EX1200-3604/
EX1200-3608

4-/8-CHANNEL DAC/
500 kHz AWG

USER'S MANUAL

P/N: 82-0127-003
Released June 11, 2020

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.

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.

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:

ATTENTION - Important safety instructions

Frame or chassis ground

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

WARNINGS

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

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

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

The mains outlet that is used to power the equipment must be within 3 meters of the device and shall be easily accessible.

Power Consumption Because of the large power requirements on the 24 V line, the number of EX1200-360x modules must be limited to ensure that the power supply's current ratings are not exceeded. For more information on the number of EX1200-360x cards that can be installed in a given EX1200 chassis, refer to *Power Requirement Considerations* in *Section 2*.

WARNINGS (CONT.)

Avoid Electric Shock

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

Ground the Product

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

Operating Conditions

To avoid injury, electric shock or fire hazard:

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

Improper Use

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

Parallel/Series Channels

Channels in current (parallel) and voltage (series) mode ***cannot*** be connected together. Channels that are connected in parallel cannot be connected to another group of channels that are connected in series. Doing so may result in damage to the EX1200-360x and its mainframe**.**

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

VTI Instruments
Lake Stevens Instrument Division

3216 Wetmore Avenue, Suite 1
Everett, WA 98201

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

VTI Instruments, Pvt. Ltd.
Bangalore Instrument Division

642, 80 Feet Road
Koramangala IV Block
Bangalore – 560 034
India

Phone: +91 80 4040 7900
Phone: +91 80 4162 0200
Fax: +91 80 4170 0200

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 EX1200-3608 and EX1200-3604 (referred to collectively as the "EX1200-360x") provide eight or four independent channels, respectively, of a digital to analog converter (DAC) with 16 bits of resolution. Each channel consists of a 16-bit DAC combined with a low-pass filter and an output amplifier. The 16-bit DAC allows these modules to achieve fine resolution at very low output range settings. Along with static output operation, the DAC modules provide an arbitrary waveform generation (AWG) mode which supports looping to build complex waveforms without the system controller's intervention. The data may be paced out of the instrument by using either a user-supplied clock or the internal programmable timer with output rates up to 500 kSa/s.

Each channel is true-differential, and has sense lines that can be used to compensate for voltage drops that occur over the length of the lead wire between the DAC output and the device under test (DUT). Channels can be connected in parallel to produce outputs in excess of ±20 mA or in series to produce outputs in excess of ±20 V. An external clock input and an external trigger input are available to synchronize output level changes with external events. When used in an EX1200 series mainframe with the optional DMM, the DAC outputs can be routed to the internal analog backplane for verification prior to critical test runs to ensure the device will perform to a high degree of accuracy.

HARDWARE OVERVIEW

The following section provides information regarding the various hardware elements that are incorporated into the EX1200-360x.

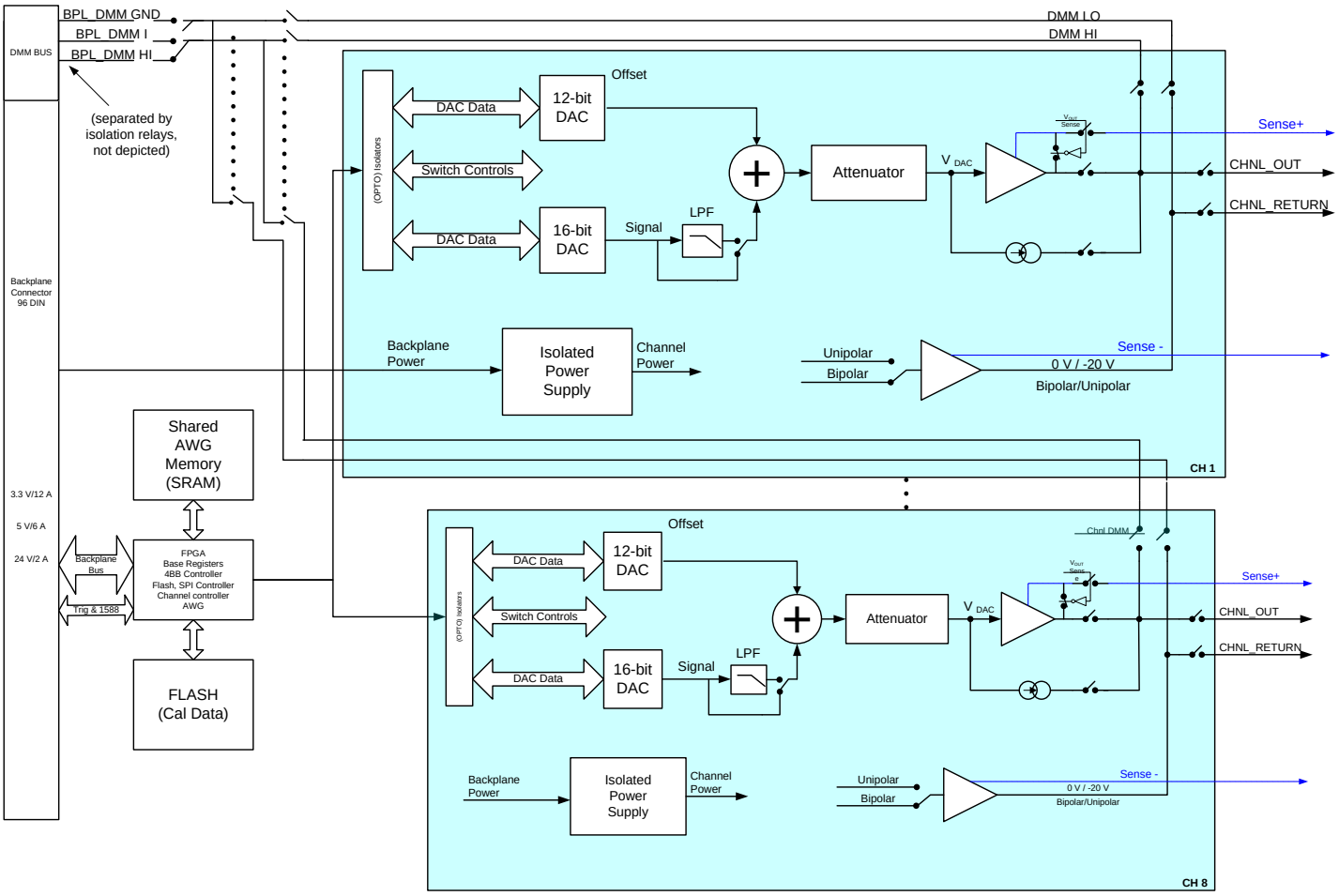


FIGURE 1-1: EX1200-3608 BLOCK DIAGRAM

Digital Section

In the digital section, the EX1200-360x channels interface to the backplane via an FPGA. The AWG data is stored into a volatile 4 MB SRAM that is shared by all channels. Access to the SRAM is managed by the FPGA. As the memory used for storing the AWG data is volatile, all information is lost if the instrument is powered off. The user must save the data before cycling power. See the SFP for details about how to do that.

In addition to the SRAM, the card is also equipped with 4 MB of non-volatile memory FLASH memory used to store calibration information and other firmware. The FLASH memory CANNOT be used to store AWG data.

Analog Section

The analog section of each isolated, EX1200-360x channel consists of the following blocks:

- Unipolar/bipolar section
- 12-bit offset DAC
- 16-bit signal DAC
- Signal DAC low-pass filter
- Amplitude conditioning circuitry
- Sense lines
- Isolated channels with voltage and current capabilities
- Isolated outputs
- Internal DMM access

Unipolar/Bipolar Section

The setup selection process determines the selection of the filter path, local or remote sensing, and voltage or current output. Signal path gain via the attenuator/amplifier stage and the unipolar/bipolar setting is determined either during static setup or when the waveform parameter file is downloaded. These settings cannot be changed dynamically during the waveform generation.

12-Bit Offset DAC

The 12-bit DAC value is determined for each signal type (voltage or current) and each range by the calibration process. This value is static and loaded when the channel setup is initiated.

16-Bit Signal DAC

Depending on the OutputMode, the 16-bit DAC value is either a static or dynamic setting. In dynamic operation, the EX1200-360x card is capable of updating the DAC every 2 µs (500 kHz update rate).

Signal DAC Analog Low-Pass Filter

The DAC output filter is a two-pole, VCVS (Voltage Controlled, Voltage Source) design with a Bessel response. The filter cutoff frequency (-3 dB pole) is set for 200 kHz. The Bessel filter is optimized for pulse response and yields maximum rise time with minimal overshoot. Figure 1-2 shows a scope snapshot of a 50 kHz square wave with the filter turned on. The default state of the filter is OFF.

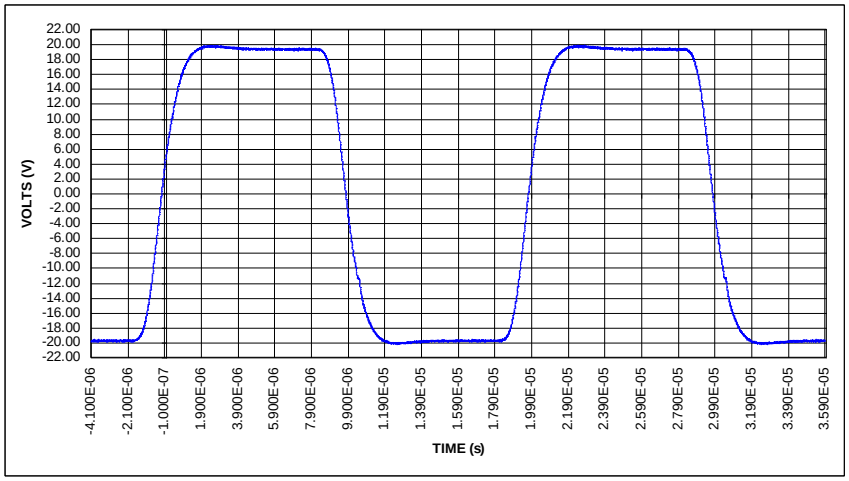


FIGURE 1-2: SQUARE WAVE WITH LPF ON

Amplitude Conditioning Circuitry

The attenuator/amplifier combination scales the DAC signal to the user-specified voltage range and conditions the signal to supply a minimum load current of ±20 mA for all voltage ranges.

The current output circuitry scales the DAC signal to the user-specified current range and conditions the signal to supply a minimum compliance voltage of ±20 V for all loads up to ±20 mA.

Sense Lines

Each channel is true-differential and has sense lines for both the high- and the low-side that can be used to compensate for voltage drops that occur over the length of the lead wire between the DAC output and the DUT. When a high current is driven into a load using long cables, it is highly recommended that the user connects and uses the sense lines. The default setting is local sensing. When the remote sensing lines are enabled, the channel should not exceed 10 Ω in either sense line.

The voltage source circuitry has internal sense lines, making the external lines unnecessary. Using the sense lines are, however, recommended to improve voltage accuracy at the load. Note that sense lines cannot be used in current mode.

For more information on using the sense lines, see *Using Sense Lines* in *Section 6.*

Isolated Channels with Voltage and Current Capabilities

Each EX1200-360x channel is galvanically isolated from each other as well as the digital circuitry. Additionally, each channel's power is isolated from the rest of the chassis as well. The channels receive their controls/commands through their dedicated isolators.

Since all channels are completely isolated, the user can connect them in series when they are in voltage mode to create higher voltages of up to 320 V peak-to-peak. Likewise, the channels can be connected in series in dc as well in dynamic mode.

In current mode, the channels can be connected in parallel to similarly generate currents of up to 160 mA. The channels can be connected in parallel in dc mode as well as in dynamic mode.

For more information on connecting the channels in series or in parallel, see *Connecting Voltage Channels in Series* and *Connecting Current Channels in Parallel* in *Section 6.*

WARNING	Channels in current (parallel) and voltage (series) mode cannot be connected together. Channels that are connected in parallel cannot be connected to another group of channels that are connected in series. Doing so may result in damage to the EX1200-360x and its mainframe.

Isolated Outputs

The outputs of every channel are isolated from the outside world, so that, at power up, the load is not driven with any voltage. By default, the state of the output switches is OFF.

Internal DMM Access

It is possible to route the voltage or current outputs of each channel to the internal EX126x series mainframe with internal DMM for self-verification. The internal DMM can also be used to verify the accuracy of the channel's settings, in an ATE environment, before the channels are connected live to the DUT. For more information, refer to the *Calibration* discussion in *Section 5*.

EX1200-360x Specifications

EX1200-360x General Specifications

Number of Channels	
EX1200-3604	4
EX1200-3608	8
Resolution	
	16 bits, 16 bits monotonic
Time/Frequency Domain	
Settling time	5 µs to 0.1% of specified value
Rise time*	3 µs typical
Slew rate	40 V/µs typical
Bandwidth	250 kHz
Crosstalk**	< 65 dbV @ 10 kHz
Phase matching	
Internal channels	< 50 ns when all channels are running synchronized on the internal clock
External channels	< 100 ns when all channels are running synchronized on the internal clock

** Measured 10% - 90% on a ±20 V square wave with 1 kΩ load, filter turned OFF.*
*** Measured on CH5 with channels 0 through 4 and 6 through 7 producing a ±20 V square wave at 10 kHz*

Time from External Trigger Receipt to First Sample Output	
	2.2 µs plus the rise time
External Clock	
Frequency	Maximum 500 kHz
Levels	LVTTL
External Trigger Input	
Pulse width	Minimum 20 ns
Levels	LVTTL
Marker Output	
Levels	LVTTL
Duration	20 ns ÷ 1.34 s in 20 ns increments
Power Consumption	
EX1200-3604	
3.3 V	0.3 A
5 V	1 A
24 V	0.8
EX1200-3608	
3.3 V	0.3 A
5 V	1 A
24 V	1.4 A

DAC Specifications, Voltage Mode

Output Voltage Ranges	
Bipolar	±1 V, ±2 V, ±5 V, ±10 V, ±20 V
Unipolar	40 V
Auto-ranging	Supported (SEE *Explanation of Specifications* for details)
Maximum output (series channels)	±160 V
Maximum frequency	DC to maximum sampling rate
Output current	±20 mA
Current protection	Current limitation circuit kicks in above 50 mA
Short circuit time*	Up to 20 minutes. No restart necessary after short circuit.

** Longer short circuit times can damage the card.*

DCV Accuracy	
1 V	±(0.05% of setting ± 1.3mV)
2 V	±(0.05% of setting ± 1.6mV)
5 V	±(0.07% of setting ± 3.5mV)
10 V	±(0.07% of setting ± 4.0 mV)
20 V	±(0.06% of setting ± 4.0 mV)
40 V	±(0.10% of setting ± 8.0 mV)
DCV Noise	
	≤ 2 mV rms
Programmable Offset Range	
	Full-scale

DAC Specifications, Voltage Mode	
Ripple Noise	
DCV	≤ 2 mV rms
Isolation	
Between channels	200 V
Voltage Remote Sensing*	
	High and low sense lines available per channel for cable length voltage drop compensation

** Maximum sense line impedance is 10 Ω in either sense.*

DAC Specifications, Current Mode	
Output Current Ranges	
Ranges	± 5 mA, ± 10 mA, ±20 mA
Maximum output (parallel channels)	±160 mA
Maximum frequency	DC to 1 kHz sampling rate DC to maximum sampling rate
Output Current Short Circuit	
Per channel	±20 mA into short circuit
Short circuit time*	Up to 20 minutes. No restart necessary after short circuit.

**Longer short circuit times can damage the card.*

DCA Accuracy	
1kΩ Load	±(0.10% of setting +0.1% of Range)
Other Load	Add + [3PPM/Ω *ABS(1kΩ-RL)] of setting
Compliance Voltage	
	20 V

AWG Specifications	
Update Rate	
Programmable	20 ns (steps)
Maximum	500 kSa/s (2 µs)
	Programmable, maximum 500 kSa/s
Trigger Sources	
	Front panel input, LXISync, software
Waveform Size	
Minimum	4 samples
Maximum	2,097,100 samples
Waveforms	
	1 to 4096 (SW limited can be increased in the future)
Sequences	
	1 to 4096 (SW limited can be increased in the future)
Waveform Repeat Count	
	1 to 2^{16} (65,536)
Memory Sequence Repeat (Burst) Count	
	1 to 2^{16} (65,536)
Steps Per Sequence	
	1 to 4096 (SW limited can be increased in the future)
Modes	
IVI-compliant	OutputModes: StandardWaveform, ArbitraryWaveform, ArbitrarySequence OperationModes: Continuous, Burst
VTI Instrument specific	OperationModes: Sequenced, SingleStep
Marker Function	
Output	Can be sourced from any of the channels
Position	Can be placed at anywhere inside a waveform
Marker Pulse Length	
	20 ns to 0.335 s
Marker Output	
	Front panel TTL compatible output
Standard Waveforms	
Supported waveforms	Sine, Ramp (up/down), triangle, and square
Initial phase	Supported for all standard waveforms
Burst mode	Supported for all standard waveforms
Duty cycle	Adjustable for all standard waveforms

AWG SPECIFICATIONS	
Channel configuration	Each channel is programmed INDEPENDENTLY in Standard or AWG modes.
CONNECTOR INFORMATION	
MATING CONNECTOR KIT	
Description	44 pin HD D-sub mating connector and backshell, with 3 ft unterminated 22 AWG wire
VTI Part Number	70-0363-502
TERMINAL BLOCK	
Description	EX1200-TB44, 44-pin DIN connector with internal CJC reference
VTI Part Number	70-0367-007
CONNECTOR	
Description	44 pin HD D-Sub mating connector, backshell and pins, crimp style
VTI Part Number	27-0390-044
CRIMPING TOOL	
Description	Crimp tooling, includes handle and positioner, 22 AWG
VTI Part Number	70-0297-001
UNTERMINATED CABLE ASSEMBLY	
Description	44-pin, unterminated cable assembly, 3 ft
VTI Part Number	70-0363-502

EXPLANATION OF SPECIFICATIONS

This section provides explanatory notes to certain elements of the EX1200-360x specifications that may be misunderstood.

Auto-ranging

Auto-ranging describes the EX1200-360x's ability to automatically adjust its range based on newly defined *gain* and *offset* values provided by the user. If set to auto-range mode and the gain is changed while the instrument is in generate mode, the output will be set to 0 V for approximately 1.5 ms before the range is set to the new value. If this transition is not acceptable in a specific application, the user should manually change the range to a higher value that will allow the gain to be increased without exceeding the range. This will effectively disable auto-ranging.

Power Consumption

Power consumption indicates the amount of current the EX1200-360x draws from the specified power supply line. Because of the large power requirements on the 24 V power supply rail, the number of EX1200-360x modules installed in an EX1200 series mainframe must be limited to ensure that the power supply's current ratings are not exceeded. For more information, refer to the *Power Requirement Considerations* discussion in *Section 2*.

SECTION 2

USING THE INSTRUMENT

UNPACKING

When an EX1200-360x is unpacked from its shipping carton, the contents should include the following items:

- An EX1200-3604 or EX1200-3608
- *LXI Quick Start Guide*
- *EX1200-3604/-3608 User's Manual* (this manual)

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 an EX1200 series mainframe.

DETERMINE SYSTEM POWER REQUIREMENTS

The power requirements of the EX1200-360x is provided in the *Specifications* section of *Section 1*. It is imperative that the EX1200 mainframe provides adequate power for the modules installed. For more information on EX1200 mainframe power consumption, please refer *Appendix B* of the *EX1200 Series User's Manual* (P/N: 82-0127-000). The user should confirm that the power budget for the system (for the chassis and all modules installed therein) is not exceeded on any voltage line.

It should be noted that if the mainframe cannot provide adequate power to the module, the instrument might not perform to specification and possibly damage the power supply. In addition, if adequate cooling is not provided, the reliability of the instrument will be jeopardized and permanent damage may occur. Damage found to have occurred due to inadequate cooling will void the warranty on the instrument in question.

Power Requirement Considerations

As described in the *Power Consumption* discussion in *Section 1*, the number of EX1200-3604 and EX1200-3608 modules that can be installed in an EX1200 mainframe is limited by the current available on the 24 V line of the power supply. Since there is approximately 3 A of current available on the 24 V rail and the EX1200-360x draws between 0.8 A and 1.4 A per card, the user must be cautious not to exceed the power supply's limit when using this instrument.

NOTE	**Prior to installing multiple EX1200-360x instruments into an EX1200 series mainframe, the power requirement calculations in Appendix B of the *EX1200 Series User's Manual* must be reviewed to ensure that the current limits of the power supply are not exceeded.**

PLUG-IN MODULE INSTALLATION

Before installing a plug-in module into an EX1200 system, make sure that the mainframe is powered down. Insert the module into the base unit by orienting the module so that the metal cover of the module can be inserted into the slot of the base unit. Position the cover so that it fits into the module's slot groove. Once the module is properly aligned, push the module back and firmly insert it into the backplane connector. See Figure 2-1 for guidance.

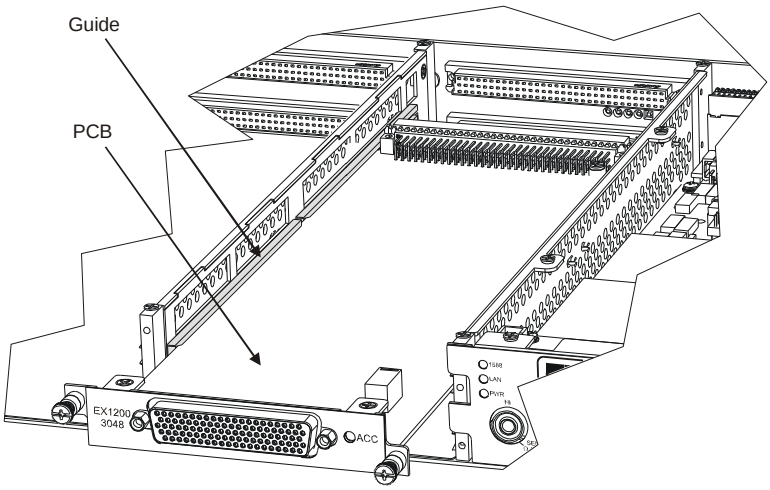


FIGURE 2-1: MODULE INSTALLATION (EX1200-3048 USED AS EXAMPLE)

WARM-UP TIME

The specified warm-up time for an EX1200 system is 30 minutes. If, however, the unit is being subjected to an ambient temperature change greater than 5 ºC, extra stabilization time is recommended to achieve maximum performance.

CONNECTOR PIN/SIGNAL ASSIGNMENT

The tables below provide signal and connector pin information for the EX1200-3604 and EX1200-3608. Please note that the pin mapping for the two cards differ. For mating connector information, please refer to the *EX1200-360x Specifications* in this manual.

Pin	Signal	Pin	Signal	Pin	Signal
1	UNUSED	**16**	UNUSED	**31**	CH1_OUT
2	UNUSED	**17**	UNUSED	**32**	CH1_RETURN
3	SHIELD	**18**	SHIELD	**33**	SHIELD
4	CH2_RETURN	**19**	CH2_REMOTE_SENSE-	**34**	CH1_REMOTE_SENSE+
5	CH2_OUT	**20**	CH2_REMOTE_SENSE+	**35**	CH1_REMOTE_SENSE-
6	UNUSED	**21**	UNUSED	**36**	GROUND*
7	UNUSED	**22**	UNUSED	**37**	MARKER_OUT
8	SHIELD	**23**	SHIELD	**38**	SHIELD
9	CH3_RETURN	**24**	CH3_REMOTE_SENSE-	**39**	EXT_CLK_IN
10	CH3_OUT	**25**	CH3_REMOTE_SENSE+	**40**	TRIG_IN
11	UNUSED	**26**	UNUSED	**41**	UNUSED
12	UNUSED	**27**	UNUSED	**42**	UNUSED
13	SHIELD	**28**	SHIELD	**43**	UNUSED
14	CH4_RETURN	**29**	CH4_REMOTE_SENSE-	**44**	UNUSED
15	CH4_OUT	**30**	CH4_REMOTE_SENSE+		
** Use as a ground reference for the I/O signals (MARKER_OUT, EXT_CLK_IN, and TRIG_IN)*					

TABLE 2-1: EX1200-3604 CONNECTOR PIN SIGNAL ASSIGNMENTS22

Pin	Signal	Pin	Signal	Pin	Signal
1	CH2_RETURN	**16**	CH2_REMOTE_SENSE-	**31**	CH1_OUT
2	CH2_OUT	**17**	CH2_REMOTE_SENSE+	**32**	CH1_RETURN
3	SHIELD	**18**	SHIELD	**33**	SHIELD
4	CH3_RETURN	**19**	CH3_REMOTE_SENSE-	**34**	CH1_REMOTE_SENSE+
5	CH3_OUT	**20**	CH3_REMOTE_SENSE+	**35**	CH1_REMOTE_SENSE-
6	CH4_RETURN	**21**	CH4_REMOTE_SENSE-	**36**	GROUND*
7	CH4_OUT	**22**	CH4_REMOTE_SENSE+	**37**	MARKER_OUT
8	SHIELD	**23**	SHIELD	**38**	SHIELD
9	CH5_RETURN	**24**	CH5_REMOTE_SENSE-	**39**	EXT_CLK_IN
10	CH5_OUT	**25**	CH5_REMOTE_SENSE+	**40**	TRIG_IN
11	CH6_RETURN	**26**	CH6_REMOTE_SENSE-	**41**	CH8_REMOTE_SENSE+
12	CH6_OUT	**27**	CH6_REMOTE_SENSE+	**42**	CH8_REMOTE_SENSE-
13	SHIELD	**28**	SHIELD	**43**	CH8_OUT
14	CH7_RETURN	**29**	CH7_REMOTE_SENSE-	**44**	CH8_RETURN
15	CH7_OUT	**30**	CH7_REMOTE_SENSE+		
** Use as a ground reference for the I/O signals (MARKER_OUT, EXT_CLK_IN, and TRIG_IN)*					

TABLE 2-3: EX1200-3608 CONNECTOR PIN SIGNAL ASSIGNMENTS

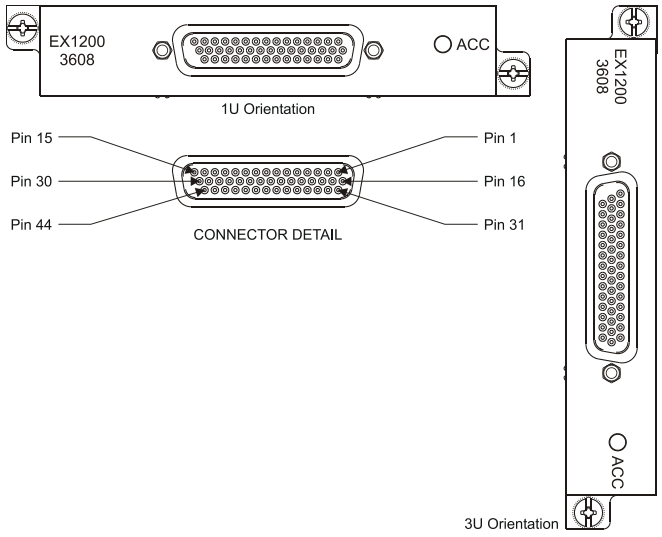


FIGURE 2-2: EX1200-3608 FRONT PANEL DETAIL

Front Panel Connector Pins Description

Pin Name	Description
CH*n*_OUT	The positive side of the channel output.
CH*n*_RETURN	The negative side of the channel output.
SHIELD	These lines are NOT connected to any internal circuitry, but all shield lines are tied to each other. These lines should be connected to earth ground at the load.
CH*n*_REMOTE_SENSE+	The sense line for the positive side.
CH*n*_REMOTE_SENSE-	The sense line for the negative side.
GROUND	The digital ground reference. This is isolated from all channels grounds and should be used as the reference for the following IO signals: MARKER, TRIG, and CLOCK. **DO NOT CONNCT TO ANY OF THE CHANNEL RETURNS.**

Pin Name	Description
MARKER_OUT	An LVTTL output with an internal 33 Ω series resistor for current protection. See the *Output Events* in *Section 2* for more information on the marker feature.
EXT_CLK_IN	An LVTTL input with a 3.3 kΩ pull-up resistor without parallel termination. The user can drive a clock into the EX1200-360x using this line that can be used to pace the AWG. **It cannot, however, be used to drive the STANDARD WAVEFORM generation**.
TRIG_IN	An LVTTL input with a 3.3 kΩ pull-up resistor without parallel termination. This input signal can be used to trigger the waveform generation. See the *Trigger Sources* for a description of how the TRIGGER feature works.

EX1200-360x Terminal Block

A differential terminal block (TB) can be purcahsed for the EX1200-360x (P/N: 70-0367-007). This module can be used to simplify cabling requirements.

TB Ref	Signal	Conn Pin	TB Ref	Signal	Conn Pin
T1	UNUSED	1	**T41**	CH3_REMOTE_SENSE+	25
T2	UNUSED	2	**T42**	CH3_REMOTE_SENSE-	24
T3	SHIELD	3	**T43**	SHIELD	23
T4	UNUSED	UNUSED	**T44**	UNUSED	UNUSED
T5	UNUSED	16	**T45**	TRIG_IN_OUT	40
T6	UNUSED	17	**T46**	EXT_CLK_IN	39
T7	UNUSED	UNUSED	**T47**	UNUSED	UNUSED
T8	UNUSED	UNUSED	**T48**	UNUSED	UNUSED
T9	CH1_RETURN	32	**T49**	UNUSED	12
T10	CH1_OUT	31	**T50**	UNUSED	11
T11	SHIELD	33	**T51**	SHIELD	13
T12	UNUSED	UNUSED	**T52**	UNUSED	UNUSED
T13	CH2_OUT	5	**T53**	UNUSED	27
T14	CH2_RETURN	4	**T54**	UNUSED	26
T15	UNUSED	UNUSED	**T55**	SHIELD	28
T16	UNUSED	UNUSED	**T56**	UNUSED	UNUSED
T17	CH2_REMOTE_SENSE+	20	**T57**	UNUSED	42
T18	CH2_REMOTE_SENSE-	19	**T58**	UNUSED	41
T19	SHIELD	18	**T59**	UNUSED	43
T20	UNUSED	UNUSED	**T60**	UNUSED	44
T21	CH1_REMOTE_SENSE-	35	**T61**	CH4_REMOTE_SENSE+	30
T22	CH1_REMOTE_SENSE+	34	**T62**	CH4_REMOTE_SENSE-	29
T23	UNUSED	UNUSED	**T63**	UNUSED	UNUSED
T24	UNUSED	UNUSED	**T64**	UNUSED	UNUSED
T25	UNUSED	7	**T65**	CH4_OUT	15
T26	UNUSED	6	**T66**	CH4_RETURN	14
T27	SHIELD	8	**T67**	UNUSED	UNUSED
T28	UNUSED	UNUSED	**T68**	UNUSED	UNUSED
T29	UNUSED	22	**T69**	UNUSED	UNUSED
T30	UNUSED	21	**T70**	UNUSED	UNUSED
T31	UNUSED	UNUSED	**T71**	UNUSED	UNUSED
T32	UNUSED	UNUSED	**T72**	UNUSED	UNUSED
T33	MARKER_OUT	37	**T73**	UNUSED	UNUSED
T34	GROUND	36	**T74**	UNUSED	UNUSED
T35	SHIELD	38	**T75**	UNUSED	UNUSED
T36	UNUSED	UNUSED	**T76**	UNUSED	UNUSED
T37	CH3_OUT	10	**T77**	UNUSED	UNUSED
T38	CH3_RETURN	9	**T78**	UNUSED	UNUSED
T39	UNUSED	UNUSED	**T79**	UNUSED	UNUSED
T40	UNUSED	UNUSED	**T80**	UNUSED	UNUSED

FIGURE 2-4: EX1200-TB44 TERMINAL BLOCK TO EX1200-3604 PIN MAPPING

TB Ref	Signal	Conn Pin	TB Ref	Signal	Conn Pin
T1	CH2_RETURN	1	T41	CH5_REMOTE_SENSE+	25
T2	CH2_OUT	2	T42	CH5_REMOTE_SENSE-	24
T3	SHIELD	3	T43	SHIELD	23
T4	UNUSED	UNUSED	T44	UNUSED	UNUSED
T5	CH2_REMOTE_SENSE-	16	T45	TRIG_IN_OUT	40
T6	CH2_REMOTE_SENSE+	17	T46	EXT_CLK_IN	39
T7	UNUSED	UNUSED	T47	UNUSED	UNUSED
T8	UNUSED	UNUSED	T48	UNUSED	UNUSED
T9	CH1_RETURN	32	T49	CH6_OUT	12
T10	CH1_OUT	31	T50	CH6_RETURN	11
T11	SHIELD	33	T51	SHIELD	13
T12	UNUSED	UNUSED	T52	UNUSED	UNUSED
T13	CH3_OUT	5	T53	CH6_REMOTE_SENSE+	27
T14	CH3_RETURN	4	T54	CH6_REMOTE_SENSE-	26
T15	UNUSED	UNUSED	T55	SHIELD	28
T16	UNUSED	UNUSED	T56	UNUSED	UNUSED
T17	CH3_REMOTE_SENSE+	20	T57	CH8_REMOTE_SENSE-	42
T18	CH3_REMOTE_SENSE-	19	T58	CH8_REMOTE_SENSE+	41
T19	SHIELD	18	T59	CH8_OUT	43
T20	UNUSED	UNUSED	T60	CH8_RETURN	44
T21	CH1_REMOTE_SENSE-	35	T61	CH7_REMOTE_SENSE+	30
T22	CH1_REMOTE_SENSE+	34	T62	CH7_REMOTE_SENSE-	29
T23	UNUSED	UNUSED	T63	UNUSED	UNUSED
T24	UNUSED	UNUSED	T64	UNUSED	UNUSED
T25	CH4_OUT	7	T65	CH7_OUT	15
T26	CH4_RETURN	6	T66	CH7_RETURN	14
T27	USR_SHIELD	8	T67	UNUSED	UNUSED
T28	UNUSED	UNUSED	T68	UNUSED	UNUSED
T29	CH4_REMOTE_SENSE+	22	T69	UNUSED	UNUSED
T30	CH4_REMOTE_SENSE-	21	T70	UNUSED	UNUSED
T31	UNUSED	UNUSED	T71	UNUSED	UNUSED
T32	UNUSED	UNUSED	T72	UNUSED	UNUSED
T33	MARKER_OUT	37	T73	UNUSED	UNUSED
T34	GND_C	36	T74	UNUSED	UNUSED
T35	USR_SHIELD	38	T75	UNUSED	UNUSED
T36	UNUSED	UNUSED	T76	UNUSED	UNUSED
T37	CH5_OUT	10	T77	UNUSED	UNUSED
T38	CH5_RETURN	9	T78	UNUSED	UNUSED
T39	UNUSED	UNUSED	T79	UNUSED	UNUSED
T40	UNUSED	UNUSED	T80	UNUSED	UNUSED

FIGURE 2-5: EX1200-TB44 TERMINAL BLOCK TO EX1200-3608 PIN MAPPING

Terminal Block Receiver

The EX1200-TBR chassis is a 1U receiver capable of housing six terminal blocks. The EX1200-TBR ships with rubber feet for table top installations, but may be fitted with rackmount ears for installation into a test rack (P/N: 70-0367-010).

To install a terminal block into the EX1200-TBR, insert the flanges on the side of the terminal block into the guide rails of the desired slot. Continue to push the terminal block into the receiver until it is secured by the rear-locking latch of the receiver. To remove the terminal block from the EX1200-TBR, hold the center thumbscrew on the terminal block, then pull the terminal block from the receiver.

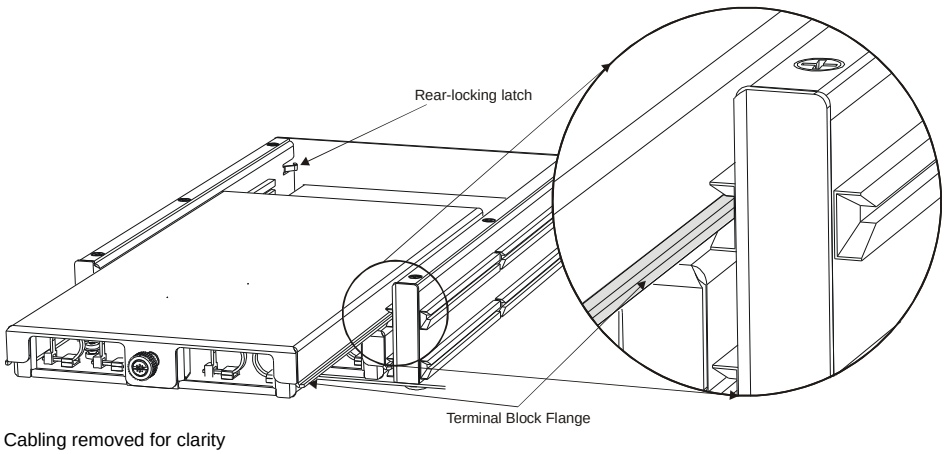


FIGURE 2-6: TERMINAL BLOCK INSTALLATION INTO THE EX1200-TBR

WAVEFORM GENERATION

The EX1200-360x employs several different "modes" of operation when generating waveforms. Three top-level modes can be identified:

- DriveMode
- OutputMode
- OperationMode

DriveMode determines whether the EX1200-360x will generate voltage or current.
Any **DriveMode** can be used with any **OutputMode**, and any **OutputMode** can be used with any **OperationMode**.

OutputMode determines the type of waveform(s) that will be generated. **OutputMode** and its settings are further described in Waveform OutputModes later in this section. Three settings be selected for this mode:

- **StandardWaveform**: This mode generates standard waveforms, such as dc, sine, square, triangle, ramp up, and ramp down. The default for this mode is dc.
- **ArbitraryWaveform**: This mode generates a single arbitrary waveform.
- **ArbitrarySequence**: This mode is a superset of the Arbitrary Waveform mode. Rather than generating a single arbitrary waveform, multiple arbitrary waveforms can be combined into a *sequence*. A sequence consists of a list of waveforms and repeat counts.

OperationMode determines how the waveforms are initiated and repeated. This mode can be set to four different values:

- **Continuous**: When set to continuous, the signal is generated continuously. This is the simplest setting and is the default for this mode. This setting is valid for all **OutputModes**.
- **Burst**: When set to Burst, a waveform/sequence is generated n times when a trigger is received, where n is equal to the value of the BurstCount property
- **Sequenced**: This mode is only available for the **ArbitrarySequence OutputMode**. When set to **Sequenced**, a trigger is used to advance through each item in the sequence. The **BurstCount** parameter does not apply when this setting is used.
- **SingleStep**: The **SingleStep** setting allows the user to use the software trigger to output the next DAC value. This is typically used to debug arbitrary waveforms and sequences.

Additional information for this mode ant its settings are further discussed in detail in *Operation Mode* later in this section.

NOTE	Waveforms and sequences CANNOT be updated while generation is in progress.

WAVEFORM OUTPUTMODES

Standard Waveform OutputMode

The EX1200-360x can generate several standard waveforms: dc, sine, ramp up, ramp down, triangle, and square. These waveforms are configured using the following parameters: frequency, duty cycle, initial phase, dc offset, and amplitude. DC offset and amplitude can be changed during standard waveform generation. The new values take effect at the end of the current clock sample.

Arbitrary Waveform Generator (AWG)

In either of the AWG modes (ArbitraryWaveform or ArbitrarySequence), channels can be set to output complex waveforms by loading one or more waveforms and linking them together. As the channels are independently isolated, the mode setting of one channel has no effect on other channels. The EX1200-360x supports a large number of waveforms and sequences, as seen below:

- Up to 4,096 waveforms can be saved on the EX1200-360x.
- Up to 4,096 distinct waveforms can be linked together to form a single sequence.
- Up to 4,096 independent sequences can be created.

The waveforms can be utilized by multiple sequences, and sequences can be used by multiple channels. Thus, any channel can run any sequence, independent of other channels. A waveform can be used only once, looped up to 65,536 times (in a repeat), or looped continuously. At the end of a burst, the last value in the sequence is maintained as the output for that channel.

Waveform generation among channels can also be configured freely by the user. Channels can operate independently of each other, or may be synchronized using the internal clock, an external clock provided through the front panel, or by using triggers, such as the EX1200 series mainframe Trigger Bus lines, the DMM backplane, or software events. The synchronization options are also configurable per channel, allowing some channels to use the internal clock, while others use an external clock signal. Sampling rates can configured in a similar, independent manner.

NOTE	Waveforms and sequences **CANNOT** be updated while generation is in progress.

Arbitrary Waveform and Arbitrary Sequence OutputMode

The EX1200-360x implements deep on-board memory to store large waveforms. Arbitrary waveforms can be created and stored in memory. These waveforms can then be combined sequentially by "linking and looping." Using this method, complex waveform sequences can be generated in a predetermined order. Using the example in Figure 2-7, the following section will describe how this is accomplished.

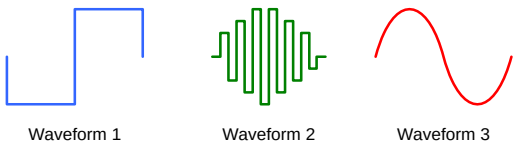


FIGURE 2-7: ARBITRARY WAVEFORMS

Linking and Looping

"Linking and looping" is the process by which waveforms and sequences are combined and repeated. The EX1200-360x provides for this functionality in two modes:

- Arbitrary waveform mode
- Arbitrary sequence mode

In ArbitraryWaveform mode, a single waveform is generated, but can be “looped”, or repeated, up to 65,536 times using the **Burst OperationMode** or looped indefinitely by using the **Continuous OperationMode** setting.

In ArbitrarySequence mode, the user can define multiple waveforms and combine, or “link”, them in any order. Each entry in the sequence must contain two parameters: waveform name and loop count. Each waveform also contains the marker position (see *Output Events* for more information).

Waveform	Loop Count
2	2x
1	1x
3	2x
1	3x

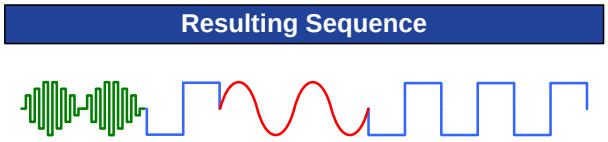


FIGURE 2-8: DEFINING A SEQUENCE WITH NAME AND LOOP COUNT

Shared Waveform and Instruction Memory

EX1200-360x uses the same physical memory for both waveform data and sequencing instructions for all channels. In most function generators, the memory for AWGs is small and separated from that of the waveform data memory, limiting the maximum number of waveforms that can be sequenced by the AWG and its overall flexibility. The EX1200-360x’s 4 MB of on-board memory stores data and sequencing instructions together, vastly expanding the space available for sequencing instructions. Shared memory provides the user the flexibility to use long sequences with small waveforms, short sequences with large waveforms, or any other combination.

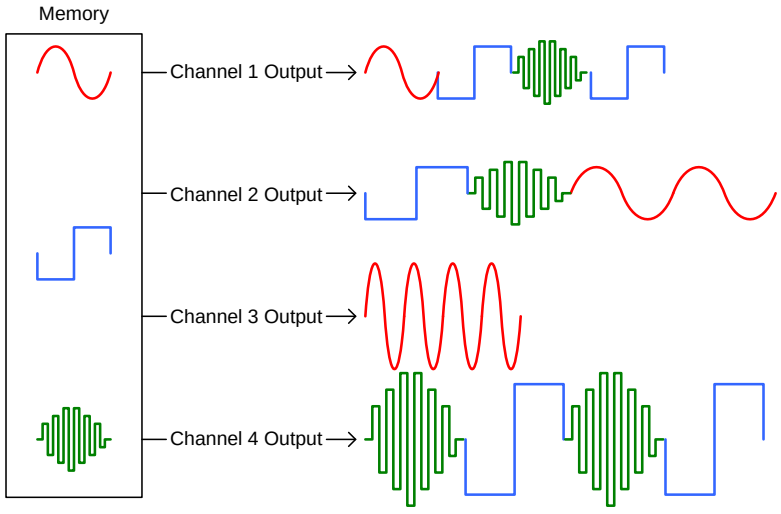


FIGURE 2-9:WAVEFORMS SHARED ACROSS CHANNELS

Note in Figure 2-9 that, although the waveforms are specifically defined in memory, the user can alter the amplitude and frequency on a per channel basis. This is done by adjusting the sample rate, offset, and gain for the channel.

Operation Mode

Operation Mode defines how the EX1200-360x behaves in response to internal and external triggers (for more information on triggers, see ***Trigger Sources***). This mode has three settings to control how waveform generation begins and ends: **Continuous**, **Burst**, and **Sequenced** (a fourth diagnostic setting, **SingleStep**, is also available). The user can use both hardware (such as front panel and backplane) and software triggers to initiate waveform generation.

Continuous

As defined by the IVI standard, the **Continuous** setting causes the entire waveform or sequence to be generated continuously. The EX1200-360x begins generation of the signal with the first waveform in the sequence. Once the waveform (or all waveform segments in a sequence) have been generated, the waveform is repeated again. This continues until generation is aborted by the user.

Figure 2-10 provides an example of how this works. A sequence comprised of three waves (Wave 1, Wave 2, and Wave 3) is set to Continuous mode. The output begins shortly after the trigger condition is satisfied. Here, rather than generating the sequence only once, the EX1200-360x loops the sequence until the generation session is stopped. Note that, at time t_2, the EX1200-360x loops back to the first waveform defined in the sequence and that the entire sequence repeats. Note that, in Continuous mode, no trigger is required to begin waveform/sequence generation. As soon as Continuous mode is enabled, generation begins.

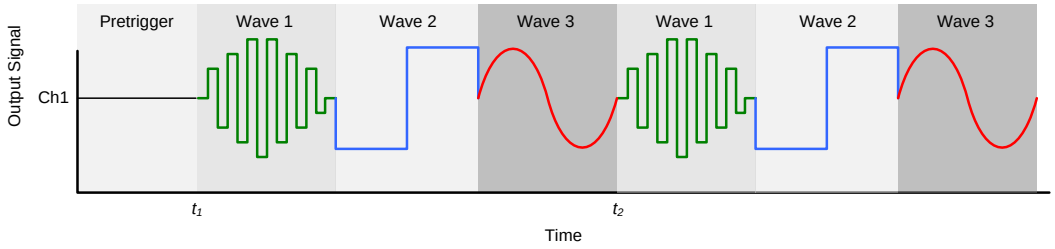


FIGURE 2-10: SEQUENCING USING THE CONTINUOUS TRIGGER

Burst

Like Continuous, **Burst** is defined by the IVI standard. In the Burst setting, the EX1200-360x generates one complete output (whether it's a standard/arbitrary waveform or sequence) *n* times (where *n* is equal to the **BurstCount**), stops, and then holds the last output value until acted upon by a trigger. At the end of a burst, the EX1200-360x holds the last value of the generated output. Essentially, this functions as a **Sequence** waveform that is run multiple times. Conversely, a Sequence can be thought of as a Burst waveform where the **BurstCount** is set to 1.

Sequenced

For the **Sequenced** setting, the EX1200-360x advances through each "sequence item" of the arbitrary sequence upon receipt of a trigger. As described in the *Arbitrary Waveform Generator (AWG)* discussion, a "sequence item" is defined by a waveform handle and a repeat count. If the repeat count of a sequence item is set to "0", that item will be repeated indefinitely until a trigger is received. If a trigger occurs after the waveform has been generated *n* times, then the next step in the sequence begins. Figure 2-11 shows a simple example where two waveforms are used in a sequence once and the trigger is received after the first waveform is completely generated. Note that, when Wave 1 and Wave 2 generation is completed, the last sample value of the wave continues to be driven until the next trigger is received.

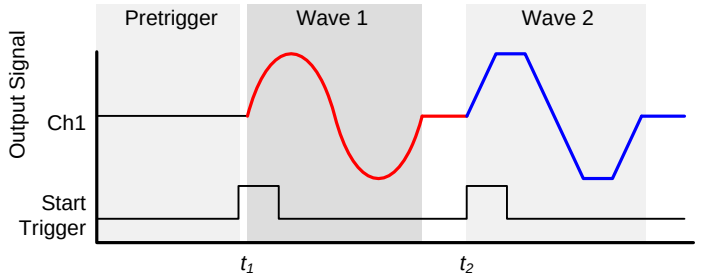


FIGURE 2-11: SEQUENCED MODE WITH TRIGGER SENT AFTER WAVEFORM COMPLETION

If, however, a trigger occurs during waveform generation, the behavior of the instrument depends on the SYNC/ASYNC setting for that channel. Advancement from one waveform to the next can be either SYNChronous or ASYNChronous When set to SYNC, the current waveform completes generation before advancing to the next waveform as shown in Figure 2-12.

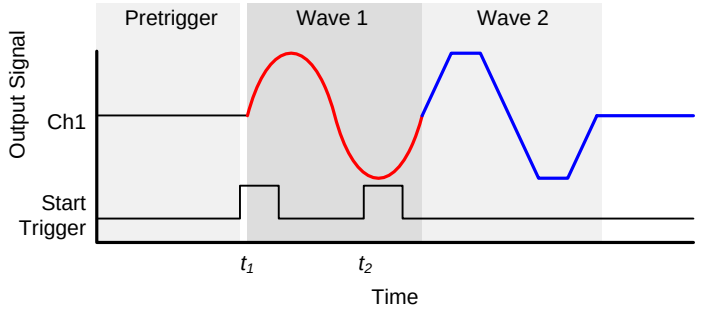


FIGURE 2-12: SYNCHRONOUS SEQUENCE MODE ADVANCEMENT

If the channel is set to ASYNC, generation of the current waveform is terminated and the next waveform in the sequence begins generation immediately. This is illustrated in Figure 2-13.

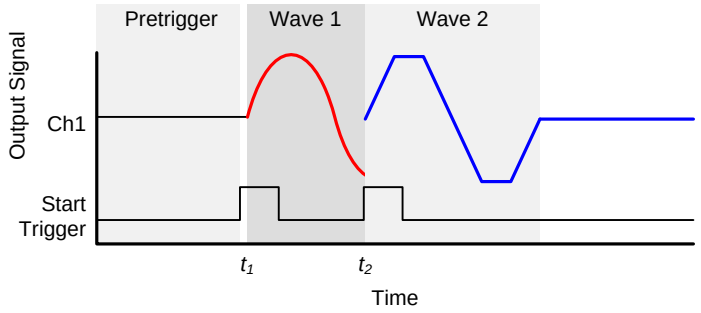


FIGURE 2-13: ASYNCHRONOUS SEQUENCE MODE ADVANCEMENT

As seen in preceding examples, the output starts on the rising edge of the start trigger. Note, however, that there is a slight delay between when the trigger occurs and when the channel 1 output begins. This delay is a function of the sampling rate and interpolation factors and is described in the specifications. Also, note that in this particular example, a sequence of two waveforms has been configured for generation. Once the last waveform in the sequence is generated, the EX1200-360x continues to drive the last sample value and will continue to do so until acted on by another trigger.

Single Step

The **SingleStep** diagnostic setting allows the user to use a software trigger to cause a single DAC update. All other trigger source settings are ignored when this setting is applied.

OUTPUT EVENTS

Events are output signals generated resulting response to changes internal to the EX1200-360x. These events are configured using the Events interface and are communicated over the EX1200 backplane through one of the BPL*n* trigger lines. The possible event sources are: WaitingForTrigger, Generating, BurstComplete, FrontPanel, and Marker, as seen in Figure 2-14. WaitingForTrigger and BurstComplete are set on a per channel basis, allowing the user to select a channel that will generate either of these signals. The front panel trigger input and the Marker output are also available as signal sources for the events interface. Events are output to one of the EX1200 backplane lines (BPL0 through BPL7) and may be inverted using the Event.Slope property**.**

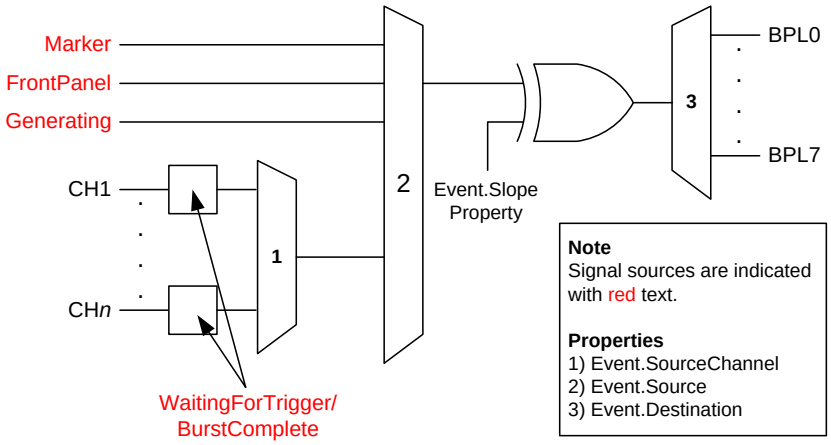


FIGURE 2-14: EVENT SELECTION

- **Marker**: Generates an event when a specific point on a waveform/sequence is encountered and follows the state of the marker output. See *Markers* for more information.
- **Front Panel**: Follows the state of the front panel trigger input.
- **Generation** state: Per the IVI standard, a function generator can be in one of the two states: CONFIGURATION (no waveforms are generated) or GENERATION (waveforms are being produced). When Generation is selected by the user, the "IsGenerating" signal becomes True on the backplane.
- **WaitingForTrigger**: This signal is True while in the Waiting For Trigger state and False after a trigger is received. Because this signal requires a triggerable mode, it is only applicable in Burst or Sequenced OperationMode.
- **BurstComplete**: When a channel is in BURST mode, the BurstComplete signal goes True on the backplane when the burst is done. This signal remains until a new sequence is initiated.

Markers

Marker are special events that are particularly useful synchronizing multiple instruments. The user can configure the MARKER_OUT line on the front panel to change its state synchronously with a configured sample number. Thus, a marker is a specific point on waveform/sequence that generates the MARKER_OUT signal allowing for communication to other instruments. Also note that a

marker output can be used as an input for the Event output, as the maker output and event outputs are independent.

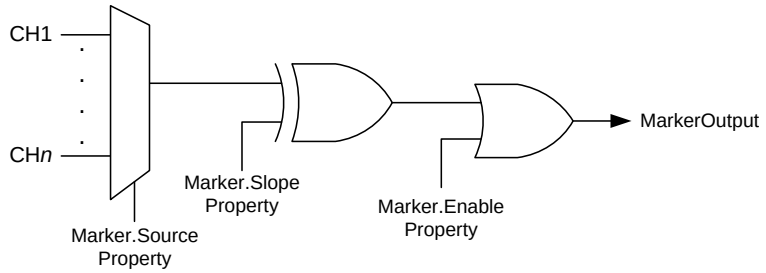


FIGURE 2-15: MARKER SELECTION

Marker events are specified by defining an offset count, measured in samples, from the beginning of the waveform. In sequence mode, one marker event is configured for each waveform in the sequence. The user can also program the length of the pulse generated by the marker. This duration is programmed in seconds. The shortest duration is 20 ns.

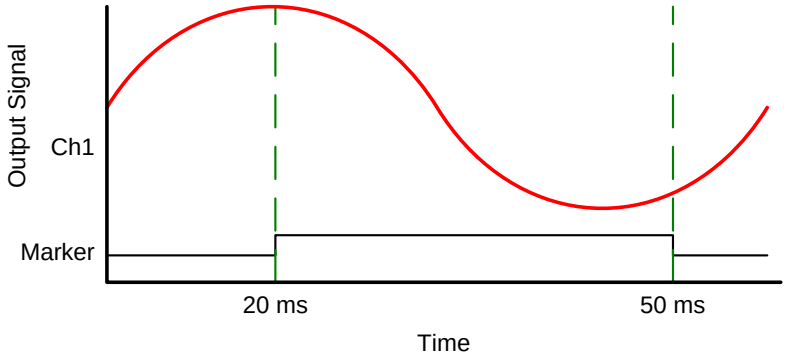


FIGURE 2-16: MARKER EVENT TIMING

TRIGGERS

Triggers are inputs to the EX1200-3608 that cause an action to occur. Triggers can be routed to either the backplane or the front panel (not currently available). As seen in Figure 2-17, each channel can independently select one of four different trigger sources:

1) Front panel trigger input
2) Internal trigger
3) Backplane (BPL0 – BPL7)
4) Software

For more information on triggers, refer to the *Trigger Sources* discussion in Section 3.

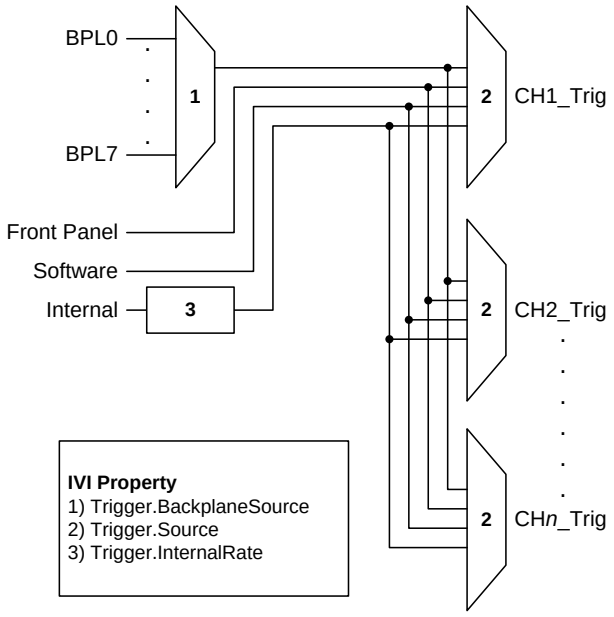


FIGURE 2-17: TRIGGER SOURCE DIAGRAM

GAIN AND OFFSET

The EX1200-360x AWG data provided in an IVI-supported format: values are normalized (-1.0 through +1.0). The user then provides an offset and gain. The normalized AWG values are processed in real-time to generate the desired amplitude and offset for the signal. The raw, normalized value is multiplied by the user-defined gain value. The user-defined offset is then added. This new "User AWG Value" is then passed through the internal calibration unit to compensate for analog circuitry non-linearities. Offset and gain can be changed during AWG generation.

A block diagram of how the data flows and is manipulated is presented in Figure 2-18.

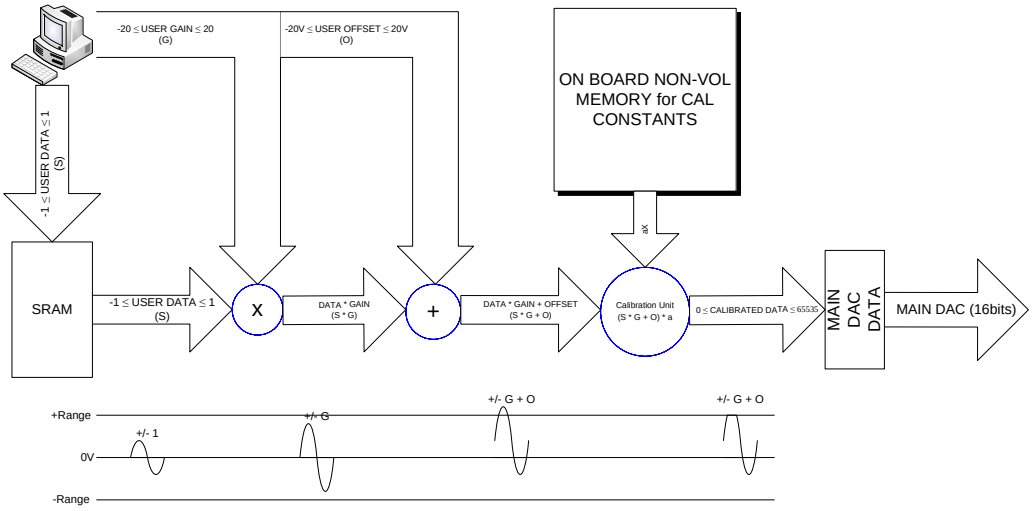


FIGURE 2-18: GAIN AND OFFSET PROCESS

Note that, if the user specifies a gain and/or offset that would create a waveform larger than the supported range, the waveform is "clipped" internally (digital format) at the maximum/minimum value.

CALIBRATION

Every EX1200-360x is factory calibrated using NIST-traceable standards. The EX1200-360x has an on-board calibration reference that corrects for environmental effects on dc gain and offset. Optionally, the EX1200-360x can be returned the factory for a complete factory calibration. VTI recommends annual factory calibration of the EX1200-360x.

SUMMARY AND APPLICATIONS

With the advanced waveform triggering and sequencing features of the EX1200-360x function generator, complex waveforms can be generated and synchronization can be achieved between multiple instruments. Moreover, the synchronized instruments are in the same physical LXI system, allowing for complex test signals to be implemented that are tightly synchronized with multiple instruments.

USING THE SOFT FRONT PANEL

Through the embedded web interface of the EX1200 system, a soft front panel (SFP) can be accessed for the EX1200-360x which can be used to operate the instrument modifying an easy-to-use graphic user interface (GUI). While not all instrument functionality is exposed in the SFP, it can be used to set up the instrument for most applications. For more information on using the SFP, see *0* of this manual.

BPL_INSFAIL BEHAVIOR

The EX1200 platform backplane has a BPL_INSFAIL line that indicates to all modules that a severe failure has occurred. When this line is asserted, some modules will put themselves into a known state, such as opening all relays. The EX1200-360x does not respond to this signal.

SECTION 3

PROGRAMMING THE INSTRUMENT

RELATED SOFTWARE COMPONENTS

IVI-COM Driver
IVI-C Driver
LabView Driver
Linux C++ Driver

USING THE DRIVER

The EX1200-360x may be used in a variety of environments including: Visual Basic, C#, C++, LabView. VTI Instruments provides an IVI-C and IVI-COM compliant driver as well as a shared object that can be used on Linux systems that comply with the Linux Standard Base (Version 3.1).

Here is how to use the driver in each environment:

1) **Visual Studio C++**
```
#import "IviDriverTypeLib.dll" no_namespace
#import "VTEXFgen.dll" no_namespace
```

2) **C#**
Add a reference to VTEXFgen.dll in the project. Include the following at the top of any code file that will access the driver:

```
using VTI.VTEXFgen.Interop;
```

3) **C/C++ on Windows**
Link against VTEXFgen.lib and include VTEXFgen.h in the file.

4) **C++ on Linux**
Link against /opt/vti/lib/libfgen.so and include all the headers in /opt/vti/include in the source file.

5) **LabView**
Copy the driver package to the <Labview>/instr.lib directory and access all relevant VIs

PROGRAMMING OVERVIEW

The EX1200-360x has three basic modes which need to be defined for each channel: **Drive Mode**, **Output Mode**, and **Operation Mode**. **Drive Mode** dictates whether a channel will generate a voltage or current signal. **Output Mode** selects how the channel's actual output level is determined: **Function** provides pre-programmed standard waveforms functions; **Arbitrary** allows the user to program any set of output levels; **Sequence** provides the ability to run several of the **Arbitrary** waveforms, one after another, without having to cease generation to reprogram the channel. **Operation Mode** determines when a channel's output is generated, whether it waits for triggers, outputs continuously, etc.

INITIALIZING\CLOSING THE INSTRUMENT

The base interface of the EX1200-360x IVI driver, VTEXFgen (LibFgen on Linux), is used to open and close connections to the instrument as well as containing pointers to all other interfaces to access the functionality of the instrument.

A coding example is provided in the *Initialization* discussion later in the *Programming Reference* section.

Option Strings

The VTEX drivers provide option strings that can be used when Initializing an instrument. The option string values exist to change the behavior of the driver. The following options strings are available on VTI IVI drivers:

- **Simulate**: Allows the user to run a program without commanding switch card or instruments. This option is useful as a debugging tool.
- **Cache**: Per the IVI specification, this option "specifies whether or not to cache the value of attributes." Caching allows IVI drivers to maintain certain instrument settings to avoid sending redundant commands. The standard allows for certain values to be cached always or never. In VTI IVI-drivers, all values used are of one of these types. As such, any values entered have no effect.
- **QueryInstrumentStatus**: Queries the instrument for errors after each call is made. As implemented in the VTI IVI drivers, instruments status is always queried regardless of the value of this property.
- **DriverSetup**: Must be last, and contains the following properties:
  - **Logfile**: Allows the user to specify a file to which the driver can log calls and other data.
  - **Logmode**: Specifies the mode in which the log file is opened. The allowed modes are:
    - **w**: truncate s the file to zero length or creates a text file for writing.
    - **a**: opens the file for adding information to the end of the file. The file is created if it does not exist. The stream is positioned at the end of the file.
  - **LogLevel**: Allows the user to determine the severity of a log message by providing a level-indicator to the log entry.
  - **Slots**: This is the most commonly used option and it allows for a slot number or a slot number and a card model to be specified.
    "Slots=(2)" - Just slot 2.
    "Slots=(2=EX1200_3048)" - slot and card model
    "Slots=(2,3)" - Multiple slots
- **InterchangeCheck**: Boolean option that enables/disables IVI Interchangeability checking. As implemented in the VTI IVI drivers, values entered for this property have no effect.
- **RangeCheck**: Boolean option that enables or disables driver validation of user-submitted values. As implemented in the VTI IVI drivers, validation of user inputs is always performed at the firmware level regardless of this property's value.
- **RecordCoercions**: Boolean option that enables driver recording of coercions. As implemented in the VTI IVI drivers, coercions are handled in the firmware and cannot be recorded.

STANDARD WAVEFORM (FUNCTION) OUTPUTMODE

Standard Waveform Mode is the interface to the "four function calculator" of waveform generation. The specification describes six waveforms: Sine, Triangle, Ramp Up, Ramp Down, Square, and DC.

The properties available in this mode are:

- Waveform (type)
- Amplitude
- DCOffset

- StartPhase
- DutyCycle

Coding examples are provided in the *Standard Waveforms* and *DC Offset* discussions later in the *Programming Reference* section.

ARBITRARY WAVEFORM OUTPUTMODE

In this mode, the user can supply an arbitrary waveform.

Using this mode requires two steps:

1) Define one or more waveforms using the Arbitrary.Waveform.Create () method.
2) Configure a channel to use a sequence using the Arbitrary.Waveform.Configure () method and a waveform from the previous step.

A waveform consists of an array of 64-bit floating point values in the range of -1.0 to +1.0. Internally in the device, the waveform will be scaled and rounded to a 16bit fractional format

Waveforms can be of any length up to the limits of SRAM. It is possible to use all of SRAM for one waveform.

The same waveform can be assigned to multiple channels.

Waveforms can be deleted if they are not in use. Once deleted, a waveform will have to be uploaded to the memory again later on, if the user desires to use it again.

It is not possible the edit or modify a waveform after it has been uploaded. To accomplish this, the user should modify their original data, delete the waveform in SRAM, and upload the modified version.

A waveform can be used my any number of channels at the same time, independently of the configuration of other channels.

A coding example is provided in the *Arbitrary Waveforms* discussion later in the *Programming Reference* section.

ARBITRARY SEQUENCE OUTPUTMODE

Sequence mode extends Arbitrary mode by allowing a “sequence” to be defined, which is a list of waveforms and an associated list of repeat counts.

Using this mode is a three stage process.

1) Define one or more waveforms using the Arbitrary.Waveform.Create() method.
2) Define one or more sequences using the Arbitrary.Sequence.Create() method and the waveforms from the previous step.
3) Configure a channel to use a sequence using the Arbitrary.Sequence.Configure() method and a sequence from the previous step.

Sequences share the same SRAM space as waveforms. Sequences can also be used by any number of channels at the same time, independently of the configuration of other channels.

The same restrictions on deleting and editing waveforms apply to sequences as well.

A coding example is provided in the *Arbitrary Sequence* discussion later in the *Programming Reference* section.

BURST OPERATIONMODE

Any of the above output modes can be generated as a trigger initiated burst. In this mode, a trigger initiates a “burst”, which is a "wave" repeated a specific number of times.

In Arbitrary Sequence mode, the whole arbitrary sequence is repeated BurstCount times.

In Arbitrary Waveform mode, the waveform is repeated BurstCount times.

In Standard Waveform mode, the selected standard waveform is repeated BurstCount times.

SEQUENCED OPERATIONMODE

This is an extension to the specification. It only applies to Sequence mode. The application of a trigger causes the sequence to advance to the next item in the sequence. If the trigger happens after the waveform is generated repeat-count times, then the next item in the sequence starts. If the trigger happens while a waveform is being generated, what happens depends on the AsyncAdvance flag. If the flag is not set, the current waveform is completely generated before advancing. If the flag is set, the current data point is finished before advancing.

The repeat counts in the sequence can be 0, which means repeat that sequence item until a trigger.

SINGLESTEP OPERATIONMODE

This VTI Extension allows the SendSoftwareTrigger() to single step though the Standard Waveform, Arbitrary Waveform, or Arbitrary Sequence.

In this operation mode, all trigger sources settings are ignored and the Software Trigger causes a single DAC update.

TRIGGER SOURCES

The device has a very flexible trigger routing model. Each channel can independently select from one of five trigger sources. Internal, External (Front Panel), Software, and Backplane. All channels connected to the same trigger source will activate at the same time.

The Internal trigger has a rate that can be adjusted. It is enabled and disabled through the AbortGeneration() and InitiateGeneration() calls, and its rate can be set using the Trigger.InternalRate property.

The External trigger is from the front panel connector.

The Backplane trigger can be sourced from any one of the eight BPL trigger lines. Set the Trigger.BackplaneSource property to choose which backplane line to route to the module.

The slope of the trigger can be selected on channel. See Trigger.Slope.

When Software is selected, the user’s program calls the Trigger.SendSoftwareTrigger() function to trigger the channel.

A coding example is provided in the *Trigger* discussion later in the *Programming Reference* section.

MARKERS

Each waveform has a single point along it called the “mark”. There is a marker output on the front panel connector that can be associated with a channel. When the associated channel is generating, the marker output will pulse when the “marked” data sample is generated.

The width and the polarity of the marker output can be adjusted though the Marker.Width and Marker Polarity properties.

The default mark is at the first sample of the waveform (sample 0), and can be adjusted through the Marker.Position property.

A coding example is provided in the *Marker* discussion later in the *Programming Reference* section.

EVENTS

The device has the ability to route a signal to a backplane trigger line. To differentiate these signals from "triggers" which are inputs, they are called "events". This signal can then be used through the System driver LxiSync interfaces to trigger another LXI device via Wired Trigger Bus or LAN Event.

Some of the possible Event sources are:

- ExternalTrigger
- Marker
- BurstComplete
- GenerationMode
- WaitingForTrigger

EX1200 INTERNAL DMM

It is possible to route any of the EX1200-360x's output channels to the EX1200 platform's internal DMM. To accomplish this, set the **Calibration.RouteToInternalDmm** property for the desired channel to either **Voltage** or **Current**, depending on which inputs of the DMM should be connected (**HI** and **LO**, or **I** and **LO**, respectively). Set the property to **None** to disconnect the channel from the DMM again. See the *EX1200 Series User Manual* for information on programming the internal DMM to take a measurement.

SECTION 4

PROGRAMMING REFERENCE

INTRODUCTION

This section provides a list of the functions available on the EX1200-360x. Additional information can be found in the driver help file. If the instrument will be used on a Linux system, a .chm viewer must be installed on the host PC (examples of these programs can be found at the following URL: http://www.linux.com/news/software/applications/8209-chm-viewers-for-linux.)

INTERFACE HIERARCHY

COM Interface Name	Generic Name	Type
InitiateGeneration	Initiate Generation	M
AbortGeneration	Abort Generation	M
Output		
ChannelOutputMode[channel]	Channel Output Mode	P
Count	Count	P
DriveMode[channel]	Output Drive Mode	P
Enabled[channel]	Output Enabled	P
Filter[channel]	Output Filter	P
Impedance[channel]	Output Impedance	P
OperationMode[channel]	Operation Mode	P
OutputMode	Output Mode	P
Range[channel]	Output Range	P
ReferenceClockSource	Reference Clock Source	P
RemoteSense[channel]	Output Remote Sense	P
Value[channel]	Output Value	P
Trigger		
SendSoftwareTrigger	Send Software Trigger	M
BackplaneSource	Backplane Source	P
BurstCount[channel]	Burst Count	P
InternalRate	Internal Trigger Rate	P
Slope[channel]	Trigger Slope	P
Source[channel]	Trigger Source	P
Event		
Destination	Event Destination	P
Slope	Event Slope	P
Source	Event Source	P
SourceChannel	Event Source Channel	P
Marker		
Enable	Marker Enable	P
Position[wfmHandle]	Marker Position	P
Slope	Marker Slope	P
Source	Marker Source	P
Width	Marker Width	P
StandardWaveform		
Configure	Configure Standard Waveform	M
Amplitude[channel]	Amplitude	P
DCOffset[channel]	DC Offset	P

COM Interface Name	Generic Name	Type
DutyCycleHigh[channel]	Duty Cycle High	P
Frequency[channel]	Frequency	P
StartPhase[channel]	Start Phase	P
Waveform[channel]	Waveform	P
Arbitrary		
ClearMemory	Clear Arbitrary Memory	M
ChannelSampleRate[channel]	Arbitrary Channel Sample Rate	P
Gain[channel]	Arbitrary Gain	P
Offset[channel]	Arbitrary Offset	P
SampleRate	Arbitrary Sample Rate	P
Waveform		
Configure	Configure Arbitrary Waveform	M
Clear	Clear Arbitrary Waveform	M
Create	Create Arbitrary Waveform	M
GetList	Get Arbitrary Waveform List	M
Retrieve	Retrieve Arbitrary Waveform	M
Frequency[channel]	Arbitrary Frequency	P
Handle[channel]	Arbitrary Handle	P
NumberOfWaveformsMax	Number Waveforms Max	P
Quantum	Waveform Quantum	P
SizeMax	Waveform Size Max	P
SizeMin	Waveform Size Min	P
Sequence		
Configure	Configure Arbitrary Sequence	M
Clear	Clear Arbitrary Sequence	M
GetList	Get Arbitrary Sequence List	M
Create	Create Arbitrary Sequence	M
Retrieve	Retrieve Arbitrary Sequence	M
AsyncAdvance[channel]	Async Advance	P
Handle[channel]	Sequence Handle	P
LengthMax	Sequence Length Max	P
LengthMin	Sequence Length Min	P
LoopCountMax	Loop Count Max	P
NumberSequencesMax	Number Sequences Max	P
Calibration		
Get	Get Calibration Constant	M
LinearFit	Calibration Linear Fit	M
Load	Load Calibration	M
Read	Calibration Read	M
Save	Save Calibration	M
Set	Set Calibration Constant	M
Write	Calibration Write	M
RawRac[channel]	Raw Dac	P
RawOffsetDac[channel]	Raw Offset Dac	P

COMMON

Common Methods.

Methods

HRESULT AbortGeneration();

Aborts signal generation and moves to Configuration State.

Channels that are currently generating waveforms stop generating.

HRESULT InitiateGeneration();

Moves to Generation State and initiates signal generation.

Channels that have OperationMode configured to a triggered mode enter the WaitingForTrigger state.

Any methods or properties that need to access SRAM will return an error in the Generation state.

OUTPUT

Properties and methods in the Output interface.

Properties

Output.ChannelOutputMode(ChannelName) - VTI Extension

Determines how the channel generator produces waveforms. This attribute determines which extension group's functions and attributes are used to configure the waveform the channel produces.

- Output Function
- Output Arbitrary
- Output Sequence

The enum values are

```
enum VTEXFgenOutputModeEnum:
     VTEXFgenOutputModeFunction(0),
     VTEXFgenOutputModeArbitrary(1),
     VTEXFgenOutputModeSequence(2)
```

Output.Count

Returns the number of available output channels.

Read Only.

Output.DriveMode(ChannelName) - VTI Extension

Specifies the output drive mode.

- Voltage
- Current

```
enum VTEXFgenDriveModeEnum:
     VTEXFgenDriveModeVoltage(1000),
     VTEXFgenDriveModeCurrent(1001)
```

Output.Enabled(ChannelName)

Specifies whether the signal the function generator produces appears at the output connector.

Output.Filter(ChannelName) - VTI Extension

Specifies if the low-pass filer should be enabled.

This is bit 16 in the Channel Control Register in the fpga.

Output.Impedance(ChannelName)

Specifies the channel output impedance.

The returned value is always 0.0. Attempting to set any value other tan 0.0 is an INVALID VALUE error.

Output.OperationMode(ChannelName)

Specifies how the function generator produces output on a channel.

- Continuous
- Burst
- Sequenced
- SingleStep

```
enum VTEXFgenOperationModeEnum:
     VTEXFgenOperationModeContinuous(0, “Continuous”),
     VTEXFgenOperationModeBurst(1, “Burst”),
     VTEXFgenOperationModeSequenced(1000, “Sequenced”),
     VTEXFgenOperationModeSingleStep(1001, “SingleStep”)
```

Output.OutputMode

Determines how the function generator produces waveforms. This attribute determines which extension group’s functions and attributes are used to configure the waveform the function generator produces.

- Output Function
- Output Arbitrary
- Output Sequence

See also Output.OutputMode(ChannelName).

Output.Range(ChannelName) - VTI Extension

Specifies the Range of the channel.

The device has six voltage ranges and three current ranges. The possible Range are:

Range	Polarity
1 V	Bipolar
2 V	Bipolar
5 V	Bipolar
10 V	Bipolar
20 V	Bipolar
40 V	Unipolar
5 mA	Bipolar
10 mA	Bipolar
20 mA	Bipolar

Separate ranges are used for Voltage DriveMode and Current DriveMode.

A negative range causes the device to go into autorange mode. When in autorange mode, the StandardWaveform.Amplitide and StandardWaveform.DCOffset or Arbitrary.Gain and Arbitrary.Offset combined with the configured waveforms or sequences determines the actual range.

Changing the range does not alter the user selected amplitudes, gains, and offsets. The output will be clamped to the hardware limits of the selected range if the users desired values places the output beyond those limits.

Output.RemoteSense(ChannelName) - VTI Extension

Specifies if the remote sense lines are enabled.

Output.Value(ChannelName) - VTI Extension

The instantaneous actual output value of the DAC channel in engineering units. (That is in volts or amps.)

STANDARD WAVEFORM

Properties and methods of the StandardWaveform interface.

Properties

StandardWaveform.Amplitude(ChannelName)

Specifies the amplitude of the standard waveform the function generator produces. When the Waveform attribute is set to Waveform DC, this attribute does not affect signal output. The units are volts.

VTI Extension: When the DriveMode is configured for Current output, the units are in amps.

StandardWaveform.DCOffset(ChannelName)

Specifies the DC offset of the standard waveform the function generator produces. If the Waveform attribute is set to Waveform DC, this attribute specifies the DC level the function generator produces. The units are volts.

VTI Extension: When the DriveMode is configured for Current output, the units are in amps.

StandardWaveform.DutyCycleHigh(ChannelName)

Specifies the duty cycle for a square waveform. This attribute affects function generator behavior only when the Waveform attribute is set to Waveform Square. The value is expressed as a percentage.

StandardWaveform.Frequency(ChannelName)

Specifies the frequency of the standard waveform the function generator produces. When the Waveform attribute is set to Waveform DC, this attribute does not affect signal output. The units are Hertz.

StandardWaveform.StartPhase(ChannelName)

Specifies the start phase of the standard waveform the function generator produces. When the Waveform attribute is set to Waveform DC, this attribute does not affect signal output. The units are degrees.

StandardWaveform.Waveform(ChannelName)

Specifies which standard waveform the function generator produces.

- Sine
- Square
- Triangle
- Ramp Up
- Ramp Down
- DC

```
enum VTEXFgenWaveformEnum:
     VTEXFgenWaveformSine(1),
     VTEXFgenWaveformSquare(2),
     VTEXFgenWaveformTriangle(3),
     VTEXFgenWaveformRampUp(4),
```

```
VTEXFgenWaveformRampDown(5),
VTEXFgenWaveformDC(6)
```

Methods

HRESULT StandardWaveform.Configure(ChannelName, Waveform, Amplitude, DCOffset, Frequenecy, StartPhase);

This function configures the attributes of the function generator that affect standard waveform generation. These attributes are the Waveform, Amplitude, DC Offset, Frequency, and Start Phase.

When the Waveform parameter is set to Waveform DC, this function ignores the Amplitude, Frequency, and Start Phase parameters and does not set the Amplitude, Frequency, and Start Phase attributes.

ARBITRARY

Properties and methods in the Arbitrary interface.

Properties

Arbitrary.ChannelSampleRate(ChannelName) - VTI Extension

Specifies the channel-specific sample rate.

See Arbitrary.SampleRate

Arbitrary.Gain(ChannelName)

Specifies the gain of the arbitrary waveform the function generator produces. This value is unitless.

Arbitrary.Offset(ChannelName)

Specifies the offset of the arbitrary waveform the function generator produces. The units are volts.

VTI Extension: When the DriveMode is configured for Current output, the units are in amps.

Arbitrary.SampleRate

Specifies the sample rate of the arbitrary waveforms the function generator produces. The units are samples per second.

See Arbitrary.Waveform.ChannelSampleRate().

Methods

HRESULT Arbitrary.ClearMemory();

Removes all previously created arbitrary waveforms and sequences from the function generator's memory and invalidates all waveform and sequence handles.

If a waveform cannot be cleared because it is currently being generated, this function returns the error Waveform In Use.

If a sequence cannot be cleared because it is currently being generated, this function returns the error Sequence In Use.

ARBITRARY WAVEFORM

Properties and methods of the Arbitrary Waveform interface.

Properties

Arbitrary.Waveform.Frequencey(ChannelName)

Specifies the rate in Hertz at which an entire arbitrary waveform is generated.

Arbitrary.Waveform.Handle(ChannelName) - VTI Extension

Read Only.

Retrieves the current waveform handle.

Arbitrary.Waveform.NumberWaveformsMax

Returns the maximum number of arbitrary waveforms that the function generator allows.

Read Only.

Returned value depends on available free memory.

Arbitrary.Waveform.Quantum

The size of each arbitrary waveform shall be a multiple of a quantum value. This attribute returns the quantum value the function generator allows. For example, if this attribute returns a value of 8, all waveform sizes must be a multiple of 8.

Read Only.

The value is 2.

Arbitrary.Waveform.SizeMax

Returns the maximum number of points the function generator allows in an arbitrary waveform.

Read Only.

Returned value depends on available free memory.

Arbitrary.Waveform.SizeMin

Returns the minimum number of points the function generator allows in an arbitrary waveform.

Read Only.

The value is 4.

Methods

HRESULT Arbitrary.Waveform.Clear(WfmHandle);

Removes a previously created arbitrary waveform from the function generator's memory and invalidates the waveform's handle.

If the waveform cannot be cleared because it is currently being generated, or it is specified as part of an existing arbitrary waveform sequence, this function returns the Waveform In Use error.

HRESULT Arbitrary.Waveform.Configure(ChannelName, WfmHandle, Gain, Offset);

Configures the attributes of the function generator that affect arbitrary waveform generation. These attributes are the arbitrary waveform handle, gain, and offset.

This is the only function that assigns a waveform to a channel. Note that the waveform must already have been downloaded using Arbitrary.Waveform.Create.

HRESULT Arbitrary.Waveform.Create(Data[], WfmHandle);

Creates an arbitrary waveform and returns a handle that identifies that waveform. You pass a waveform handle as the `WfmHandle` parameter of the Configure Arbitrary Waveform function to produce that waveform. You also use the handles this function returns to create a sequence of arbitrary waveforms with the Create Arbitrary Sequence function.

If the function generator cannot store any more arbitrary waveforms, this function returns the error No Waveforms Available.

The *Data* parameter specifies the array of data to use for the new arbitrary waveform. The array's elements must be normalized between -1.00 and +1.00.

HRESULT Arbitrary.Waveform.GetList([out] SAVEARRAY(int32) *WaveHandles); - VTI Extension

Retrieves the list of created waveform handles.

HRESULT Arbitrary.Waveform.Retrieve([in] LONG waveHandle, [out] SAFEARRAY(double) *Data); - VTI Extension

Retrieve the waveform data from the waveform handle.

ARBITRARY SEQUENCE

Properties

Arbitrary.Sequence.AsyncAdvance(ChannelName) - VTI Extension

Specifies the how the Sequence will advance on a trigger when in Sequenced OperationMode.

If True, the sequence will advance after the current sample finishes.

If False, the sequence will advance after the current waveform finishes.

Arbitrary.Sequence.LengthMax

Returns the maximum number of arbitrary waveforms that the function generator allows in an arbitrary sequence.

Read Only.

Returned value depends on available free memory.

Arbitrary.Sequence.LengthMin

Returns the minimum number of arbitrary waveforms that the function generator allows in an arbitrary sequence.

Read Only.

The minimum sequence length is 1.

Arbitrary.Sequence.LoopCountMax

Returns the maximum number of times that the function generator can repeat a waveform in a sequence.

Read Only.

Currently 0xffff.

Arbitrary.Sequence.NumberSequencesMax

Returns the maximum number of arbitrary sequences that the function generator allows.

Read Only.

Returned value depends on available free memory.

Methods

HRESULT Arbitrary.Sequence.Clear(SeqHandle);

Removes a previously created arbitrary sequence from the function generator's memory and invalidates the sequence's handle.

If the sequence cannot be cleared because it is currently being generated, this function returns the error Sequence In Use.

HRESULT Arbitrary.Sequence.Configure(ChannelName, SeqHandle, Gain, Offset);

Configures the attributes of the function generator that affect arbitrary sequence generation. These attributes are the arbitrary sequence handle, gain, and offset.

This is the only function that assigns a sequence to a channel. Note that the sequence and waveforms must already have been downloaded using Arbitrary.Sequence.Create and Arbitrary.Waveform.Create.

HRESULT Arbitrary.Sequence.Create(WfmHandle[], LoopCount[], SeqHandle);

Creates an arbitrary waveform sequence from an array of waveform handles and a corresponding array of loop counts. The function returns a handle that identifies the sequence. You pass a sequence handle to the `Handle` parameter of the *Configure Arbitrary Sequence* function to produce that sequence.

If the function generator cannot store any more arbitrary sequences, this function returns the error No Sequences Available.

HRESULT Arbitrary.Sequence.GetList([out] SAVEARRAY(int32) *sequenceHandles); - VTI Extension

Retrieves the list of created sequence handles.

HRESULT Arbitrary.Sequence.Retrieve([in] LONG sequenceHandle, [out] SAFEARRAY(int32) *waveHandles, [out] SAFEARRAY(int32) *repeatCounts); - VTI Extension

Retrieve the waveform handles and repeats counts in the sequence.

TRIGGER

Trigger.BackplaneSource - VTI Extension

Specifies the BPL trigger line that is the source for the Backplane Trigger.Source

```
enum VTEXFgenBackplaneSourceEnum:
     VTEXFgenBackplaneSourceNone(0, "None"),
     VTEXFgenBackplaneSourceBPL0(1000, "BPL0"),
     VTEXFgenBackplaneSourceBPL1(1001, "BPL1"),
     VTEXFgenBackplaneSourceBPL2(1002, "BPL2"),
```

```
     VTEXFgenBackplaneSourceBPL3(1003, "BPL3"),
     VTEXFgenBackplaneSourceBPL4(1004, "BPL4"),
     VTEXFgenBackplaneSourceBPL5(1005, "BPL5"),
     VTEXFgenBackplaneSourceBPL6(1006, "BPL6"),
     VTEXFgenBackplaneSourceBPL7(1007, "BPL7")
```

Trigger.BurstCount(ChannelName)

Specifies the number of waveform cycles that the function generator produces after it receives a trigger.

Trigger.InternalRate

Specifies the rate at which the function generator's internal trigger source produces a trigger, in triggers per second.

Trigger.Slope(ChannelName) - VTI Extension

Specifies the Slope of the channels trigger input.

```
enum VTEXFgenTriggerSlopeEnum:
     VTEXFgenTriggerSlopeRise(1000, "Rise"),
     VTEXFgenTriggerSlopeFall(1001, "Fall")
```

Trigger.Source(ChannelName)

Specifies the trigger source. After the function generator receives a trigger from this source, it produces a signal.

- Internal
- External
- Software
- Backplane (VTI Extension)

```
enum VTEXFgenTriggerSourceEnum:
     VTEXFgenTriggerSourceExternal(1),
     VTEXFgenTriggerSourceSoftware(2),
     VTEXFgenTriggerSourceInternal(3),
     VTEXFgenTriggerSourceBackplane(1000)
```

The "internal" trigger source is a clock maintained by the card. The "external" trigger source is the front panel connector.

HRESULT Trigger.SendSoftwareTrigger();

Any channel that is configured for Software trigger and is in the WaitingForTrigger state is activated.

EVENT

Event.Destination - VTI Extention

Specifies the backplane trigger line that the Event will be routed to.

```
enum VTEXFgenBackplaneDestinationEnum:
     VTEXFgenBackplaneDestinationNone(0, "None"),
     VTEXFgenBackplaneDestinationBPL0(1000, "BPL0"),
     VTEXFgenBackplaneDestinationBPL1(1001, "BPL1"),
     VTEXFgenBackplaneDestinationBPL2(1002, "BPL2"),
     VTEXFgenBackplaneDestinationBPL3(1003, "BPL3"),
     VTEXFgenBackplaneDestinationBPL4(1004, "BPL4"),
     VTEXFgenBackplaneDestinationBPL5(1005, "BPL5"),
     VTEXFgenBackplaneDestinationBPL6(1006, "BPL6"),
     VTEXFgenBackplaneDestinationBPL7(1007, "BPL7")
```

Event.Slope - VTI Extension

Specifies if the Event output edge is Rising or Falling.

Event.Source - VTI Extension

Specifies the Event signal that drives the backplane trigger line.

- Marker
- FrontPanel
- WaitingForTrigger
- BurstComplete
- Generating

```
enum VTEXFgenEventEnum:
        VTEXFgenEventMarker(1000, "Marker"),
        VTEXFgenEventFrontPanel(1001, "Front Panel"),
        VTEXFgenEventBurstComplete(1002, "Burst Complete"),
        VTEXFgenEventWaitingForTrigger(1003, "Waiting For
Trigger"),
        VTEXFgenEventGenerating(1004, "Generating")
```

Event.SourceChannel - VTI Extension

Specifies the channel that provides the signals for Event.Source.

MARKER

Marker.Enable - VTI Extension

Specifies if the marker output should be enabled.

Marker.Position(WfmHandle) - VTI Extension

Specifies the marker position for a waveform. By default, the marker position is at 0.

Marker.Slope - VTI Extension

Specifies if the Marker output edge is Rising or Falling.

Marker.Source - VTI Extension

Specifies the channel that drives the marker output.

Marker.Width - VTI Extension

Sets the width of the marker pulse, in seconds.

APPLICATION EXAMPLES

The following examples provide a basis for understanding how the EX1200-360x APIs can be used together to program the instrument. These examples are written in C++ can be found in the IVI driver directory (typically C:\Program Files\IVI Foundation\IVI\Drivers\VtexFgen\ Examples).

INITIALIZATION

```
#include "stdafx.h"

#using <mscorlib.dll>

using namespace System::Reflection;
using namespace System::Runtime::CompilerServices;

//
// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
//
[assembly:AssemblyTitleAttribute("")];
[assembly:AssemblyDescriptionAttribute("")];
[assembly:AssemblyConfigurationAttribute("")];
[assembly:AssemblyCompanyAttribute("")];
[assembly:AssemblyProductAttribute("")];
[assembly:AssemblyCopyrightAttribute("")];
[assembly:AssemblyTrademarkAttribute("")];
[assembly:AssemblyCultureAttribute("")];

//
// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the value or you can default the Revision and Build Numbers
// by using the '*' as shown below:

[assembly:AssemblyVersionAttribute("1.0.*")];

//
// In order to sign your assembly you must specify a key to use. Refer to the
// Microsoft .NET Framework documentation for more information on assembly signing.
//
// Use the attributes below to control which key is used for signing.
//
// Notes:
//   (*) If no key is specified, the assembly is not signed.
//   (*) KeyName refers to a key that has been installed in the Crypto Service
//       Provider (CSP) on your machine. KeyFile refers to a file which contains
//       a key.
//   (*) If the KeyFile and the KeyName values are both specified, the
//       following processing occurs:
//       (1) If the KeyName can be found in the CSP, that key is used.
//       (2) If the KeyName does not exist and the KeyFile does exist, the key
//           in the KeyFile is installed into the CSP and used.
//   (*) In order to create a KeyFile, you can use the sn.exe (Strong Name) utility.
//        When specifying the KeyFile, the location of the KeyFile should be
//        relative to the project directory.
//   (*) Delay Signing is an advanced option - see the Microsoft .NET Framework
```

```
//       documentation for more information on this.
//
[assembly:AssemblyDelaySignAttribute(false)];
[assembly:AssemblyKeyFileAttribute("")];
[assembly:AssemblyKeyNameAttribute("")];
```

OUTPUT CONFIGURATION

```
// This is the main project file for VC++ application project
// generated using an Application Wizard.

#include "stdafx.h"
#using <mscorlib.dll>
#import "IviDriverTypeLib.dll" no_namespace
#import "VTEXFgen.dll" no_namespace

using namespace System;

int _tmain()
{
    ::CoInitialize(NULL); //Start the COM layer
   /*We want to instantiate a pointer to the driver in a try/catch block so that we
fail
   properly if the driver is not found in the COM registry.*/

   try
   {
      IVTEXFgenPtr fgen(__uuidof(VTEXFgen));

      /*We want to do the Initialization in a try/catch block so that our test code
      doesn't run if we fail to initialize.*/
      try
      {
         /*We chose to give the driver an empty options string. More information on
            option strings can be found in any of the VTEX programmer's manuals.
         Note that we also set the Reset bit so that we get a clean start to work
         from.*/
         fgen->Initialize("TCPIP::10.1.4.55::INSTR",VARIANT_TRUE,VARIANT_TRUE, "");

         //Get the available channel names for the specified DAC card
         for(int i=0; i<fgen->Output->Count; i++)
         {
            //Note that, the channel names are 1-based (start from 1)
            printf(fgen->Output->Name[i+1]);
         }

         //Disable or enable channel output
         fgen->Output->Enabled["CH1"] = VARIANT_TRUE;
         fgen->Output->Enabled["CH2"] = VARIANT_FALSE;
         //These will enable the output of "CH1" and disable the output of "CH2"

         //Set the output mode for all the channels
         fgen->Output->OutputMode = VTEXFgenOutputModeFunction;

         /*The driver also allows the user to set the output mode for a specified
         channel*/
         fgen->Output->ChannelOutputMode["CH1"] = VTEXFgenOutputModeFunction;
           fgen->Output->ChannelOutputMode["CH2"] = VTEXFgenOutputModeArbitrary;
           fgen->Output->ChannelOutputMode["CH3"] = VTEXFgenOutputModeSequence;

         //Set the drive mode for a specified channel (either voltage or current)
         fgen->Output->DriveMode["CH1"] = VTEXFgenDriveModeVoltage;

         //Enable or disable the filter for a specified channel
```

```
          fgen->Output->FilterEnabled["CH1"] = VARIANT_FALSE;

          //Set the range of the output signal on a specified channel
          fgen->Output->Range["CH1"] = 10.0;

             //Use Range -1 to indicate autorange -- the range will be automatically
             //set whenever you change Gain, Amplitude, Offset, DCOffset, Waveform, or
             //when you configure an Arbitrary waveform or sequence to the smallest
             //range that can contain your output without clipping, or the largest
             //range if none will.
             fgen->Output->Range["CH1"] = -1.0;

             //query ActualRange to determine which range has been chosen for you
             double myRange = fgen->Output->ActualRange["CH1"];

             //Control the external sense lines
             fgen->Output->SenseEnabled["CH1"] = VARIANT_FALSE;
             //When enabled, the sense leads for the channel should be connected at
             //the load,and the function generator will compensate for the impedance
             //of the wiring
             fgen->Output->SenseEnabled["CH1"] = VARIANT_TRUE;

             //Get the current, instantaneous, output value. Note that this is
             //calculated based on current settings and calibration constants, not
             //measured.
             double outputVoltage = fgen->Output->Value["CH1"];

          //Close the initialized session
          fgen->Close();
       }
       catch(_com_error &e)
       {
          ::MessageBox(NULL, e.Description(), e.ErrorMessage(), MB_ICONERROR);
       }
    }
    catch(...)
    {
       /*We put this here to catch any error the program generates.*/
       //Do something to intelligently deal with errors
    }
}
```

STANDARD WAVEFORMS

```
// StandardWaveforms.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#import "IviDriverTypeLib.dll" no_namespace
#import "VTEXFgen.dll" no_namespace

int _tmain(int argc, _TCHAR* argv[])
{
   ::CoInitialize(NULL); //Start the COM layer

   try
   {
      IVTEXFgenPtr fgen(__uuidof(VTEXFgen));

      try
      {
         //Initialize a new session
         fgen->Initialize("TCPIP::10.1.4.55::INSTR",VARIANT_TRUE,VARIANT_TRUE, "");
```

```
            /*
                * In this example, we will exercise all of the options available with
the Function option.
                * This option allows the card to generate several pre-defined
waveforms, including sine, square,
                * triangle, and ramp, based on only a few user-supplied
charactaristics.
                */

            //First, set the OutputMode property to Function
            fgen->Output->ChannelOutputMode["CH1"] = VTEXFgenOutputModeFunction;

            //Enable the front-panel output of the channel
            fgen->Output->Enabled["CH1"] = VARIANT_TRUE;

            //Select a waveform and its basic attributes
            fgen->Output->Range["CH1"] = 5.0; // use the ±5V output range
            fgen->StandardWaveform->Waveform["CH1"] = VTEXFgenWaveformSine; // the
waveform function
            fgen->StandardWaveform->Amplitude["CH1"] = 5.0; // amplitude will be 5VPP
(+/- 2.5V)
            fgen->StandardWaveform->DCOffset["CH1"] = 2.5; // the output will be
centered around +2.5V
            fgen->StandardWaveform->Frequency["CH1"] = 10000; // frequency of 10KHz
            fgen->StandardWaveform->StartPhase["CH1"] = 90.0; // start the waveform at
a phase of 90 degrees

            //Alernatively, most of these options can be passed to
StandardWaveform.Configure()
            //To set up a triangle wave, with no offset, 20VPP, 5Hz, and 0 phase:
            fgen->Output->Range["CH1"] = 10.0;
            fgen->StandardWaveform->Configure("CH1", VTEXFgenWaveformTriangle, 20.0,
0.0, 5.0, 0.0);

            //In addition to the above options, the Square function can also be
configured with a Duty Cycle.
            fgen->Output->Range["CH1"] = 5.0;
            fgen->StandardWaveform->Configure("CH1", VTEXFgenWaveformSquare, 10.0,
0.0, 1000.0, 0.0);
            fgen->StandardWaveform->DutyCycleHigh["CH1"] = 75.0; // the high level of
the square wave will last for 75% of its period. Since our frequency is 1KHz, this is
750us, while the low level will last 250us.

         //Close the initialized session
         fgen->Close();
      }
      catch(_com_error &e)
      {
         ::MessageBox(NULL, e.Description(), e.ErrorMessage(), MB_ICONERROR);
      }
   }
   catch(...)
   {
      /*We put this here to catch any error the program generates.*/
      //Do something to intelligently deal with errors
   }
}
```

DC OFFSET

```
// This is the main project file for VC++ application project
// generated using an Application Wizard.

#include "stdafx.h"
```

```
#using <mscorlib.dll>
#import "IviDriverTypeLib.dll" no_namespace
#import "VTEXFgen.dll" no_namespace

using namespace System;

int _tmain()
{
    ::CoInitialize(NULL); //Start the COM layer
   /*We want to instantiate a pointer to the driver in a try/catch block so that we
fail
   properly if the driver is not found in the COM registry.*/

   try
   {
      IVTEXFgenPtr fgen(__uuidof(VTEXFgen));

      /*We want to do the Initialization in a try/catch block so that our test code
      doesn't run if we fail to initialize.*/
      try
      {
         /*We chose to give the driver an empty options string. More information on
            option strings can be found in any of the VTEX programmer's manuals.
         Note that we also set the Reset bit so that we get a clean start to work
         from.*/
         fgen->Initialize("TCPIP::10.1.4.55::INSTR",VARIANT_TRUE,VARIANT_TRUE, "");

         //In this example, you are to use the function generator to
         //produce DC voltage and current output.
         //CH1 will be used to produce DC voltage.
         //CH2 will be used to produce DC current.

         //To produce Voltage, set the drive mode to voltage
         fgen->Output->DriveMode["CH1"] = VTEXFgenDriveModeVoltage;
         //To produce Current, set the drive mode to current
         fgen->Output->DriveMode["CH2"] = VTEXFgenDriveModeCurrent;

         //Set the range for the output signal for the channels
         fgen->Output->Range["CH1"] = 10.0;
         fgen->Output->Range["CH2"] = 0.01;

         //Set the output mode of the channels
         fgen->Output->ChannelOutputMode["CH1"] = VTEXFgenOutputModeFunction;
         fgen->Output->ChannelOutputMode["CH2"] = VTEXFgenOutputModeFunction;

         //Setup CH1 and CH2 to output DC waveform
         fgen->StandardWaveform->Waveform["CH1"] = VTEXFgenWaveformDC;
         fgen->StandardWaveform->Waveform["CH2"] = VTEXFgenWaveformDC;

         //Set the DC offset of the channels
         fgen->StandardWaveform->DCOffset["CH1"] = 5.0;
         fgen->StandardWaveform->DCOffset["CH2"] = 0.005;

         //To physically produce the signal, you need to enable the channels
         fgen->Output->Enabled["CH1"] = VARIANT_TRUE;
         fgen->Output->Enabled["CH2"] = VARIANT_TRUE;

         //Close the initialized session
         fgen->Close();
      }
      catch(_com_error &e)
      {
         ::MessageBox(NULL, e.Description(), e.ErrorMessage(), MB_ICONERROR);
      }
```

```
    }
    catch(...)
    {
        /*We put this here to catch any error the program generates.*/
        //Do something to intelligently deal with errors
    }
}
```

ARBITRARY SEQUENCE

```
// ArbitrarySequences.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <math.h>
#import "IviDriverTypeLib.dll" no_namespace
#import "VTEXFgen.dll" no_namespace

SAFEARRAY * CreateArray(double data[], int len) {
    SAFEARRAY *a = ::SafeArrayCreateVector (VT_R8, 0, len);

    for (long i = 0; i < len; i++) {
        ::SafeArrayPutElement(a, &i, (void*)&data[i]);
    }

    return a;
}

SAFEARRAY * CreateArray(long data[], int len) {
    SAFEARRAY *a = ::SafeArrayCreateVector (VT_I4, 0, len);

    for (long i = 0; i < len; i++) {
        ::SafeArrayPutElement(a, &i, (void*)&data[i]);
    }

    return a;
}


int _tmain(int argc, _TCHAR* argv[])
{
    ::CoInitialize(NULL); //Start the COM layer

    try
    {
        IVTEXFgenPtr fgen(__uuidof(VTEXFgen));

        try
        {
            //Initialize a new session
            fgen->Initialize("TCPIP::10.1.4.55::INSTR",VARIANT_TRUE,VARIANT_TRUE, "");


            /*
            * In this example, we will exercise all of the options available with the
Arbitrary option.
            * This option allows the card to generate any user-specified waveform.
            */

            //First, set the OutputMode property to Function
            fgen->Output->ChannelOutputMode["CH1"] = VTEXFgenOutputModeSequence;
            fgen->Output->ChannelOutputMode["CH2"] = VTEXFgenOutputModeSequence;
            fgen->Output->ChannelOutputMode["CH3"] = VTEXFgenOutputModeSequence;
```

```
          /*
          * Generation must be enabled or waveform data will never advance.
          * However, note that modifying any waveform data while generation is enabled
will cause
          * generation to be temporarily disabled automatically, starting all
waveforms over from the beginning.
          * To minimize the effect of this, you can abort and re-initiate generation
yourself:
          */
          fgen->AbortGeneration();

          //Enable the front-panel output of the channels
          fgen->Output->Enabled["CH1"] = VARIANT_TRUE;
          fgen->Output->Enabled["CH2"] = VARIANT_TRUE;
          fgen->Output->Enabled["CH3"] = VARIANT_TRUE;

          //A sequence consists of several waveforms, so first create some waveforms

          double w[] = {-1.0, -7.0/9.0, -5.0/9.0, -3.0/9.0, -1.0/9.0, 1.0/9.0,
3.0/9.0, 5.0/9.0, 7.0/9.0, 1.0};
          SAFEARRAY *wave = CreateArray(w, 10);
          int tenSteps = fgen->Arbitrary->Waveform->Create(&wave);
          ::SafeArrayDestroy(wave);

          wave = ::SafeArrayCreateVector (VT_R8, 0, 100);
          double x;
          for (long i = 0; i < 100; i++)
          {
             x = sin(2.0 * 3.141592653589793 * (double)i / 100.0);
             ::SafeArrayPutElement(wave, &i, (void*)&x);
          }
          int sine = fgen->Arbitrary->Waveform->Create(&wave);
          ::SafeArrayDestroy(wave);

          double sq[] = {-1.0, -1.0, 1.0, 1.0};
          wave = CreateArray(sq, 4);
          int square = fgen->Arbitrary->Waveform->Create(&wave);
          ::SafeArrayDestroy(wave);

          //Next, create some sequences from these

          //This sequence will output each waveform once before repeating
          long h1[] = {tenSteps, sine, square};
          long c1[] = {1, 1, 1};
          SAFEARRAY *handles = CreateArray(h1, 3);
          SAFEARRAY *counts = CreateArray(c1, 3);
          int onceEach = fgen->Arbitrary->Sequence->Create(&handles, &counts);
          ::SafeArrayDestroy(handles);
          ::SafeArrayDestroy(counts);

          //This will output two sines, then output square forever
          long h2[] = { sine, square };
          long c2[] = { 10, 0 };
          handles = CreateArray(h2, 2);
          counts = CreateArray(c2, 2);
          int tenSinesThenSquare = fgen->Arbitrary->Sequence->Create(&handles,
&counts);
          ::SafeArrayDestroy(handles);
          ::SafeArrayDestroy(counts);

          //This will output tenSteps forever, but with other settings can be more
useful...
          long h3[] = {tenSteps, sine, square};
          long c3[] = {0, 0, 0};
```

```
          handles = CreateArray(h3, 3);
          counts = CreateArray(c3, 3);
          int infiniteRepeats = fgen->Arbitrary->Sequence->Create(&handles, &counts);
          ::SafeArrayDestroy(handles);
          ::SafeArrayDestroy(counts);

          //Now, apply a sequence to a channel -- this works much the same way as for
waveforms
          fgen->Output->Range["CH1"] = 5.0;
          fgen->Arbitrary->Sequence->Configure("CH1", onceEach, 5.0, 0.0);
          fgen->Arbitrary->ChannelSampleRate["CH1"] = 500000; // setting frequency is
not available for sequence mode, so set sample rate directly

          fgen->Output->Range["CH2"] = 10.0;
          fgen->Arbitrary->Sequence->Configure("CH2", tenSinesThenSquare, 7.5, 0.0);
          fgen->Arbitrary->ChannelSampleRate["CH2"] = 1000;

          fgen->Output->Range["CH3"] = 1.0;
          fgen->Arbitrary->Sequence->Configure("CH3", infiniteRepeats, 0.5, 0.0);
          fgen->Arbitrary->ChannelSampleRate["CH3"] = 15000;

          //Turn generation on again
          fgen->InitiateGeneration();

          //A special operation mode exists for sequences
          fgen->Output->OperationMode["CH3"] = VTEXFgenOperationModeSequenced;

          //In this mode, triggers will advance to the next waveform in the sequence,
even if the repeat count for a sequence is infinite
          fgen->Trigger->Source["CH3"] = VTEXFgenTriggerSourceSoftware;
          fgen->Trigger->SendSoftwareTrigger(); // advance to the sine wave
          Sleep(1000);
          fgen->Trigger->SendSoftwareTrigger(); // advance to the square wave
          Sleep(1000);
          fgen->Trigger->SendSoftwareTrigger(); // return to the step function

          //The Async flag will control whether the waveform changes immediately or
after the current iteration of the waveform finishes
          fgen->Arbitrary->Sequence->AsyncAdvance["CH3"] = VARIANT_TRUE; // advance
immediately
          fgen->Arbitrary->Sequence->AsyncAdvance["CH3"] = VARIANT_FALSE; // advance
after the current iteration of the waveform finishes

          //If the repeat count is not infinite and the waveform's repeat count
completes before receiving a trigger,
          //the channel will hold the last value as a DC output until it receives the
trigger
          fgen->Output->OperationMode["CH2"] = VTEXFgenOperationModeSequenced;
          fgen->Trigger->Source["CH2"] = VTEXFgenTriggerSourceSoftware;

          fgen->Trigger->SendSoftwareTrigger(); // we should be in the infinite square
by now -- advance back to the sine wave
          Sleep(1000);
          //by now CH2 will have outputted two periods of sine wave and be held at DC
0V
          fgen->Trigger->SendSoftwareTrigger(); // advance to the square wave and let
it run


          //Cleaning up after ourselves works just as for waveforms. First, ensure no
channels are using the sequence:
          fgen->Arbitrary->Sequence->Configure("CH3", onceEach, 1.0, 0.0);
          //or
          fgen->Output->ChannelOutputMode["CH3"] = VTEXFgenOutputModeFunction;
```

```
          //Then we can delete it
          fgen->Arbitrary->Sequence->Clear(infiniteRepeats);

          //to delete the consituent waveforms however, no sequences may exist that
include them
          //so to delete tenSteps we first need to get rid of onceEach as well
          fgen->Output->ChannelOutputMode["CH1"] = VTEXFgenOutputModeFunction;
          fgen->Arbitrary->Sequence->Clear(onceEach);
          fgen->Arbitrary->Waveform->Clear(tenSteps);

          //and to remove the other waveforms we must delete all sequences that
include them
          fgen->Output->ChannelOutputMode["CH2"] = VTEXFgenOutputModeFunction;
          fgen->Arbitrary->Sequence->Clear(tenSinesThenSquare);
          fgen->Arbitrary->Waveform->Clear(sine);
          fgen->Arbitrary->Waveform->Clear(square);

          //alternatively, we can clear all memory with one command -- but no
sequences or waveforms may be in use
          fgen->Arbitrary->ClearMemory();

          //Close the initialized session
          fgen->Close();
       }
       catch(_com_error &e)
       {
          ::MessageBox(NULL, e.Description(), e.ErrorMessage(), MB_ICONERROR);
       }
    }
    catch(...)
    {
       /*We put this here to catch any error the program generates.*/
       //Do something to intelligently deal with errors
    }
}
```

ARBITRARY WAVEFORMS

```
// ArbitraryWaveforms.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <cstdlib>
#import "IviDriverTypeLib.dll" no_namespace
#import "VTEXFgen.dll" no_namespace

SAFEARRAY * CreateArray(double data[], int len) {
    SAFEARRAY *a = ::SafeArrayCreateVector (VT_R8, 0, len);

    for (long i = 0; i < len; i++) {
        ::SafeArrayPutElement(a, &i, (void*)&data[i]);
    }

    return a;
}

int _tmain(int argc, _TCHAR* argv[])
{
    ::CoInitialize(NULL); //Start the COM layer

    try
    {
       IVTEXFgenPtr fgen(__uuidof(VTEXFgen));
```

```
      try
      {
         //Initialize a new session
         fgen->Initialize("TCPIP::10.1.4.55::INSTR",VARIANT_TRUE,VARIANT_TRUE, "");

         /*
         * In this example, we will exercise all of the options available with the
Arbitrary option.
         * This option allows the card to generate any user-specified waveform.
         */

         //First, set the OutputMode property to Function
         fgen->Output->ChannelOutputMode["CH1"] = VTEXFgenOutputModeArbitrary;
         fgen->Output->ChannelOutputMode["CH2"] = VTEXFgenOutputModeArbitrary;
         fgen->Output->ChannelOutputMode["CH3"] = VTEXFgenOutputModeArbitrary;
         fgen->Output->ChannelOutputMode["CH4"] = VTEXFgenOutputModeArbitrary;

         /*
         * Generation must be enabled or waveform data will never advance.
         * However, note that modifying any waveform data while generation is enabled
will cause
         * generation to be temporarily disabled automatically, starting all
waveforms over from the beginning.
         * To minimize the effect of this, you can abort and re-initiate generation
yourself:
         */
         fgen->AbortGeneration();

         //Enable the front-panel output of the channels
         fgen->Output->Enabled["CH1"] = VARIANT_TRUE;
         fgen->Output->Enabled["CH2"] = VARIANT_TRUE;
         fgen->Output->Enabled["CH3"] = VARIANT_TRUE;
         fgen->Output->Enabled["CH4"] = VARIANT_TRUE;

         //All data for aritrary wavefrorms must be normalized to between -1.0 and
1.0
         //All waveforms must also be at least SizeMin long, and have a length evenly
divisible by Quantum (for ex1200-360x these are 4 and 2, respectively)
         int sizeMin = fgen->Arbitrary->Waveform->SizeMin;
         int quantum = fgen->Arbitrary->Waveform->Quantum;

         //Create a simple 4-level step function
         double data[] = {-1.0, -0.3333, 0.3333, 1.0};
         SAFEARRAY *steps = CreateArray(data, 4);


         //Upload the function to the card and save a handle to it
         int stepHandle = fgen->Arbitrary->Waveform->Create(&steps);
         ::SafeArrayDestroy(steps);

         //Now, apply the waveform to a channel
         //Here, we will command CH1 to output first 0V, 1V, 2V, and finally 3V
         fgen->Output->Range["CH1"] = 5.0; // choose a range that is wide enough
         fgen->Arbitrary->Waveform->Configure("CH1", stepHandle, 3.0, 1.5); // your
normalized data will be scaled via this gain and offset

         //Turn generation on again
         fgen->InitiateGeneration();
         //We'll leave generation on from this point on

         //By default, the card will step through our four values at its default
sample rate (500KHz for the ex1200-360x)
         //But, you can choose a different sample rate if desired, either for all
channels at once:
```

```
          fgen->Arbitrary->SampleRate = 10000;
          //or for just one channel at a time
          fgen->Arbitrary->ChannelSampleRate["CH1"] = 100;
          //Alternatively, as a conveniece, frequency can be set.
          //This will automatically caluclate a new sample rate for you based on your
waveform's length and the desired frequency
          fgen->Arbitrary->Waveform->Frequency["CH1"] = 200; // because len=4, the new
sample rate will be 200*4=800Hz

          //Many channels can output the same waveform with different settings,
without using any additional memory:
          fgen->Output->Range["CH2"] = 10.0;
          fgen->Arbitrary->Waveform->Configure("CH2", stepHandle, 10.0, 0.0);
          fgen->Arbitrary->Waveform->Frequency["CH2"] = 500;

          fgen->Output->Range["CH3"] = 20.0;
          fgen->Arbitrary->Waveform->Configure("CH3", stepHandle, 10.0, 10.0);
          fgen->Arbitrary->Waveform->Frequency["CH3"] = 100;

          fgen->Output->Range["CH4"] = 40.0;
          fgen->Arbitrary->Waveform->Configure("CH4", stepHandle, 15.0, 20.0);
          fgen->Arbitrary->Waveform->Frequency["CH4"] = 2000;

          //Waveform length is only limited by available memory
          //Let's fill up all of remaining memory with random noise
          int len = fgen->Arbitrary->Waveform->SizeMax;
          SAFEARRAY *noise = ::SafeArrayCreateVector (VT_R8, 0, len);

          double x;
          for (long i = 0; i < len ; i++)
          {
             x = (rand() * 2.0 / (double)RAND_MAX) - 1.0;
             ::SafeArrayPutElement(noise, &i, (void*)&x);
          }
          int noiseHandle = fgen->Arbitrary->Waveform->Create(&noise);
          ::SafeArrayDestroy(noise);

          //now we'll output our random noise between +/-10mV on CH2 at the fastest
sample rate:
          fgen->Output->Range["CH2"] = 1.0;
          fgen->Arbitrary->Waveform->Configure("CH2", noiseHandle, 0.01, 0.0);
          fgen->Arbitrary->ChannelSampleRate["CH2"] = 500000;

          //But now we can't create any more waveforms.
          //Once we're done with the noise output, we can get rid of it to free up
some memory.
          //First, ensure that no channels are using it:
          fgen->Arbitrary->Waveform->Configure("CH2", stepHandle, 1.0, 0.0);
          //or
          fgen->Output->ChannelOutputMode["CH2"] = VTEXFgenOutputModeFunction;
          //Then we can delete it
          fgen->Arbitrary->Waveform->Clear(noiseHandle);

          //Close the initialized session
          fgen->Close();
       }
       catch(_com_error &e)
       {
          ::MessageBox(NULL, e.Description(), e.ErrorMessage(), MB_ICONERROR);
       }
    }
    catch(...)
    {
       /*We put this here to catch any error the program generates.*/
```

```
        //Do something to intelligently deal with errors
    }
}
```

MARKER

```
// Marker.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#import "IviDriverTypeLib.dll" no_namespace
#import "VTEXFgen.dll" no_namespace

SAFEARRAY * CreateArray(double data[], int len) {
    SAFEARRAY *a = ::SafeArrayCreateVector (VT_R8, 0, len);

    for (long i = 0; i < len; i++) {
        ::SafeArrayPutElement(a, &i, (void*)&data[i]);
    }

    return a;
}

SAFEARRAY * CreateArray(long data[], int len) {
    SAFEARRAY *a = ::SafeArrayCreateVector (VT_I4, 0, len);

    for (long i = 0; i < len; i++) {
        ::SafeArrayPutElement(a, &i, (void*)&data[i]);
    }

    return a;
}


int _tmain(int argc, _TCHAR* argv[])
{
    ::CoInitialize(NULL); //Start the COM layer

    try
    {
        IVTEXFgenPtr fgen(__uuidof(VTEXFgen));

        try
        {
            //Initialize a new session
            fgen->Initialize("TCPIP::10.1.4.55::INSTR",VARIANT_TRUE,VARIANT_TRUE, "");

              /*
                  * The Marker interface provides the ability to drive the front panel
marker signal.
                  * This example will show how to configure and use this interface.
                  */

              //Markers are assigned to positions in arbitrary waveforms. Only one
marker may be set per waveform.

              //First, create and configure a waveform
              double w1[] = {-1.0, -0.8, -0.6, -0.4, -0.2, 0.0, 0.2, 0.4, 0.6, 0.8};
            SAFEARRAY *wave = CreateArray(w1, 10);
            int upSteps = fgen->Arbitrary->Waveform->Create(&wave);
            ::SafeArrayDestroy(wave);

              fgen->Output->ChannelOutputMode["CH1"] = VTEXFgenOutputModeArbitrary;
              fgen->Output->Range["CH1"] = 5.0;
```

```
            fgen->Arbitrary->Waveform->Configure("CH1", upSteps, 5.0, 0.0);
            fgen->Arbitrary->Waveform->Frequency["CH1"] = 10.0;

            //Next, set the marker position for the waveform
            fgen->Marker->Position[upSteps] = 6; // set the position to index 6, which
is our 0.2 level

            //Finally, configure the marker output driver
            fgen->Marker->Source = "CH1"; //look at the waveform on CH1 to drive the
marker output
            fgen->Marker->Slope = VTEXFgenMarkerSlopeRise; //output will be normally
low, pulsed high when position is reached
            fgen->Marker->Width = 0.001; // the high pulse will last for 1 millisecond
before returning to a low level
            fgen->Marker->Enabled = VARIANT_TRUE; // turn on the output

            //Any time a waveform is being generated, its marker can be outputted (it
is at position 0 if not otherwise set).
            //This includes in sequence mode -- the existing marker for each
constituent waveform will cause the output to be pulsed.

            //Let's make use of this two create a step function that goes both up and
down, outputting a marker at the same level both times
            double w2[] = {1.0, 0.8, 0.6, 0.4, 0.2, 0.0, -0.2, -0.4, -0.6, -0.8};
          wave = CreateArray(w2, 10);
          int downSteps = fgen->Arbitrary->Waveform->Create(&wave);
          ::SafeArrayDestroy(wave);

            fgen->Marker->Position[downSteps] = 4; // again at 0.2, which is position
4 this time
            long h[] = {upSteps, downSteps};
            long c[] = {1, 1};

          SAFEARRAY *handles = CreateArray(h, 2);
          SAFEARRAY *counts = CreateArray(c, 2);
            int stepper = fgen->Arbitrary->Sequence->Create(&handles, &counts);
          ::SafeArrayDestroy(handles);
          ::SafeArrayDestroy(counts);

            fgen->Output->Range["CH2"] = 1.0;
            fgen->Arbitrary->ChannelSampleRate["CH2"] = 5000.0;
            fgen->Output->ChannelOutputMode["CH2"] = VTEXFgenOutputModeSequence;
            fgen->Arbitrary->Sequence->Configure("CH2", stepper, 1.0, 0.0);

            fgen->Marker->Source = "CH2";
            fgen->Marker->Slope = VTEXFgenMarkerSlopeFall;
            fgen->Marker->Width = 1.0e-5;

            //now we will get a 10us low pulse at position 6 of upSteps and at
position 4 of downSteps, which happens to be 0.2 in each

          //Close the initialized session
          fgen->Close();
       }
       catch(_com_error &e)
       {
          ::MessageBox(NULL, e.Description(), e.ErrorMessage(), MB_ICONERROR);
       }
    }
    catch(...)
    {
       /*We put this here to catch any error the program generates.*/
       //Do something to intelligently deal with errors
    }
```

```
}
```

TRIGGER

```
// Trigger.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#import "IviDriverTypeLib.dll" no_namespace
#import "VTEXFgen.dll" no_namespace

int _tmain(int argc, _TCHAR* argv[])
{
   ::CoInitialize(NULL); //Start the COM layer

   try
   {
      IVTEXFgenPtr fgen(__uuidof(VTEXFgen));

      try
      {
         //Initialize a new session
         fgen->Initialize("TCPIP::10.1.4.55::INSTR", VARIANT_TRUE, VARIANT_TRUE, "");

           //This example will illustrate the triggering capabilities of the VTEXFgen
driver

           //The OperationMode setting controls what effect triggers have on the
device

           //In Continuous mode, triggers have no effect and the chosen output is
repeated endlessly.
           fgen->Output->OperationMode["CH1"] = VTEXFgenOperationModeContinuous;

           //In Single Step mode, the internal sample clock is ignored, and only the
SendSoftwareTrigger call
           //will advance output to the next level -- this is most useful for
debugging the card's outputs.
           fgen->Output->OperationMode["CH1"] = VTEXFgenOperationModeSingleStep;
           fgen->Trigger->SendSoftwareTrigger();
           fgen->Trigger->SendSoftwareTrigger();
           fgen->Trigger->SendSoftwareTrigger();

           //Sequenced mode is a special mode that only works with the Sequence
OutputMode setting.
           //See the AribtrarySequences example for more details.
           fgen->Output->OperationMode["CH1"] = VTEXFgenOperationModeSequenced;

           //In Burst mode, a trigger will cause the channel to generate a set number
of iterations of the chosen output
           fgen->Output->OperationMode["CH1"] = VTEXFgenOperationModeBurst;
           fgen->Trigger->BurstCount["CH1"] = 5; //generate the output 5 times when
triggered, then wait for another trigger

           //Choose which trigger causes the channel to be triggered

           //Software allows the user application to trigger the channel through the
API
           fgen->Trigger->Source["CH1"] = VTEXFgenTriggerSourceSoftware;

           //The internal trigger is a built-in periodic timer with a programmable
rate, in units of Hz
           fgen->Trigger->Source["CH1"] = VTEXFgenTriggerSourceInternal;
           fgen->Trigger->InternalRate = 100.0;
```

```
            //The external trigger is a front-panel input pin and can trigger on
either rising or falling edges
            fgen->Trigger->Source["CH1"] = VTEXFgenTriggerSourceExternal;
            fgen->Trigger->Slope["CH1"] = VTEXFgenTriggerSlopeRise;

            //Finally, Backplane allows use of any of the backplane triggers of the
chassis the card is in
            fgen->Trigger->Source["CH1"] = VTEXFgenTriggerSourceBackplane;
            //Because only one trigger line can be connected to each card at a time,
the selection
            //of which backplane line to use is not a channel property, but global to
the card.
            fgen->Trigger->BackplaneSource = VTEXFgenBackplaneSourceBPL3;
            //Which slope to trigger on is configurable for backplane triggers as
well.
            fgen->Trigger->Slope["CH1"] = VTEXFgenTriggerSlopeFall;

          //Close the initialized session
          fgen->Close();
       }
       catch(_com_error &e)
       {
          ::MessageBox(NULL, e.Description(), e.ErrorMessage(), MB_ICONERROR);
       }
    }
    catch(...)
    {
       /*We put this here to catch any error the program generates.*/
       //Do something to intelligently deal with errors
    }
}
```

SECTION 5

SFP OPERATION

INTRODUCTION

EX1200s offer an embedded web page which provides network configuration control, time configuration, and the ability to perform firmware upgrades. To facilitate discovery of the mainframe, VTI provides the **L**AN **In**strument **C**onnection and **U**pgrade (**LInC-U**) utility on the *VTI Instruments Corp. Drivers and Product Manuals CD* included with the EX1200 mainframe in the *EX Platforms Requisites* directory.

To open the embedded web page, start the **LInC-U** utility by navigating to **Start → Programs → VTI Instruments Corporation → LInC-U Utility → LInC-U Utility**. Once the utility is run, LInC-U will scan the network to discover all LAN-based VTI instruments. Once the scan is complete, the **Discovery Devices** tab will appear and show the instruments that were discovered, as shown in Figure 5-1. To open the web page, click on the hostname hyperlink in the **Discover Devices** tab. The IP address of the EX1200 can also be viewed from this window as well as its firmware version.

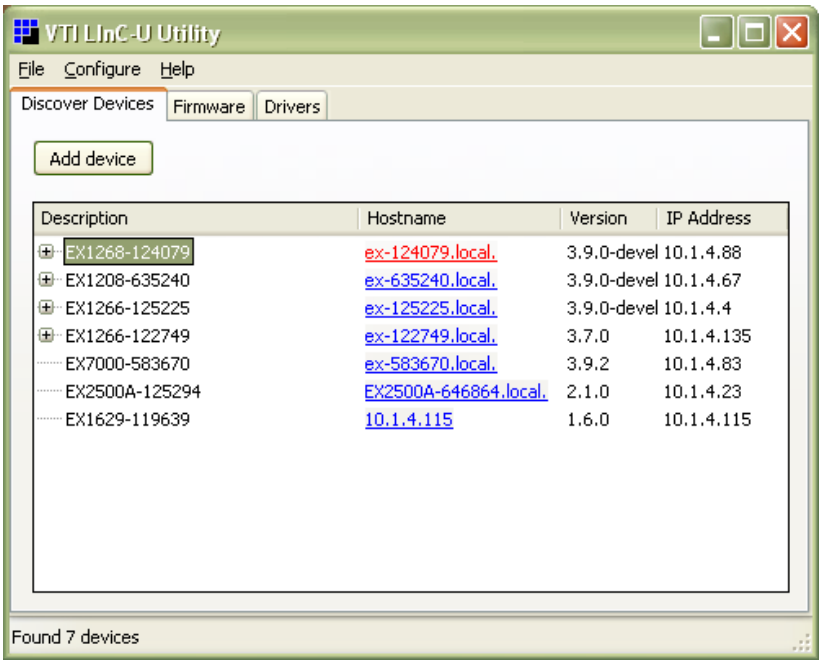


FIGURE 5-1: LINC-U DISCOVERY TAB WITH AN EX1268 SELECTED

Alternatively, the EX1200 may also be discovered using Internet Explorer's Bonjour for Windows plug-in, by entering the mainframe's IP address into the address bar of any web browser to view the embedded web page, or using VXI-11. For more information on discovery methods, refer to the *EX1200 Series User's Manual* (P/N: 82-0127-000).

GENERAL WEB PAGE OPERATION

When initial connection is made to the EX1200, the instrument home page, **Index**, appears (see **Error! Reference source not found.**). This page displays instrument-specific information i ncluding:

- Model
- Manufacturer
- Serial Number
- Description
- LXI Class
- LXI Version
- Hostname
- MAC Address
- IP Address
- Netmask
- Instrument Address String
- Firmware Version
- IEEE-1588 Time

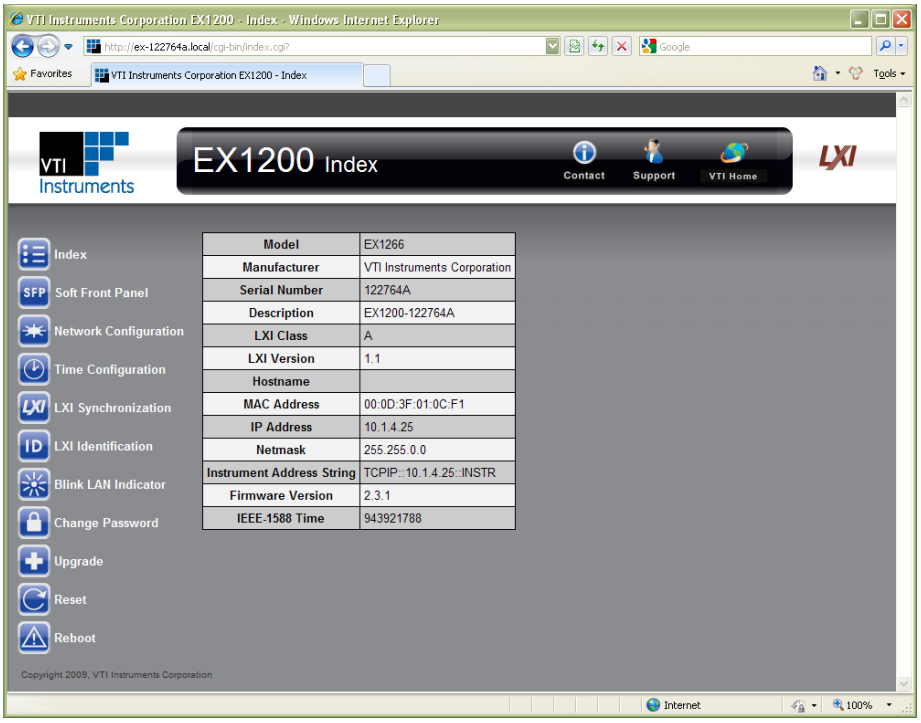


FIGURE 5-2: EX1200 MAIN WEB PAGE

The **Index** is accessible from any other instrument page by clicking on the EX1200 web page header. The EX1200 **Command Menu** is displayed on the left-hand side of every internal web page. The entries on the command menu represent three types of pages:

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

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

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

Use of the entry-type web pages in the EX1200 are governed by a common set of operational characteristics:

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

Notes on Web Page Use

If a window needs to be resized, this should be done when the window opens. Resizing requires a refresh which causes the current state to be lost.

VTI Instruments Logo

The VTI Instruments logo that appears on the upper left of all EX1200 web pages is a link to the VTI Instruments corporate website: http://www.vtiinstruments.com.

The remainder of this discussion will focus on the EX1200-360x soft front panel. For more information on other EX1200 soft front panel elements, please refer to the *EX1200 Series User's Manual*.

EX1200-3608 SOFT FRONT PANEL

To navigate to the EX1200-360x soft front panel, click on **Soft Front Panel** in the **Command Menu** (see Figure 5-3). Next, select **Fgen ex1200-360x** from the list of instruments installed in the EX1200.

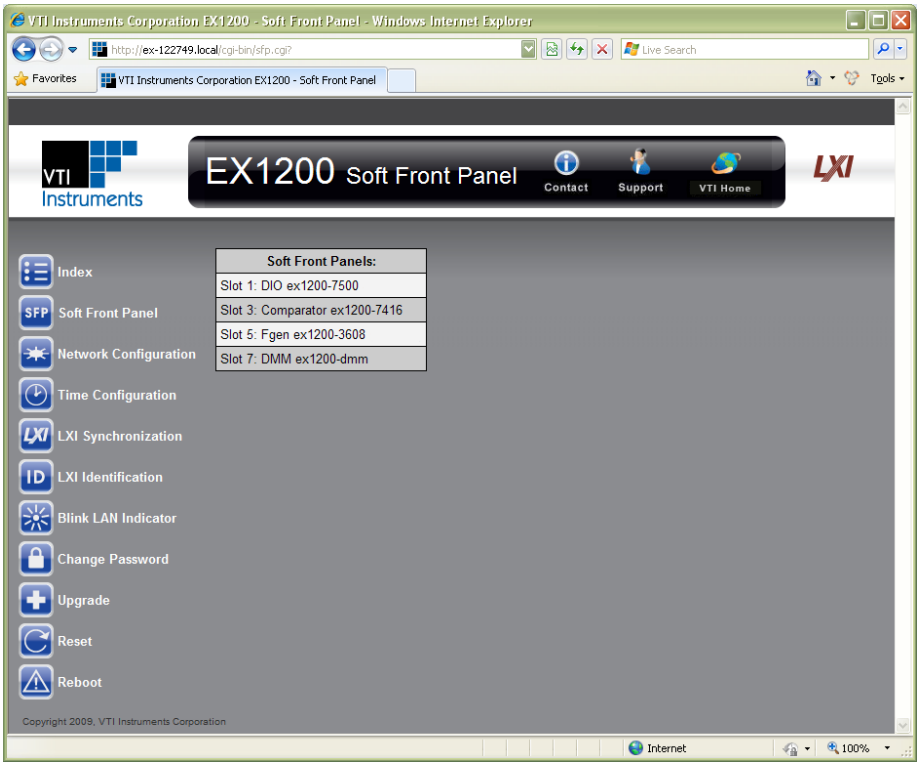


FIGURE 5-3: EX1200 SOFT FRONT PANEL MAIN PAGE

MONITOR AND CONTROL PAGE

By default, the EX1200-360x SFP opens to the **Monitor and Control** view. From this view, the user can define a channel's output mode, waveform, range, etc. Although the SFP does not expose the entire functionality of the EX1200-360x, the SFP can be used to set up the instrument in **most** applications.

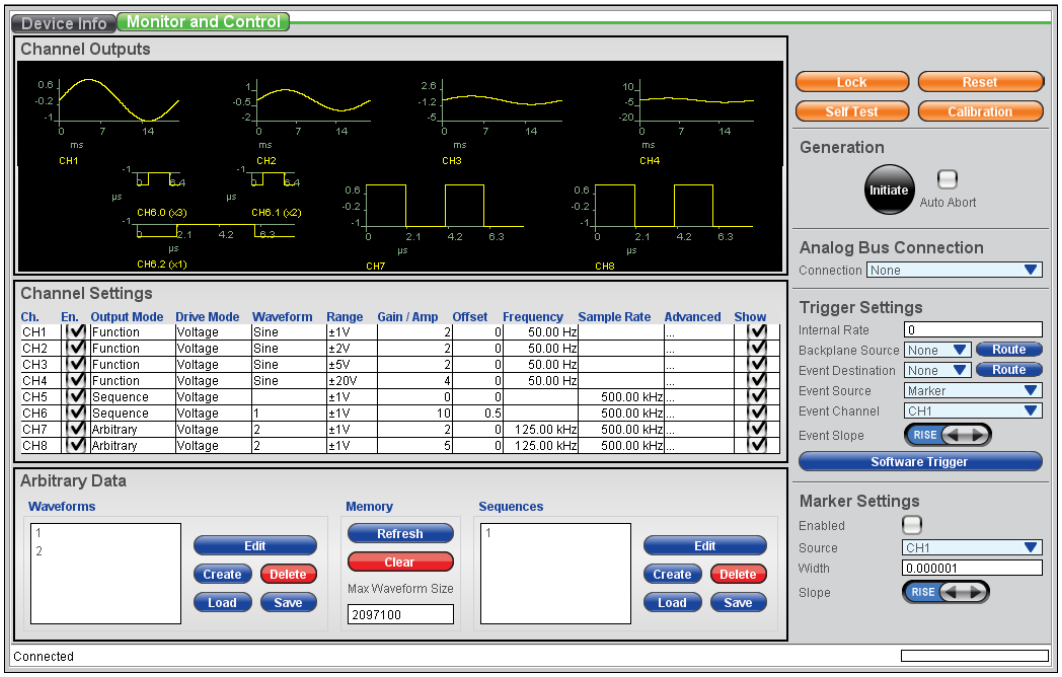


FIGURE 5-4: EX1200-360X SOFT FRONT PANEL

Channel Outputs

The **Channel Outputs** section provides a graphic illustration of signal that is being output on each EX1200-360x channel.

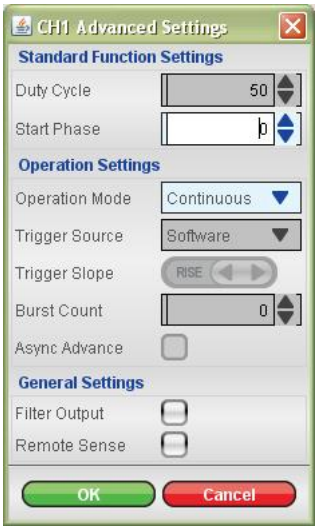
Channel Settings

The **Channel Settings** section is used to configure the output for each EX1200-360x channel. The options provided are dependent on the **Output Mode** selected. Each **Output Mode** will be addressed separately.

Function

- **Ch**: Indicates the channel that is defined by this row.
- **En** checkbox: When selected, indicates that the channel is enabled for output generation.
- **Output Mode**: For this section, it has been set to **Function**.
- **Drive Mode**: Specifies the output drive mode as being either **Voltage** or **Current**.
- **Waveform**: Defines type of function that will be output on the channel. Valid values include: **Sine**, **Square**, **Triangle**, **Ramp Up**, **Ramp Down**, and **DC**.
- **Range**: Defines the output range for the channel. The values available for the user depend on the **Drive Mode** selection.
  - **Voltage** selected: Auto, ±**1 V**, ±**2 V**, ±**5 V**, ±**10 V**, ±**20 V**, and **40 V** (unipolar only).
  - **Current** selected: Auto, **±5 mA**, **±10 mA**, or **±20 mA**.

- **Gain/Amp**: Defines the specified amplitude, or the peak-to-peak size, of a waveform. This value should not exceed twice the selected **Range** value for bipolar channels or the **Range** for unipolar channels. Adjusting the **Gain/Amp** to where the signal exceeds the bounds of the channel's range results in clipping the output signal.
- **Offset**: Defines the offset for the channel. This value should not exceed the selected **Range** value. Adjusting the **Offset** to where the signal exceeds the bounds of the channel's range results in clipping the output signal.
- **Frequency**: Indicates the waveforms frequency. Note, for a **DC** waveform, this is always **0**.
- **Sample Rate**: This field is not configurable when **Function** is selected.
- **Advanced**: When this field is clicked, a dialog box opens allow for additional configuration of the waveform. The fields that are available for configuration vary depending on the **Waveform** that is selected.

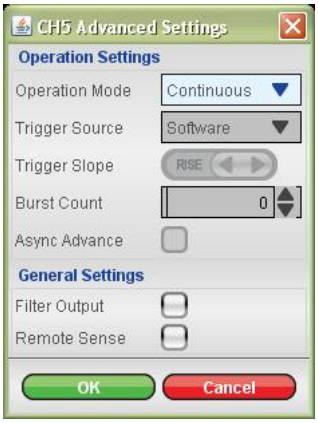


  - **Duty Cycle**: Sets the duty cycle as a percentage (square waveform only)
  - **Start Phase**: Indicates the start phase for a waveform in degrees (not applicable to dc waveforms).
  - **Operation Mode**: Sets the operation mode for the waveform. Options include **Continuous**, **Sequence**, **Burst**, and **SingleStep**.
  - **Trigger Source**: This option is only available if **Burst** or **Sequence** is selected for Operation Mode. This defines the trigger which will cause a burst to occur or a sequence to be generated. The user can select from **External**, **Software**, **Internal**, or **Backplane**.
  - **Trigger Slope**: Determines the slope that will cause the trigger defined in **Trigger Source** to be generated. This option only applies to **Backplane** and **External** trigger sources.
  - **Burst Count**: When **Burst** is the selected **Operation Mode**, it specifies the number of times a waveform is repeated sequentially after receiving a trigger.
  - **Async Advance**: This option is not applicable in **Function** mode.
  - **Filter Output**: Enables the output anti-aliasing filter.
  - **Remote Sense**: Enables the remote sense inputs to compensate for lead-wire impedance.
- **Show** checkbox: When this is not selected, the channel is no longer visible in the Channel Settings section of the SFP.

Arbitrary

- **Ch**: Indicates the channel that is defined by this row.
- **En** checkbox: When selected, indicates that the channel is enabled for output generation.
- **Output Mode**: For this section, it has been set to **Arbitrary**.
- **Drive Mode**: Specifies the output drive mode as being either **Voltage** or **Current**.
- **Waveform**: Contains a list of all defined arbitrary waveform segments.

- **Range**: Defines the output range for the channel. The values available for the user depend on the **Drive Mode** selection.
  - **Voltage** selected: Auto, ±**1 V**, ±**2 V**, ±**5 V**, ±**10 V**, ±**20 V**, and **40 V** (unipolar only).
  - **Current** selected: Auto, **±5 mA**, **±10 mA**, or **±20 mA**.
- **Gain/Amp**: Defines the specified amplitude, or the peak-to-peak size, of a waveform. This value should not exceed the selected **Range** value for the channel. Adjusting the **Gain/Amp** to where the signal exceeds the bounds of the channel's range results in clipping the output signal.
- **Offset**: Defines the offset for the channel. This value should not exceed the selected **Range** value. Adjusting the **Offset** to where the signal exceeds the bounds of the channel's range results in clipping the output signal.
- **Frequency**: Always equal to **SampleRate**/**WaveformLength**. Note that this value can only be set when the EX1200-360x is not generating.
- **Sample Rate**: Determines how often the channel moves to the next sample in the segment. If the embedded checkbox is selected, output changes will be based on the external clock input.
- **Advanced**: When this field is clicked, a dialog box opens allow for additional configuration of the waveform.



  - **Operation Mode**: Sets the operation mode for the waveform. Options include **Continuous**, **Sequence**, **Burst**, and **SingleStep**.
  - **Trigger Source**: This option is only available if **Burst** or **Sequence** is selected for **Operation Mode**. This defines the trigger which will cause a burst to occur or for a sequence to be generated. The user can select from **External**, **Software**, **Internal**, or **Backplane**.
  - **Trigger Slope**: Determines the slope that will cause the trigger defined in **Trigger Source** to be generated. This option only applies to **Backplane** and **External** trigger sources.
  - **Burst Count**: When **Burst** is the selected **Operation Mode**, it specifies the number of times a waveform is repeated sequentially after receiving a trigger.
  - **Async Advance**: In **Sequence** mode, enabling this feature causes the channel to move to the next waveform in the sequence immediately upon receiving a trigger, rather than first completing the waveform. This option not applicable only in **Sequence** mode.
  - **Filter Output**: Enables the output anti-aliasing filter.
  - **Remote Sense**: Enables the remote sense inputs to compensate for lead-wire impedance.
- **Show** checkbox: When this is not selected, the channel is no longer visible in the **Channel Settings** section of the SFP.

Sequence

- **Ch**: Indicates the channel that is defined by this row.
- **En** checkbox: When selected, indicates that the channel is enabled for output generation.
- **Output Mode**: For this section, it has been set to **Sequence**.
- **Drive Mode**: Specifies the output drive mode as being either **Voltage** or **Current**.

- **Waveform**: Contains a list of all defined sequences.
- **Range**: Defines the output range for the channel. The values available for the user depend on the **Drive Mode** selection.
  - **Voltage** selected: Auto, ±**1 V**, ±**2 V**, ±**5 V**, ±**10 V**, ±**20 V**, and **40 V** (unipolar only).
  - **Current** selected: Auto, **±5 mA**, **±10 mA**, or **±20 mA**.
- **Gain/Amp**: Defines the specified amplitude, or the peak-to-peak size, of a waveform. This value should not exceed the selected **Range** value for the channel. Adjusting the **Gain/Amp** to where the signal exceeds the bounds of the channel's range results in clipping the output signal.
- **Offset**: Defines the offset for the channel. This value should not exceed the selected **Range** value. Adjusting the **Offset** to where the signal exceeds the bounds of the channel's range results in clipping the output signal.
- **Frequency**: This is not available for sequence configuration.
- **Sample Rate**: Determines often the channel moves to the next sample in the segment. If the embedded checkbox is selected, output updates will be based on the external clock input.
- **Advanced**: When this field is clicked, a dialog box opens allow for additional configuration of the sequence. Advanced sequence configuration is identical to advanced arbitrary waveform configuration discussed previously.
- **Show** checkbox: When this is not selected, the channel is no longer visible in the **Channel Settings** section of the SFP.

Arbitrary Data

The **Arbitrary Data** section is used to create waveforms and sequences that will be used to generate arbitrary waveform. The following options are provided.

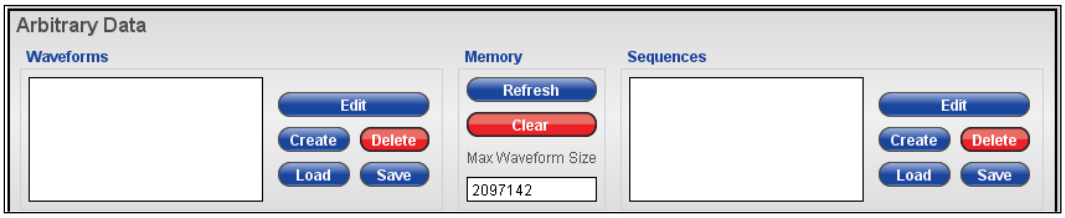


FIGURE 5-5: ARBITRARY DATA SECTION

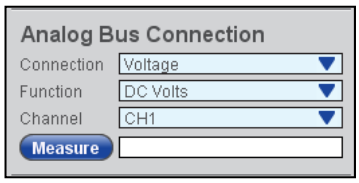
- **Waveforms/Sequences**: Indicates the waveforms/sequences available to the EX1200-360x. The corresponding button allow the user to do the following:
  - **Edit**: When a waveform is selected from the **Waveforms/Sequence** text box, clicking on this button will open the waveform for editing.
  - **Create**: Launches the waveform/sequence editor, allowing the user to manually define new waveform/sequence data.
  - **Load**: Allows the user to load waveform/sequence data from a CSV file.
  - **Delete**: Removes the selected waveform(s)/sequence(s) from the EX1200-360x's memory.
  - **Save**: Allows the user to export the selected waveform/sequence as a CSV file.
- **Memory**: Allows the user to manage stored waveform and sequence data.
  - **Refresh**: When clicked, all of the EX1200-360x's memory is read and the lists of waveforms and sequences are updated accordingly. Because waveform and sequence data can be very large, they are not periodically updated in the **Channels Output** display as other settings are.
  - **Clear**: Clears all of the EX1200-360x's memory, deleting all waveforms and sequences
  - **Max Waveform Size**: The maximum number of steps currently available for new waveforms.

Generation

The **Generation** section initializes waveform generation for all channels. The following options are provided.

- **Initiate/Abort** button: Generation begins, or ceases, respectively, on all channels when this button is clicked.
- **Auto Abort** checkbox: When selected, the user can temporarily abort waveform initiation in order to access waveform memory. Since accessing the memory of the EX1200-360x is not allowed during generation, this function is provided as a convenience. If this is not selected, the user must manually abort generation prior to creating or editing waveforms or sequences.

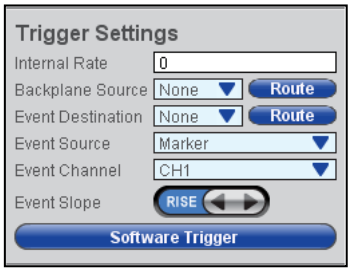
Analog Bus Connections



The **Analog Bus Connections** section defines the analog bus lines to which specified EX1200-360x channels will connect. Connecting the EX1200-360x to the analog bus allows the user measure the voltage/current being generated by the EX1200-360x using the internal DMM. When a Connection other than **None** is selected, the following options are provided:

- **Connection**: Defines the connections that are made between the EX1200-360x and the EX1200 mainframe's analog backplane.
  - **None**: Indicates that no backplane connections will be made.
  - **Voltage**: Connects the selected **Channel** to the "HI" and "LO" lines of the EX1200's analog bus. This allows the integrated DMM to measure the voltage generated on the selected **Channel**.
  - **Current**: connect the selected channel to the "I" and "LO" lines of the EX1200's analog bus. This allows the integrated DMM to measure the current generated on the selected **Channel**.
- **Function**: Sets the EX1200 DMM measurement function used when the **Channel**'s output is measured.
- **Channel**: Selects the EX1200-360x channel that will be connected to the analog bus.
- **Measure** button: Configures and initiates a single measurement using the integrated DMM. The measurement result is shown in the adjacent text box.

Trigger Settings



The **Trigger Settings** section defines the EX1200-360x's trigger mechanism and determines how the instruments will progress through the trigger model.

- **Internal Rate**: When **Internal** is selected as the **Trigger Source** under **Advanced** settings, it defines the internal periodic trigger rate.
- **Backplane Source**: When **Backplane** is selected as the **Trigger Source** under **Advanced** settings, it defines the backplane trigger line that will be used to receive a trigger event.
- **Event Destination**: Defines the backplane trigger line where the EX1200-360x will drive a trigger event output.
- **Route** buttons: Configures the EX1200 mainframe to route the selected BPL line to an external trigger (ALARM0, LXI0 through LXI7, LAN0 through LAN7, DIO0 through DIO7).
- **Event Source**: Defines the signal that causes the **Event Destination** to change its state. The user selects from **Marker**, **Front Panel**, **Burst Complete**, **Waiting For Trigger**, or **Generating**.
- **Event Channel**: For the **Burst Complete** and **Waiting For Trigger Event Sources**, defines the channel that will monitor for trigger events.

- **Event Slope**: Defines whether the **Event Destination** is driven by a rising or falling edge when the event occurs.
- **Software Trigger** button: Sends a software trigger event, causing any channel configured with the **Software** trigger source to be triggered.

Marker Settings

The **Marker Settings** section allows the user to configure the MARKER_OUT line. The following options are provided:

- **Enabled** checkbox: Enables marker output when selected.
- **Source**: Selects the channel from which the marker signal is taken.
- **Width**: Sets the pulse width of the marker output in seconds.
- **Slope**: When set to **Rise**, the marker output is normally low and asserts a high pulse. When **Fall** is selected, the marker output is normally high and asserts a low pulse.

System Buttons

The four orange buttons at the top right of the **Status Display** are system buttons that affect the functionality of the EX1200-360x.

Lock/Unlock

The **Lock** button requests (or releases) exclusive access to the EX1200-360x. If the function generator will be calibrated using the SFP, a lock should be established to prevent any unintended access to the instrument.

Reset

Clicking the **Reset** button returns the EX1200-360x to its power-on default settings and values.

Self Test

The **Self Test** button executes the EX1200-360x's **SelfTest** driver function. The self-test checks for open, shorted, or "stuck" data and address lines and checks the address lines to ensure the integrity of waveform data storage.

Calibration

The calibration button launches a dialog box, shown in Figure 5-6, that allows the user to review and modify the currently loaded calibration constants and to generate an entirely new set of calibration constants using the internal DMM as a reference.

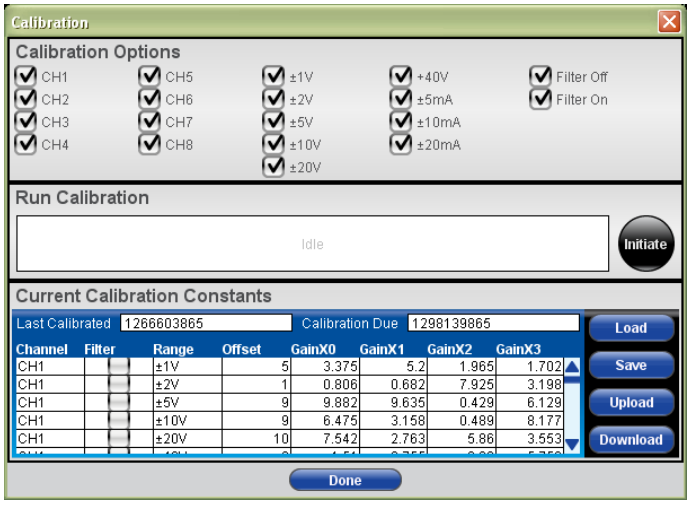


FIGURE 5-6: EX1200-360X CALIBRATION WEB PAGE

- **Calibration Options**: These checkboxes allow specific channels, ranges, and filter options to be included/excluded when generating new calibration constants.
- **Run Calibration**: Click the **Initiate** button in this field to initiate the calibration process in order to generate new calibration constants. The progress bar shows the percentage of the calibration process that is so far complete, along with a short message describing the current operation.
- **Current Calibration Constants**: This table lists the currently loaded calibration constants for each channel, range, and filter option. The **Last Calibrated** field display the IEEE 1588 time when the displayed calibration constants were generated, while the **Calibration Due** field identifies when the constants expire and the instrument should be recalibrated. The first three columns (**Channel**, **Filter**, and **Range**) are read-only and identify the options applicable to the selected row. The **Offset**, **GainX0**, **GainX1**, **GainX2**, and **GainX3** columns are the actual calibration adjustments for each channel/filter/range combination.

NOTE	Although the **Offset**, **GainX0**, **GainX1**, **GainX2**, and **GainX3** values can be modified, changing the constants is not advised, as incorrect values will adversely affect the accuracy of the EX1200 360x's outputs. Modifications of this type should only be performed by qualified personnel.

  - **Load**: Clicking this button allows the user to load a set of calibration constants stored in the EX1200-360x's non-volatile memory which will be used for future outputs. The non-volatile memory provides eight user calibration slots and one factory calibration. When clicked, a dialog box is launched which prompts the user to select a slot to load.
  - **Save**: Clicking this button allows the currently active calibration constants to be saved to non-volatile memory. If the user attempts to overwrite factory calibration, a password prompt will appear. Overwriting factory calibration constants is highly discouraged. The password can be provided by VTI Customer Support if necessary. User calibration slots, however, can be overwritten without a password.
  - **Upload**: Clicking this button allows the user to upload a complete calibration file to the EX1200-360x's non-volatile memory. A dialog box similar to the Save button will appear. Again, although it is possible to overwrite the factory calibration, this action is

password protected. Once a slot is selected, a prompt appears allowing the user to select the desired calibration file.

- **Download**: Clicking this button allows the user to generate a CSV file with the calibration constants from one of the non-volatile calibration files. A dialog box allows the user to select the calibration slot to download data from which is then followed by a dialog box to choose where the file will be saved.

Although the calibration though the soft front panel provides the user the ability to perform in-house calibration, annual factory calibration of the EX1200-360x is still recommended, as factory calibration involves use of an 8.5 digit multimeter which allows for a full assessment of the output current accuracy. Some of the EX1200-360x's specification are otherwise limited by the specifications of the EX1200 DMM.

SECTION 6

APPLICATION NOTES

USING INTERNAL DMM TO CALIBRATE THE INSTRUMENT

To reduce system downtime, the EX1200-360x can be calibrated in the field when paired with an EX126x mainframe. This unique feature allows the user to maintain high levels of accuracy and confidence in measurement, as the specifications can be confirmed just prior to running a test, eliminating any temperature variations or time drifts that may occur from test to test.

For information on how to perform calibration using the EX1200 DMM and embedded web page, refer to the *Calibration* discussion in *Section 5*.

TYPICAL WAVEFORM EXAMPLES

Figure 6-1 below shows a typical example of square wave generation. This is a 50 kHz square wave, operating at full voltage output with a full load of 20 mA.

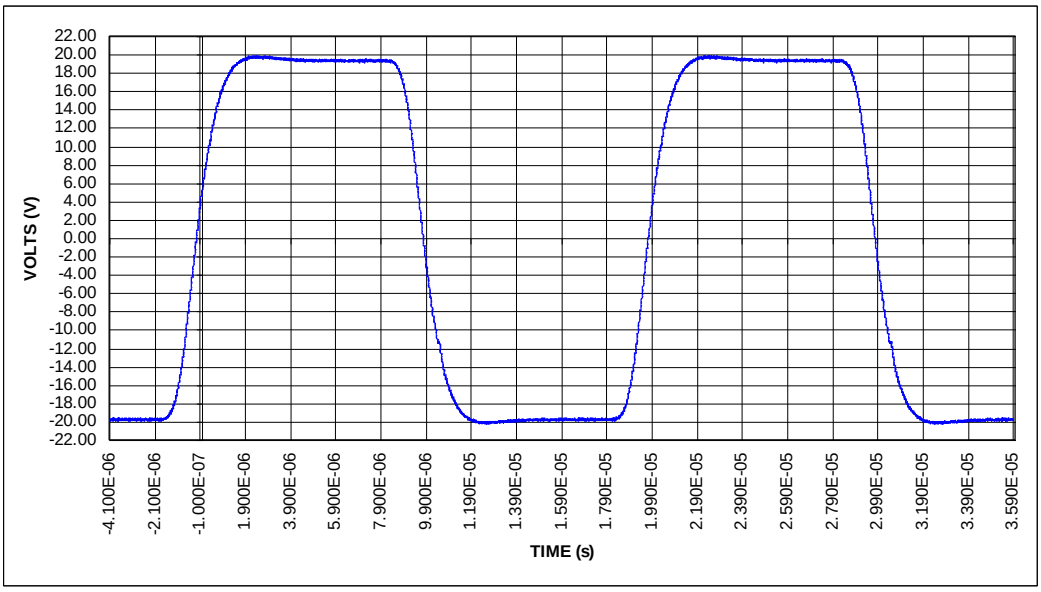


FIGURE 6-1: 50 KHZ SQUARE WAVE GENERATED BY EX1200-360X

The examples below show typical sine waves. These are 50 kHz (Figure 6-2), 10 kHz (Figure 6-3), and 1 kHz (Figure 6-4) sine waves outputting full voltage with a full (1 kΩ) load of 20 mA. The

figures show the response with and without the filter. As the plots indicate, it is highly recommended that the filter be used for frequencies above 1 kHz.

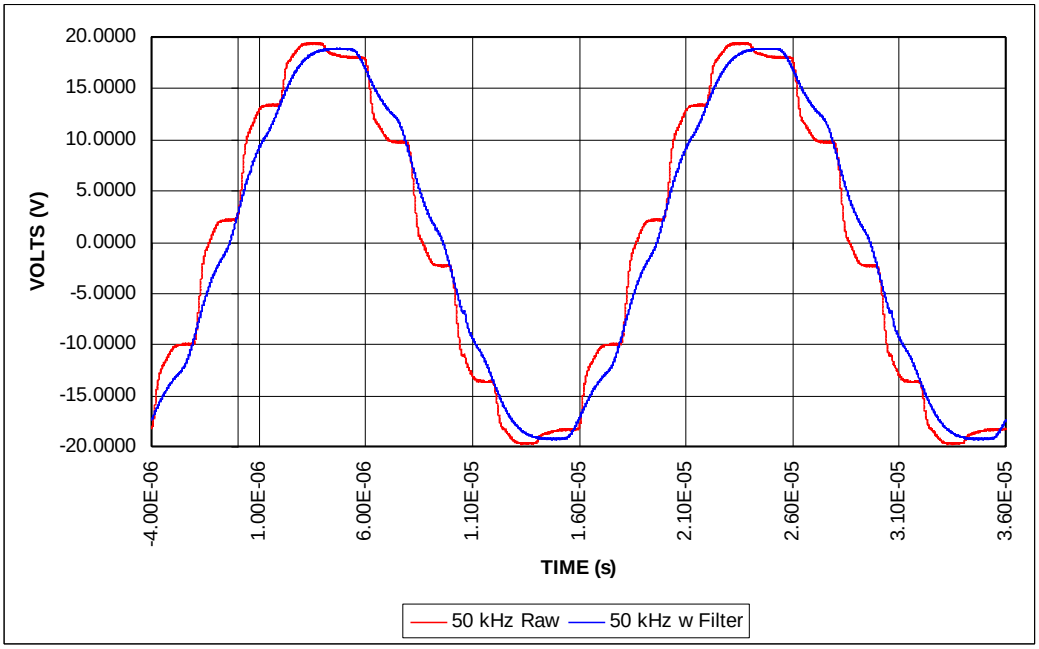


FIGURE 6-2: 50 KHZ SINE WAVE GENERATED BY EX1200-360X

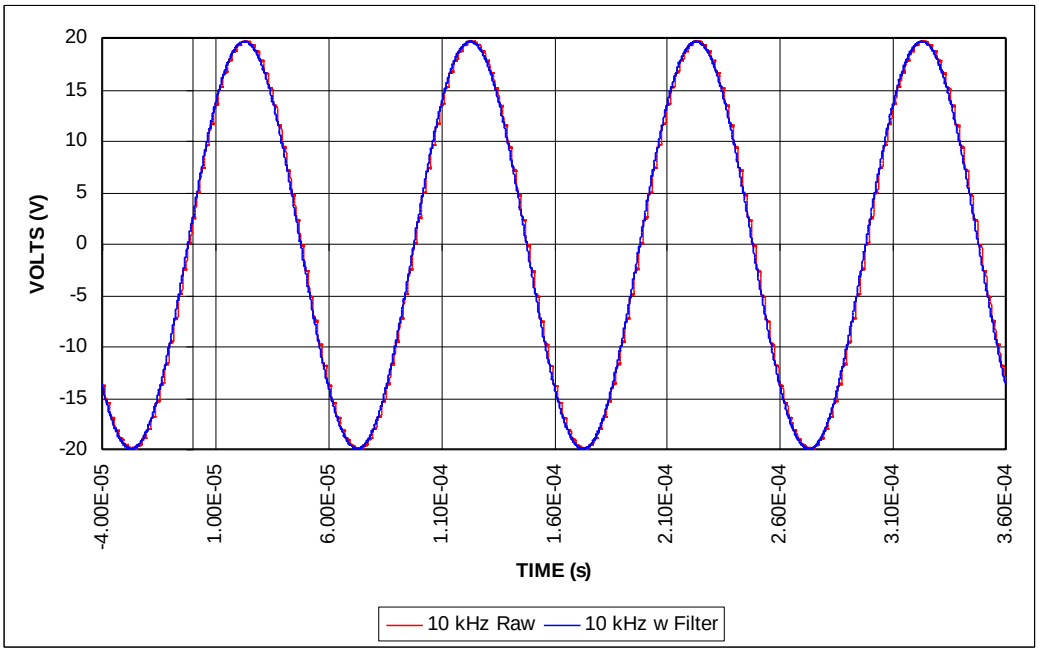


FIGURE 6-3: 10 KHZ SINE WAVE GENERATED BY EX1200-360X

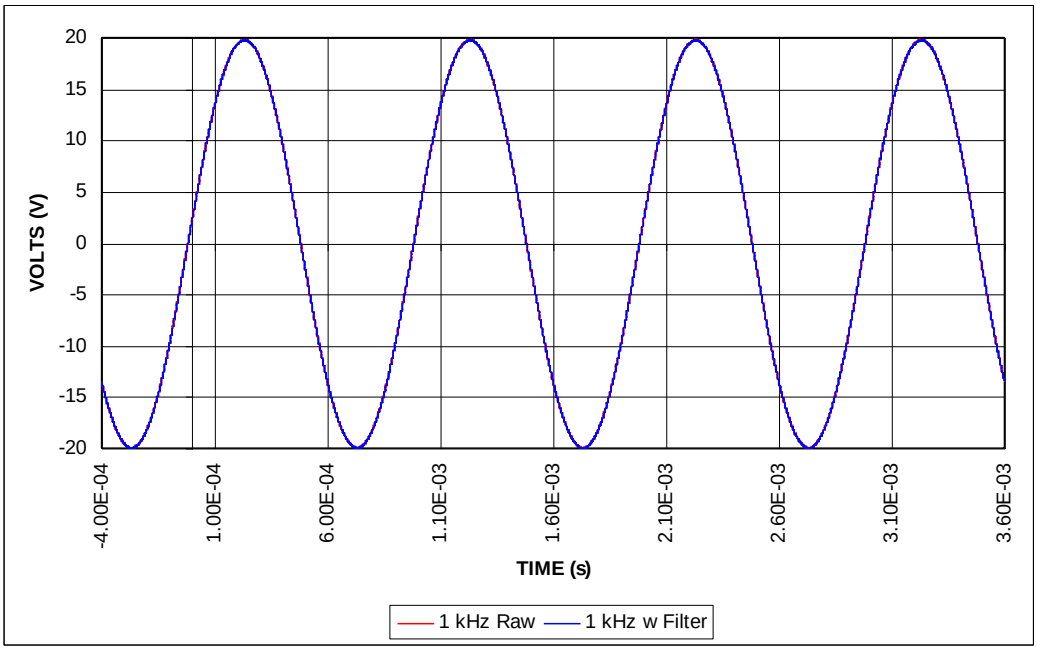


FIGURE 6-4: 1 KHZ SINE WAVE GENERATED BY EX1200-360X

CONNECTING VOLTAGE CHANNELS IN SERIES

WARNING	High-voltage waveforms can be potentially dangerous. Use extreme caution when wiring any EX1200-360x channels in series.

Because all channels of the EX1200-360x card are isolated, it is possible to wire voltage channels in series to generate high-voltage waveforms (up to 320 V peak-to-peak). When wiring channels in series, all channels must have the same waveform. As an example, if the goal is to produce a sine wave, then each channel in the series must be set to the same frequency and the same phase. Each channel must also have the same filter setting to eliminate a possible slew rate conflict between the channels.

The channels must also have similar amplitudes to ensure that the waveform will be equally distributed among the channels. For example, if eight channels are wired in series and the desired peak-to-peak signal is 80 V, then each channel should be set to 10 V.

The waveforms in Figure 6-5 show an EX1200-3608 with all eight channels wired in series. Because each channel has some common mode capacitance relative to chassis, bandwidth will be limited. In the examples shown below, the waveform is set to 200 Hz. When fully loaded to 20 mA, a small reduction in the square wave slew rate can be seen in the blue trace. As shown by the green trace, applying the filter at this low frequency has virtually no effect on the waveform.

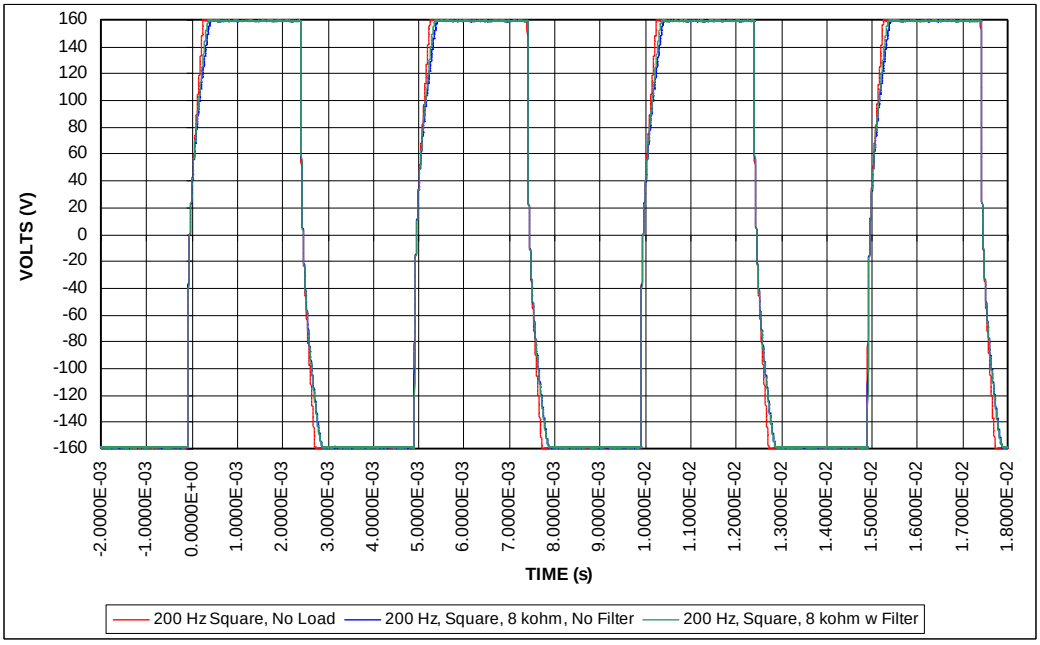


FIGURE 6-5: SLEW RATE REDUCTION FOR CHANNELS IN SERIES

For the single sine wave shown in Figure 6-6, the slew rate induced error is missing due to the reduced rate of change as the waveform exceeds ±50% of its programmed amplitude. The frequency is still 200 Hz.

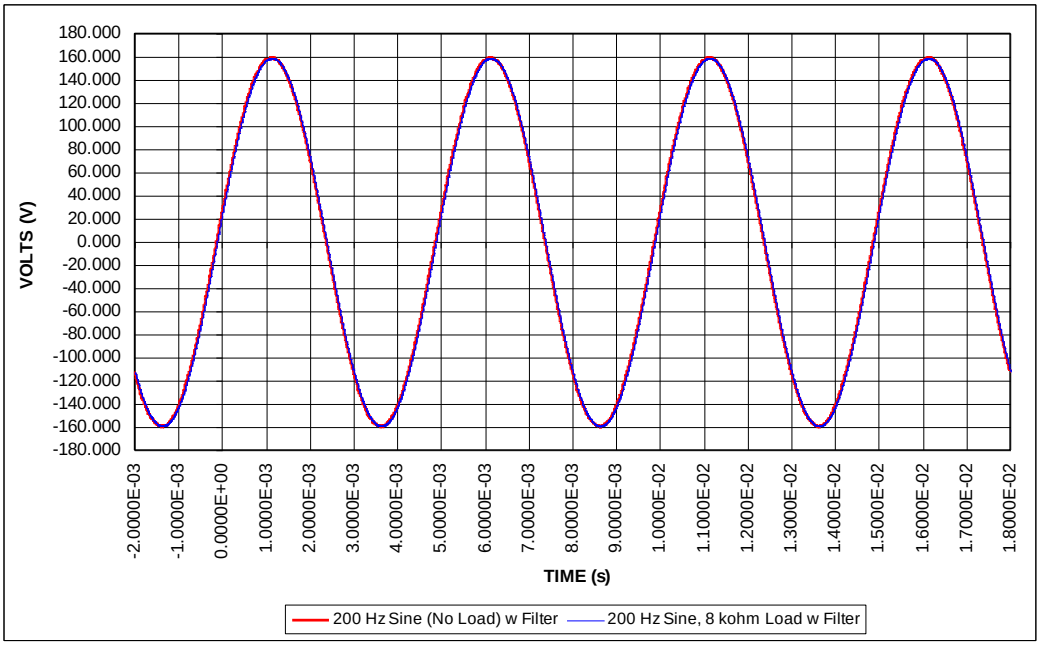


FIGURE 6-6: CHANNELS IN SERIES WITH SLEW RATE ERROR REMOVED

MULTIPHASE STIMULUS

In Figure 6-7, channels 1 and 2, 3 and 4, and 5 and 6 are in series. Each phase is set to a nominal output of ±40 peak-to-peak (±20 V peak-to-peak for each channel). Each signal has a load of ±20 mA peak. While this type of application does not generate a large amount of power (0.8 W peak), it does allow for multiphase signals to be applied to the input of control loops. In this type of application, control loops for multi phase power converters may be stimulated without the need for a large (and potentially dangerous) power source.

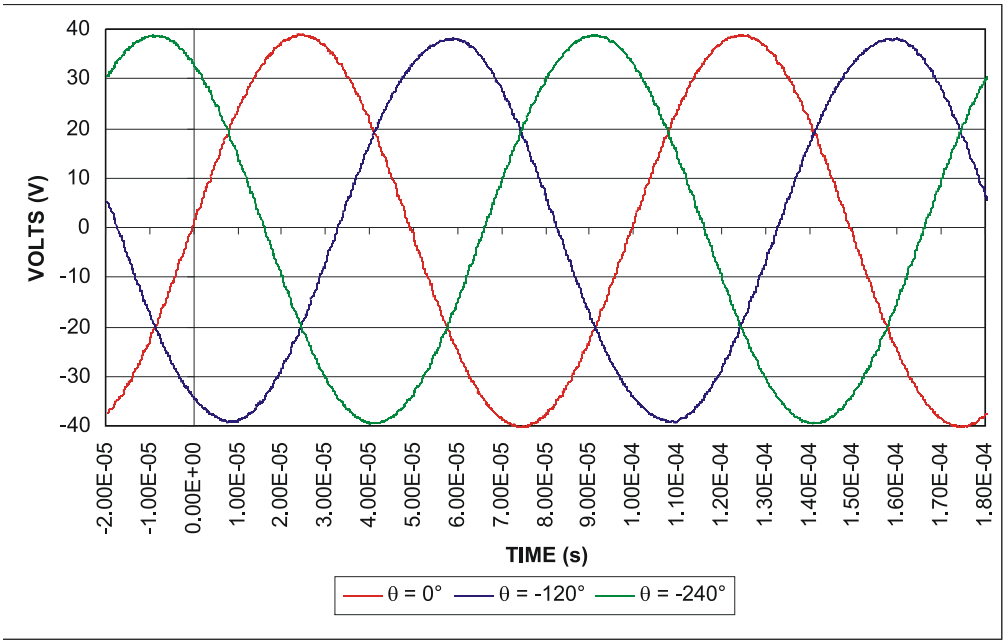


FIGURE 6-7: MULTIPHASE STIMULUS

CONNECTING CURRENT CHANNELS IN PARALLEL

WARNING	High-current waveforms potentially stress and damage the load (UUT). Use caution when wiring any load to a high current-source to prevent UUT over-current conditions.

As previously mentioned, all channels of the EX1200-360x card are isolated, allowing for channels to be wired in parallel and generate high-current waveforms (up to 160 mA). As was true when wiring voltage channels in series, channels wired in parallel wiring must 1) be set to output the same waveform, 2) be set to the same frequency, 3) be in the same phase, and 4) be set to the same filter setting. All of the parallel channels must have similar amplitudes to ensure that the waveform is equally distributed among the channels. As an example, if eight channels are wired in parallel, and the desired peak-to-peak signal is 0.080 A, each channel should be set to 0.010 A.

The waveforms below show all eight channels of the EX1200-3608 card wired in parallel with a load of 124.5 Ω. Because each channel has the same small common mode capacitance, relative to chassis, the bandwidth will be much higher than for voltage mode. In the examples shown below, the waveforms are set to 50 kHz, 10 kHz, and 1 kHz. All waveforms look excellent up to 50 kHz.

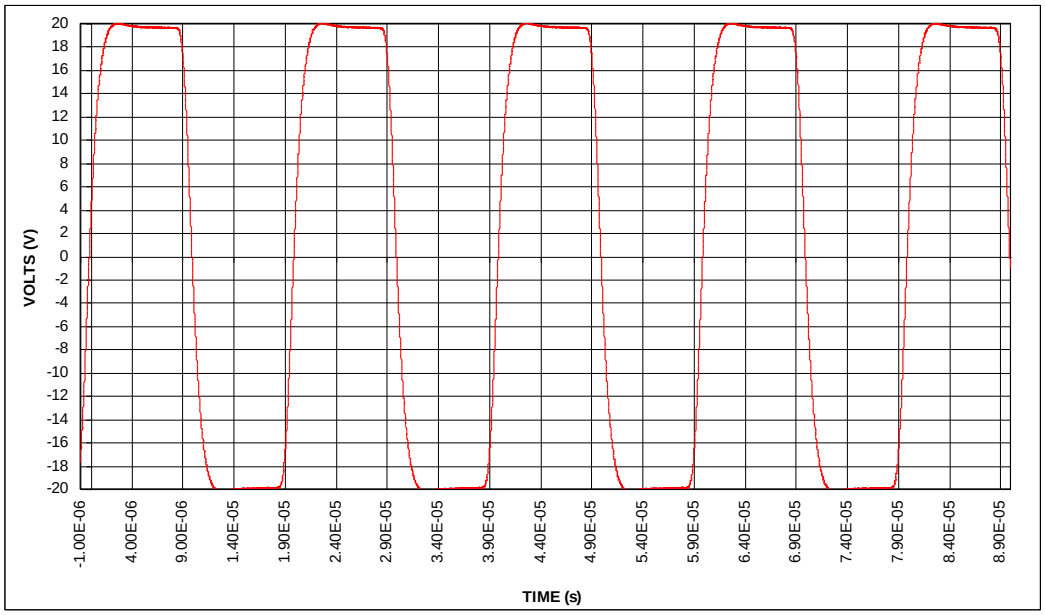


FIGURE 6-8: 50 KHZ SQUARE WAVE WITH EIGHT CHANNELS IN PARALLEL

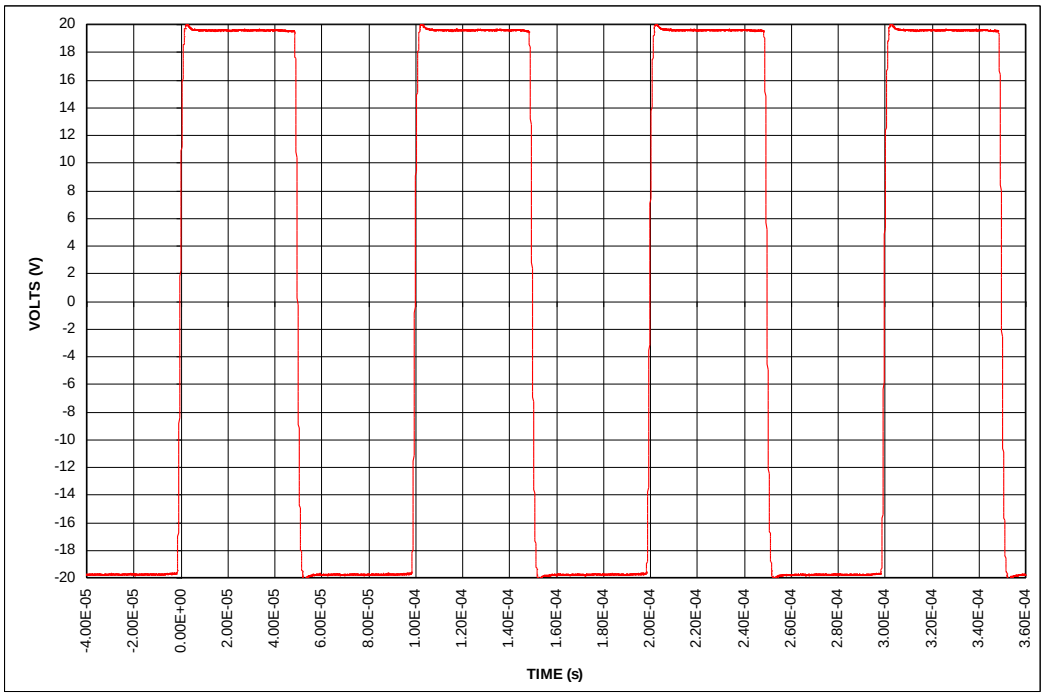


FIGURE 6-9: 10 KHZ SQUARE WAVE WITH EIGHT CHANNELS IN PARALLEL

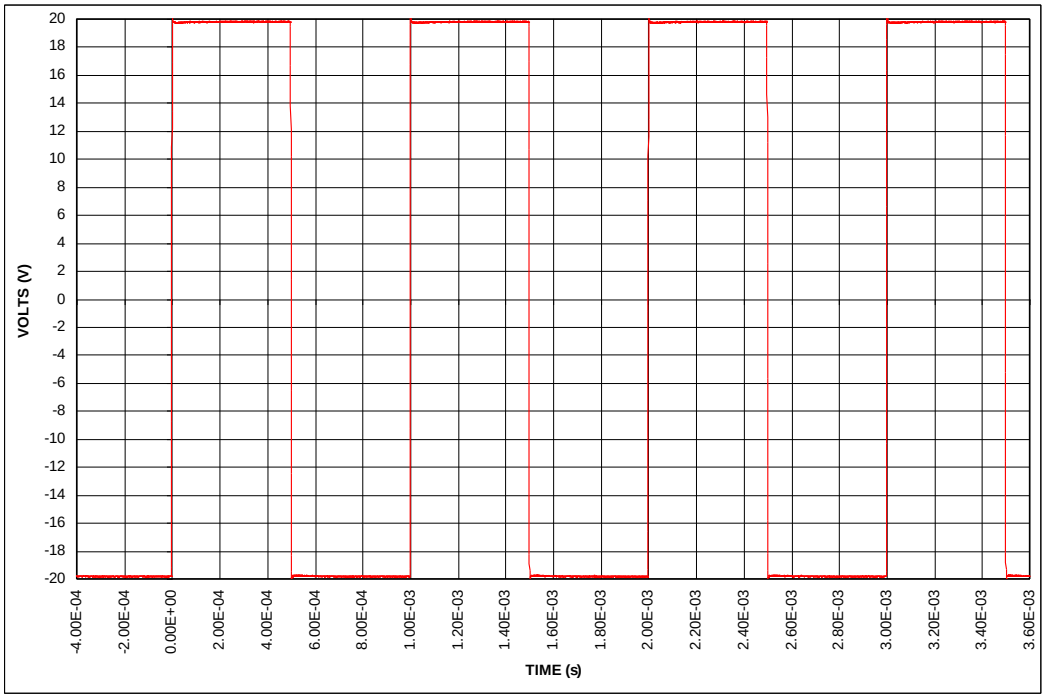


FIGURE 6-10: 1 KHZ SQUARE WAVE WITH EIGHT CHANNELS IN PARALLEL

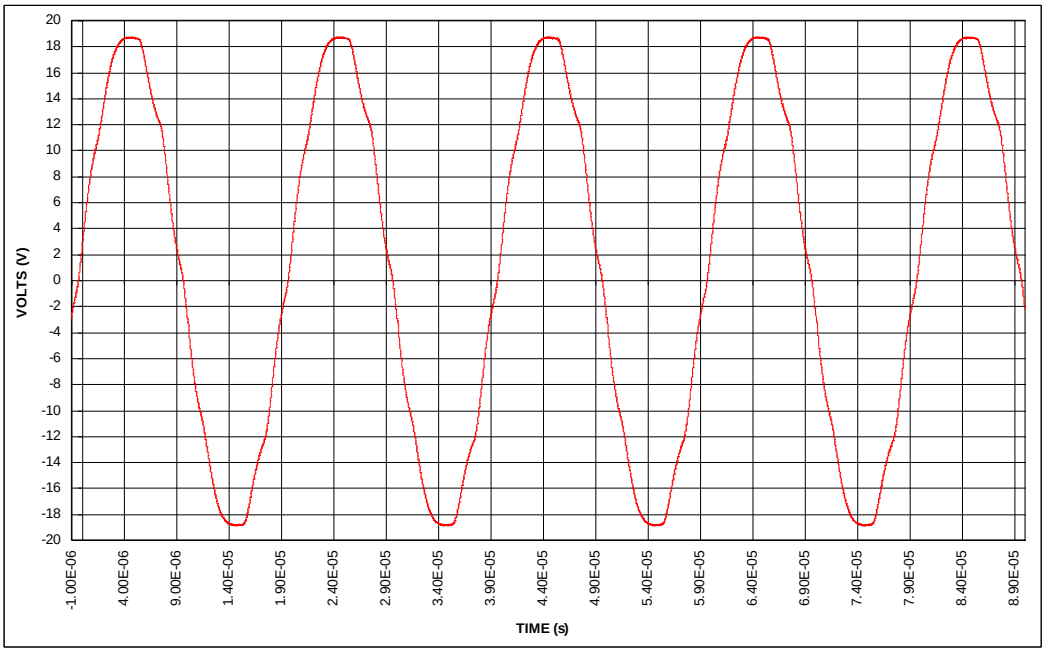


FIGURE 6-11: 50 KHZ SINE WAVE WITH EIGHT CHANNELS IN PARALLEL

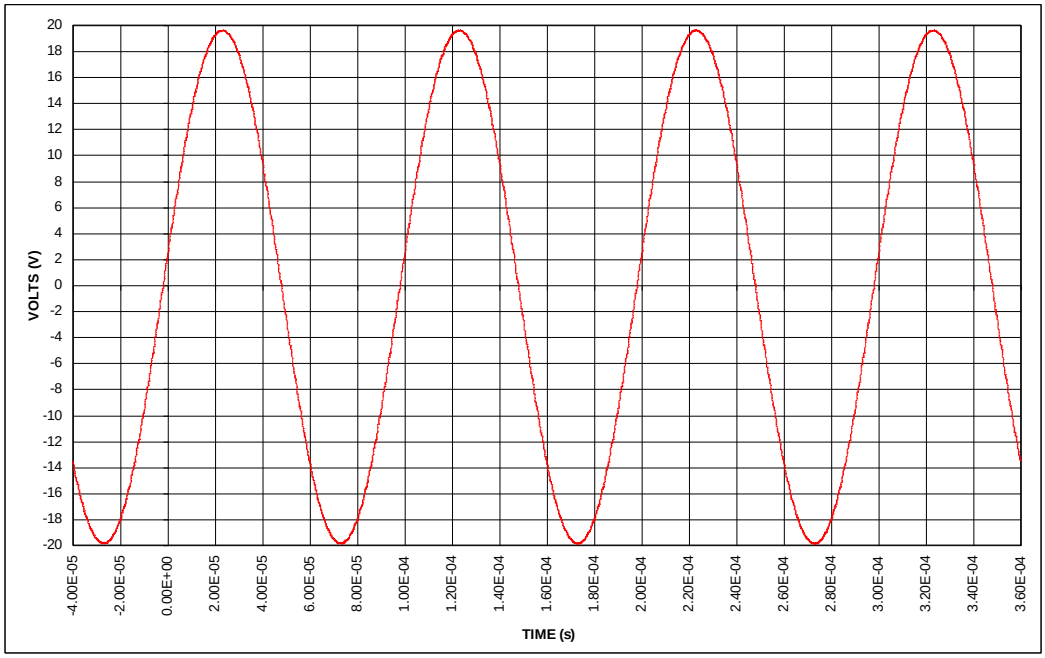


FIGURE 6-12: 10 KHZ SINE WAVE WITH EIGHT CHANNELS IN PARALLEL

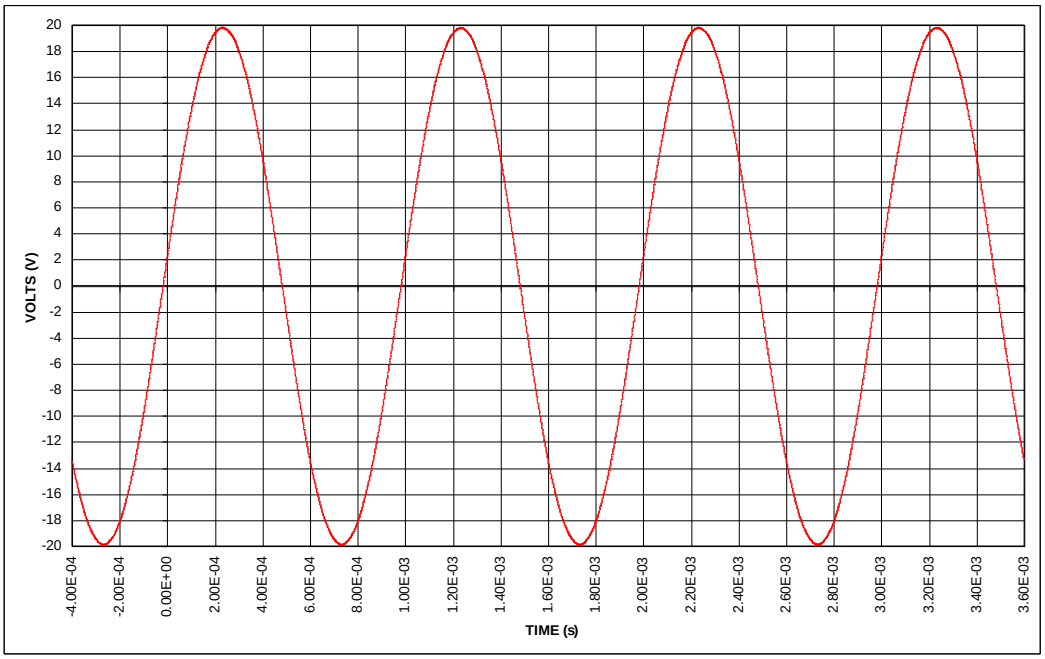


FIGURE 6-13: 1 KHZ SINE WAVE WITH EIGHT CHANNELS IN PARALLEL

USING SENSE LINES

Normally, a voltage source operating in a constant voltage mode achieves optimum line and load regulation, the lowest output impedance, the lowest drift, and the fastest transient recovery performance at the power supply output terminals. If the load is separated from the output terminals by any line length, the current flowing through this line will create a voltage drop proportional to the impedance of the load leads. This results in the degradation of the optimal characteristics at the load terminals previously mentioned.

With remote error sensing, it is possible to connect the sense terminals, thus defining the input of the voltage feedback amplifier, directly to the load so that the power supply regulates the voltage at the load rather than at the voltage source output terminals. Thus, the voltage at the voltage source output terminals shifts to compensate the voltage drop in the load leads, thereby maintaining a constant voltage at the load terminals.

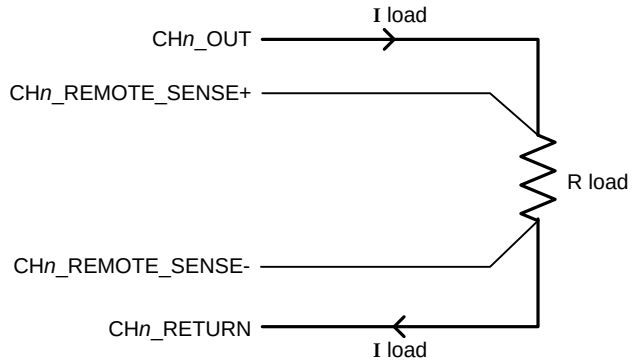


FIGURE 6-14: CONNECTING REMOTE SENSE LINES

When using the remote sense lines, there are several points to consider:

- The sense and power lines should **always** be interconnected.
- The dc-common point should be as close as possible at the load(s) end.
- Arrange the sense and power lines in the same cable form to minimize noise pick-up. Sense line wires should run parallel or be twisted.
- In noisy environments or when long sense lines (>1 m) are used, shielded cables should be used to protect the sense signals from external noise.
- The maximum voltage drop which can be compensated for by remote sensing is limited (see *EX1200-360x Specifications* for details). This limitation sometimes dictates the use of a larger wire cross sections than would be required by the current rating or impedance considerations.
- Although dc and low-frequency performance is improved by remote sensing, over-/under-compensation associated with long load and sense leads can affect the stability of the feedback loop seriously enough to cause oscillation during waveform generation, especially at higher frequency and voltages,. This problem cannot directly be overcome. The only measure to prevent oscillation is the reduction of voltage drops and line impedances.
- When connecting several channels ***in series***, the only the sense lines that need to be connected to the load are the ones from the first and last channels in the series. See Figure 6-15.
- Channels in ***current*** mode do not need the sense lines.

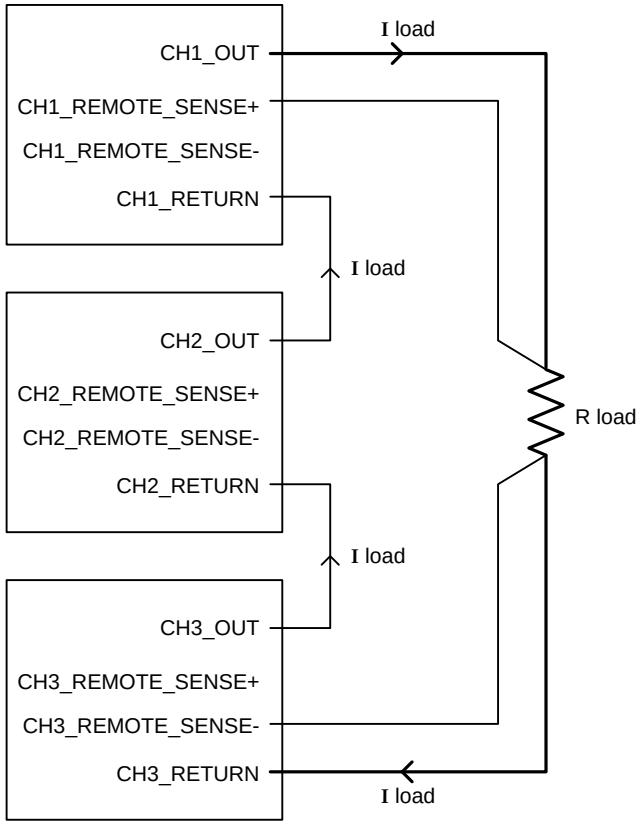


FIGURE 6-15: USING THE SENSE LINES WITH CHANNELS CONNECTED IN SERIES

INDEX

T

W