EX1200-7008

PROGRAMMABLE RESISTOR AND RTD SIMULATION MODULE

USER'S MANUAL

P/N: 82-0127-008
Released May 5th, 2017

VTI Instruments Corp.

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

TABLE OF CONTENTS

CERTIFICATION

VTI Instruments Corp. (VTI) certifies that this product met its published specifications at the time of shipment from the factory. VTI further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology (formerly National Bureau of Standards), to the extent allowed by that organization's calibration facility, and to the calibration facilities of other International Standards Organization members. Note that the contents of this document are subject to change without notice.

WARRANTY

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

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

RESTRICTED RIGHTS LEGEND

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.

WARNINGS (CONT.)

Avoid Electric Shock

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

Ground the Product

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

Operating Conditions

To avoid injury, electric shock or fire hazard:

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

Improper Use

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

SUPPORT RESOURCES

Support resources for this product are available on the Internet and at 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
Divyasree N R Enclave, Block A,
4th Floor, Site No 1, EPIP Industrial Area,
Whitefield, Bengaluru 560066, India
Phone: +91 (0) 80 6782 3200
Fax: +91 (0) 80 6782 3232

Asia Support
Phone: +852 9177 6127

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 EX1200-7008 is a multichannel resistance temperature detector (RTD) sensor simulation card with the ability to accurately simulate a variety of RTDs on a single module. This card provides eight independent resistance output channels, which can be configured programmatically to characterize the behavior of a measurement and/or signal conditioning system. Multiple simulation channels can be synchronized to create virtual thermal environments which can be used to evaluate the performance of safety systems.

The majority of RTD simulators on the market use series relay ladders that switch relays in and out of circuit to achieve the desired value. These modules are subject to relay contact bounces that can create oscillations on the output before settling to the commanded resistance value. These oscillations appear as unwanted "thermal shocks" to the system under test. Ageing and contact degradation will further deteriorate the performance of these simulators. The EX1200-7008 implements an advanced, solid-state servomechanism to actively simulate resistance values. This method produces "bounceless" changes in resistance and smooth ramps at the required rate of change. Ageing effects are nullified, as the resistance value is continuously monitored in a closed loop and corrected for errors.

Each EX1200-7008 channel can be programmatically configured to simulate platinum (Pt) and meets the DIN, IEC, and ITS-90 curve-fitting standards. Required temperature values can be directly commanded, eliminating the need to derive conversion algorithms in application software. Resistance values from 4 Ω to 10 kΩ can be realized, with 25 mΩ resolution throughout the range. Two or more channels can be combined to simulate any type of thermistor. It can also accept continuous or pulsed type current excitation sources, making it truly "universal".

The EX1200-7008 is part of the EX1200 family of products and can be mixed and matched with other EX1200 series modules to configure high-density measurement and switching systems. This allows users the ability to accommodate a wide range of mixed signals into a standard EX1200 series mainframe and allows for the creation of a complete measurement system in as small as a 1U rack space.

FEATURES

- Eight isolated, universal, 2-/4-wire RTD simulator channels.
- Fast, monotonic, glitch-free resistance value programming.
- Simulation of resistors and platinum and copper RTDs:
  - PT100 (385)
  - PT200 (385)
  - PT500 (385)
  - PT1000 (385)
  - PT100 (392)
  - PT200 (392)
  - PT500 (392)
  - Cu10
  - Cu100
- Pulsed and continuous excitation inputs.
- Direct temperature value programming per DIN, IEC, and ITS-90 standards.
- Extensive triggering and marker output capability.
- Synchronization of level changes with input measurements to facilitate.
- Static and ramp resistance and preset point functions are supported.

PROGRAMMER'S REFERENCE

Please refer to the driver's online help documentation for assistance with programming the EX1200-7008 (typically found in installed drivers root directory: **Start → VTI Instruments → VTEXRTDSimulator**).

Please check the latest driver availability in relevant product page at VTI Instruments web site (http://www.vtiinstruments.com/).

EX1200-7008 SPECIFICATIONS

PHYSICAL SPECIFICATIONS	
FRONT PANEL CONNECTOR	
	44-pin D-sub connector
POWER CONSUMPTION	
3.3 V	350 mA
5 V	10 mA
24 V	540 mA
GENERAL SPECIFICATIONS	
NUMBER OF CHANNELS	
	8
SUPPORTED MODES	
	Open, Short, Static Resistance, Static Temperature, Resistance Ramp, Resistance Preset, and Temperature Preset.
SIMULATION MODES	
	Resistance and temperature modes
SUPPORTED RTD	
	PT100 (385), PT200 (385), PT500 (385), PT1000 (385), PT100 (392), PT200 (392), PT500 (392), Cu10 and Cu100
COMPLIANCE VOLTAGE	
	10 V
EXCITATION / INPUT CURRENT	
	+/-10.5mA(max) (pulsed/continuous), 1 mA to 10 mA Max (DC to 2.5 ms pulse width with min 10mSec PRI)
SIMULATION STEP	
Resistance mode	20 mΩ steps (programmable)
Temperature mode	0.1 °C steps
RESISTANCE ACCURACY	
	0.1 % +- 0.075 Ohm
TEMPERATURE ACCURACY	
	0.1 °C (PT100, 385 type)
COMMON MODE INPUT	
	250 V_{PEAK}
RESISTANCE SIMULATION RANGE	
	4 Ω to 10 kΩ (open and short)
OPEN CIRCUIT RESISTANCE	
	> 10 MΩ
SHORT CIRCUIT RESISTANCE	
	< 500 mΩ
ISOLATION	
	Channel-to-channel, galvanic isolation (300 V)
SETTLING TIME	
	10 ms
FRONT PANEL TRIGGER VOLTAGE LEVEL	
	TTL Compatible
WIRE MODES	
	2-wire and 4-wire modes

RAMP SPECIFICATIONS	
RAMP TIME	
	10 ms to 4 days 1 ms intervals
MARKER OUTPUT	
	BPL or front panel triggers on configured value and configured channel
TIME RESOLUTION	
	10 ms with values coerced to the nearest millisecond
TRIGGER SOURCE	
	Software, BPL lines, LXI and LAN triggers (through VTEXSystem driver) and front panel trigger
PRESET SPECIFICATIONS	
ONBOARD MEMORY	
	10 points per channel
SLOPE	
	Positive and negative
INCREMENT TRIGGER SOURCE	
	Software, BPL lines, LXI, and LAN triggers (through VTEXSystem driver) and front panel trigger
MAXIMUM TRIGGER FREQUENCY	
	100 Hz
ACCESSORIES	
MATING CONNECTOR (SOLDER CRIMP)	
Description	44-pin male connector
VTI part number	27-0061-044
Manufacturer/part number	17EHD-044PAA-000 (Positronic and AMP)
MATING CONNECTOR (CRIMP-STYLE)	
Description	44-pin male connector with plastic hood and crimp pins (22 AWG – 30 AWG)
VTI part number	27-0390-044
Manufacturer/part number	ODD44M10Z0Z (Positronic and AMP)
MATING CONNECTOR (PLASTIC HOOD)	
Description	44-pin connector backshell
VTI part number	27-0086-044
Manufacturer/part number	5745833-1 (Positronic and AMP)
CRIMP TOOL	
Description	Final assembly, crimp tool set, Positronic ODD series, 22 AWG
VTI part number	70-0297-001
Manufacturer/part number	(Positronic 9507 and 9502-4-0-0)
PRE-ASSEMBLED, UN-TERMINATED WIRING HARNESS	
Description	Unterminated Cable assembly
VTI part number	70-0363-502
TERMINAL BLOCK	
Description	Final assembly, EX1200-TB44 (terminal block)
VTI part number	70-0367-007

SECTION 2

USING THE INSTRUMENT

UNPACKING

When an EX1200-7008 is unpacked from its shipping carton, the contents should include the following items:

An EX1200-7008
LXI Quick Start Guide (P/N: 82-0123-000)
EX1200-7008 User's Manual (this manual)
EX1200-7008 IVI, Linux, or LabVIEW Driver (included on Distribution CD)

All components should be immediately inspected for damage upon receipt of the unit. ESD precautions should be observed while unpacking and installing the instrument into an EX1200 series mainframe.

DETERMINE SYSTEM POWER REQUIREMENTS

The power requirements of the EX1200 mainframes are provided in the *Specifications* section of Section 1. It is imperative that the mainframe provide adequate power for the modules installed. For more information on EX1200 mainframe power consumption, please refer to the *EX1200 Series User's Manual* (P/N: 82-0127-000) for more information. 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.

NOTE For more information on power requirement calculations, refer to and review *Appendix B* in the *EX1200 Series User's Manual* 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.

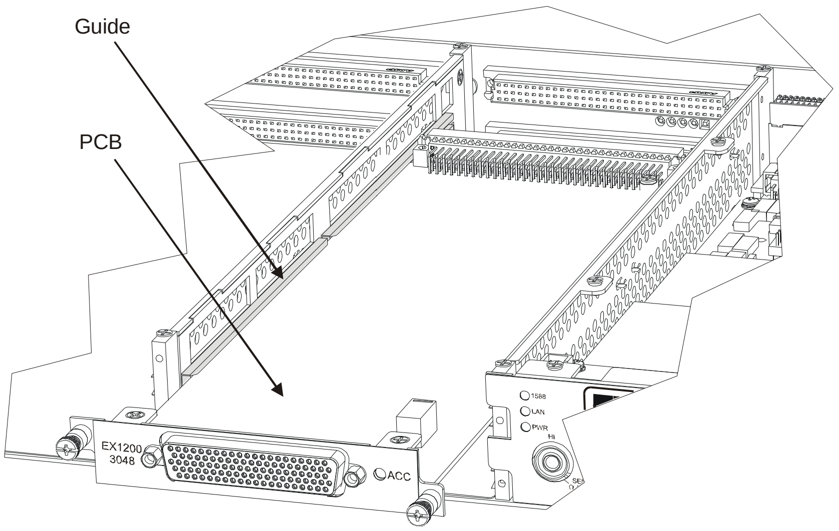


FIGURE 2-1: MODULE INSTALLATION (EX1200-3048 USED AS EXAMPLE)

MAXIMIZING MEASUREMENT PERFORMANCE

This section discusses tips and procedures that can help maximize the actual performance realized with the EX1200-7008 and aid the user in avoiding some common pitfalls associated with making measurements.

Warm-up Time

Warm-up time is not required for EX1200-7008. However, if combined with other EX1200 family products, refer to appropriate instruments user manuals for recommended warm-up time.

Maximum Input Source Capacitance

EX1200-7008 uses solid state-based resistance simulation technique. It monitors a) source excitation and output voltage and b) source current excitation and output simulation voltage. These two properties are input to an algorithm which controls the simulation servo loop for maximum accuracy. If ***input source capacitance*** is high, it makes closed loop servo controls feedback signals to oscillate and makes the system unstable.

CONNECTOR PIN/SIGNAL ASSIGNMENT

The connector pins and their signal assignments are shown in Table 2-1 and Figure 2-2 below. For mating connector and accessory information, please see the *Accessories* section of the *EX1200-7008 Specifications*.

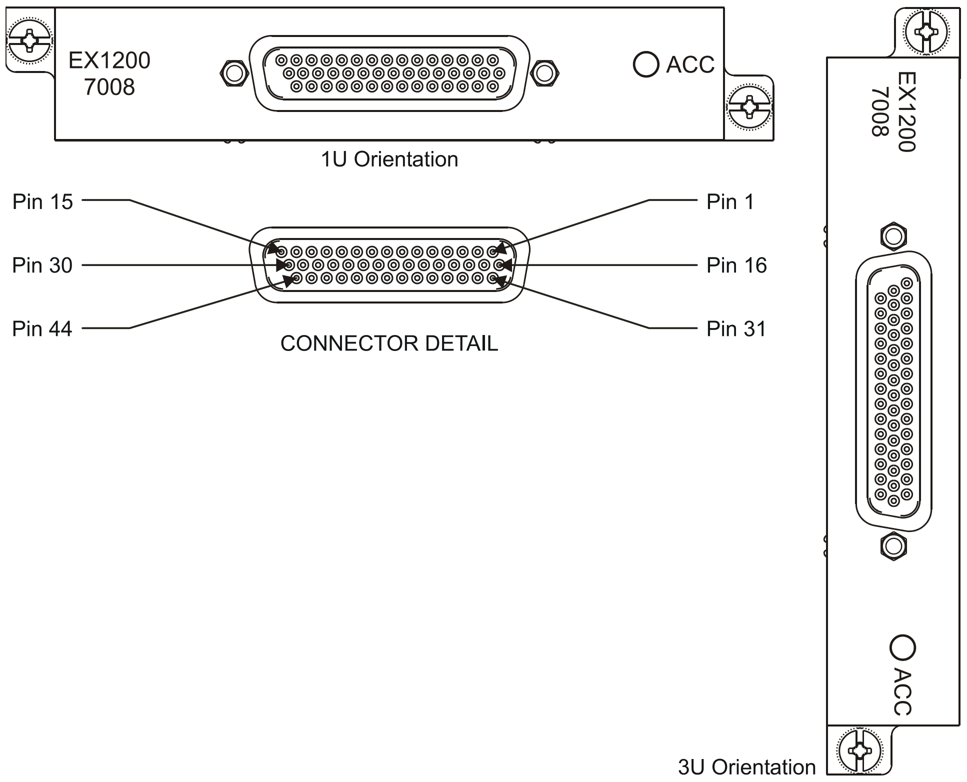


FIGURE 2-2: EX1200-7008 FRONT PANEL DETAIL

Pin	Signal	Pin	Signal	Pin	Signal
1	LO_CH2	**16**	SNSLO_CH2	**31**	HI_CH1
2	HI_CH2	**17**	SNSHI_CH2	**32**	LO_CH1
3	USR_SHIELD	**18**	USR_SHIELD	**33**	USR_SHIELD
4	LO_CH3	**19**	SNSLO_CH3	**34**	SNSHI_CH1
5	HI_CH3	**20**	SNSHI_CH3	**35**	SNSLO_CH1
6	LO_CH4	**21**	SNSLO_CH4	**36**	GND_D
7	HI_CH4	**22**	SNSHI_CH4	**37**	TRIGGER_OUT
8	USR_SHIELD	**23**	USR_SHIELD	**38**	USR_SHIELD
9	LO_CH5	**24**	SNSLO_CH5	**39**	EXT_CLK_IN
10	HI_CH5	**25**	SNSHI_CH5	**40**	TRIG_IN_OUT
11	LO_CH6	**26**	SNSLO_CH6	**41**	SNSHI_CH8
12	HI_CH6	**27**	SNSHI_CH6	**42**	SNSLO_CH8
13	USR_SHIELD	**28**	USR_SHIELD	**43**	HI_CH8
14	LO_CH7	**29**	SNSLO_CH7	**44**	LO_CH8
15	HI_CH7	**30**	SNSHI_CH7	▨	▨

TABLE 2-1: EX1200-7008 CONNECTOR PIN SIGNAL ASSIGNMENT

TB Ref	Signal	Conn Pin	TB Ref	Signal	Conn Pin	TB Ref	Signal	Conn Pin
T1	LO_CH2	1	**T23**	NC		**T45**	TRIG_IN_OUT	40
T2	HI_CH2	2	**T24**	NC		**T46**	EXT_CLK_IN	39
T3	USR_SHIELD	3	**T25**	HI_CH4	7	**T47**	NC	
T4	NC		**T26**	LO_CH4	6	**T48**	NC	
T5	SNSLO_CH2	16	**T27**	USR_SHIELD	8	**T49**	HI_CH6	12
T6	SNSHI_CH2	17	**T28**	NC		**T50**	LO_CH6	11
T7	NC		**T29**	SNSHI_CH4	22	**T51**	USR_SHIELD	13
T8	NC		**T30**	SNSLO_CH4	21	**T52**	NC	
T9	LO_CH1	32	**T31**	NC		**T53**	SNSHI_CH6	27
T10	HI_CH1	31	**T32**	NC		**T54**	SNSLO_CH6	26
T11	USR_SHIELD	33	**T33**	TRIGGER_OUT	37	**T55**	USR_SHIELD	28
T12	NC		**T34**	GND_D	36	**T56**	NC	
T13	HI_CH3	5	**T35**	USR_SHIELD	38	**T57**	SNSLO_CH8	42
T14	LO_CH3	4	**T36**	NC		**T58**	SNSHI_CH8	41
T15	NC		**T37**	HI_CH5	10	**T59**	HI_CH8	43
T16	NC		**T38**	LO_CH5	9	**T60**	LO_CH8	44
T17	SNSHI_CH3	20	**T39**	NC		**T61**	SNSHI_CH7	30
T18	SNSLO_CH3	19	**T40**	NC		**T62**	SNSLO_CH7	29
T19	USR_SHIELD	18	**T41**	SNSHI_CH5	25	**T63**	NC	
T20	NC		**T42**	SNSLO_CH5	24	**T64**	NC	
T21	SNSLO_CH1	35	**T43**	USR_SHIELD	23	**T65**	HI_CH7	15
T22	SNSHI_CH1	34	**T44**	NC		**T66**	LO_CH7	14

The purpose of each signal is defined in **TABLE 2-2**.

Signal	Description
HI_CHx	Positive input current
LO_CHx	Negative input current
SNS_HI_CHx	Voltage sense output
SNS_LO_CHx	Voltage sense output reference
USR_SHIELD	The chassis ground connects to EX1200 mainframe. This pin is usually connected to cable shield.

TABLE 2-2: SIGNAL PIN DEFINITIONS

NOTE	Polarity of the input channel should be strictly followed. Negative excitation is not supported.

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, all of the input channels on the EX1200-7008 will be opened, with the resistance of > 10 kΩ. Once BPL_INSFAIL is asserted, card will not respond to any API commands. To reset the EX1200-7008, it is necessary to make a call to the Reset API. .

NOTE	Card will not generate BPL_INSFAIL signal and will only respond to BPL_INSFAIL signal.

CALIBRATION

Every EX1200-7008 is factory calibrated and traceable to NIST standards. Adjustments and verification procedure is given in the appendix for field use. Optionally, the EX1200-7008 can be returned to the factory for a complete factory calibration. Refer to *Section 6* for the manual calibration procedure. **VTI recommends annual factory calibration of the EX1200-7008.**

SECTION 3

THEORY OF OPERATION

INTRODUCTION

The EX1200-7008 contains internal, high-precision, solid-state based resistance simulation combined with DAC input. It is capable of producing any resistance value from 4 Ω to 10 kΩ (at ≤ 1 mA of source current) with a resolution of 25 mΩ programmable via API commands. Voltage is generated through a DAC based on the input channel's excitation current. Output voltage is continuously monitored and any voltage loss due to source impedance is compensated through servo control based on a closed loop mechanism.

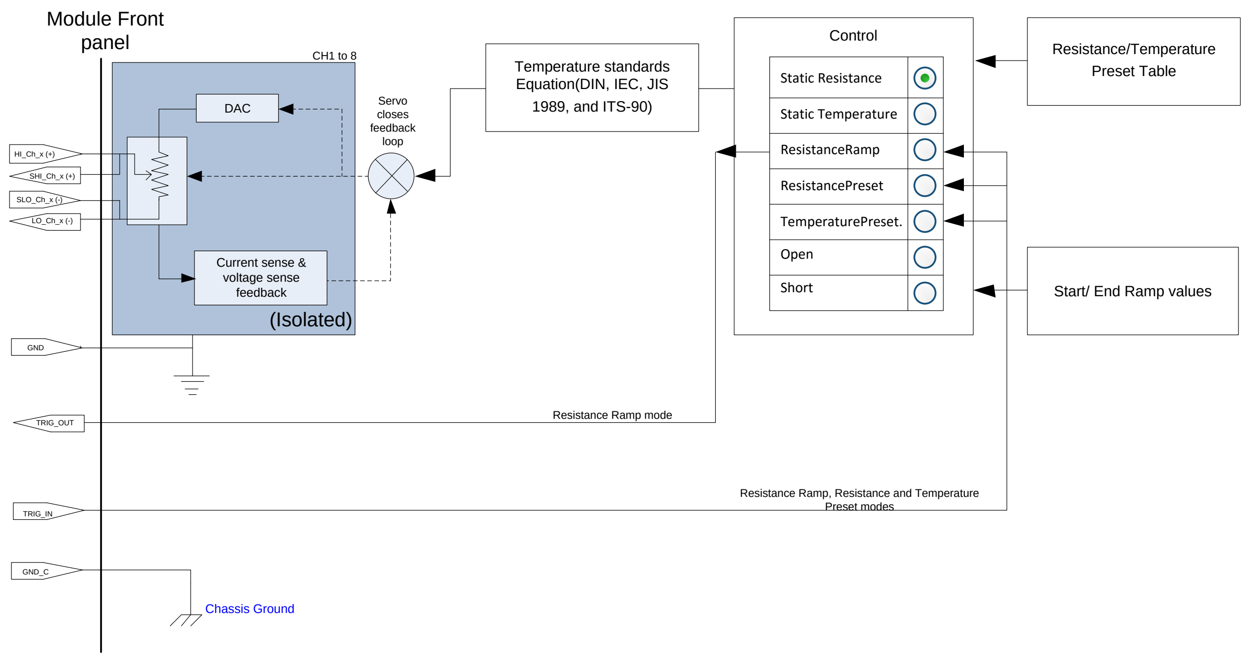


FIGURE 3-1: EX1200-7008 BLOCK DIAGRAM

Channels can be configured for open, short, static resistance, static temperature, resistance ramp, resistance preset, and temperature preset modes.

NOTE	All channels are configured to the same **Mode** property. This is not selectable on per channel basis.

MODES OF OPERATION

Resistance and Temperature Modes

In the resistance and temperature modes, resistance/temperature is updated within 1.5ms after receiving the source/input current. If an output value is not set before the channel is enabled, the default value of open circuit is set. Using the delay property, output value can be kept on hold for the user-specified period (in multiples of 10ms), even after the source/input current has been disabled. This enables the user to use a pulsed current source too. Trigger input and output are not supported in this mode. To simulate open or short, set the values +99999 and -99999, respectively.

Resistance Ramp Mode

In resistance ramp mode, resistance is ramped from a starting resistance value (`RTDSimulatorChannel.Output/VTEXRTDSIMULATOR_ATTR_OUTPUT`) to end value (`RTDSimulatorChannelRamp.EndResistance/VTEXRTDSIMULATOR_ATTR_END_RESISTANCE`) within the set ramp time. If the starting resistance is lower than the end resistance, it simulates positive ramp (while the converse simulates a negative ramp). Start and end resistance values cannot be open or short values. As soon as the EX1200-7008 is initiated, it outputs the starting resistance value and waits for a trigger input. Once a trigger is received, the EX1200-7008 begins the ramping operation, moving to the end resistance value, then returning to the starting resistance value. Once the cycle is complete, the EX1200-7008 waits for another trigger. The trigger input can come from software, any of the BPL lines, or the front panel triggers inputs. Only linear ramping is supported.

A programmable digital marker signal is supported to ensure sophisticated trigger synchronization and can be configured for one of the output channels. Markers are used to send a trigger from the trigger output channel once the set resistance level is reached.

The time need to move from one ramp-point to the next is determined with 10 milliseconds resolution. Ramp time is divided into 10 milliseconds resolution resulting in a smoother ramp curve as the ramp time is increased.

Resistance Preset and Temperature Mode

In the resistance preset and temperature modes, the EX1200-7008 outputs a predefined array of resistance or temperature points. Prior to initiation, channels are placed in the reset state (open). As soon as EX1200-7008 is initiated, all the channels starts from the first predefined point and then advanced to next point once an input trigger is asserted. The trigger comes from software, any of the BPL lines, or from front panel triggers. Trigger output is not supported in this mode. Each channel may contain a maximum of 10 preset points. Preset points may contain open and/or short simulation points. To simulate an open or short, set the value to +99999 and -99999, respectively.

Open and Short Simulation

All channels can be set to either an open or short by setting RTDSimulator mode to VTEXRTDSimulatorModeOpen or VTEXRTDSimulatorModeShort. It is also possible to set a channel to an open or short by setting the RTDSimulator Channel output to +99999, to simulate an open condition, or -99999, to simulate a short condition.

SECTION 4

PROGRAMMING THE INSTRUMENT

INTRODUCTION

This section provides programming examples for the EX1200-7008. 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.)

NOTE Programming examples are installed with the **IVI Driver – RTDSimulator**, typically located in the "<HDD Designation>**Program Files\IVI Foundation\IVI\Drivers\VTEXRTDSimulator\Examples**" folder.

RELATED SOFTWARE COMPONENTS

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

USING THE DRIVER

The EX1200-7008 may be used in a variety of environments including: Visual Basic, C#, C++, Labwindows/CVI, LabView. VTI Instruments provides 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 "VTEXRTDSimulator.dll" no_namespace
```
2) **C#**
Add a reference to VTEXRTDSimulator.dll COM object in the project. Include the following at the top of any code file that will access the driver:
```
using VTI.VTEXRTDSimulator.Interop;
```
3) **C/C++ on Windows**
Link against VTEXRTDSimulator.lib and include VTEXRTDSimulator.h in the file.
4) **C++ on Linux**
Link against /opt/vti/lib/librtdsimulator.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 from Instrument I/O function panel

USING THE EX1200-7008

INITIALIZING\CLOSING A DRIVER SESSION

The base interface of the EX1200-7008 IVI driver, VTEXRTDSimulator (LibRTDSimulator 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.

Prior to using the EX1200-7008, an instrument driver connection must be made to the EX1200-7008. Once a connection is made using the Initialize call, the user can execute their test code. Before the program exits, the user should release the resources using the Close call. Users familiar with other VTI Instrument Drivers for the EX1200 series should find this driver is very similar to ones they have used before.

C++

```
#import "IviDriverTypeLib.dll" no_namespace
#import " VTEXRTDSimulator.dll" no_namespace

int main() {
   ::CoInitialize(NULL); // start COM layer
   // try/catch so driver not found situations are properly handled
   try {
      IVTEXRTDSimulator Simulator(__uuidof(VTEXRTDSimulator));
      try { // second error layer for failed initialize
         /* using an empty options string here; see Option Strings below
      for more infomration;also using Reset bit to get a clean init */
         Simulator->Initialize("TCPIP::10.1.4.55::INSTR", VARIANT_TRUE,
   VARIANT_TRUE, "");

         // test code goes here

         Simulator->Close();
      } catch (_com_error &e) {
         ::MessageBox(NULL, e.Description(), e.ErrorMessage(),
   MB_ICONERROR);
      }
   } catch (...) {
      // handle errors here, depending on needs
   }
}
```

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. To simulate use `“simulate=true”`
- **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 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. For example, following string will write driver calls to rtd1og.txt file.
  `"DriverSetup=logfile= rtdlog.txt"`
- **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_7008)"` - slot and card model number
  `"DriverSetup= slots=(3=RTDSimulator)"` - slot and card model name
  `"Slots=(2,3)"` - Multiple slots

RESISTANCE OUTPUT

This example shows how to simulate a resistance value. Note that the resistance and delay should be set first before enabling a channel. This prevents enabling the channel with the default open circuit resistance. Like the mode property, all channels will be configured with the same delay time. It is not configurable on a per channel basis.

C++

```
//This example simulates the resistance Output on channel 1 and channel 2

#include <iostream>
#include <vector>
#include "libRTDSimulator.h"

using namespace std;

#define MAXCOUNT 10
int main(int argc, char* argv[])
{
  try
  {
    LibRTDSimulator* rtdSimulator = NULL;
    /*We want to do the Initialization in a try/catch block so that our test code
    doesn't run if we fail to create.*/
    rtdSimulator = LibRTDSimulator::Create();

    try
    {
      /*We chose to give the driver an empty options string. You may want to
      give your driver options - check the manual to see the available settings.
      Note that we also set the Reset bit so that we get a clean start to work
      from.*/
      rtdSimulator->Initialize("TCPIP::10.30.1.54::INSTR", true, true, "");

      //Configure mode to resistance
      rtdSimulator->Mode = VTEXRTDSimulatorModeResistance;

      /* Based on the type of excitation source, set the appropriate delay time.
         Note that the “Delay” sets the time delay(ΔT) for the output to go zero after
         the input current has gone to zero.
         ΔT = (Delay + 1) * 10 ms
         If Delay is set to 255 (max value) then the output is kept constant and
         continuous irrespective of the input
      */
      rtdSimulator->Delay = 1; // ΔT = 20ms

      //Set the resistance output on channel 1 and enable the channel.
      rtdSimulator->Channels->Item["CH1"]->Output = 2000;

      //Resistance input in Ohm
      rtdSimulator->Channels->Item["CH1"]->Enabled = true;
      //Set the resistance output on channel 2 and enable the channel.
      rtdSimulator->Channels->Item["CH2"]->Output = 3000;
      //Resistance input in Ohm
      rtdSimulator->Channels->Item["CH2"]->Enabled = true;


      cout<<"Simulated output on channel: 1 "<<rtdSimulator->Channels->Item["CH1"]-
>Output<<endl;
      cout<<"Simulated output on channel: 2 "<<rtdSimulator->Channels->Item["CH2"]-
>Output<<endl;

      cout<<"\nCurrent Measured on Channel's\n"<<endl;
      //Get the exitation Current on channel 1 and channel 2
      //User can enter the MAXCOUNT number to get the measured current
      for(int i=0; i< MAXCOUNT; i++)
      {
        //Measuring current on channel 1 and channel 2
        cout<<"Current on Channel: 1 is\t"<<rtdSimulator->Channels->Item["CH1"]-
>MeasuredCurrent<<endl;
        cout<<"Current on Channel: 2 is\t"<<rtdSimulator->Channels->Item["CH2"]-
>MeasuredCurrent<<endl;
        sleep(1);
      }

      //Close the initialized session
      rtdSimulator->Close();
    }
    catch (VTEXException e)
    {
      cout<<"Error code: \t"<<e.errorCode<<endl;
      cout<<"Error message: \n"<<e.errorMessage.c_str()<<endl;
    }
  }

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

  cout<<"\nDone - Press Enter to Exit"<<endl;
  getchar();

  return 0;
}
```

TEMPERATURE OUTPUT

This example covers selecting a static temperature point for simulation. This is very similar to the resistance output example except it outputs the corresponding resistance per the selected RTD type and output value.

C++

```
//This example simulates the temperature output on channel 1

#include "stdafx.h"

#import "IviDriverTypeLib.dll" no_namespace
#import "VTEXRTDSimulator.dll" no_namespace


int _tmain(int argc, _TCHAR* argv[])
```

```
{
  /*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.*/
  ::CoInitialize(NULL);//Start the COM layer

  try
  {
    /*We want to do the Initialization in a try/catch block so that our test code
    doesn't run if we fail to initialize.*/
    IVTEXRTDSimulatorPtr rtdSimulator(__uuidof(VTEXRTDSimulator));

    try
    {
      /*We chose to give the driver an empty options string. You may want to
      give your driver options - check the manual to see the available settings.
      Note that we also set the Reset bit so that we get a clean start to work
      from.*/
      rtdSimulator->Initialize("TCPIP::10.30.1.54::INSTR", VARIANT_TRUE, VARIANT_TRUE, 
"");

      //Configure mode to temperature
      rtdSimulator->Mode = VTEXRTDSimulatorModeTemperature;

      /* Based on the type of excitation source, set the appropriate delay time.
         Note that the “Delay” sets the time delay(ΔT) for the output to go zero after
         the input current has gone to zero.
         ΔT = (Delay + 1) * 10 ms
         If Delay is set to 255 (max value) then the output is kept constant and 
         continuous irrespective of the input
      */
      rtdSimulator->Delay = 254; // ΔT = 2.55 s

      //Set one of the standard RTD type, should be Platinum or Copper
      //EUConversion => engineering convesion unit.
      //User can set one of the supported conversion units 
VTEXRTDSimulatorEUConversionDegreeC, \
      //
VTEXRTDSimulatorEUConversionDegreeK, \
      //
VTEXRTDSimulatorEUConversionDegreeF
      rtdSimulator->Channels->Item["CH1"]->RTDType = VTEXRTDSimulatorRTDTypePT100_385;
      rtdSimulator->Channels->Item["CH1"]->EUConversion = 
VTEXRTDSimulatorEUConversionDegreeC;

      //Set the temperature output on channel 1.
      rtdSimulator->Channels->Item["CH1"]->Output = 100;

      //Enable channel 1.
      rtdSimulator->Channels->Item["CH1"]->Enabled = VARIANT_TRUE;
      cout<<"Simulated Output on Channel: 1 "<<rtdSimulator->Channels->Item["CH1"]-
>Output<<endl;

      cout<<"\nCurrent Measured on Channel: 1\n"<<endl;
      cout<<"Hit any key to stop\n"<<endl;
      //Get the exitation Current on channel 1
      while(!_kbhit())
      {
        //Measuring current on channel 1
        cout<<"Current on Channel: 1 is\t"<<rtdSimulator->Channels->Item["CH1"]-
>MeasuredCurrent<<endl;
        Sleep(1000);
      }
      //Close the initialized session
      rtdSimulator->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
  }

  ::CoUninitialize();

  cout<<"\nDone - Press Enter to Exit"<<endl;
  getchar();

  return 0;
}
```

TEMPERATURE PRESET

The EX1200-7008 has a preset mode that supports up to 10 preset points per channel. In this mode, the instrument outputs the set temperature points in succession.

C++

```
//This example simulates the preset temperature points on channel 1

#include "stdafx.h"

#import "IviDriverTypeLib.dll" no_namespace
#import "VTEXRTDSimulator.dll" no_namespace

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


  try
  {
    /*We want to do the Initialization in a try/catch block so that our test code
    doesn't run if we fail to initialize.*/
    IVTEXRTDSimulatorPtr rtdSimulator(__uuidof(VTEXRTDSimulator));

    try
    {
      /*We chose to give the driver an empty options string. You may want to
      give your driver options - check the manual to see the available settings.
      Note that we also set the Reset bit so that we get a clean start to work
      from.*/
      rtdSimulator->Initialize("TCPIP::10.30.1.54::INSTR", VARIANT_TRUE, VARIANT_TRUE,
"");

      //Set mode to temperature preset
      rtdSimulator->Mode = VTEXRTDSimulatorModeTemperaturePreset;

      //Set one of the standard RTD type, should be Platinum or Copper
      //EUConversion => engineering convesion unit.
      //User can set one of the supported conversion units
VTEXRTDSimulatorEUConversionDegreeC, \
      //
VTEXRTDSimulatorEUConversionDegreeK, \
      //
VTEXRTDSimulatorEUConversionDegreeF
      rtdSimulator->Channels->Item["CH1"]->RTDType = VTEXRTDSimulatorRTDTypePT100_385;
      rtdSimulator->Channels->Item["CH1"]->EUConversion =
VTEXRTDSimulatorEUConversionDegreeC;

      //Set Preset mode to cyclic
      //Cyclic Mode => keeps cycling through the set preset temperature points
      //End Mode    => Keeps sending the last preset temperature point as trigger
continues
      rtdSimulator->Channels->Item["CH1"]->Preset->Mode =
VTEXRTDSimulatorPresetModeCyclic;

      //Set the TriggerIntput property to software
      //User needs to send the software trigger to update the next value
      rtdSimulator->Trigger->TriggerInput = VTEXRTDSimulatorTriggerInputSoftware;

      //User should specify the maximum values that needs to be created.
      //Valid preset points should contain at least 2 and goes up to maximum of 10.
      int maxsize = 3;
      long i = 0;
      double preset1 = 50.2;
      double preset2 = 120.0;
      double preset3 = 200.8;
      SAFEARRAY* presetPoints = NULL;
      //Here we are creating a maximum size of preset points.
        presetPoints = ::SafeArrayCreateVector(VT_R8, 0, maxsize);

      ::SafeArrayPutElement(presetPoints, &i, (void*)&preset1); i++;
      ::SafeArrayPutElement(presetPoints, &i, (void*)&preset2); i++;
      ::SafeArrayPutElement(presetPoints, &i, (void*)&preset3);

      //Sets the preset temperature points on channel 1
      rtdSimulator->Channels->Item["CH1"]->Preset->PutPresetPoints(&presetPoints);

      //Enable the channel 1
      rtdSimulator->Channels->Item["CH1"]->Enabled = VARIANT_TRUE;


      //Initiate the card
      rtdSimulator->Initiate();

      int count  = 10; //Specifies the number for temperature output.
      for(int i = 0;  i < count;  i++)
      {
        //SendSoftwareTrigger, keeps sending the temperature data on the Front panel
for the specified count
        //Minimum update rate is 10 milli seconds.
        //User has to give at the time interval which should be > 10 milli seconds
between two triggers
        rtdSimulator->Trigger->SendSoftwareTrigger();
        Sleep(100);
      }

      cout<<"\nCurrent Measured on Channel: 1\n"<<endl;
      cout<<"Hit any key to stop\n"<<endl;
      while(!_kbhit())
      {
        //Measuring current on channel 1
        cout<<"Current on Channel: 1 is\t"<<rtdSimulator->Channels->Item["CH1"]-
>MeasuredCurrent<<endl;
        Sleep(1000);
      }

      //Aborts the initilized card
      rtdSimulator->Abort();
      //Close the initialized session
      rtdSimulator->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
  }

  ::CoUninitialize();

  cout<<"\nDone - Press Enter to Exit"<<endl;
  getchar();

  return 0;
}
```

RESISTANCE PRESET

The EX1200-7008 has preset mode which supports up to 10 points preset per channel. In this mode, the instrument outputs the set resistance points one after another.

C++

```
//This example simulates the preset resistance points on channel 1

#include "stdafx.h"

#import "IviDriverTypeLib.dll" no_namespace
#import "VTEXRTDSimulator.dll" no_namespace

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

  try
  {

    /*We want to do the Initialization in a try/catch block so that our test code
    doesn't run if we fail to initialize.*/
    IVTEXRTDSimulatorPtr rtdSimulator(__uuidof(VTEXRTDSimulator));

    try
    {

      /*We chose to give the driver an empty options string. You may want to
      give your driver options - check the manual to see the available settings.
      Note that we also set the Reset bit so that we get a clean start to work
      from.*/
      rtdSimulator->Initialize("TCPIP::10.30.1.54::INSTR", VARIANT_TRUE, VARIANT_TRUE,
"");

      //Set mode to resistance preset and Preset mode to cyclic
      //Cyclic Mode => keeps cycling through the set preset resistance points
      //End Mode    => Keeps sending the last preset resistance points as trigger
continues
      rtdSimulator->Mode = VTEXRTDSimulatorModeResistancePreset;
      rtdSimulator->Channels->Item["CH1"]->Preset->Mode =
VTEXRTDSimulatorPresetModeCyclic;

      //Set the TriggerIntput property to software
      //User needs to send the software trigger to update the next value
      rtdSimulator->Trigger->TriggerInput = VTEXRTDSimulatorTriggerInputSoftware;

      //User should specify the maximum values that needs to be created.
      //Valid preset points should contain at least 2 and goes up to maximum of 10.
      int maxsize = 3;
      long i = 0;
      double preset1 = 500.00;
      double preset2 = 2000.0;
      double preset3 = 4000.2;
      SAFEARRAY* presetPoints = NULL;
      //Here we are creating a maximum size of preset points.
        presetPoints = ::SafeArrayCreateVector(VT_R8, 0, maxsize);

      ::SafeArrayPutElement(presetPoints, &i, (void*)&preset1); i++;
      ::SafeArrayPutElement(presetPoints, &i, (void*)&preset2); i++;
      ::SafeArrayPutElement(presetPoints, &i, (void*)&preset3);

      //Sets the preset resistance points on channel 1
      rtdSimulator->Channels->Item["CH1"]->Preset->PutPresetPoints(&presetPoints);

      //Enable the channel 1
      rtdSimulator->Channels->Item["CH1"]->Enabled = VARIANT_TRUE;


      //Initiate the card
      rtdSimulator->Initiate();

      int count = 10; //Specifies the number for resistance output.
      for(int i = 0;  i < count;  i++)
      {
        //SendSoftwareTrigger, keeps sending the resistance data on the Front panel
for the specified count
        //Minimum update rate is 10 milli seconds.
        //User has to give at the time interval which should be > 10 milli seconds
between two triggers
        rtdSimulator->Trigger->SendSoftwareTrigger();
        Sleep(100);
      }

      cout<<"\nCurrent Measured on Channel: 1\n"<<endl;
      cout<<"Hit any key to stop\n"<<endl;
      //Get the exitation Current on channel 1
      while(!_kbhit())
      {
        //Measuring current on channel 1
        cout<<"Current on Channel: 1 is\t"<<rtdSimulator->Channels->Item["CH1"]-
>MeasuredCurrent<<endl;
        Sleep(1000);
      }

      //Aborts the initilized card
      rtdSimulator->Abort();
      //Close the initialized session
      rtdSimulator->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
  }

  ::CoUninitialize();

  cout<<"\nDone - Press Enter to Exit"<<endl;
  getchar();

  return 0;
}
```

RESISTANCE RAMP

This example shows how a resistance ramp is created from a set start and end point over a defined amount of time. (Temperature ramp mode is not supported by EX1200-7008.)

C++

```
//This example simulates the resistance ramp on channel 1

#include "stdafx.h"

#import "IviDriverTypeLib.dll" no_namespace
#import "VTEXRTDSimulator.dll" no_namespace

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

  try
  {
    /*We want to do the Initialization in a try/catch block so that our test code
    doesn't run if we fail to initialize.*/
    IVTEXRTDSimulatorPtr rtdSimulator(__uuidof(VTEXRTDSimulator));

    try
    {
      /*We chose to give the driver an empty options string. You may want to
      give your driver options - check the manual to see the available settings.
      Note that we also set the Reset bit so that we get a clean start to work
      from.*/
      rtdSimulator->Initialize("TCPIP::10.30.1.54::INSTR", VARIANT_TRUE, VARIANT_TRUE,
"");
```

```
      //Set mode to resistance Ramp
      rtdSimulator->Mode = VTEXRTDSimulatorModeResistanceRamp;

      //For ramp function we need to set the start and end resistance
      //Property Output is used to set the start resistance and property EndResistance
is for setting the end resistance.
      rtdSimulator->Channels->Item["CH1"]->Output = 200.0;
      rtdSimulator->Channels->Item["CH1"]->Ramp->EndResistance = 2000.0;

      //Sets the time to ramp from start to end resistance
      rtdSimulator->Channels->Item["CH1"]->Ramp->RampTime = 10;

      //User can set this property for trigger output
      //The card starts ramping from start resistamce towards end. Upon reaching the
trigger level,
      //marker output trigger pulse will be sent out of the card. User needs to make
sure that the trigger level will be set between start and end resistance.
      // ( Start resistance < Triggerlevel  < End resistance )
      rtdSimulator->Channels->Item["CH1"]->Ramp->TriggerOutputLevel = 1000;

      //Set the TriggerIntput property to software and TriggerOutput property to BPL0
      //User needs to send the software trigger to get the resistance output on front
panel
      //Set the output reference channel to identify channel referred for trigger
output
      rtdSimulator->Trigger->TriggerInput = VTEXRTDSimulatorTriggerInputSoftware;
      rtdSimulator->Trigger->TriggerOutput = VTEXRTDSimulatorTriggerOutputFrontpanel;
      rtdSimulator->Trigger->TriggerOutputReferenceChannel = "CH1";

      //Enable the channel 1
      rtdSimulator->Channels->Item["CH1"]->Enabled = VARIANT_TRUE;


      //Initiate the card
      rtdSimulator->Initiate();

      //SendSoftwareTrigger should be send to start ramp
      rtdSimulator->Trigger->SendSoftwareTrigger();

      cout<<"\nCurrent and Resistance Measured on Channel: 1\n"<<endl;
      cout<<"Hit any key to stop\n"<<endl;
      //Get the exitation Current on channel 1
      while(!_kbhit())
      {
        //Measuring current on channel 1
        //Get the resistance output at that instance of call on Channel 1
        cout<<"Current on Channel: 1  is\t"<<rtdSimulator->Channels->Item["CH1"]-
>MeasuredCurrent<<endl;
        cout<<"Resistance on Channel: 1 is\t"<<rtdSimulator->Channels->Item["CH1"]-
>MeasuredResistance<<endl;
        Sleep(1000);
      }

      //Aborts the initilized card
      rtdSimulator->Abort();
      //Close the initialized session
      rtdSimulator->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
  }

  ::CoUninitialize();

  cout<<"\nDone - Press Enter to Exit"<<endl;
  getchar();

  return 0;
}
```

DRIVER INTERFACES

The following is a list of programmatic interfaces to the EX1200-7008 with a description of the functionality they provide. Please refer to the help file installed with the driver for a programming reference that includes all methods with their parameters as well as all enumerations.

- **IVTEXRTDSimulator**: Used to open and close connections to the instrument. Also contains Failsafe information that is per-card, such as Temperature.
- **Channels**: Used to configure or query individual channel state such as resistance value, temperature value, Preset and Ramp interface etc.
- **IIviDriver ->DriverOperation**: IVI standard. Provides control over the manner in which the driver operates.
- **IIviDriver ->Identity**: IVI standard. Provides detailed version information about the driver, connected hardware, and firmware versions.
- **IIviDriver ->Platform**: Used to Log driver function calls and to retrieve the serial number of the instrument.
- **IVTEXRTDSimulator ->Trigger**: Used to configure trigger condition and select a backplane trigger line to be asserted.
- **IIviDriver ->Utility**: IVI standard. Provides useful functionality not specific to this driver, such as Reset.

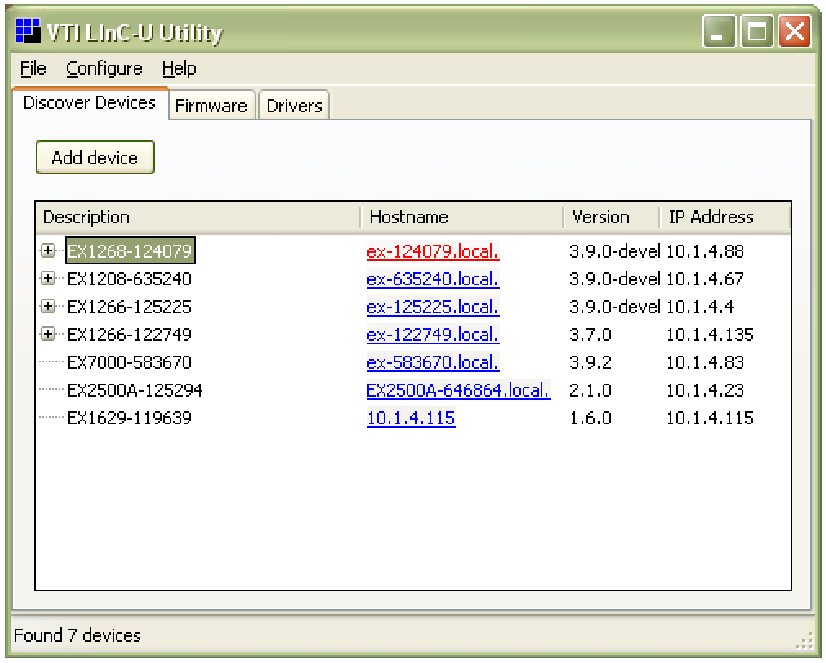
SECTION 5

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.



Description	Hostname	Version	IP Address
EX1268-124079	ex-124079.local.	3.9.0-devel	10.1.4.88
EX1208-635240	ex-635240.local.	3.9.0-devel	10.1.4.67
EX1266-125225	ex-125225.local.	3.9.0-devel	10.1.4.4
EX1266-122749	ex-122749.local.	3.7.0	10.1.4.135
EX7000-583670	ex-583670.local.	3.9.2	10.1.4.83
EX2500A-125294	EX2500A-646864.local.	2.1.0	10.1.4.23
EX1629-119639	10.1.4.115	1.6.0	10.1.4.115

Found 7 devices

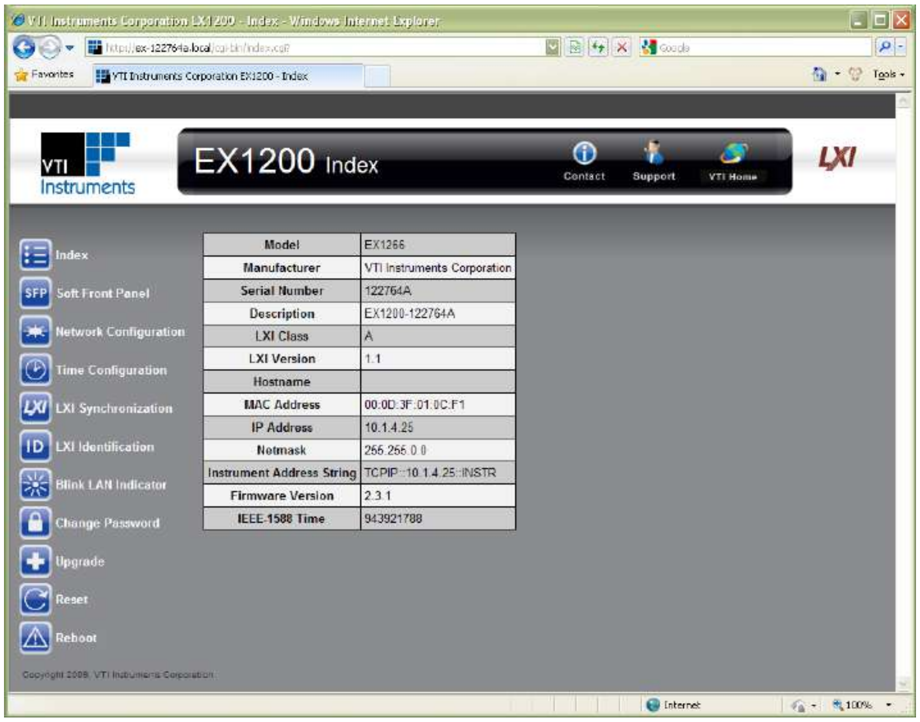
FIGURE 5-1: LINC-U DISCOVERY TAB WITH AN EX1268 SELECTED

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 Figure 5-2). This page displays instrument-specific information including

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



Model	EX1266
Manufacturer	VTI Instruments Corporation
Serial Number	122764A
Description	EX1200-122764A
LXI Class	A
LXI Version	1.1
Hostname	
MAC Address	00:0D:3F:01:0C:F1
IP Address	10.1.4.25
Netmask	255.255.0.0
Instrument Address String	TCPIP::10.1.4.25::INSTR
Firmware Version	2.3.1
IEEE-1588 Time	943921788

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.

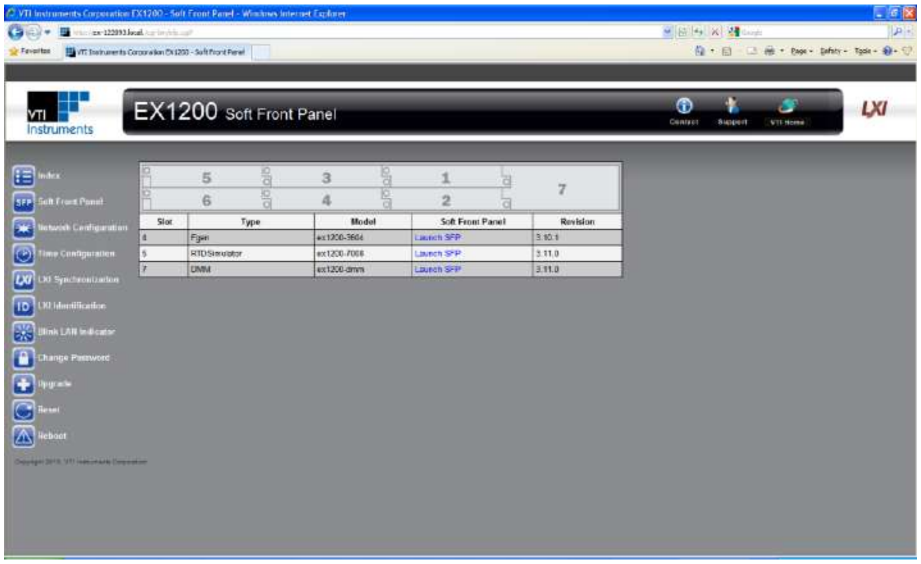
VTI Instruments Logo

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 EX1200-7008 soft front panel. For more information on other EX1200 soft front panel elements, please refer to the *EX1200 Series User's Manual*.

EX1200-7008 SOFT FRONT PANEL

To navigate to the EX1200-7008 soft front panel, click on **Soft Front Panel** in the **Command Menu** (see Figure 5-3). Next, select **RTDSimulator, ex1200-7008** from the list of instruments installed in the EX1200.

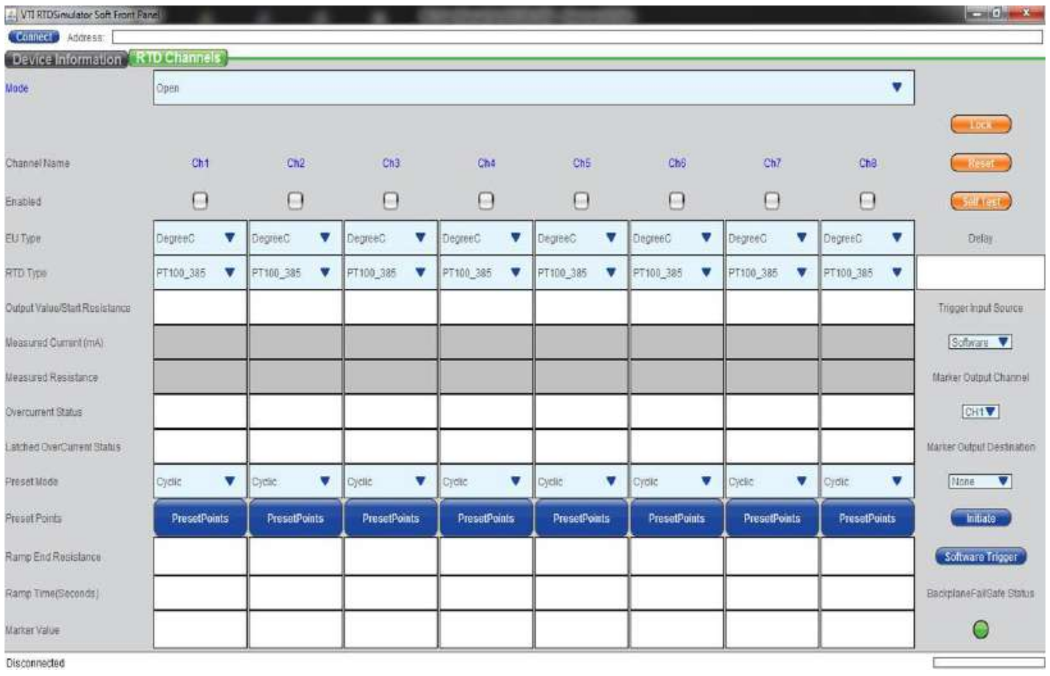


Slot	Type	Model	Soft Front Panel	Revision
4	Fgen	ex1200-3604	Launch SFP	3.10.1
5	RTDSimulator	ex1200-7008	Launch SFP	3.11.0
7	DMM	ex1200-dmm	Launch SFP	3.11.0

FIGURE 5-3: EX1200 SOFT FRONT PANEL MAIN PAGE

RTD CHANNELS PAGE

By default, the EX1200-7008 SFP opens to the **RTD Channels** view. From this view, the user can define a channel's output mode, RTD type, range, etc. Although the SFP does not expose the entire functionality of the EX1200-7008, the SFP can be used to set up the instrument in **most** applications.



Mode: Open

Channel Name	Ch1	Ch2	Ch3	Ch4	Ch5	Ch6	Ch7	Ch8
Enabled	☐	☐	☐	☐	☐	☐	☐	☐
EU Type	DegreeC	DegreeC	DegreeC	DegreeC	DegreeC	DegreeC	DegreeC	DegreeC
RTD Type	PT100_385	PT100_385	PT100_385	PT100_385	PT100_385	PT100_385	PT100_385	PT100_385
Output Value/Start Resistance								
Measured Current (mA)								
Measured Resistance								
Overcurrent Status								
Latched OverCurrent Status								
Preset Mode	Cyclic	Cyclic	Cyclic	Cyclic	Cyclic	Cyclic	Cyclic	Cyclic
Preset Points	PresetPoints	PresetPoints	PresetPoints	PresetPoints	PresetPoints	PresetPoints	PresetPoints	PresetPoints
Ramp End Resistance								
Ramp Time(Seconds)								
Marker Value								

Delay:
Trigger Input Source: Software
Marker Output Channel: CH1
Marker Output Destination: None

FIGURE 5-4: EX1200-7008 SOFT FRONT PANEL

Channel Settings

The *channel settings* area is used to configure EX1200-7008 channel outputs. In order for a channel to output, its **Enabled** checkbox must be selected (see Figure 5-4). The user can also enter the "Delay" time, which sets the time delay(ΔT) for the output to go zero after the input current has gone to zero. This is useful when pulsed current sources are used.

$$\Delta T = (\text{Delay} + 1) * 10 \text{ ms}$$

Delay can vary from 0 (default value) to 255 (max value). When set to 255, the output is kept constant and continuous irrespective of the input signal.

The allowed settings are dependent on the **Mode** selected. Each **Mode** is addressed separately below. **Measured resistance** and **measured current** work independently of the selected **Mode** and measure excitation current and feedback resistance when a channel is enabled. If a setting is not mentioned in a **Mode** description below, that setting is disabled and it does not affect the mode being described.

Open Mode

- **Channel Name**: Indicates the channel that is defined by the current column. This setting is not user-configurable and provided only as an indicator.

Short Mode

- **Channel Name**: Indicates the channel that is defined by the current column. This setting is not user-configurable and provided only as an indicator.

- **Enable** checkbox: When selected, indicates that the channel is enabled for short simulation. If a channel is not enabled, it will be in default (open) state.

Resistance Mode

- **Channel Name**: Indicates the channel that is defined by the current column. This setting is not user-configurable and provided only as an indicator.
- **Enable** checkbox: When selected, indicates that the channel is enabled for simulation. Output resistance will be set as soon as the channel is enabled.
- **Output/Start Resistance:** Defines the output resistance for the channel.

Temperature Mode

- **Channel Name**: Indicates the channel that is defined by the current column. This setting is not user-configurable and provided only as an indicator.
- **Enable** checkbox: When selected, indicates that the channel is enabled for simulation. Output temperature will be set as soon as the channel is enabled.
- **EU Type:** Sets the unit of measurement as either **DegreeC**, **DegreeF,** or **Kelvin**.
- **RTD Type:** Defines the RTD type that will be simulated on the given channel.
- **Output/Start Resistance:** Defines the output temperature for the channel as per the EU type and RTD settings.

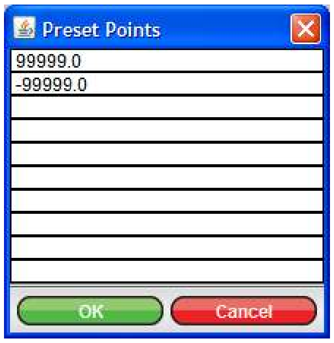
Resistance Ramp Mode

- **Channel Name**: Indicates the channel that is defined by the current column. This setting is not user-configurable and provided only as an indicator.
- **Enable** checkbox: When selected, indicates that the channel is enabled for simulation.
- **Output/Start Resistance:** Defines the starting resistance value for the ramping function. Once ramping completes, the channel is also reset this value.
- **Ramp End Resistance:** Defines the resistance value where ramping will end.
- **Ramp Time:** Time (in seconds) to complete ramping operation from **Start** to **End Resistance**.
- **Marker Value:** Defines a resistance value that, once achieved during the ramp operation, causes a trigger to be sent from the trigger output channel.
- **Trigger Input Source:** Sets the trigger source for the channel. Once initiated, the instrument will wait for a trigger from this source before beginning its ramp operation.
- **Marker Output Channel:** Defines the channel where the trigger output will be generated when this channel reaches the defined **Marker Value**. Any channel can be configured as the marker output source channel.
- **Marker Output Source:** Defines the backplane trigger line where the EX1200-7008 will drive a marker output.
- **Initiate** button**:** Once this clicked, arming begins on all channels. Hereafter, outputs are generated once a trigger is received.
- **Software Trigger** button**:** Once clicked, a software trigger event is sent. This triggers all channels if their **Trigger Input Source** = **Software**.

Resistance Preset Mode

- **Channel Name**: Indicates the channel that is defined by the current column. This setting is not user-configurable and provided only as an indicator.
- **Enable** checkbox: When selected, indicates that the channel is enabled for simulation.
- **Trigger Input Source:** Sets the reference trigger source. Once initiated, the first preset point is set and then the instruments waits for an input trigger from this source before proceeding to the next preset point.
- **Preset Mode:** Cyclic selection repeats the point when one cycle is completed. End selection will stop with last preset point when cycle completes.

- **Preset Points:** Allows loading preset resistance array points. 99999 and -99999 values will simulate open and short circuit conditions, respectively.



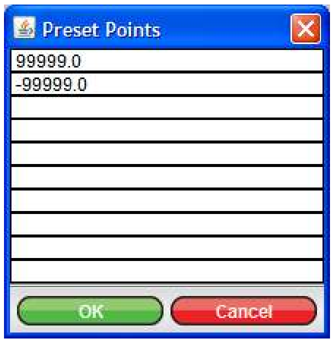
99999.0
-99999.0

FIGURE 5-5: PRESET POINT DATA TABLE

- **Initiate:** Arming begins, on all channels when this button is clicked. Hereafter, Output starts as soon as trigger is received
- **Software Trigger** button**:** Once clicked, a software trigger event is sent. This triggers all channels if their **Trigger Input Source** = **Software**.

Temperature Preset Mode

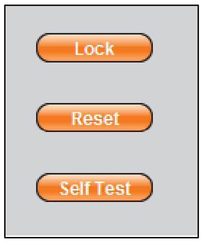
- **Channel Name**: Indicates the channel that is defined by the current column. This setting is not user-configurable and provided only as an indicator.
- **Enable** checkbox: When selected, indicates that the channel is enabled for simulation.
- **EU Type:** To indicate whether set value is in Celsius, Fahrenheit or Kelvin.
- **RTD Type:** Defines RTD type required for simulation.
- **Trigger Input Source:** Reference trigger source. Once initiated, sets first preset point and waits for input trigger for each preset point from configured (this selection) source.
- **Preset Mode:** cyclic selection repeats the point when one cycle is completed. End selection will stop with last preset point when cycle completes.
- **Preset Points:** Allows loading preset temperature array points. 99999 and -99999 values will simulate open and short circuit conditions respectively. Resistance will be calculated as per EU type and RTD selections (see Figure 5-5).



99999.0
-99999.0

- **Initiate:** Arming begins, on all channels when this button is clicked. Hereafter, Output starts as soon as trigger is received.
- **Software Trigger:** Sends a software trigger event. when this button is clicked , causes channels to be triggered if *Trigger input source* is software.

System Buttons



The three orange buttons at the top right of the **Status Display** are system buttons that affect the functionality of the EX1200-7008.

Lock/Unlock

The **Lock** button requests (or releases) exclusive access to the EX1200-7008. SFP can be used to monitor channel status while application software uses the card for debugging purpose.

Reset

Clicking the **Reset** button returns the EX1200-7008 to its power-on default settings and values.

Self Test

The **Self Test** button executes the EX1200-7008's **SelfTest** driver function. The self-test checks for the existence of configuration and calibration data.

SECTION 6

CALIBRATION PROCEDURE

Resources Required:

Fluke 5700/20/30 calibrator (or equivalent)

Keithley 2002/Agilent 3458A (or equivalent)

Note: Make sure the EX1200-7008 and other instruments are powered ON and kept it for enough warmup time (Typical 1 hr).

1. Initialization

1.1 Initialize the ex1200-7008 RTD simulator, calibrator, and the DMM.

2. Current Input Channel Calibration

The purpose of this test is to calculate and store the gain and offset of the current measurement path, for the all eight channels.

Required equipment: External current source with a 100 µA to 10.5 mA
Recommended equipment: Fluke 5700/20/30 calibrator (or equivalent)

2.1 Enable the calibration mode

```
Session.Calibration.CalMode = true;
```

2.2 Configure the calibrator to DC current output mode.

2.3 Connect the current calibrator HI_CH1, referenced to LO_CH1.

2.4 Set the Raw DAC and DigiPOT Values.

```
Session.Calibration.Card.set_RawDac("CH1", 0xFFAF);
Session.Calibration.Card.set_RawDigipot("CH1", 0x45);
```

2.5 Enable channel 1.

```
Session.Channels.get_Item("CH1").Enabled = true;
```

2.6 Turn ON the calibrator Output.

2.7 Set the Input Current gain to *X1*.

```
Session.Calibration.Card.set_CurrentGain("CH1",
  VTEXRTDSimulatorCalibrationGainEnum.Gainx1);
```

2.8 Set the current calibrator to 10.5mA (*A1*).

2.9 Wait for 3 seconds or wait for calibrator output to become stable.

2.10 Average 1000 samples of Raw ADC and record (as *C1*)

```
C1 = Session.Calibration.Card.get_RawCurrent("CH1");
```

2.11 Set the current calibrator to 100 µA (*A2*).

2.12 Wait for 3 seconds or wait for calibrator output to become stable

2.13 Average 1000 samples of Raw ADC and record (as *C2*)

```
C2 = Session.Calibration.Card.get_RawCurrent("CH1");
```

2.14 Follow the below calculations to get the Current ADC Gain and offset.

2.15 Disable the Channel 1.

```
Session.Channels.get_Item("CH1").Enabled = false;
```

2.16 Check the calculated values are within the limits specified.

2.17 Repeat the above steps for remaining channels and store the values accordingly.

2.18 The following are the test inputs:

Set Gain	Set Input Current (Amps)	Measured ADC Current (Counts)
X1	**0.0105 (A1)**	*C1*
	0.0001 (A2)	*C2*

2.19 Calculate $GainX1 = 1000 \times (2^{24}) \times \frac{(C2-C1)}{(A2-A1)}$ and $OffsetX1 = \left(\frac{1000 \times (2^{24}) \times C1}{GainX1}\right) - A1$

```
Session.Calibration.Write("CHx",VTEXRTDSimulatorCalibrationConstantEnum.CurrentADCGain
  1x, GainX1);
Session.Calibration.Write("CHx",VTEXRTDSimulatorCalibrationConstantEnum.CurrentADCOffs
  et1x, OffsetX1);
```
where `CHx -> CH1 to CH8.`

2.20 Store the gain and offset in the calibration file in any locations from 0 to 6.

```
Session.Calibration.Save(0, 0x12007008);
```

2.21 Disable the calibration mode.

```
Session.Calibration.CalMode = false;
```

3. Voltage Input Channel Calibration.

The purpose of this test is to calculate and store the gain and offset of the voltage measurement path, for the all eight channels.

Required equipment: Voltage calibrator
Recommended equipment: Fluke 5700/20/30 calibrator (or equivalent)

3.1 Enable the calibration mode.

```
Session.Calibration.CalMode = true;
```

3.2 Configure the calibrator to DC Voltage 0.0V output mode and Turn ON the output.

3.3 Connect the Voltage calibrator to SENSHI_CH1 with the return to SENSLO_CH1.

3.4 Set the Raw DAC to 0 and the digital trimpot to 0.

```
Session.Calibration.Card.set_RawDac("CH1", 0xFFFF);
Session.Calibration.Card.set_RawDigipot("CH1", 0xFF);
```

3.5 Enable the Channel 1.

```
Session.Channels.get_Item("CH1").Enabled = true;
```

3.6 Set the input voltage gain for *X1*.

```
Session.Calibration.Card.set_VoltageGain("CH1",
  VTEXRTDSimulatorCalibrationGainEnum.Gainx1);
```

3.7 Set the Voltage calibrator to 10V (*V1*).

3.8 Wait for 3 seconds or wait for calibrator output to become stable.

3.9 Average 1000 samples of Raw ADC and record (as *C1*)

```
C1 = Session.Calibration.Card.get_RawVoltage("CH1");
```

3.10 Set the Voltage calibrator to 0.9V (*V2*).

3.11 Wait for 3 seconds or wait for calibrator output to become stable.

3.12 Average 1000 samples of Raw ADC and record (as *C2*)

```
C2 = Session.Calibration.Card.get_RawVoltage("CH1");
```

3.13 Follow the below calculations to get the Voltage ADC Gain and offset.

3.14 Disable the Channel 1.

```
Session.Channels.get_Item("CH1").Enabled = false;
```

3.15 Check the calculated values are within the limits specified.

3.16 Repeat the above steps for *GainX10* and the Calibrator Voltage as follows.

```
Session.Calibration.Card.set_VoltageGain("CH1",
  VTEXRTDSimulatorCalibrationGainEnum.Gainx10);
```

3.17 Repeat this test for remaining channels and store the values.

3.18 The following are the test inputs:

Set Gain	Set Calibrator Voltage (Volts)	Measured ADC Voltage (Counts)
X1	**10.0 (V1)**	*C1*
	0.9 (V2)	*C2*
X10	**1.0 (V3)**	*C3*
	0.01 (V4)	*C4*

3.19 Calculate *GainX10* = $2^{24} \times \frac{(C2-C1)}{(V2-V1)}$ and *OffsetX10* = $(\frac{2^{24} \times C1}{GainX10}) - V1$

```
Session.Calibration.Write("CH1",VTEXRTDSimulatorCalibrationConstantEnum.VoltageADCGain
  1x, GainX1);
Session.Calibration.Write("CH1",VTEXRTDSimulatorCalibrationConstantEnum.VoltageADCOffs
  et1x, OffsetX1);
```

3.20 Calculate *GainX1* = $2^{24} \times \frac{(C4-C3)}{(V4-V3)}$ and *OffsetX1* = $(\frac{2^{24} \times C3}{GainX10}) - V3$

```
Session.Calibration.Write("CH1",VTEXRTDSimulatorCalibrationConstantEnum.VoltageADCGain
  1x, GainX1);
Session.Calibration.Write("CH1",VTEXRTDSimulatorCalibrationConstantEnum.VoltageADCOffs
  et1x, OffsetX1);
```

3.21 Write calibration date to the today (Calculate the total seconds from 01/01/1970 to today and write).

```
Session.Calibration.Write("CH1",VTEXRTDSimulatorCalibrationConstantEnum.Caldate,
  Today);
```

3.22 Similarly write calibration due date to today + 1 year.

```
Session.Calibration.Write("CH1",VTEXRTDSimulatorCalibrationConstantEnum.Caldue,  Today
  + 1year);
```

3.23 Store the Gain and offset in the calibration file 0 to 6.

```
Session.Calibration.Save(0, 0x12007008);
```

3.24 Disable the calibration mode.

```
Session.Calibration.CalMode = false;
```

CALIBRATION VERIFICATION

1. Current ADC Verification

The purpose of this test is to verify the current measurement accuracy, after the calibration.
Hardware Requirement: External current source, 100 µA to 10.5 mA.
Recommended equipment: Fluke 5700/20/30 calibrator (or equivalent)

1.1 Load the calibration file from 0 to 6 in which the cal constants stored during calibration.

```
Session.Calibration.Load(0);
```

1.2 Disable the calibration mode.

```
Session.Calibration.CalMode = false;
```

1.3 Set the mode to Resistance output.

```
Session.Mode = VTEXRTDSimulatorModeEnum.VTEXRTDSimulatorModeResistance;
```

1.4 Enable the Channel 1.

```
Session.Channels.get_Item("CH1").Enabled = true;
```

1.5 Set output resistance to 50 Ω.

```
Session.Channels.get_Item("CH1").Output = 50;
```

1.6 Configure the calibrator to DC Current output mode.

1.7 Connect the current calibrator to HI_CH1 and LO_CH1.

1.8 Set the current calibrator to the Set Current value in the below table.

1.9 Turn on the calibrator output.

1.10 Wait for 3 seconds or wait for calibrator output to become stable.

1.11 Record the ADC value in amps.

```
Measured = Session.Channels.get_Item("CH1").MeasuredCurrent
```

1.12 Try to collect much data points and average the readings.

1.13 Turn off the calibrator.

1.14 Disable the channel.

```
Session.Channels.get_Item("CH1").Enabled = false;
```

1.15 Check the current readings are within the limits 0.09% of the set current.

1.16 Repeat the above steps for the below values and repeat it for remaining channels.

1.17 The following are the test inputs.

Set Current (Amps)	Lower Limit	Measured (Amps)	Upper Limit
0.01	0.00999	Enter the reading	0.01001
0.009	0.008991		0.009009
0.008	0.007992		0.008008
0.007	0.006993		0.007007
0.006	0.005994		0.006006
0.005	0.004995		0.005005
0.004	0.003996		0.004004
0.003	0.002997		0.003003
0.002	0.001998		0.002002
0.001	0.000999		0.001001

0.0009	0.0008955		0.000905
0.0008	0.000796		0.000804
0.0007	0.0006965		0.000704
0.0006	0.000597		0.000603
0.0005	0.0004975		0.000503
0.0004	0.000398		0.000402
0.0003	0.000297		0.000303
0.0002	0.000198		0.000202
0.0001	0.000099		0.000101

2. Voltage ADC Verification.

The purpose of this test is to verify the ADC reading, after the calibration.

Required equipment: External voltage source
Recommended equipment: Fluke 5700/20/30 calibrator (or equivalent)

2.1 Load the calibration file from 0 to 6 in which the cal constants stored during calibration.

```
Session.Calibration.Load(0);
```

2.2 Enable the Calibration Mode.

```
 Session.Calibration.CalMode = true;
```

2.3 Connect the External DMM to HI_CH1 and LO_CH1.

2.4 Set the raw DAC value to 0.

```
Session.Calibration.Card.set_RawDac("CH1", 0x0);
```

2.5 Set the Raw DigiPOT to 0.

```
Session.Calibration.Card.set_RawDigipot("CH1", 0x0);
```

2.6 Enable the channel 1.

```
Session.Channels.get_Item("CH1").Enabled = true;
```

2.7 Set the input voltage gain for *X1*.

```
Session.Calibration.Card.set_VoltageGain("CH1",
  VTEXRTDSimulatorCalibrationGainEnum.Gainx1);
```

2.8 Configure the calibrator to DC Voltage output mode.

2.9 Set the current calibrator to the Set Voltage value in the below table.

2.10 Turn on the calibrator output.

2.11 Wait for 3 seconds or wait for calibrator output to become stable.

2.12 Check the Measured ADC Value from the RAW Voltage.

```
Raw Voltage = Session.Calibration.Card.get_RawVoltage("CH1");
ActualVoltage = ((Raw Voltage + ADCOffSet1x)/16777215)* VoltageADCGain1x;
```

2.13 Repeat the above steps for gain X10. Refer the below table.

2.14 Disable the Channel.

```
Session.Channels.get_Item("CH1").Enabled = false;
```

2.15 Check the Measured readings are within the limits 0.1% of the Set Voltage.

2.16 Repeat this test for remaining channels.

2.17 The following are the test inputs.

Set Voltage (Volts)	Lower Limit	Measured (Volts)	Upper Limit
10	**9.99**	**Enter the reading**	**10.01**
9	**8.991**		**9.009**
8	**7.992**		**8.008**
7	**6.993**		**7.007**
6	**5.994**		**6.006**
5	**4.995**		**5.005**
4	**3.996**		**4.004**
3	**2.997**		**3.003**
2	**1.998**		**2.002**
1	**0.999**		**1.001**
1	**0.999**		**1.001**
0.9	**0.8991**		**0.9009**
0.8	**0.7992**		**0.8008**
0.7	**0.6993**		**0.7007**
0.6	**0.5994**		**0.6006**
0.5	**0.4995**		**0.5005**
0.4	**0.3996**		**0.4004**
0.3	**0.2997**		**0.3003**
0.2	**0.1998**		**0.2002**
0.1	**0.0999**		**0.1001**

2.18

3. Voltage DAC Verification.

The purpose of this test is to verify the DAC Output, after the calibration.

Required equipment: 8.5 Digit DMM
Recommended equipment: Keithley 2002/Agilent 3458A (or equivalent)

3.1 Configure the DMM to 4 Wire Resistance measurement.

3.2 Connect the External DMM 4W Measurement points to HI_CH1, LO_CH1, SENSEHI_CH1 and SENSELO_CH1.

3.3 Enable the channel 1.

```
Session.Channels.get_Item("CH1").Enabled = true;
```

3.4 Set output resistance to any values from 4 ohms to 10000 Ohms.(Test for 10 test points)

```
Session.Channels.get_Item("CH1").Output = Resistance;
```

3.5 Check the Measured DMM Reading.

3.6 Disable the Channel.

```
Session.Channels.get_Item("CH1").Enabled = false;
```

3.7 Check the Measured readings are within the limits 0.01%±75 mOhm.

3.8 Repeat this test for remaining channels.

3.9 Close all instruments sessions.

APPENDIX A

ONBOARD MEMORY

MEMORY LISTINGS AND CLEARING PROCEDURES

The following table provides information regarding memory contained in the EX1200-7008. Protocol for clearing the memory is also included.

Component	Volatile?	Contains	User Writeable?	Clear Procedure
32 MB Flash	No	FPGA Image	No	None
64 MB Flash	No	Calibration	Yes	Use the Calibration APIs to overwrite the data
		SFP Image	No	None
16 MB (1M x 16) SRAM	Yes	Buffered Data	No	Power cycle instrument

INDEX