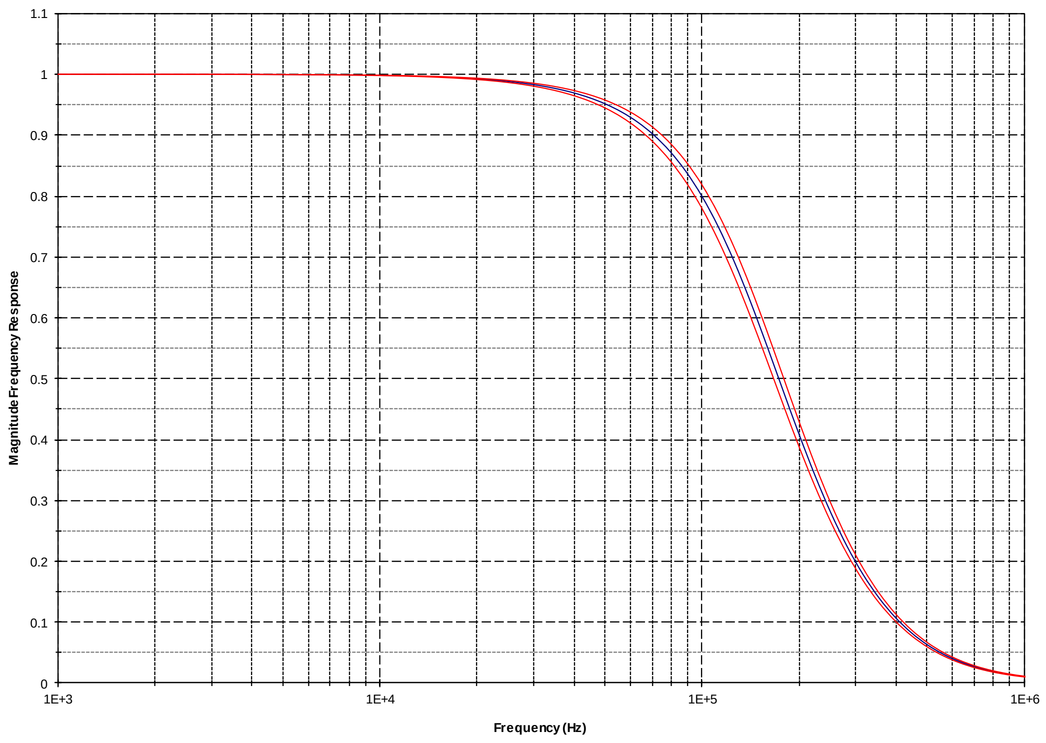


Figure 4-374: EMX-4016M Magnitude Frequency Response

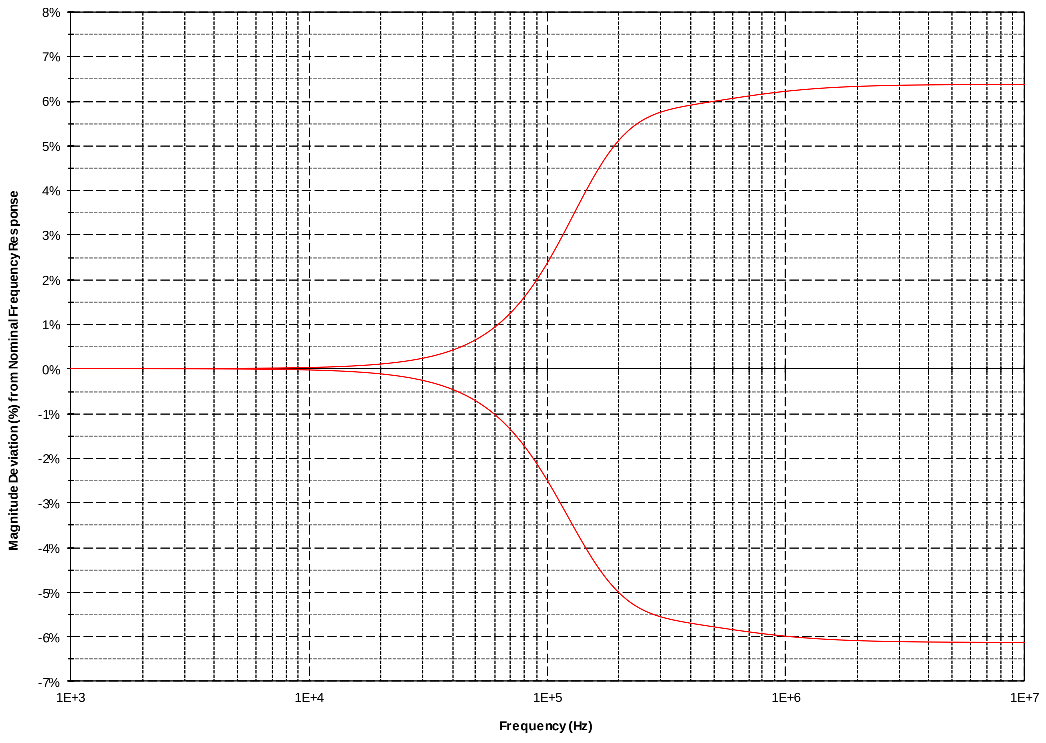


Figure 4-385: EMX-4016M Magnitude % Deviation from Nominal vs. Frequency

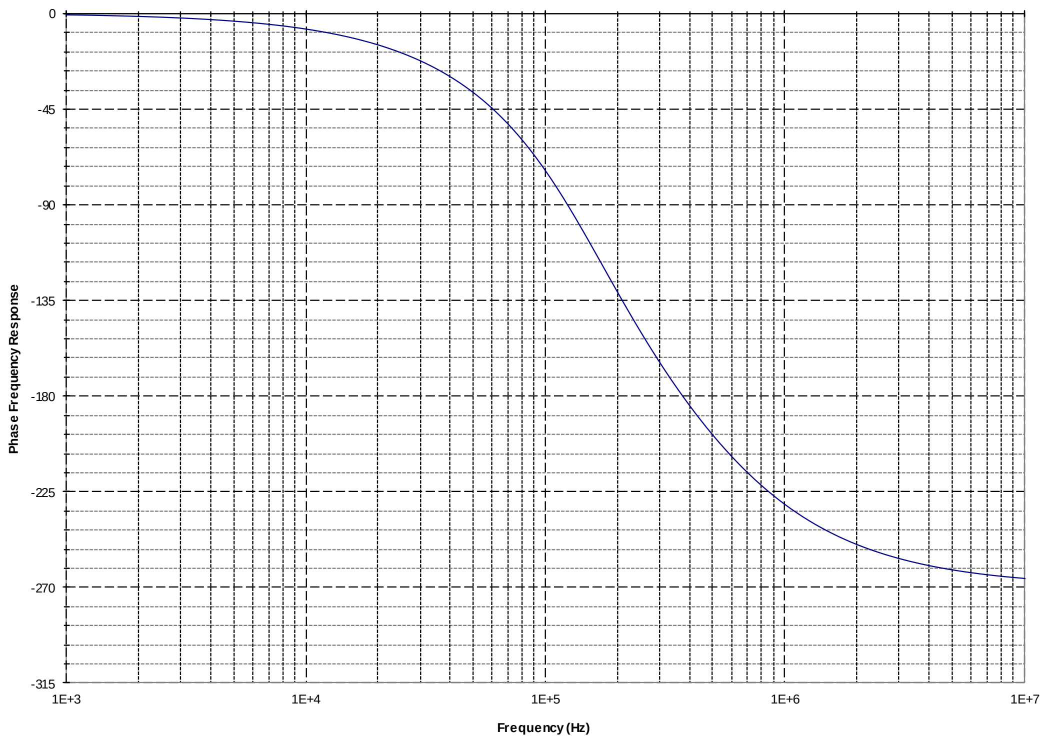


Figure 4-395: EMX-4016M Nominal Phase Frequency Response

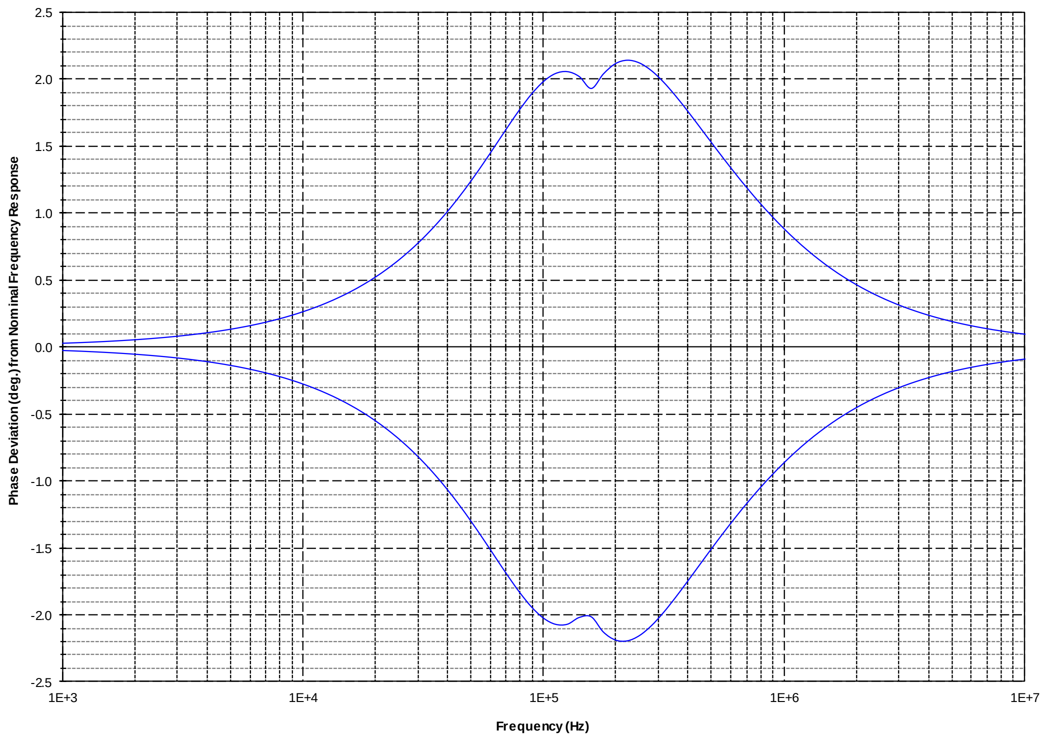


Figure 4-406: EMX-4016M Phase Deviation (Degrees) from Nominal vs. Frequency

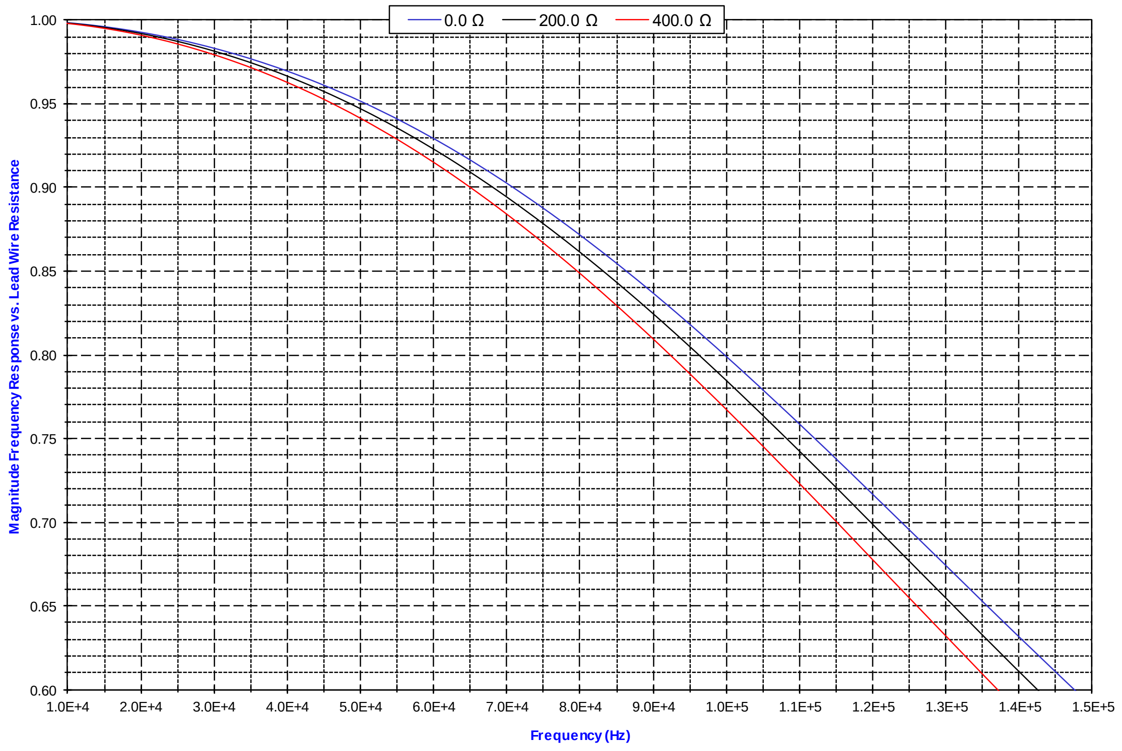


Figure 4-417: EMX-4016M Magnitude Frequency Response vs. Lead Wire Resistance

ACCESSORIES

BREAK-OUT-BOX (BOB)

VTI offers a series of Break-Out-Boxes (BOB) for use with the EMX-4250 and EMX-4251. There are 8-channel (EMX-4008), 16-channel (EMX-4016 & EMX-4116), and 32-channel (EMX-4032) versions. The BOBs convert the EMX-4250 micro-D Molex input connector into BNC input connectors. The shell of the BNC connectors are floating from chassis ground, hence taking advantage of the pseudo-differential inputs provided by the EMX-425X modules.

Fault LED indicators are also provided on the BOB for each BNC connector. The LEDs are powered and controlled by the EMX-425X digitizer modules. LEDs will be GREEN during normal operation and will turn RED if an OPEN or SHORT is detected by the digitizer module when the input is set to IEPE.

There are two BNC connectors in the rear panel: one for a trigger input and one for a calibration output. The calibration output is used to calibrate the built-in calibration signals used by the EMX-425X module to perform self-calibration. The trigger input is used for trigger the digitizer by an external signal.

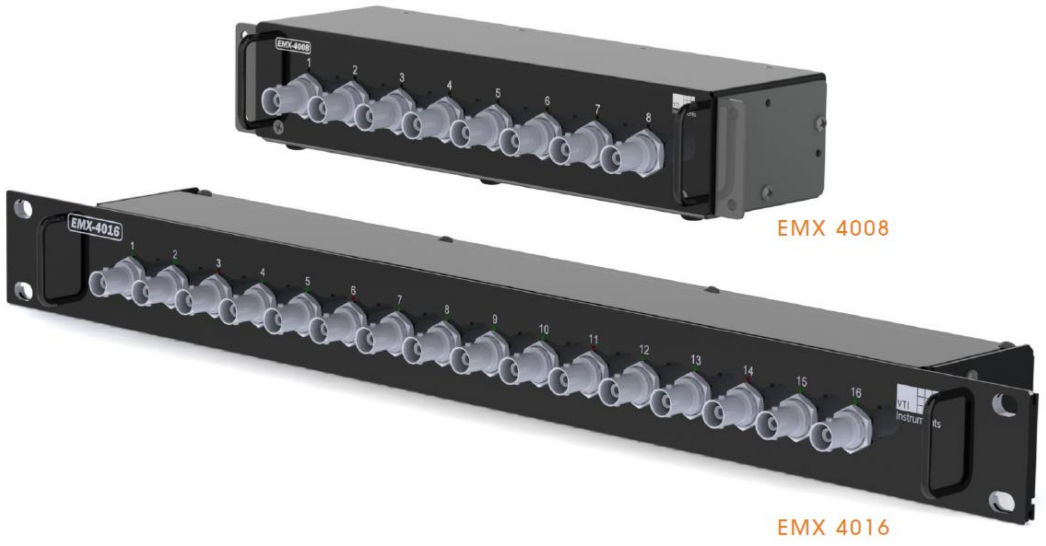


Figure 4-28: EMX-4016 & EMX-4008 Break-Out-Box

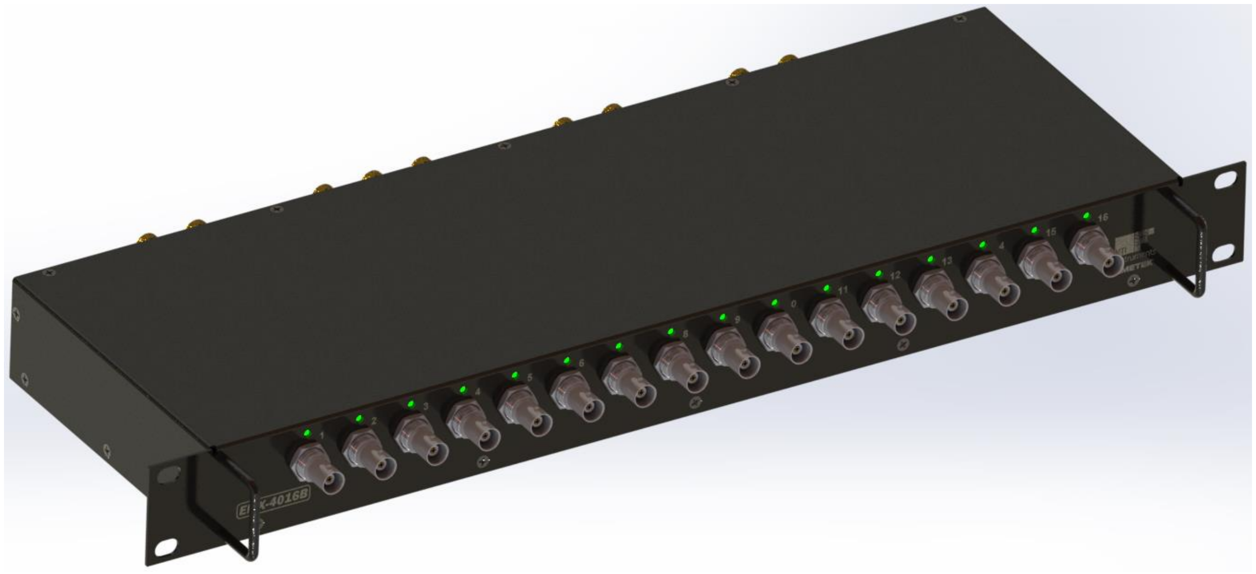


Figure 4-429: EMX-4016B Break-Out-Box

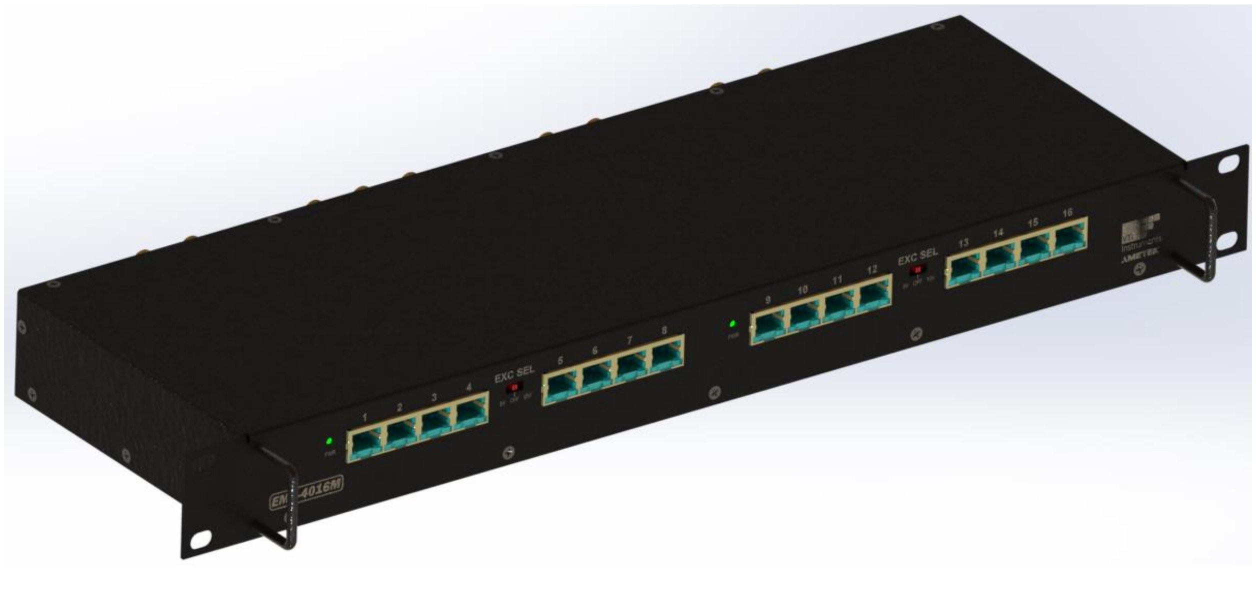


Figure 4-30: EMX-4016M Break-Out-Box

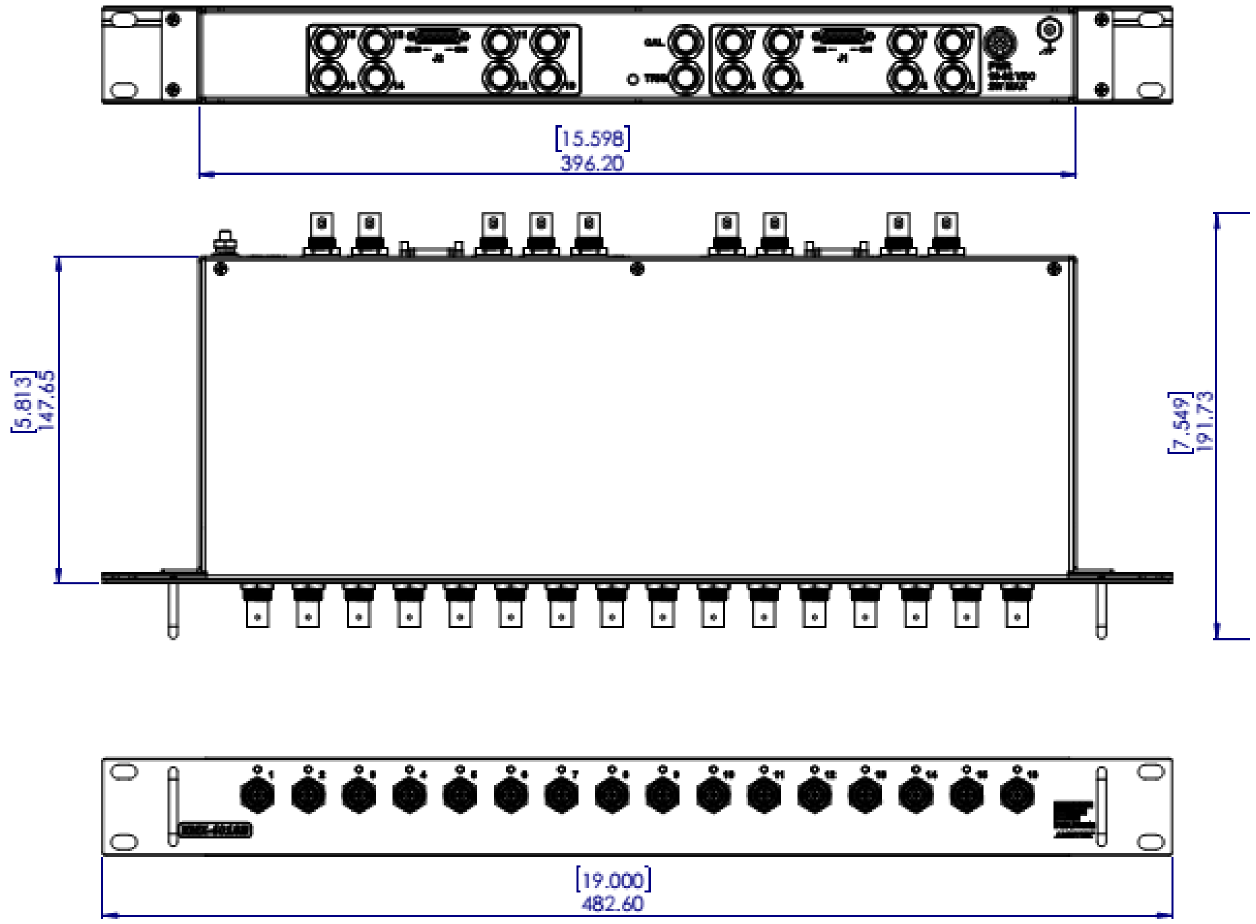
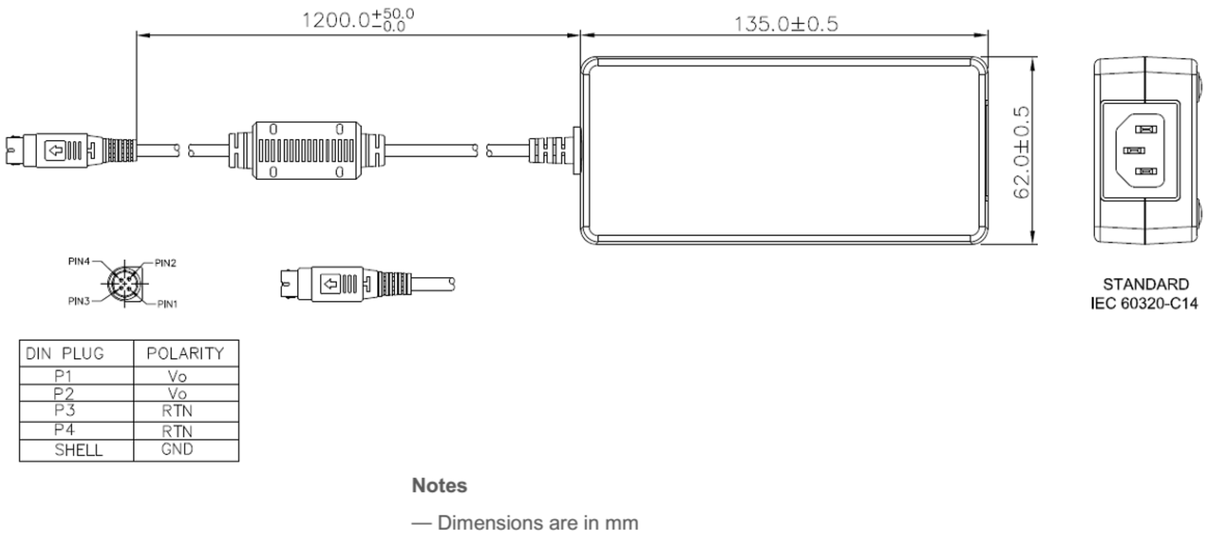


Figure 4-31: EMX4016B Outline Drawing



DIN PLUG	POLARITY
P1	Vo
P2	Vo
P3	RTN
P4	RTN
SHELL	GND

Figure 4-32:Drawing Represent

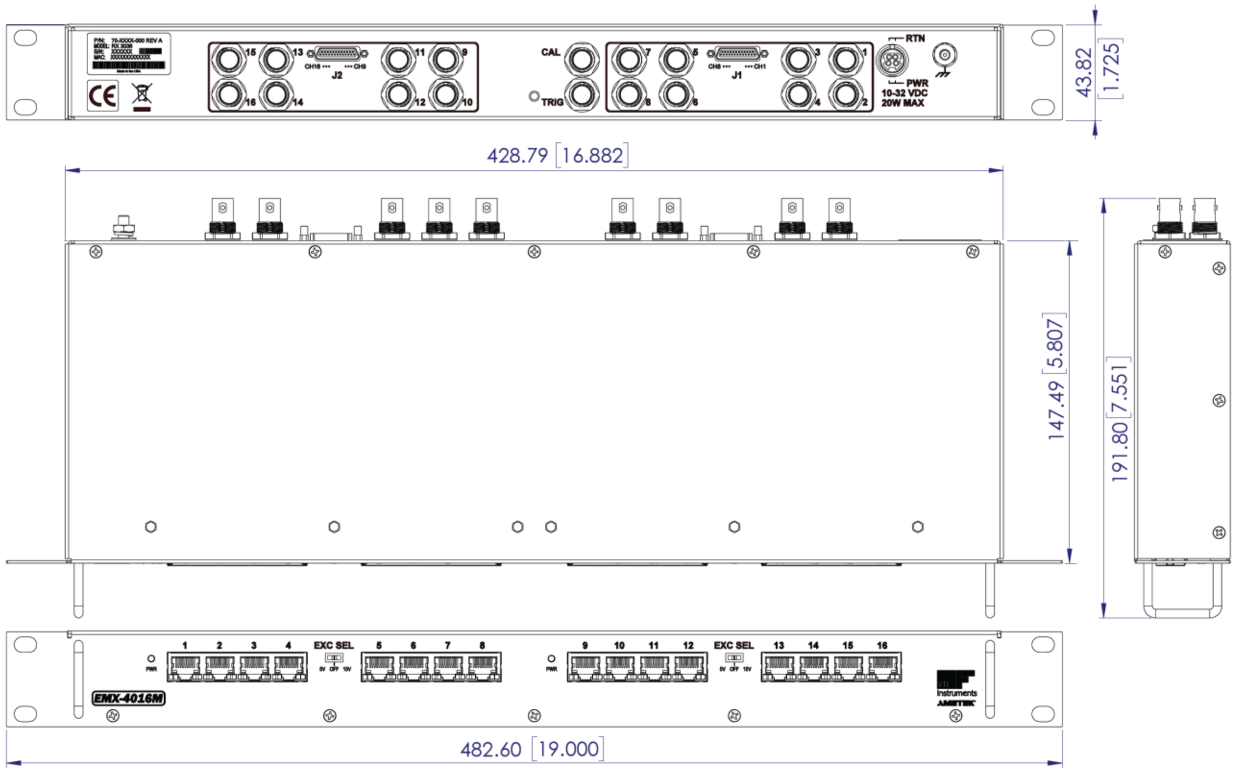


Figure 4-33: EMX4016M Outline Drawing

ORDERING INFORMATION

Model No.	Description	P/N
EMX-4250	Digitizer, 24-bit, 16Ch, 204.8 kSa/s IEPE/Volts	70-0409-004
EMX-4251	Digitizer, 24-bit, 8Ch, 204.8 kSa/s IEPE/Volts	70-0409-012
EMX-4350	Digitizer, 24-bit, 4Ch, 625.0 kSa/s IEPE/Volts	70-0409-004
EMX-4380	Digitizer, 24-bit, 4Ch, 625.0 kSa/s IEPE/Volts/Charge	70-0409-011
EMX-4016	Break-Out-Box (BOB), 16Ch, for EMX-4250/4251, 1U 19” Rack mount	70-0409-015
EMX-4016B	Break-Out-Box (BOB), 16Ch, w Buffer output ,for EMX-4250/4251, 1U 19” Rack mount	70-0409-215
EMX-4016M	Break-Out-Box (BOB), 16Ch, Bridge SC, w Buffer Out, for EMX-4250/4251, 1U 19” Rack mount	70-0409-315
EMX-4032	Break-Out-Box (BOB), 32Ch, for EMX-4250/4251, 1U 19” Rack mount	70-0409-016
EMX-4008	Break-Out-Box (BOB), 8Ch, for EMX-4250/4251, Table Top	70-0409-010
▨	Break-Out-Cable (BOC) Micro-D to 8Ch BNC, for EMX-4250/4251	53-0515-020
▨	Cable Assy. Micro-D to Micro-D for EMX-4250/4251 to BOB	53-0515-020
▨	Adapter 10-32 Female Microdot to BNC Male	27-0577-000
▨	Plug, 25-pin Female Micro-D, ITT Cannon MDSM-25SC-Z11-VS1	27-0295-025
EMX-4008V	8CH BOB Plugin with Fault LED	70-0409-018
BMX01	CHASSIS 1X BOB EMX-4008V	70-0409-019
BMX04	CHASSIS 4X BOB EMX-4008V	70-0409-020
BMX08	CHASSIS 8X BOB EMX-4008V	70-0409-021
EMX-4008	CHASSIS 8CH BREAK-OUT BOX, 1U	70-0409-010

APPENDIX A

DIGITAL ANTI-ALIAS FILTER AND MEASUREMENT SPAN

OVERVIEW

The EMX-4250/4350/4380 uses analog-to-digital converter (ADC) based on delta-sigma architecture. The analog signal is first filtered by an analog anti-aliasing filter at a fixed corner frequency (see *Anti-Alias Filter* in page 71) to attenuate signals with frequencies close to the delta-sigma converter oversampling clock frequency. The digitized signal is filtered by a digital FIR filter in the ADC chip for the highest span data. For lower span (sample rate) data, signal is further filtered by up to sixteen stages of digital decimation filters to avoid aliasing error.

The EMX-4250/4350/4380 implements ÷2 decimation filters along with an optional ÷5 decimation filter. These filters are linear phase FIR filter with very low pass-band ripple and high dynamic range even after sixteen stages of filtering.

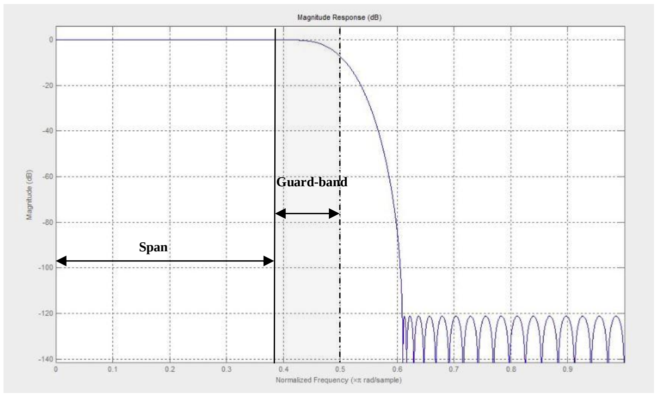


Figure A-5-1: ÷2 Decimation Filter

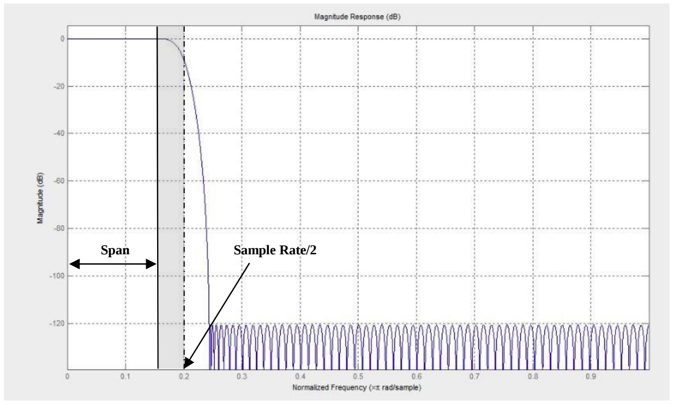


Figure A-5-2: ÷5 Decimation Filter

SAMPLE RATES AND NOMINAL SPANS

The sample rate is the data rate that user gets after decimation. While the Nyquist frequency is <sample rate>/2, the usable frequency range after the decimation filter, without alias error, is the nominal span. The nominal spans are calculated as <sample rate>/2.56. The frequency region above the pass-band (Span) can be alias contaminated. It is called guard-band.

Model	**EMX-4350/4380**						**EMX-4250/4251**			
Clock Freq	409600 Hz*		524288 Hz*		625000 Hz*		131072 Hz**		204800Hz**	
Top Span	160000***		204800***		244140.625***		51200***		80000***	
1/5	▨	32000	▨	40960	▨	48828.13	▨	10240	▨	16000
½- Stage1	80000	16000	102400	20480.00	122070.3	24414.06	25600	5120	40000	8000
Stage2	40000	8000	51200	10240.00	61035.16	12207.03	12800	2560	20000	4000
Stage3	20000	4000	25600	5120.00	30517.58	6103.52	6400	1280	10000	2000
Stage4	10000	2000	12800	2560.00	15258.79	3051.76	3200	640	5000	1000
Stage5	5000	1000	6400	1280.00	7629.40	1525.88	1600	320	2500	500
Stage6	2500	500	3200	640.00	3814.70	762.94	800	160	1250	250
Stage7	1250	250	1600	320.00	1907.34	381.47	400	80	625	125
Stage8	625	125	800	160.00	953.67	190.73	200	40	312.5	62.5
Stage9	312.5	62.5	400	80.00	476.83	95.37	100	20	156.25	31.25
Stage10	156.25	31.25	200.00	40.00	238.42	47.68	50.00	10.00	78.13	15.63
Stage11	78.13	15.63	100.00	20.00	119.21	23.84	25.00	5.00	39.06	7.81
Stage12	39.06	7.81	50.00	10.00	59.60	11.92	12.50	2.50	19.53	3.91
Stage13	19.53	3.91	25.00	5.00	29.80	5.96	6.25	1.25	9.77	1.95
Stage14	9.77	1.95	12.50	2.50	14.90	2.98	3.13	0.63	4.88	0.98
Stage15	4.88	0.98	6.25	1.25	7.45	1.49	1.56	0.31	2.44	0.49
Stage16	2.44	0.49	3.13	0.63	0.37	0.07	0.78	0.16	1.22	0.24

**EMX-4350/4380 clock frequencies can be divided by 2, 4, or 8 to obtain lower spans.*
***EMX-4250 clock frequencies can be divided by 2 or 4 to obtain lower spans*
****While the programming API defines nominal span = sample rate/2.56(0.39fs), the actual pass-band at the top span is 0.424 fs for 4350/4380, 0.4545fs for 4250/51.*

Table A-1: Nominal Span (Sample Rate / 2.56)

GROUP DELAYS

With EMX-4250/51, a filter in A/D converter introduces 39 samples group delay, and with for EMX-4350/4380 it is 28 samples. This delay is about 45 µs at 625000 Hz.

For lower decimated spans, at each stage of 1/2 decimation filter adds 31.5 samples group delay, and 1/5 decimation filter adds 73 samples delay. The total group delay is shown in the table below.

Model	**EMX-4350/4380**						**EMX-4250/4251**			
Clock Freq	409600 Hz		524288 Hz		625000 Hz		131072 Hz		204800 Hz	
Top Span	6.83594E-05		5.34058E-05		0.0000448		0.000297546		0.00019043	
1/5		0.00025		0.00019		0.00016		0.00085		0.00055
½- Stage1	0.000145	0.00063	0.000113	0.00049	0.000095	0.00041	0.00054	0.0020	0.000344	0.00132
Stage2	0.000299	0.00140	0.00023	0.0011	0.000196	0.00092	0.00102	0.0045	0.000652	0.00285
Stage3	0.000607	0.00294	0.000474	0.00230	0.000398	0.0019	0.00198	0.0093	0.00127	0.00593
Stage4	0.00122	0.00601	0.000955	0.0047	0.000800	0.0039	0.0039	0.019	0.0025	0.012
Stage5	0.00245	0.0122	0.00192	0.0095	0.00161	0.0080	0.0078	0.038	0.0050	0.0244
Stage6	0.00491	0.0245	0.00384	0.0191	0.00322	0.0160	0.0154	0.0765	0.0099	0.0490
Stage7	0.00984	0.0491	0.00768	0.0383	0.00645	0.0322	0.031	0.153	0.0197	0.0982
Stage8	0.0197	0.0983	0.0154	0.0768	0.0129	0.0644	0.0616	0.307	0.0394	0.197
Stage9	0.0394	0.197	0.0308	0.154	0.0258	0.1289	0.123	0.615	0.0788	0.394
Stage10	0.0787	0.394	0.0615	0.308	0.0516	0.2580	0.246	1.230	0.158	0.787
Stage11	0.157	0.787	0.123	0.615	0.103	0.5160	0.492	2.460	0.315	1.575
Stage12	0.315	1.575	0.246	1.230	0.206	1.0321	0.984	4.922	0.630	3.150
Stage13	0.630	3.15	0.492	2.461	0.413	2.0643	1.969	9.843	1.260	6.300
Stage14	1.26	6.30	0.984	4.922	0.826	4.1287	3.938	19.687	2.520	12.60
Stage15	2.52	12.6	1.969	9.844	1.652	8.2574	7.875	39.375	5.040	25.20
Stage16	5.04	25.2	3.937	19.69	3.303	16.515	15.75	78.75	10.08	50.40

Table A-2: Group Delay (Seconds)

APPENDIX B

PHASE MEASUREMENT AND CORRECTION

OVERVIEW

The Fourier transform of the time domain signal acquired by EMX-4250/4350/4380 gives the amplitude and phase of each frequency components. The amplitude indicates the magnitude of the phenomena and the phase represents the time shift. When the instrument acquire signal, the analog signal goes through multiple stages of analog and digital filters before the data reaches to the user's application. Each filter adds a certain time delay to the signal.

When you are trying to measure the time difference between two signals of the same frequency, if both signal goes through the same analog and digital filters, then the amount of time shift can be considered to be equal, since digital filters can achieve the perfect phase match. In this case, the phase difference between two signal does not require any special correction.

On the other hand, if one of signal does not go through these filters, the phase of the other signals must be corrected for the delay added in each filtering stages. This is the case when the user triggers the measurement with a phase reference signal, such as TDC (Top Dead Center) pulse. The trigger signal is directly detected at the trigger detection circuit for the trigger inputs, while all the analog signals go into ADC channels and they are filtered to the desired frequency span. Thus, delay is introduced. Fortunately, our digital filters are all linear phase FIR filters that adds a constant time shift independent of the signal frequency, so they can be corrected mathematically. The timestamps associated to the data samples are already corrected for delays introduced by analog and digital anti-aliasing filters. However, there are other phase corrections that the users have to consider, depending on the test setup and the accuracy requirement.

SUB-SAMPLE TRIGGER DELAY

When the measurement is triggered by a trigger signal sent directly to the front panel trigger connector, or backplane trigger line, the trigger event occurs asynchronous to the ADC sampling clock. This trigger event time is measured by the timestamp clock and recorded, and returned to the user. This information can be obtained by parsing *AdditionalData* string returns at the *Measurement.Read* (or *MemoryRead* for streaming) method. The *Read* method also returns the timestamp of the first sample in the data record. The difference between the trigger timestamp and the data timestamp can be used to correct for more accurate phase measurement.

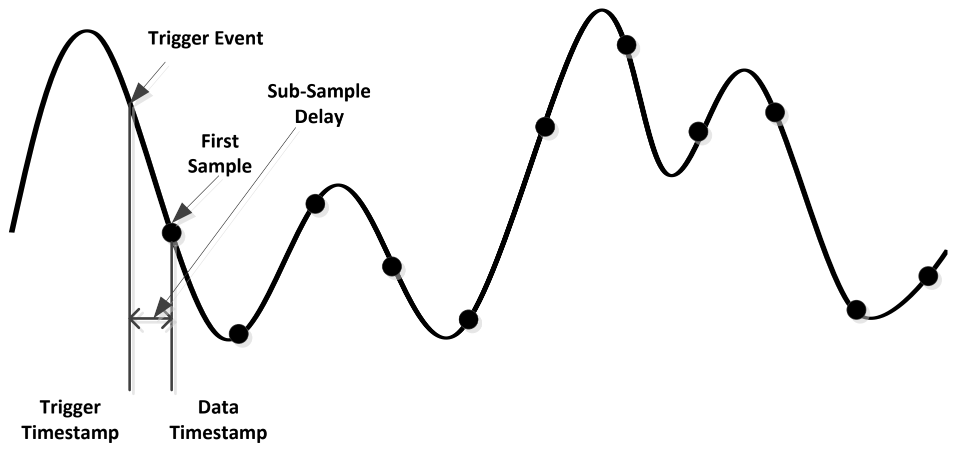


Figure B-6-1: Sub-sample Trigger Delay

AC Coupling Filter

When the analog signal is measured in AC coupling mode, the analog AC coupling, high-pass filter adds non-linear phase. This delay can be significant, and the amount of delay is frequency dependent. See Module Information section for the AC coupling filter phase performance.

Transducer Phase Delay

Additional delay can be introduced by the transducer being used, depending on the transducer architecture. This may also needed to be considered depending on the measurement accuracy requirement. This is beyond the scope of this manual. Contact transducer manufacturer for more information.

APPENDIX C

TEDS

OVERVIEW

TEDS (Transducer Electronic Data Sheets) is a smart transducer standard defined as IEEE 1451.4. The TEDS is a memory device attached to the IEEE 1451.4 compatible transducer that stores the information of the smart transducer including the transducer identification, sensitivity, calibration, and manufacturer information.

The IEEE 1451.4 standard defines two classes of interfaces. The class 1 sensors (IEPE) uses coax connector and the analog and the digital (TEDS) are on the same line. The class 2 is used in bridge sensor and the analog and the digital lines are separate.

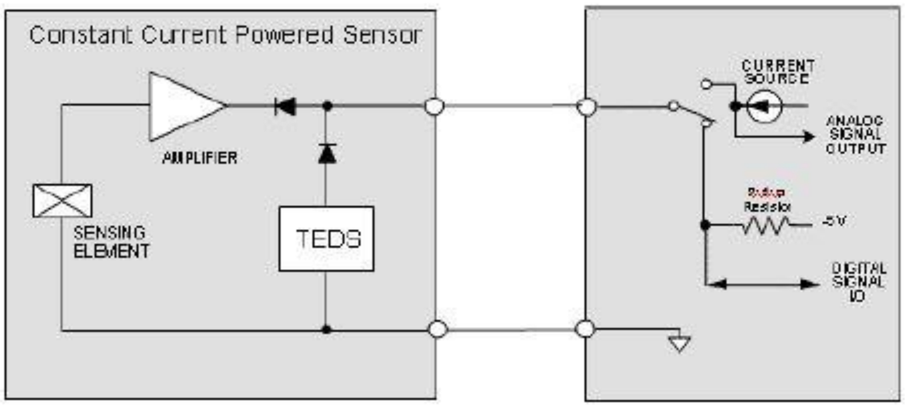


Figure C-7-1: IEEE 1451.4 Class 1 (IEPE)

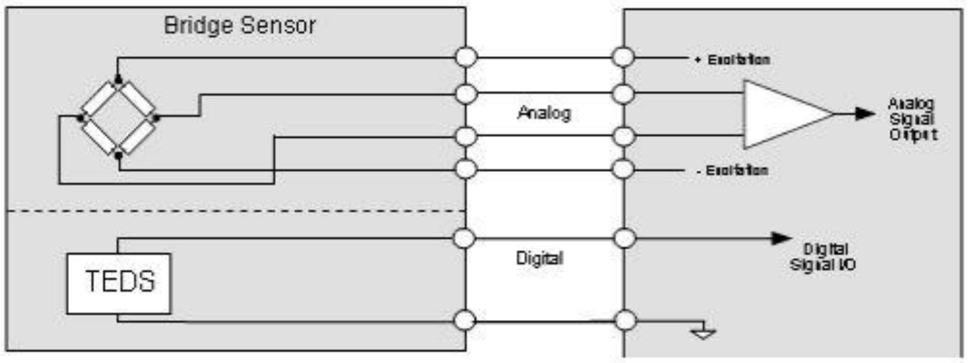


Figure C-7-2: IEEE 1451.4 Class 2

TEDS data is stored in EEPROM. The first 64-bit is "Basic" TEDS data followed by the transducer specific data. The transducer specific information must be parsed using the template provided by the transducer manufacturer.

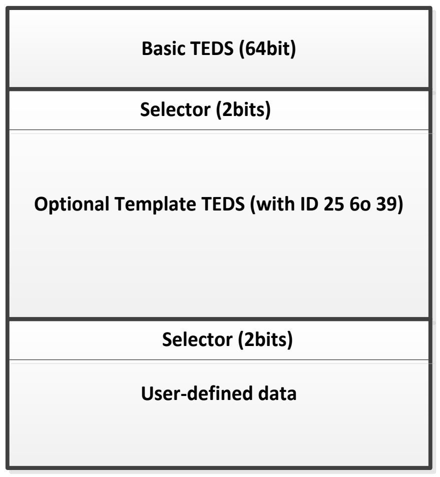


Figure C-7-3: TEDS Data Structure

The Basic TEDS fields are defined by IEEE 1451.4 specification.

Field	Content and Range	Size
Manufacturer ID	Number (17-16381)	14 bits
Model Number	Number (0-32767)	15 bits
Version Letter	Character (A-Z)	5 bits
Version Number	Number (0-63)	6 bits
Serial Number	Number (0-16777215)	24 bits

Table C-1: Basic TEDS

The TEDS data bit stream is usually 256-bit including 8-bit checksum to ensure the correct data transfer.

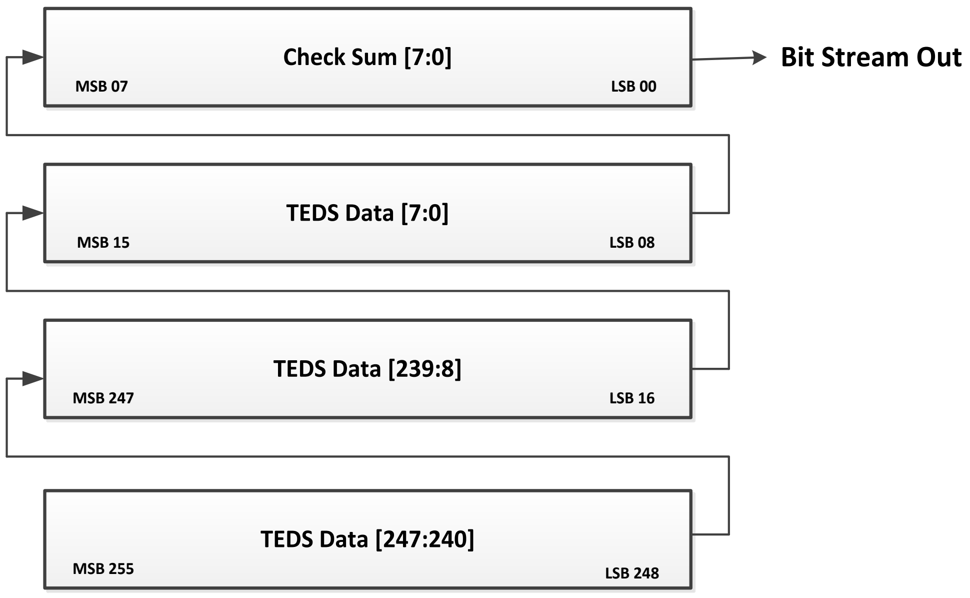


Figure C-7-4: TEDS Bit Stream

MICROLAN (MLAN)

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 instrument has a 1-wire bus master. Under typical operation on the instrument, 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. 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 instrument, or wait a significant amount of time between commands.*** VTI Instruments 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

The Digitizer and DSA drivers provide access to TEDS on a per-channel basis. First, TEDS must be enabled on the desired channel by setting the *Channel.TEDS.Enabled* property to True. Only one channel per card can have TEDS enabled at a time. All other channels must be set to False. After this occurs, the *Channel.TEDS* interface's *ReadURN()*, *WriteTEDS()*, and *ReadTEDS()* functions can be used. The *WriteTEDS()* and *ReadTEDS()* functions allow the EMX-4250/4350/4380 card to act as an MLAN Repeater (see Maxim APPLICATION NOTE 2966 for the MLAN Repeater specification). Use *WriteTEDS()* to send MLAN command strings and use *ReadTEDS()* to read the response. *ReadURN()* is a convenience function that implements the MLAN commands for identifying the first TEDS device on the 1-wire bus. This allows simple applications that only have to read the URN of a single device for each channel to function without having to construct the MLAN command for discovery and identification every time.

Below, example code for each function of the 2430 and 2431 is discussed. The code example shown here is for the purpose of discussion. Up-to-date example codes are installed with the Digitizer and DSA driver.

First, 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

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(string name)
{

       CComSafeArray<short> WriteBuf;
       WriteBuf.Create();

       WriteBuf.Add((short)0); // reserve first byte for length
       // clear the search state so we find the 'first' device
       WriteBuf.Add(CMD_RESET);
       WriteBuf.Add(CMD_GETBUF);
       // set the length
       WriteBuf[0] = (short)WriteBuf.GetCount() - 1;

       // send the commands
       digitizer->Channels->Item[name]->TEDS-
>WriteTEDS(WriteBuf.GetSafeArrayPtr());

       // retrieve the response
       CComSafeArray<short> ReadBuf;
       SAFEARRAY* psaReadBuf;
       digitizer->Channels->Item[name]->TEDS-
>ReadTEDS(&psaReadBuf);
}
```

This function performs an MLAN bus master reset, that is, it resets the MLAN repeater inside the instrument. 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(string name)
{

       CComSafeArray<short> WriteBuf;
       WriteBuf.Create();

       WriteBuf.Add((short)0); // reserve first byte for length
       // clear the search state so we find the 'first' device
       WriteBuf.Add(CMD_RESET);
       // do a reset, search and then read results
       WriteBuf.Add(CMD_ML_RESET);
       WriteBuf.Add(CMD_ML_SEARCH);
       WriteBuf.Add((short)DATA_ID);
       WriteBuf.Add((short)0);
       // request the result buffer as the last command
       WriteBuf.Add(CMD_GETBUF);
       // set the length
       WriteBuf[0] = (short)WriteBuf.GetCount() - 1;
```

```
        // send the commands
        digitizer->Channels->Item[name]->TEDS-
>WriteTEDS(WriteBuf.GetSafeArrayPtr());
        std::wcout << "Read URN (WriteData)" << std::endl;
        PrintPacket(WriteBuf);

        // retrieve the response for each command
        CComSafeArray<short> ReadBuf;
        SAFEARRAY* psaReadBuf;
        digitizer->Channels->Item[name]->TEDS-
>ReadTEDS(&psaReadBuf);
        std::wcout << "Read URN (ReadData)" << std::endl;
        ReadBuf.Attach(psaReadBuf);
        PrintPacket(ReadBuf);
}
```

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 *ReadURN* method of the Digitizer driver. It is included for clarity and as one of the simpler examples of MLAN programming.

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(IVTEXDigitizerPtr digitizer, const
char* data)
{
    int recLen = 0;
    CComSafeArray<short> WriteBuf;
    WriteBuf.Create();
    if(strlen(data) != ((DS2430_SCRATCHPAD_LEN * 2) +
(DS2430_SCRATCHPAD_LEN-1)))
    {
        //(SCRATCHPAD_LEN*2)+(SCRATCHPAD_LEN-1) = 95
        printf("Data was not the right length (wanted 95, got
%i)\n", (int)strlen(data));
        return -1;
    }

    WriteBuf.Add((short)0); // reserve first byte for length
    // access the current device with address in DATA_ID
    WriteBuf.Add(CMD_ML_ACCESS);
    // construct a block of communication to MicroLAN
    WriteBuf.Add(CMD_ML_DATA);
    WriteBuf.Add(3+DS2430_SCRATCHPAD_LEN); // block length
    WriteBuf.Add(2+DS2430_SCRATCHPAD_LEN); // data length
    // send the write scratchpad command
    WriteBuf.Add(WRITE_SCRATCHPAD);
    // send the address byte
    WriteBuf.Add((short)0);
    // the bytes of data to write
    char byte[3];
    for(int i = 0; i < ((2*DS2430_SCRATCHPAD_LEN) +
(DS2430_SCRATCHPAD_LEN-1)); i+=3)
    {
        strncpy_s(byte, 3, &data[i], 2);
        byte[2]='\0';
        WriteBuf.Add((unsigned char)strtoul(byte, NULL, 16));
//convert to hex
    }
    // request the result buffer as the last command
    WriteBuf.Add(CMD_GETBUF);
    // set the length
    WriteBuf[0] = (short)WriteBuf.GetCount() - 1;

    // send the commands
    digitizer->Channels->Item[name]->TEDS-
>WriteTEDS(WriteBuf.GetSafeArrayPtr());
    std::cout << "DS2430 Write Scratchpad (WriteData)" <<
std::endl;
    PrintPacket(WriteBuf);

    // retrieve the response
    CComSafeArray<short> ReadBuf;
    SAFEARRAY* psaReadBuf;
    digitizer->Channels->Item[name]->TEDS->ReadTEDS(&psaReadBuf);
    std::cout << "DS2430 Write Scratchpad (ReadData)" <<
std::endl;
    ReadBuf.Attach(psaReadBuf);
    PrintPacket(ReadBuf);

    recLen = (int)ReadBuf.GetCount();
    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(IVTEXDigitizerPtr digitizer)
{
     int recLen = 0;
     CComSafeArray<short> WriteBuf;
     WriteBuf.Create();

     WriteBuf.Add((short)0); // reserve first byte for length
     // access the current device with address in DATA_ID
     WriteBuf.Add(CMD_ML_ACCESS);
     // construct a block of communication to MicroLAN
     WriteBuf.Add(CMD_ML_DATA);
     WriteBuf.Add(3); // block length
     WriteBuf.Add(2+DS2430_SCRATCHPAD_LEN); // data length of read
     // send the read scratchpad command
     WriteBuf.Add(READ_SCRATCHPAD);
     // send the address byte
     WriteBuf.Add((short)0);
     // request the result buffer as the last command
     WriteBuf.Add(CMD_GETBUF);
     // set the length
     WriteBuf[0] = (short)WriteBuf.GetCount() - 1;

     // send the commands
     digitizer->Channels->Item[name]->TEDS-
>WriteTEDS(WriteBuf.GetSafeArrayPtr());
     std::cout << "DS2430 Read Scratchpad (WriteData)" <<
std::endl;
     PrintPacket(WriteBuf);

     // retrieve the response
     CComSafeArray<short> ReadBuf;
     SAFEARRAY* psaReadBuf;
     digitizer->Channels->Item[name]->TEDS->ReadTEDS(&psaReadBuf);
     std::cout << "DS2430 Read Scratchpad (ReadData)" <<
std::endl;
     ReadBuf.Attach(psaReadBuf);
     PrintPacket(ReadBuf);

     recLen = (int)ReadBuf.GetCount();
     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(IVTEXDigitizerPtr digitizer)
{
    int recLen = 0;
    CComSafeArray<short> WriteBuf;
    WriteBuf.Create();

    WriteBuf.Add((short)0); // reserve first byte for length
    // access the current device with address in DATA_ID
    WriteBuf.Add(CMD_ML_ACCESS);
    // construct a block of communication to MicroLAN
    WriteBuf.Add(CMD_ML_DATA);
    WriteBuf.Add(3); // block length
    WriteBuf.Add(2); // data length
    // send the copy scratchpad command
    WriteBuf.Add(COPY_SCRATCHPAD);
    // send the validation key
    WriteBuf.Add(0xA5);
    // delay for 128ms
    WriteBuf.Add(CMD_DELAY);
    WriteBuf.Add((short)0);
    WriteBuf.Add(DELAY_128 | DELAY_MS);
    // set the length
    WriteBuf[0] = (short)WriteBuf.GetCount() - 1;

    // send the commands
    digitizer->Channels->Item[name]->TEDS-
>WriteTEDS(WriteBuf.GetSafeArrayPtr());
    std::cout << "DS2430 Copy Scratchpad (WriteData)" <<
std::endl;
    PrintPacket(WriteBuf);

    // retrieve the response
    CComSafeArray<short> ReadBuf;
    SAFEARRAY* psaReadBuf;
    digitizer->Channels->Item[name]->TEDS->ReadTEDS(&psaReadBuf);
    std::cout << "DS2430 Copy Scratchpad (ReadData)" <<
std::endl;
```

```
    ReadBuf.Attach(psaReadBuf);
    PrintPacket(ReadBuf);

    recLen = (int)ReadBuf.GetCount();
    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(IVTEXDigitizerPtr digitizer)
{
    int recLen = 0;
    CComSafeArray<short> WriteBuf;
    WriteBuf.Create();

    WriteBuf.Add((short)0); // reserve first byte for length
    // access the current device with address in DATA_ID
    WriteBuf.Add(CMD_ML_ACCESS);
    // construct a block of communication to MicroLAN
    WriteBuf.Add(CMD_ML_DATA);
    WriteBuf.Add(3); // block length
    WriteBuf.Add(2+DS2430_MEMORY_LEN); // data length with 32
bytes of reads
    // send the read memory command
    WriteBuf.Add(READ_MEMORY);
    // send the address byte
    WriteBuf.Add((short)0);
    // request the result buffer as the last command
    WriteBuf.Add(CMD_GETBUF);
    // set the length
    WriteBuf[0] = (short)WriteBuf.GetCount() - 1;
    // send the commands
    digitizer->Channels->Item[name]->TEDS-
>WriteTEDS(WriteBuf.GetSafeArrayPtr());
    std::cout << "DS2430 Read Memory (WriteData)" << std::endl;
    PrintPacket(WriteBuf);

    // retrieve the response
    CComSafeArray<short> ReadBuf;
    SAFEARRAY* psaReadBuf;
    digitizer->Channels->Item[name]->TEDS->ReadTEDS(&psaReadBuf);
    std::cout << "DS2430 Read Memory (ReadData)" << std::endl;
    ReadBuf.Attach(psaReadBuf);
    PrintPacket(ReadBuf);

    recLen = (int)ReadBuf.GetCount();
    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 **Error! Reference source not found.**command 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(IVTEXDigitizerPtr digitizer,
const char* data)
{
    int recLen = 0;
    CComSafeArray<short> WriteBuf;
    WriteBuf.Create();

    if(strlen(data) != ((DS2430_SCRATCHPAD_LEN * 2) +
(DS2430_SCRATCHPAD_LEN-1)))
    {
        printf("Data was not the right length (wanted 95, got
%i)\n", (int)strlen(data));
        return -1;
    }

    WriteBuf.Add((short)0); // reserve first byte for length
    // access the current device with address in DATA_ID
    WriteBuf.Add(CMD_ML_ACCESS);
    // construct a block of communication to MicroLAN
    WriteBuf.Add(CMD_ML_DATA);
    WriteBuf.Add(3+DS2430_SCRATCHPAD_LEN); // block length
    WriteBuf.Add(4+DS2430_SCRATCHPAD_LEN); // data length
    // send the write scratchpad command
    WriteBuf.Add(WRITE_SCRATCHPAD);
    // send the address byte
    WriteBuf.Add((short)0);
    // the 5 bytes of data to write
    char byte[3];
    for(int i = 0; i < ((2*DS2430_SCRATCHPAD_LEN) +
(DS2430_SCRATCHPAD_LEN-1)); i+=3)
    {
        strncpy_s(byte, 3, &data[i], 2);
        byte[2]='\0';
        WriteBuf.Add((unsigned char)strtoul(byte, NULL, 16));
//convert to hex
    }
    // the copy command
    // access the current device with address in DATA_ID
    WriteBuf.Add(CMD_ML_ACCESS);
    // construct a block of communication to MicroLAN
    WriteBuf.Add(CMD_ML_DATA);
    WriteBuf.Add(3); // block length
    WriteBuf.Add(2); // data length
    // send the copy scratchpad command
    WriteBuf.Add(COPY_SCRATCHPAD);
    // send the validation key
    WriteBuf.Add(0xA5);
    // delay for 128ms
    WriteBuf.Add(CMD_DELAY);
    WriteBuf.Add((short)0);
    WriteBuf.Add(DELAY_128 | DELAY_MS);

    // set the length
    WriteBuf[0] = (short)WriteBuf.GetCount() - 1;

    // send the commands
    digitizer->Channels->Item[name]->TEDS-
>WriteTEDS(WriteBuf.GetSafeArrayPtr());
    std::cout << "DS2430 Write & Copy Scratchpad (WriteData)" <<
std::endl;
    PrintPacket(WriteBuf);
    // retrieve the response
    CComSafeArray<short> ReadBuf;
    SAFEARRAY* psaReadBuf;
    digitizer->Channels->Item[name]->TEDS->ReadTEDS(&psaReadBuf);
    std::cout << "DS2430 Write & Copy Scratchpad (ReadData)" <<
std::endl;
    ReadBuf.Attach(psaReadBuf);
    PrintPacket(ReadBuf);

    recLen = (int)ReadBuf.GetCount();
    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

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. Examples of these difference can be seen in the TEDS example program included with the Digitizer or DSA driver.

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 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.

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(CComSafeArray<short> buf)
```

```
{
    int len = buf.GetCount();
    printf("Packet length: %i\n", (int)len);
    for(int i = 0; i < len; i++)
    {
        printf("%02X ", buf[i]);
    }
    printf("\n");
}
```

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(CComSafeArray<short> buf, int offset, int len)
{
    int i, j, k;
    short byte;
    unsigned int r = 0;
    for(k = (offset-1); k < ((offset + len)-1); k++)
    {
        // CRC16 CALCULATION
        byte = buf[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 = (unsigned short)~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 instrument 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.

APPENDIX D

CALIBRATION

OVERVIEW

In order to ensure the measurement accuracy, the analog gain and offset must be regularly calibrated. This is especially true when significant temperature changes occurred since the last time the instruments was calibrated. With the EMX-4250/51, EMX-4350, and EMX-4380, the user can perform calibration using internal reference signal. This is called self-calibration. For self-calibration to work properly, the reference calibration signal should be calibrated annually. This is called factory-calibration.

SELF-CALIBRATION

Self-calibration can be performed without any external equipment or special cabling. It is performed by the Digitizer or DSA driver's *Calibration.Self* interface. During the self-calibration, the internal reference signal is automatically routed to the analog front end and the complete signal path is calibrated. Once the self-calibration is performed, the calibration constants (*Adjustments*) are generated in volatile memory and automatically applied during the data acquisition. Self-calibration constants can be saved or restored to the non-volatile memory using the *SaveAdjustments* and *LoadAdjustments* methods. Otherwise, this information is lost after the instrument is turned off. Self-calibration is primarily used to calibrate drift due to temperature variations. Self-calibration can be done with users signal connected to the input connectors.

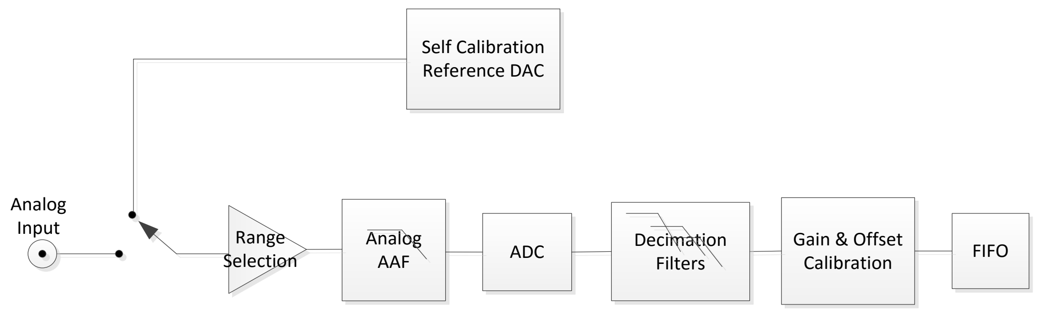


Figure D-8-1: Self-Calibration

FACTORY CALIBRATION

Factory calibration calibrates the reference signal as well as IEPE current source and charge amplifier (EMX-4380). Factory calibration constants are stored in non-volatile memory when the instrument is shipped from VTI or re-calibrated at authorized facilities. Factory calibration constants are password protected to avoid their being accidental overwritten. These constants are automatically loaded and applied to the calibration reference signal during the self-calibration. Optionally, the factory calibration constants can be copied, modified, and stored into a separate non-volatile memory. It is called full calibration. When the *Full.LoadByDefault* property is set to true, the full calibration constants are used instead of the factory calibration constants. Calibration adjustments are stored in human readable JSON format. If necessary, the user can edit the constants and keep them as full-calibration.

To perform factory calibration, a fully calibrated precision instrument and setup is required. The factory calibration is usually generated by VTI or authorized calibration facility. If the user wishes to perform factory calibration at their own facility, contact to VTI for more information.

SELF/FULL CALIBRATION ADJUSTMENTS FILE

Calibration adjustments files are JSON documents organized as a series of nested JSON Objects (lists of key-value pairs). The following list of keys and the description of their values mimics the nested structure of the calibration files. Many keys are only present on some models.

- Date: YYYY-MM-DD HH:mm:SS
- Type: Full or Self
- Instrument:
  - Model: model number
  - Serial: serial number
  - Version: version number
  - Daughter Card:
    - Model: model number
    - Serial: daughter card serial number
- Adjustments:
  - Channels:
    - CH1 - CHN: list of calibration objects, each containing:
      - Range: The range that these constants apply to
      - Clock Frequency: The setting of *SamplingClockFrequency* that these constants apply to
      - Offset: Calibration offset in ADC counts
      - Offset_AC: If present, the offset to use when Coupling=AC
      - Gain: Calibration gain, unitless
  - IEPE Channels:
    - CH1 - CHN:
      - High: If present, the value of the high IEPE source in Amps
      - Low: If present, the value of the low IEPE source in Amps
      - Offset: If present, the calibration offset of the IEPE source, in counts
      - Gain: If present, the calibration gain of the IEPE source, in amps per count
      - Load: A test value obtained by running the IEPE source through an internal load resistor
  - Charge_Gain:
    - CH1 - CHN: Additional gain that is combined with channel gain when Function=Charge
- References:
  - Channels:
    - CH1 - CHN:
      - nHz: reference measurements for the specified Clock Frequency
        - Various model-specific reference voltage measurements, as measured by an external DMM

SELF CALIBRATION LOG

The calibration log simply consists of a series of informational messages about the progress of calibration, each accompanied by a timestamp. The following are some of the types of message that can appear:

- connect to localhost:9900: Setting up the socket to read data from the instrument
- setup: Setting up the measurement settings needed during calibration
- warmup: Waiting a few seconds to ensure that any previously disabled ADCs are warmed up and settled
- measure X freq Y range Z: Measuring voltage reference X using clock frequency Y and range Z
- restore user configuration: Restoring all settings to the values they had before calibration was started
- save results: Creating new results file and storing in volatile memory
- check limits: Comparing calculated calibration adjustments to minimum and maximum valid values
- apply adjustments: Overwriting currently loaded self calibration adjustments with the newly calculated ones
- failed: an error has occurred
  - Unable to synchronize: was not able to synchronize ADCs.
  - Current full calibration does not match this card: The currently loaded full calibration does not match this card's programmed model and serial numbers.
  - Current full calibration date is in the future: The currently loaded full calibration can't be valid because it hasn't happened yet.
  - Current date/time is invalid: The current time is invalid because it's before the release of this firmware.
  - CHX Y Hz, range Z: A of B exceeds limits of C - D: When calculating adjustment A on CHX at a clock frequency of Y Hz and range Z, the value was B, which falls either below the minimum value of C or above the maximum of D.
  - failed I/O: Unable to retrieve measurement data from the ADCs.
  - crashed: the self calibration code has a bug.
- success: Completed all steps successfully.

CALIBRATION RESULTS

The calibration results file contains all the same information as the Calibration Adjustments. It *also* includes more information about the measurements that went into calculating each adjustment value.

That additional information takes the following form:

- Measurements:
  - Channels:
    - CH1 - CHN: a list of measurement objects, each containing:
      - Reference: The name of the voltage reference being measured
      - Range: The range being used to do the measurement
      - Clock Frequency: The clock frequency used to do the measurement
      - Min: A list of the minimum value from each of the records acquired during the measurement
      - Max: A list of the maximum value from each of the records acquired during the measurement

- Stdev: A list of the standard deviations from each of the records acquired during the measurement
- Reading: The final measurement value, obtained by averaging all the samples of all records acquired during the measurement

SELF/FULL CALIBRATION REPORT

The calibration report file is a human-readable version of all the information found in the calibration results file. It is organized as a comma-separated list (CSV) file. It has the following sections, each consisting of the listed columns:

- Self/Full Calibration Report
  - Date – the date the calibration was done
  - Model – the model of the card being calibration
  - Serial – the serial number of the main board
  - Daughter Card Serial – the serial number of the daughter board
  - Firmware Version – the firmware version of the card
- Internal References
  - Channel – the channel that the measurement is for
  - Clock Frequency – the clock frequency used when making the measurement
  - Reference – the name of the voltage reference being measured
  - Value – the value of the reference as measured by an external DMM during full calibration
- Channel Measurements (Filtered ADC Counts)
  - Channel – the channel that the measurement was taken with
  - Clock Frequency – the clock frequency used to make the measurement
  - Range – the range used to make the measurement
  - Reference – the name of the voltage reference being measured
  - Measurement – The final averaged measurement value, in units of ADC counts
- New Channel Adjustments
  - Channel – the channel that the adjustment is for
  - Clock Frequency – the clock frequency that the adjustment is for
  - Range – the range that the adjustment is for
  - Adjustment – the name of the adjustment
  - Value – The new calculated value of the adjustment
  - Minimum – the minimum allowed value of the adjustment
  - Maximum – the maximum allowed value of the adjustment
  - Error Ratio – The ratio of the new value's deviation from nominal to the allowable error range ((value - nominal) / ((maximum - minimum)/2)); a value of 1.0 indicates equal to the maximum, -1.0 equal to the minimum.
  - PASS/FAIL – PASS if |Error Ratio| < 1.0, otherwise FAIL
- IEPE Adjustments
  - Channel – the channel that the adjustment is for
  - Adjustment – the name of the adjustment
  - Value – The new calculated value of the adjustment
  - Minimum – the minimum allowed value of the adjustment
  - Maximum – the maximum allowed value of the adjustment
  - Error Ratio – The ratio of the new value's deviation from nominal to the allowable error range ((value - nominal) / ((maximum - minimum)/2)); a value of 1.0 indicates equal to the maximum, -1.0 equal to the minimum.
  - PASS/FAIL – PASS if |Error Ratio| < 1.0, otherwise FAIL

- Charge Gain
    - Channel – the channel that the adjustment is for
    - Value – The new calculated value of the adjustment
    - Minimum – the minimum allowed value of the adjustment
    - Maximum – the maximum allowed value of the adjustment
    - Error Ratio – The ratio of the new value’s deviation from nominal to the allowable error range ((value - nominal) / ((maximum - minimum)/2)); a value of 1.0 indicates equal to the maximum, -1.0 equal to the minimum.
    - PASS/FAIL – PASS if |Error Ratio| < 1.0, otherwise FAIL

APPENDIX E

MULTIPLE CHASSIS SYSTEM

OVERVIEW

When a large number of measurement channels is required, or when a distance between measurement points are large, instruments may need to be distributed among more than one chassis. In order to achieve synchronized, simultaneous the data acquisition between instruments, both sampling clock and the measurement state machine transition must be synchronized. When all instruments are installed in a single PXIe chassis, they can share the same sampling clock and timing signal using chassis' backplane.

When more than one chassis are involved, these signals need to be somehow synchronized together. One method to achieve this, is by physically connect them using dedicated cables. While this approach gives the most consistent and best synchronization performance, it may not be practical when chassis are separated by long distance. The other method is to synchronize them using LAN messages over Ethernet. In this method, the sampling clock of each chassis are synchronized to a grandmaster clock using IEEE1588-2008 protocol, and state machine transition are coordinated by LAN messages between chassis as defined in LXI specification. See also *Multiple Cards, Segments and Chassis* in Section 4 of this manual.

While the synchronization between chassis is automatically performed by instrument's driver and it is transparent to the user, it is important to understand the tradeoffs and limitations.

MASTER AND SLAVE

Regardless of synchronization method, one of the chassis becomes a master and others become slaves. The trigger state machine transition is paced by a master chassis, and all slave chassis follows it. Because of this, arm and trigger events are only detected by cards in a master chassis[2]. The analog trigger signal, TTL external trigger, PXI or LAN arm or trigger event must be sent to a master chassis.

When chassis are synchronized by clock and trigger cables, the master chassis must be the first chassis of the resource string being used at the driver initialization. The trigger signal must be routed from a backplane trigger line to EMX2500 front panel connector for a master chassis. For slave chassis, the trigger signal needs to be routed from front panel connector to the backplane trigger line.

When LAN synchronization is used, the first chassis in the resource string becomes a master by default. However, when any analog channel or external trigger channel is assigned as an arm or trigger source, the chassis contains the channel automatically becomes master. It is invalid configuration to specify arm or trigger channels from more than one chassis.

[2] When a master chassis has multiple trigger segments, only one of segments becomes a master.

SYNCHRONIZATION PERFORMANCE

Two histograms show typical synchronization error between two chassis using trigger cable, and LAN event[3]. When the chassis are synchronized by dedicated cable, the error is mainly determined by signal transmission delay caused by signal routing and cable length.

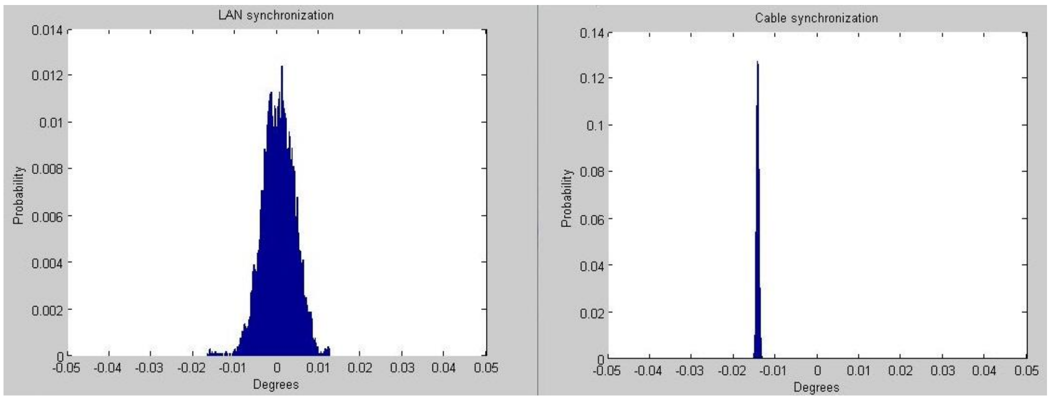


Figure E-1: Typical chass-to-chasss phase error (in degrees) at 1kHz

With LAN synchronization, unlike by cable, the error is usually distributed around zero. This is because IEEE 1588 algorithm corrects for the time delay of LAN packet transmission between master clock and slave. As far as the delay is equal for both directions, the average error becomes zero. The error variance largely depends on the network configuration. The quality of network switches, amount of network traffic, and the stability of master clock, greatly affect the actual synchronization performance, or phase error. A dedicated network for the test system with a stable GPS Grandmaster clock (PTP v2), connected with boundary/transparent clock switches[4] provides the best and the most consistent performance.

[3] The test result with Dell PowerConnect™ 2724 Gigabit Ethernet switch.
[4] e.g., Meinberg LANTIME™, Hirschmann MACH1000™

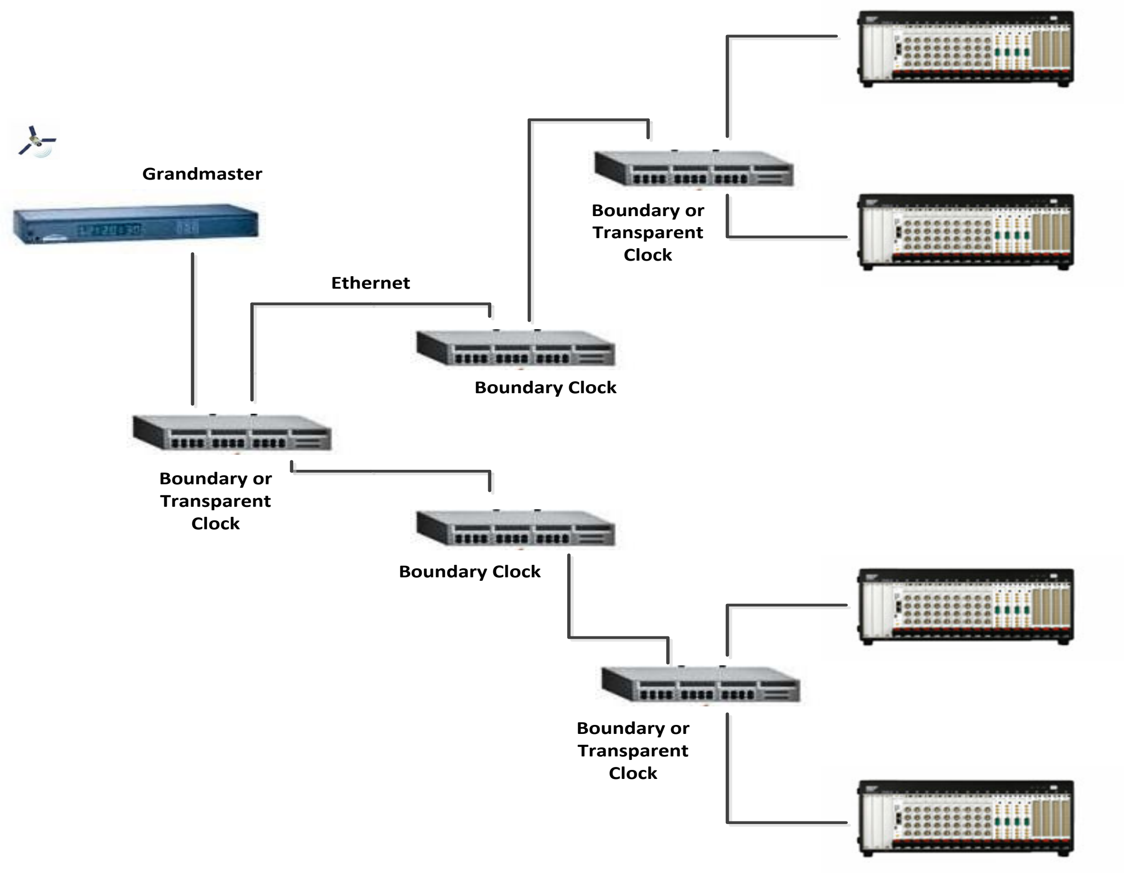

Grandmaster
Ethernet
Boundary or Transparent Clock
Boundary Clock
Boundary or Transparent Clock
Boundary Clock
Boundary or Transparent Clock

Figure E-2: IEEE1588 Network Topology

APPENDIX F

MEMORY LISTING AND CLEARING PROCEDURE

OVERVIEW

The below table provides the Flash components information used in different EMX plugin cards. It also contains clearing procedure details for different modules inside the flash components and whether the user can perform writable operation on them or not.

Component	Volatile?	Contains	User Writeable?	Clear Procedure	Models
1x16 MB Flash (MFG: Spansion P/N: S25FL128SAGNFI90x (EMX-4250/4251) or S25FL129P0XNFI00 (EMX-4350/4380))	No	Firmware	No	None	EMX-4250 EMX-4251 EMX-4350 EMX-4380
		Backup Firmware	No	None	
		Instrument Identity	No	None	
		Factory Full Calibration	No	None	
		User Full Calibration	Yes	VTEXDigitizer.Calibration. Full.ClearSavedAdjustments()	
		Self Calibration	Yes	VTEXDigitizer.Calibration. Self.ClearSavedAdjustments()	
		Stored Instrument Configuration	Yes	VTEXDigitizer.Configuration. ClearSavedConfiguration()	
1x128 MB DDR2 SDRAM (MFG: Micron Tech P/N: MT47H64M16)	Yes	Runtime Data	Yes	Power cycle machine	EMX-4250 EMX-4251 EMX-4350 EMX-4380
1x8 kB IIC EEPROM (MFG: ST Micro P/N: M24C64)	No	Mezzanine Board Identity	No	None	EMX-4250
1x64 kB Flash (MFG: Atmel P/N: AT25F512AN-10SH-2.7)	No	Mezzanine Board Identity	No	None	EMX-4350 EMX-4380

INDEX