N2101A Option H10
5 Gb/s Bit Error Rate Tester

User's Guide

Agilent Technologies

Notices

Manual Part Number

N2101-90001

Edition

January 2007
Printed in USA

Agilent Technologies, Inc.
Digital Signal Analysis Division
1400 Fountaingrove Parkway
Santa Rosa, CA 95403, USA

Warranty

The material contained in this document is provided "as is," and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

LZW compression/decompression: Licensed under U.S. Patent No. 4,558,302 and foreign counterparts. The purchase or use of LZW graphics capability in a licensed product does not authorize or permit an end user to use any other product or perform any other method or activity involving use of LZW unless the end user is separately licensed in writing by Unisys.

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Agilent Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

Safety Notices

CAUTION

Caution denotes a hazard. It calls attention to a procedure which, if not correctly performed or adhered to, could result in damage to or destruction of the product. Do not proceed beyond a caution sign until the indicated conditions are fully understood and met.

WARNING

Warning denotes a hazard. It calls attention to a procedure which, if not correctly performed or adhered to, could result in injury or loss of life. Do not proceed beyond a warning sign until the indicated conditions are fully understood and met.

Trademark Acknowledgements

Microsoft is a U.S. registered trademark of Microsoft Corporation.

Windows and MS Windows are U.S. registered trademarks of Microsoft Corporation.

Instrument Markings

The CE mark is a registered trademark of the European Community.

Contents

1

General Information

Introduction

The N2101A BERT implements a full function Bit Error Rate testing system in a double-wide 3U PXI module. It consists of a high accuracy clock source, data pattern generator and error detector.

NOTE

Please note that this former PXIT product (PX2000-336) is now part of Agilent Technologies. Although most references have been changed to Agilent Technologies, this manual may contain some references to PXIT Inc.

The N2101A provides the following features:

- Differential data generation and analysis
- Standard data patterns PRBS 2^n-1, n = 7, 9, 10, 11, 15, 23, 31; K28.5, K28.7, CRPAT
- User specified patterns up to 256 kbits
- Internal clock rates for the following time bases: - SONET, GBE, Fiber Channel, SONET FEC for rates up to 6 Gb/s.
- External clock inputs for Tx and Rx
- Clock trigger out
- Pattern trigger out
- Data pattern generation - industry standard or user selectable.
- Bit error measurements: Count rate for the following data rates: 1.0625 Gb/s, 1.25 Gb/s, 2.125 Gb/s, 2.48832 Gb/s (OC-48/STM-16), 2.50 Gb/s, 4.25 Gb/s, 5.00 Gb/s.
- Single error and error rate injection.
- Auto-eye crossing search function with user settable BER threshold from 10^{-3} to 10^{-10}.
- Total error counts over definable time or number of bits.
- Continuous bit error rate measurement (real time error rate counter) with historical display of when errors and loss-of-sync events ocurred.
- Internal frequency modulation of transmitted signal
- Eye opening measurement at defined error rates
- Display of bathtub curve

CAUTION The N2101A is shipped in materials which prevent damage from static. The module should only be removed from the packaging in an anti-static area ensuring that correct anti-static precautions are taken.

WARNING **This product is NOT tested for use in medical or clinical applications.**

WARNING **No operator serviceable parts inside. Refer servicing to qualified service personnel.**

WARNING **Use appropriate caution when using Agilent products for testing lasers.**

WARNING **Laser Safety Notice The N2101A is used to measure optical signals. When connecting and disconnecting optical cables or equipment, all optical sources MUST be disabled. Failure to take proper safety precautions may result in eye damage. All un-used optical ports MUST be covered when not in use to prevent light leakage or contamination.**

CAUTION Do not operate the N2101A without attenuators attached to the output connectors. This will degrade the N2101A's performance over time, and will void the Agilent factory warranty.

Electrostatic Discharge Information

CAUTION

Electrical channel input circuits and the trigger input circuit can be damaged by electrostatic discharge (ESD). Therefore, avoid applying static discharges to the front-panel input connectors. Prior to connecting any coaxial cable to the connectors, momentarily short the center and outer conductors of the cable together. Avoid touching the front-panel input connectors without first touching the frame of the instrument. Be sure that the instrument is properly earth-grounded to prevent buildup of static charge. Wear a wrist-strap or heel-strap.

Electrostatic discharge (ESD) can damage or destroy electronic components. All work on electronic assemblies should be performed at a static-safe work station. The following figure shows an example of a static-safe work station using two types of ESD protection:

- Conductive table-mat and wrist-strap combination.
- Conductive floor-mat and heel-strap combination.

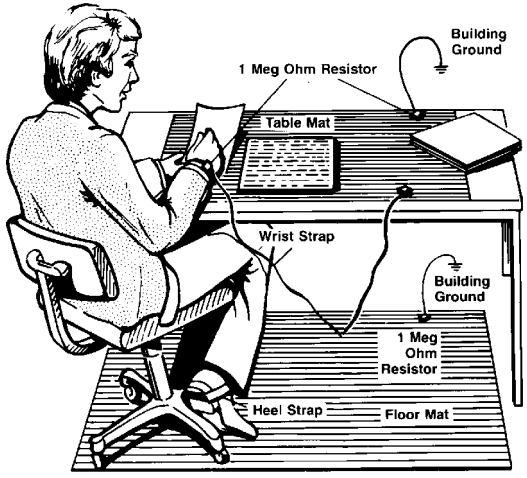


Figure 1-1. Static-safe Work Station

Both types, when used together, provide a significant level of ESD protection. Of the two, only the table-mat and wrist-strap combination provides adequate ESD protection when used alone. To ensure user safety, the static-safe accessories must provide at least 1 MΩ of isolation from ground. Refer to Table 2 for information on ordering static-safe accessories.

WARNING **These techniques for a static-safe work station should not be used when working on circuitry with a voltage potential greater than 500 volts.**

Table 2. Static-Safe Accessories

Agilent Part Number	Description
9300-0797	Static control mat, 0.6 m x 1.2 m (2 ft x 4 ft) and 4.6 cm (15 ft) ground wire. (The wrist-strap and wrist-strap cord are not included. They must be ordered separately.)
9300-0980	Wrist-strap cord 1.5 m (5 ft).
9300-1367	Wrist-strap, adjustable, without cord.
9300-1126	ESD heel-strap

Connector Care

Advances in measurement capabilities make connectors and connection techniques more important than ever. Observing simple precautions can ensure accurate and reliable measurements.

Handling and storage

- Keep connectors clean
- Extend sleeve or connector nut
- Use plastic endcaps during storage
- Do not touch mating plane surfaces
- Do not set connectors contact-end down

Visual inspection

- Inspect all connectors carefully before every connection
- Look for metal particles, scratches, and dents
- Do not use damaged connectors

Cleaning

- Clean with compressed air first
- Clean the connector threads
- Do not use abrasives
- Do not get liquid onto the plastic support beads

Making connections

- Use connector savers
- Align connectors carefully
- Make preliminary connection lightly
- To tighten, turn connector nut only
- Do not apply bending force to connection
- Do not over tighten preliminary connection
- Do not twist or screw in connectors
- Use a torque wrench, and do not tighten past the "break" point of the torque wrench

3.5 mm and SMA Connectors

Precision 3.5 mm microwave connectors are compatible with an SMA connector within its specification. Due to the variable quality of the SMA connector, mating with an SMA can sometimes cause severe damage to the 3.5 mm connector. You can use SMA connectors if special care is taken when mating the connectors, and all connectors are undamaged and clean. Before each use, check the mechanical dimensions of all connectors with a connector gauge to make sure that the center conductors are positioned correctly.

CAUTION A male SMA connector pin that is too long can smash or break the delicate fingers on the precision 3.5 mm female connector.

CAUTION Some precision 3.5 mm female connector fingers are very tight and can pull the center pin of their mates out past specifications when the connectors are disconnected. If such a male pin is inserted into a female connector, it can cause considerable damage by pushing the female center conductor back too far. Be aware of this possibility and check all connectors before mating them again.

Returning the N2101A to Agilent

The instructions in this section show you how to properly package the instrument for return to an Agilent Technologies service office. If the instrument is still under warranty or is covered by an Agilent maintenance contract, it will be repaired under the terms of the warranty or contract. If the instrument is no longer under warranty or is not covered by an Agilent maintenance plan, Agilent will notify you of the cost of the repair after examining the unit.

When an instrument is returned to an Agilent service office for servicing, it must be adequately packaged and have a complete description of the failure symptoms attached.

When describing the failure, please be as specific as possible about the nature of the problem. Include copies of any instrument failure settings, data related to instrument failure, and error messages along with the instrument being returned.

Please notify the service office before returning your instrument for service. Any special arrangements for the instrument can be discussed at this time. This will help the Agilent service office repair and return your instrument as quickly as possible.

Call Center

For technical assistance, contact your local Agilent Call Center. In the Americas, call 1 (800) 829-4444. In other regions, visit http://www.agilent.com/find/assist. Before returning an instrument for service, you must first call the Call Center at 1 (800) 829-4444.

Preparing the product for shipping

1. Write a complete reason for returning the product and attach it to the instrument. Include any specific performance details related to the problem.
2. Pack the product. Use original packaging or comparable. Original materials are available through any Agilent office. Or, follow these recommendations:
   - Use a double-walled, corrugated cardboard carton of 159 kg (350 lb) test strength. The carton must allow approximately 7 cm (3 inches) on all sides

of the kit for packing material and be strong enough to accommodate the weight of the kit.

- Surround the kit with approximately 7 cm (3 inches) of packing material, to protect the kit and prevent it from moving in the carton. If packing foam is not available, the best alternative is S.D-240 Air Cap™ from Sealed Air Corporation (Commerce, California 90001). Air Cap looks like a plastic sheet filled with air bubbles. Use the pink (antistatic) Air Cap™ to reduce static electricity. Wrapping the kit several times in this material will protect the kit and prevent it from moving in the carton.

3 Seal the carton with strong nylon adhesive tape.

4 Mark the carton “FRAGILE, HANDLE WITH CARE”.

5 Retain copies of all shipping papers.

2

Installation

Introduction

The PXI chassis can be controlled using either embedded PXI controller or an external PC using a PCI - cPCI/PXI remote bridge (such as the NI MXI-4 product). If an external PC is used, the PC must meet the following specification:

- Windows 2000 or XP operating system
- 128 MB RAM
- Pentium, 133 MHz or greater

CAUTION Do not operate the N2101A without attenuators attached to the output connectors. This will degrade the N2101A's performance over time, and will void the Agilent factory warranty.

NOTE When using an external PC, if the sequence of the following installation steps is not followed, the PC BIOS will not be able to locate the instruments in the PXI chassis.

Your new N2101A is compatible with the current version of the control panel software. If your PXI chassis includes older N2101A (PX2000-336) instruments, you may need to upgrade the instrument's firmware as described in "Upgrading the Instrument's Firmware" on page 3-14.

Step 1. Inspect the Shipment

1 Inspect the shipping container and kit for damage. Keep the shipping container and cushioning material until you have inspected the contents of the shipment for completeness and have checked the kit mechanically and electrically.

2 Locate the shipping list. Verify that you have received all of the items listed.

To contact Agilent Technologies for technical assistance, contact your local Agilent Call Center. In the Americas, call 1 (800) 829-4444. In other regions, visit http://www.agilent.com/find/assist. Before returning an instrument for service, you must first call the Call Center at 1 (800) 829-4444.

3 Do not install the N2101A in the PXI chassis at this time.

WARNING **Ensure that the PXI chassis is connected to the specified power source using the correct power cord (noting country of use).**

WARNING **Ensure that the PXI chassis containing the N2101A provides adequate earth grounding.**

WARNING **Ensure that the air supply to the chassis is working correctly. The N2101A requires an optimal air flow within the chassis. It is recommended to regularly change filters on PXI Chassis.**

Step 2. Install the Instrument Driver Software

1 If using an external PC and remote bridge, turn the PXI chassis power off. If using an embedded controller, remove all N2101A modules from the chassis.

NOTE This step ensures that the PC BIOS will be able to locate the instruments in the PXI chassis.

2 Log onto the PC with administrator privileges, so that you can install the software.

3 Go to the Agilent website: www.agilent.com/find/pxit

4 Click on the Technical Support link and then the Drivers link.

5 Download the latest version of the following driver:

```
Agilent N2101A (PX2000-336) BERT Driver
```

6 Once the download has completed, run the file, N2101AInstall.exe. During the installation, you will enter the user name and organization. Select the all users option to ensure the software is available to all users of the PC. Click **Next**.

7 When install is finished click **Finish**. The N2101A control software is now installed.

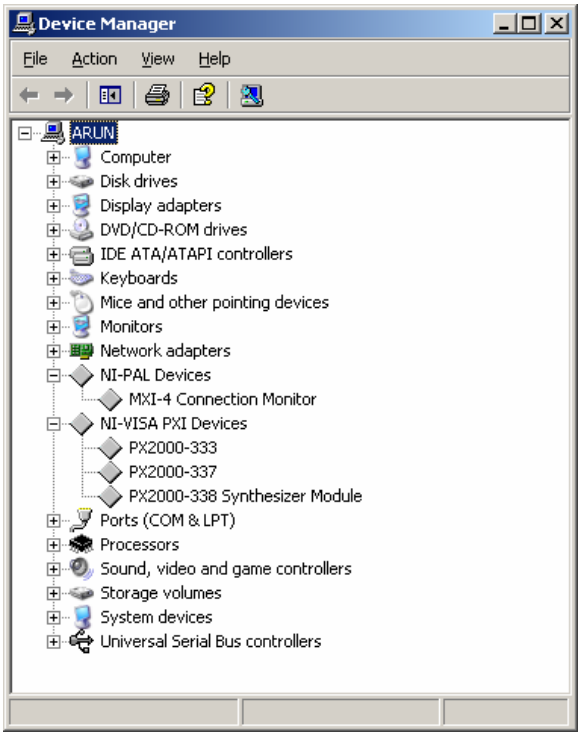
Step 3. Install the N2101A

1 With the PC and chassis powered off, install the N2101A module in an available slot in a PXI chassis.

2 Power on the PXI chassis and wait for the power up sequence to complete.

3 Turn on the PC.

If needed, you can use Windows Device Manager to determine if the instruments have been correctly identified by the BIOS. There should be an NI-VISA PXI Devices entry with your N2101A/PX2000 series instrument as shown in the following figure.



3

Using the Control Panel

Introduction

This chapter describes how to start and configure the Windows Control Panel application. All aspects of the N2101A can be controlled through the Control Panel. Use the top row of buttons to configure the module, I/O, and connect to the module. The mode of operation of the N2101A is controlled using the list box in the top left corner of the main form.

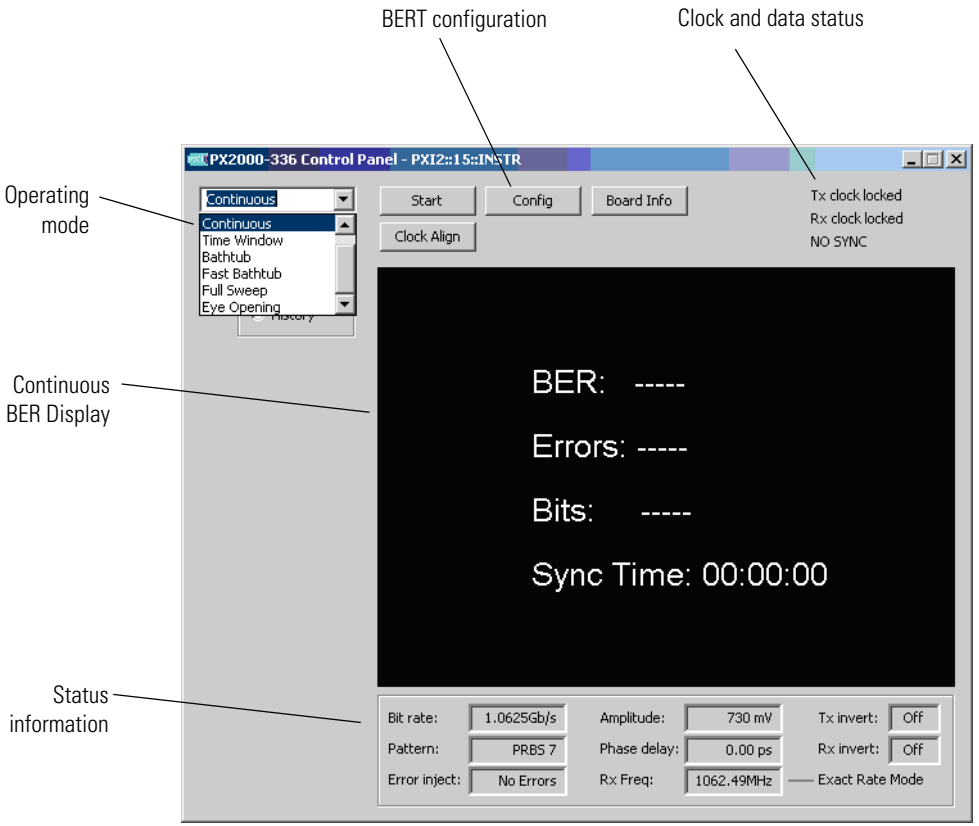


Figure 3-1. Control Panel

CAUTION

Do not operate the N2101A without attenuators attached to the output connectors. This will degrade the N2101A's performance over time and will void the Agilent factory warranty.

In continuous test mode, select **Current** to display the continuous BER results as shown in Figure 3-1. Or, select **History** to for a scrolling display of bit errors and out-of-sync conditions that is updated in real-time as shown in Figure 3-2. Controls are also available to stop, manually scroll, and scale the time base of the display, without interrupting the current test. Clicking **Live Update** resumes automatic real-time scrolling.

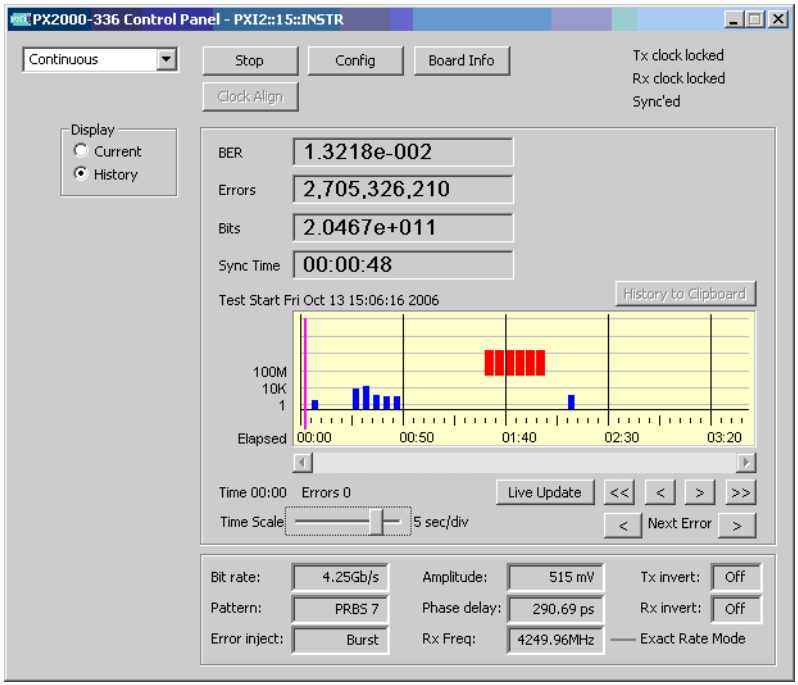


Figure 3-2. Historical Display of Bit Errors

The running test time is derived from current time minus the test start time in the display. Cumulative Error Counts, BER, and Sync Time are also displayed. Clicking **History to Clipboard** captures any non-zero bit error count time intervals, or LOS conditions, in ASCII readable form, as shown in the following example:

```
Test Start Mon Sep 18 15:42:31 2006
Test Time 00:03:54
Resolution 1 sec
00:00:06  33
00:00:08  33
00:00:12  66
00:00:16  132
00:00:17  66
00:01:28  LOS
00:01:29  LOS
00:03:13  4
```

In the Control Panel, click **Config** to select the N2101A and configure it for measurements. The **Rx Clock Source External** selection is unsupported. Rx Clock Source must be set to **Internal**, and it will track the Tx Clock Source provided (for example, Internal or External).

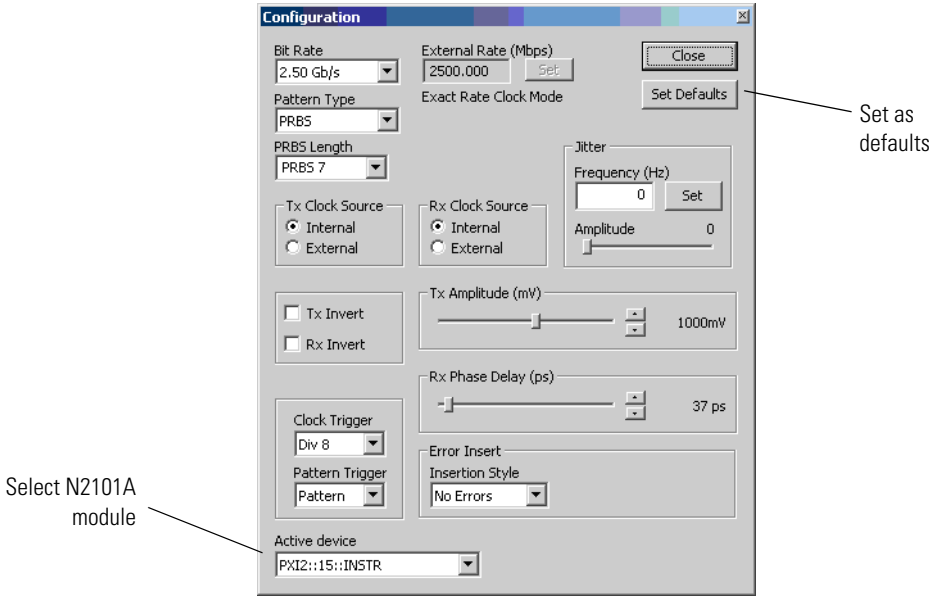


Figure 3-3. Configuration dialog box

To start the Control Panel

1. Click the Window's Start menu.
2. Click **All Programs**, **PXIT**, and then **PX2000-336**.
3. Click **PX2000-336 Control Panel**.
4. Click the **Config** button, which is located at the top of the Control Panel.
5. Select the **Active device** (N2101A module) using the drop-down list.
6. Set all other configuration settings as needed.

Powering Off the Instrument

1. Close the Control Panel.
2. Soft power down the PC.
3. Switch off power to PXI chassis.

User Patterns

You can use your own user patterns with the N2101A. The Control Panel application includes a user pattern editor, and you also save and import text pattern-files to and from the N2101A. To use user patterns, open the Configuration dialog box, and select **User** from the **Pattern Type** list; the N2101A immediately begins generating and synchronizing to any specified pattern.

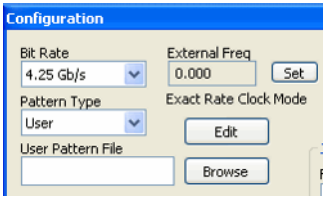


Figure 3-4. Pattern Type set to User

Click **Edit** to view the User Pattern Script Editor shown in Figure 3-5. The editor uses an abbreviated script syntax, which is explained in the editor. Use the Script field to enter the syntax for the bit pattern. Click **Set Pattern** to verify the syntax and display the resulting pattern of bits in the Pattern field.

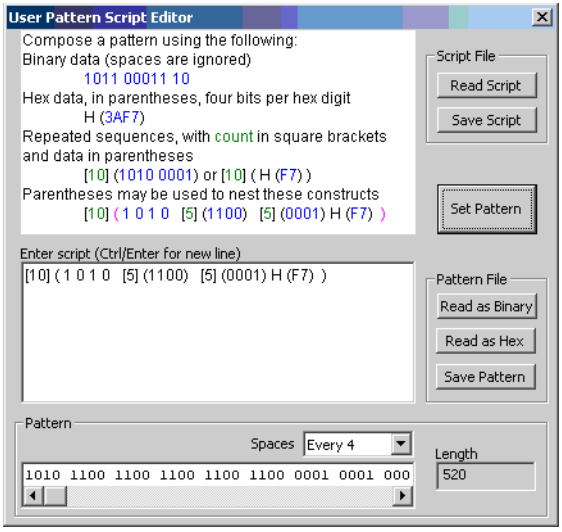


Figure 3-5. User Pattern Editor

User pattern lengths must meet the following criteria:

- Integers from 1 to 2048
- Even numbers up to 4,096
- Multiples of 4 up to 8,192
- Multiples of 8 up to 16,384
- Multiples of 16 up to 32,764
- Multiples of 32 up to 65,536
- Multiples of 64 up to 131,072 (or 2^{17}).

Quick Confidence Check

Physical loopback test allows you to verify the basic functionality of the instrument.

1. In the Control Panel, select the **Time Window** or **Continuous** from the drop down menu.
2. Click the **Config** button.
3. Select the following typical settings:

   Bit Rate . . . . . 1.0625Gb/s (or a suitable frequency)
   Pattern Type . . . . . PRBS
   PRBS Length PRBS7

4. Set Tx and Rx clock sources to internal.
5. Close the dialog box.

CAUTION Do not remove the attenuators from the transmitter output connectors.

6. Install loopback cables between the transmiter and receiver connectors. This should be done either using both the differential signal pairs or by making a single Tx to Rx connection and terminating the unused connectors with a 50 ohm cap.
7. Click the **Clock Align** to set the Rx phase delay to the center of the eye. The alignment process is complete when the phase delay values stop incrementing.
8. Click on the **Start** and the form shown in Figure 3-6 on page 3-9 should appear. Check that no errors are being reported.

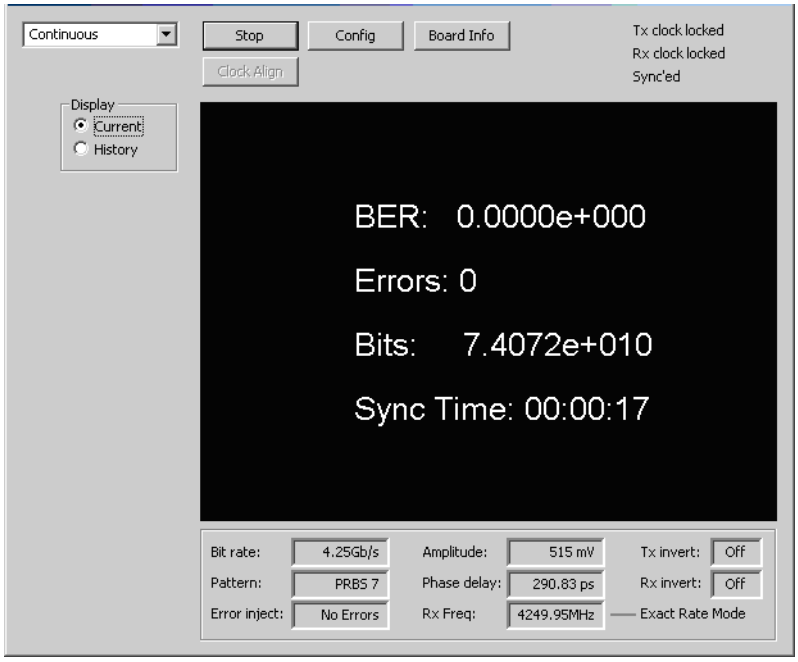


Figure 3-6. BER Screen with Zero Errors

9 Click the **Config** button.

10 Inject a single error as follows. Select single burst **Error Insert** from the configuration panel. Set the Count to 1.

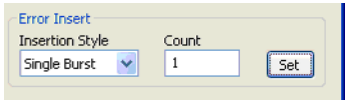


Figure 3-7. Single Error Injection (Configuration Panel)

11 Click **Set** once.

12 Close the dialog box.

13 Verify correct insertion and error detection on the following screen.

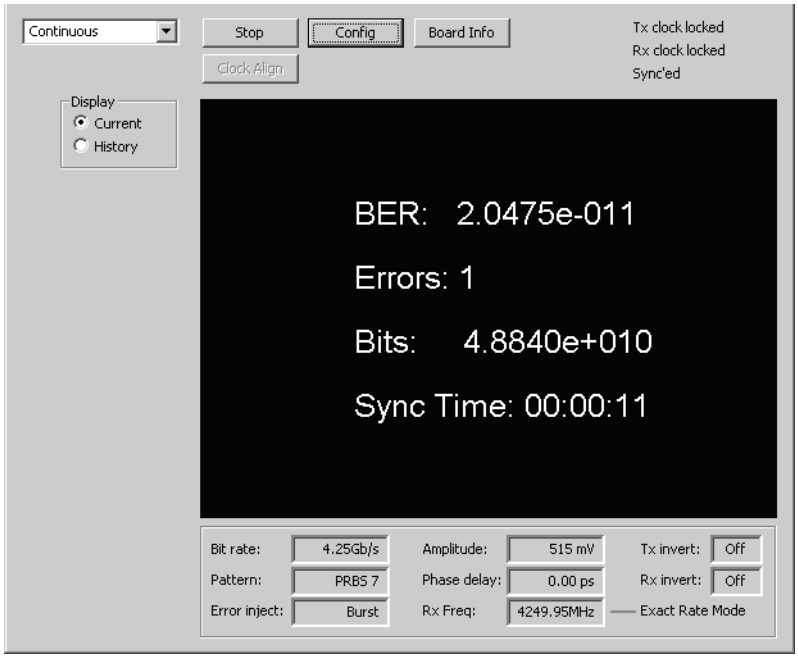


Figure 3-8. BER Screen with a Single Error Detected

Measuring Bit Error Rate

You can use the Control Panel to measure Bit Error Rate.

1. In the Control Panel, select **Continuous** from the drop down menu. Or, you could select **Time Window** and set the Window Size to the desired time window as well as set the units to µs, ms or s.
2. Click the **Config** button.
3. Select the following settings:

Bit Rate	1.0625Gb/s,1.25Gb/s, 2.125Gb/s, 2.48832Gb/s, 2.5Gb/s, 4.25Gb/s (Default), 5.0Gb/s, External.
Pattern	Select data pattern sent on the transmit signal, and synchronized to on the receive signal. Each pattern type has further parameters specific to its type. PRBS (default), DC, User.
Pseudo Random Bit Sequence (PRBS)	PRBS pattern lengths: 2^7–1 (default), 2^9–1, 2^{11}–1, 2^{15}–1, 2^{23}–1, 2^{31}–1.
Tx clock source	Can be external or internal. Internal uses the BERT internal clock (default). External uses a clock connected from an external source to the BERT.
Rx Clock Source	Can be external or internal. Internal uses the BERT internal clock (default). External uses a clock connected from an external source to the BERT. External Rx clock source is not currently supported. Set Rx clock to internal and it will track the Tx clock source (either internal or external) that is set above.
Rx Clock Frequency	This field is only active when either Tx or Rx is set to External. Specifies the desired bit rate clock . For bit rates where 2.5 GHz ≤ ≤3.3 GHz, you must inject a clock at double the rate. For example, to use 2.7 Gb/s, the user must provide a 5.4 GHz clock at the TX-CLK-IN inputs.
Tx Invert	Transmit signal polarity is inverted if this box is checked. (default - not inverted)
Rx Invert	Receive signal polarity is inverted if this box is checked. (default - not inverted)

Jitter Frequency	Specifies the frequency in Hz (0 to 10 MHz max, 10 kHz typical) (default = 0). This feature injects jitter at an uncalibrated rate and is not fully characterized. It is provided to the customer on an "As is" basis.
Jitter Amplitude	Specifies the amplitude in UI DAC units (255 is typically 0.2 UI max) (default = 0) . This feature allows the user to inject DJ into the transmitted signal. Agilent has not characterized nor calibrated this function. It is provided "As Is"
Tx Amplitude (mV)	Amplitude of transmit signal specified in millivolts. (default = 1000 mV)
Phase Delay	This option is used to specify a phase delay between the signal and receiver clocks. The range of the programmable phase delay is 1.2 ns specified in 1023 increments. This option holds the number of such increments. (default= 0)
Error Insert	Allows user to inject errors into data pattern, can select: Single burst of K errors: where K is user specified. Continuous (Rates of 10^-N for N = 3 to 10). None (Default)
Clock trigger	Clock Trigger is a divided down version of the transmit clock. Can be set to either of the following Div8 (default), Div9, RefClk - 106.25, 155.52, or 156.25 MHz depending on bit rate as shown in Table 3-1 on page 3-13.
Pattern Trigger	Can be set to either of the following Pattern (generates a trigger pulse synchronous with the transmitted pattern (Default). Div128 (Generates a divided down version of the bit rate clock
Window Size	Specifies a time window during which the BER is measured (Default = 250 ms)

4 Click on **Start** to begin the BER measurement. When the measurement is complete, the Control Panel displays the error rate.

Table 3-1. Clock Trigger Output Frequency Options

Internal Bit Rate Clock (Gbps)	Div 8 (MHz)	Div 9 (MHz)	RefClk (MHz)
1.0625	132.81	118.06	106.25
1.25	156.25	138.89	156.25
2.125	265.63	236.11	106.25
2.48832	311.04	276.48	155.52
2.50	312.50	277.78	156.25
4.25	531.25	472.22	106.25
5.00	625.00	555.56	156.25

Upgrading the Instrument's Firmware

Your new N2101A is compatible with the current version of the control panel software. If your PXI chassis includes older N2101A (PX2000-336) instruments, you may need to upgrade the firmware in those older instruments as described in this section. To update the firmware and execute the FPGA Loader program, perform the following steps:

1. Close any open Control Panels.
2. On the Windows Start menu, click All Programs, PXIT, PX2000-336, PX2000-336 FW-FPGA Loader.
3. Press the appropriate buttons to connect to a given module, and note the current module Serial Number, API, and Firmware Versions.
4. Select the new firmware and FPGA files located in the installation directory (typically C:\Program Files\PXIT\PX2000-336\fw), and press start.

   A progress bar appears as the module is updated.
5. Once complete, shut down the host PC and cycle the power on the PXI chassis.
6. After rebooting the PC, connect to the module and obtain Module Configuration, to confirm proper software upload.

4

Programming

Introduction

A Dynamic Link Library (DLL) provides an API through which custom applications can be easily developed for the N2101A. For further ease of programming and compatibility, an Active-X Control is also provided to access the N2101A. The Active-X Control extends programmability to any Active-X container application or COM compliant programming environment. This includes Visual C++, Visual Basic, C#, LabView and many other environments. The Active-X methods are listed in the Active-X Programmer's Guide which is installed with the software.

The general sequence of operation needed to perform a bit error measurement is as follows:

1. Configure the traffic generator. Refer to “Configuring the Traffic Generator” on page 4-3.
2. Configure the receiver. Refer to “Configuring the Signal Receiver” on page 4-7.
3. Select the type of measurement to be carried out and execute it. Making Measurements 4-10.

Configuring the Traffic Generator

Calling the function PXIT336ConfigureTx() configures the traffic generator. The first parameter is a VISA handle that identifies the instrument to be configured. Refer to “Obtaining the N2101A VISA Handle” on page 4-16. The complete configuration of the traffic generator is passed as the second parameter to this function, with the following options:

Bit Rate

Specifies the bit rate of the generated signal. The possible values are:

- 1.0625 Gb/s,
- 1.25 Gb/s,
- 2.125 Gb/s,
- 2.48832 Gb/s,
- 2.50 Gb/s,
- 4.25 Gb/s,
- 5.00 Gb/s

Pattern

Specifies the Pattern of the generated signal. The pattern can be one of 3 types: PRBS, DC, or User. Each pattern type has further parameters specific to its type.

Tx clock source and frequency

Can be external or internal. Internal uses the BERT internally generated clock, and will match the Bit Rate indicated above. External uses a clock connected from an external source to the BERT. In this mode, the bit rate is derived from the TX-CLK-IN input into the module. Depending on the external frequncy specified, the BERT may be required to operate in "Double Clocking Mode". The extern clock will have to be 2x the desired bit rate clock (e.g. 5.4 Ghz for 2.7 Gb/s rate).

Signal Amplitude

Output signal amplitude specified in mV. The actual output range of the N2101A is from 500 mV to 2V. Requesting the output signal amplitude to be set below the minimum will result in the output signal being set to 500 mV. Requesting a value above the output amplitude range's maximum will result in the output amplitude being set to 2V. The resolution of the output signal is 5 mV.

Injected Errors

Specifies the type of errors to inject if any. Refer to "Jitter Insertion" on page 4-15.

Data Invert

Allows to invert the transmit signal.

Pattern Type

The following parameters are used to specify the data pattern type.

PRBS pattern lengths can be one of:

- PRBS 2^{7}-1,
- PRBS 2^{9}-1,
- PRBS 2^{11}-1,
- PRBS 2^{15}-1,
- PRBS 2^{23}-1,
- PRBS 2^{31}-1.

DC pattern type can be one of the following:

- K28.5
- K28.7,
- CRPAT,

A user pattern is defined by a handle to the file containing the pattern and the pattern length. Refer to "BertUserPattern" on page 4-27.

Transmitter External Clocking Configuration Example

```
#include <stdlib.h>
#include <windows.h>
#include <pxit336.h>

main ()
{
     ConfigureTheTransmitter();
}

void ConfigureTheTransmitter(void)
{
    ViSession instHandle;            VISA handle for the BERT.
    BertTxConfigExt txConfigExt;      BERT transmitter
configuration.
    PXIT336GetInstHandle(&instHandle, 0); Get the VISA
instrument handle to the first BERT module found in the system.
    if (instHandle != NULL)
    {
//select the clock source.
         txConfigExt.ClkSrc = 1;
         txConfigExt.ClkRef = 2500.0;
         txConfigExt.Pattern.Type = DC;
         txConfigExt.Pattern.DcPattern = K28dot7;select data
pattern.
         txConfigExt.DataInvert = 1;data inversion
         txConfigExt.Amplitude = 500;in millivolts
//Instruct the BERT to use the new transmitter parameters.
         PXIT336ConfigureTxExt(instHandle, &txConfigExt);
    }
}
```

Generate Internally Clocked 4.25Gb/s PRBS-7 Example

```
#include <stdlib.h>
#include <windows.h>
#include <pxit336.h>

main ()
{
     GeneratePRBS7();
}

void GeneratePRBS7(void)
{
    ViSession instHandle;   VISA handle for the BERT.
    BertTxConfig txConfig;  BERT transmit configuration.

   PXIT336GetInstHandle(&instHandle, 0);Get the VISA instrument
handle to the first BERT module found in the system.
    if (instHandle != NULL)
    {
      PXIT336Reset(instHandle);Reset the BERT module.
     PXIT336GetTxConfig(instHandle, &txConfig); Get the current
BERT transmit configuration.
      txConfig.BitRate = _4dot25_Gbps;
      txConfig.Pattern.Type = PRBS;
      txConfig.Pattern.PrbsLength = PRBS_7;Set the BERT
configuration to generate a PRBS 7 pattern.
PXIT336ConfigureTx(instHandle, &txConfig);Instruct the BERT to
use the new transmit parameters.
    }
}
```

Configuring the Signal Receiver

Calling the function PXIT336ConfigureRx configures the BERT receiver. The first parameter is a VISA handle identifying the BERT module to configure. Refer to “Obtaining the N2101A VISA Handle” on page 4-16. A pointer to an instance of the BertRxConfig structure is passed as second parameter. The content of the BertRxConfig structure fully defines the configuration to be applied to the BERT's receiver.

Bit Rate

Specifies the bit rate of the generated signal. Refer to “BertBitRate” on page 4-26 for values. The possible bit rates are:

- 1.0625Gb/s,
- 1.25Gb/s,
- 2.125Gb/s,
- 2.48832Gb/s,
- 2.50Gb/s,
- 4.25Gb/s,
- 5.00Gb/s

Pattern

Specifies the Pattern of the of the signal to receive. The pattern can be one of 3 types: PRBS, DC, or User. Each pattern type has further parameters specific to its type.

Rx Clock Source

Can be external or internal. Internal uses the BERT internal clock. (Default). External uses a clock connected from an external source to the BERT. External Rx clock source is not currently supported. Set to Internal, the Rx clock will track the Tx clock source set above.

Data Invert

This option allows inverting the received signal.

Phase Delay

This option is used to specify a phase delay between the signal and receiver clocks. The range of the programmable phase delay is 1.2 ns specified in 1023 increments. This option holds the number of such increments. Refer to “PXIT336SetRxPhaseDelay” on page 4-22 for more detail.

Receiver Configuration Example

```
#include <stdlib.h>
#include <windows.h>
#include <pxit336.h>

main ()
{
     ConfigureTheReceiver();
}

void ConfigureTheReceiver(void)
{
    ViSession instHandle;            VISA handle for the BERT.
    BertRxConfig rxConfig;      BERT receiver configuration.
    PXIT336GetInstHandle(&instHandle, 0); Get the VISA
instrument handle to the first BERT module found in the system.
    if (instHandle != NULL)
    {
//select the receiver's clock source.
         rxConfig.UseExternalClkSrc = Internal;
        rxConfig.BitRate = _1dot25_Gbps;select the receiver's
bit rate.
         rxConfig.Pattern.Type = DC;
       rxConfig.Pattern.DcPattern = K28dot7;select the receiver
pattern.
         rxConfig.DataInvert = 0;no data inversion
         rxConfig.PhaseDelay = 0;no phase delay
//Instruct the BERT to use the new receiver parameters.
         PXIT336ConfigureRx(instHandle, &rxConfig);
    }
}
```

Configuring the Trigger Outputs

There are two trigger outputs on the N2101A: Clock Trigger and Pattern Trigger. Clock trigger is a divided down version of the internal bit rate clock. It can be set to one of three settings: Transmit clock divided by 8, transmit clock divided by 9, or internal PLL reference clock. Note that in external double clocking mode, the divide by 8 and divide by 9 are referenced to the doubled clock input. Refer to “PXIT336SetClockTrigger” on page 4-22. Pattern trigger can be programmed to generate a pulse at the start of the transmit pattern or toggle every 128 bits. Refer to “PXIT336SetPatternTrigger” on page 4-23.

Making Measurements

Bit Error Rate Measurement

Used to measure the bit error rate, BER is defined as the ratio of the number of errors detected to the total number of received bits.

Procedure

1. Configure the traffic generator by calling the function PXIT336ConfigureTxExt().
2. Configure the receiver by calling the function PXIT336ConfigureRx().
3. Select the type of measurement to be carried out by calling the function PXIT336StartMeasure().
4. Wait for the measure to be completed by polling the BERT's status using PXIT336GetStatus(). Alternatively you can set a call back function to be called on completion of the measure, as one of the parameters to the PXIT336StartMeasure() function.
5. Read the measure result(s) by calling the function PXIT336GetBitErrorCount() if only one bit error count result is expected. For measures resulting in a series of bit error count results, inspect the result data structure passed as parameter to PXIT336StartMeasure().

Results

A structure of the type BertBitErrorCount, this contains the number of error bits and the total transmitted bits.

Eye Opening Measurement

Used to measure the eye opening for a specified BER value.

Parameters

This measurement only takes two parameters the bit error rate (BER) value for which to determine the eye opening and a pointer to a function that will be called once the eye opening has been determined. See example code below.

Results

A structure of the type BertEyeOpening. The eye opening result is given in picoseconds.

Example

```
#include <stdlib.h>
#include <windows.h>
#include <pxit336.h>


void EyeOpeningMeasureExample(void);
Double eyeOpeningResult;  Variable to store eye opening
main ()
{
EyeOpeningMeasureExample();
      printf("Eye opening: %d ps", eyeOpeningResult);
}

void EyeOpeningMeasureExample(void)
{
    ViSession instHandle;              VISA handle for the BERT.
     PXIT336GetInstHandle(&instHandle, 0);Retrieve instrument
handle

  BertEyeOpeningParam eyeOpeningParameters;

// Indicate specific values for the eye opening search.
     eyeOpeningParameters.BathtubParam = [… desired values];

// Specify the bit error rate for which to measure the eye
opening.
     eyeOpeningParameters.TargetBer = 0.0001;
// Indicate where to store the measurement results.
     eyeOpeningParamers.pEyeOpening = &eyeOpeningResult;
// Initiate the eye opening measurement.
     PXIT336StartMeasure(instHandle, EyeOpening,
&eyeOpeningParameters);
}
```

Bathtub Measurement

This measurement is used to generate a Bathtub diagram with the phase delay on the X-axis and the bit error rate on the Y-axis. The phase resolution and number of measured points are configurable.

Parameters

The input parameters for this measurement are as follows:

- Phase increment. This is the number of phase delay increments between each individual measure points. This is the resolution on the X-axis of the Bathtub diagram.
- Window size. This is the maximum length of time spent measuring the bit error count at each phase delay intervals. The actual window size can be shorter depending on the BER detected. This window size is specified in microseconds (1.0E-06 seconds).
- Call back function. This is a pointer to a function called on completion of the Bathtub measure.
- Number of measure points. This is the total number of measure points to be included in the Bathtub curve.

Results

An array of structures of type BertMeasurePoint. Please note that it is the caller's responsibility to allocate the memory for this array.

Time Window Measurement

This measurement is used to report the bit error rate in a sliding time window. A call to PXIT336GetBitErrorCount() will report the number of bit errors during the last elapsed time window when using this measure type. The time window is specified in microseconds. It has a range of 0 to 221 seconds. A call back function can optionally be called at the end of every measurement window.

Parameters

This measurement only takes two parameters. The first parameter is the window size specified in microseconds. The second parameter is a pointer to a function that will be called every time a new measurement is ready. This pointer can be set to NULL if you don't wish to use this feature.

Results

A structure of the type BertBitErrorCount. This variable contains the number of bit errors and the number of compared bits. The bit error rate can be derived from these by dividing the number of bit errors with the number of compared bits.

Generating Traffic Errors

Errors can be injected in the traffic generator output by calling PXIT336InsertError() with a pointer to a structure of type BertErrorInsert. Continuous rate injection can be turned off by setting the style to NoErrors. Each call with style of singleBurst will inject Count bit errors.

Configuration Option

Style. Specifies the style of error insertion. Possible values are: NoErrors, SingleBurst, Continuous.

Rate. Specifies the rate at which errors should be inserted.

Count. Specifies the number of errors to be inserted for each call to this function.

Jitter Insertion

Although not supported, this feature is provided for users to characterize their equipment under test. The output signal characteristics are not fully defined. Deterministic Jitter can be inserted onto the data output and clock signals. The rate of jitter (frequency up to 10MHz) and amplitude in time (about 20% of UI), can be varied using the control panel and the API. Refer to "PXIT336SetJitterFrequency" on page 4-25 and "PXIT336SetJitterAmplitude" on page 4-25. The magnitude of this jitter is not calibrated nor fully characterized at this time. Jitter can only be inserted on the internal clock source.

Configuration Option

Frequency. Specifies the frequency in Hz (10 MHz max). Typical values are 10 kHz. Represents the rate of frequency modulation of the nominal bit rate clock.

Amplitude. Specifies DJ amplitude in DAC units (255 max). Uncalibrated, but typically 0.2 UI max.

Obtaining the N2101A VISA Handle

All calls to the BERT API take a VISA handle as first parameter. This handle identifies the module to be accessed by the function call. This handle can be obtained by directly using the VISA API.

The VISA handle can also be obtained by calling the function PXIT336GetInstHandle(). The first parameter to this function call is a pointer to the location where the VISA handle will be stored. The second parameter is used to enumerate the BERT modules present in the system. Setting this second parameter to 0 results in the VISA handle of the first BERT module detected in the system to be returned. Incrementing this parameter in subsequent calls will return the VISA handle of other BERT modules detected in the system.

All BERT modules present in the system can be enumerated by calling PXIT336GetInstHandle in a loop. Incrementing the value of the second parameter until the VISA handle returned is NULL will allow to obtain a VISA handle for each BERT module present in the system.

Example 1

Obtain the VISA handle of the first or only BERT module present in the system and reset it.

```
#include <stdlib.h>
#include <windows.h>
#include <pxit336.h>

main ()
{
 ViSession instHandle;           VISA handle for the BERT.
 PXIT336GetInstHandle(&instHandle, 0);Retrieve instrument
handle
      if (instHandle != NULL)
    {
         PXIT336Reset(instHandle);Reset the BERT module
    }
 }
```

Example 2

Enumerate all the BERT modules present in the system and reset them.

```
#include <stdlib.h>
#include <windows.h>
#include <pxit336.h>
```

```
main ()
{
ViSession instHandle;           VISA handle for the BERT.
int inc;define local variable
    inc = 0;
    do
    {
// Obtain the VISA handle for the Nth BERT module.
        PXIT336GetInstHandle(&instHandle, inc++);
        if (instHandle != NULL)
        {
// Reset the BERT module identified by VISA handle insHandle.
          PXIT336Reset(instHandle);
        }
    } while(instHandle != NULL);
}
```

DLL API Reference

This section documents the functions that are included in the DLL. Table 4-1 specifies bits 16 to 31 of the methods return values. You should mask off the lowest 16 bits of the return values before to compare it to one of the major error codes described in this table. Bits 16-31 of the errors codes may contain info about the cause of the error.

Table 4-1. Function Return Codes

Mnemonic	Value	Description
SUCCESS	0x00000000	Operation completed successfully.
UNEXPECTED_ERROR	0x00010000	Unexpected error. Contact support with a description of how this error occurred.
MEASURE_IN_PROGRESS_ERROR	0x00020000	The requested operation cannot be completed because there is a measurement currently running. Try calling the StopCurrentMeasure() method before retrying.
UNKNOWN_DEVICE	0x00030000	The instrument handle used is not recognised as a valid handle. Make sure there is at least one PATTERN GENERATOR present in the system, that the PXI chassis is powered up and that the PC has been rebooted after the chassis was powered up.
INVALID_PARAMETER	0x00040000	One of the parameters passed to the method is invalid or is not supported by the revision of hardware present in the system.
BOARD_REGISTER_ERROR	0x00050000	Cannot access the N2101A board registers. Power cycle the system. Contact support if problem persists.
BOARD_DEVICE_ERROR	0x00060000	Cannot access one of the devices present on the board. Usually happens when one of the I2C devices present on the board is not responding. Bits 0-7 contain the bus address and bits 8-15 contain the bus number of the device causing the error.

PXIT336GetInstHandle

Obtains a VISA handle identifying a BERT module detected in the system. This handle is stored in the location passed as parameter. The content of this location is set to NULL if no BERT module was detected for the specified index.

Function Prototype int PXIT336GetInstHandle(ViSession *instHandle, int index);

instHandle Pointer to the location where the VISA handle of the first BERT module detected in the system will be stored.

index Index of BERT module for which a VISA handle is requested. This index starts at 0 and can be incremented after each call to PXIT336GetInstHandle to enumerate all the BERT modules present in the system.

PXIT336Reset

Resets the BERT hardware. The traffic generator and receiver are configured to a known "sensible" state that can be retrieved by calls to PXIT336GetTxConfig or PXIT336GetRxConfig.

Function Prototype int PXIT336Reset(ViSession inst);

inst VISA handle of the BERT module to be reset.

PXIT336GetHardwareVersion

Reads the BERT hardware version number.

Function Prototype int PXIT336GetHardwareVersion(ViSession inst, BertVersion *versionNb);

inst VISA handle of the BERT module from which to read the hardware version.

versionNb Pointer to a BertVersion variable into which the function will write the hardware version number.

PXIT336GetSoftwareVersion

Reads the BERT software version number.

Function Prototype void PXIT336GetSoftwareVersion(BertVersion *versionNb);

versionNb Pointer to a BertVersion variable into which the function will write the software DLL version number. The verision number is represented as a 32 bit integer. For example, 0x02050A00 corresponds to version 2.5.10.0.

PXIT336GetStatus

Returns the current status of the BERT.

Function Prototype int PXIT336GetStatus(ViSession inst,BertStatus *status);

inst VISA handle of the BERT module from which to read the hardware version.

status A pointer to an instance of BertStatus where the status of the BERT will be stored.

PXIT336GetTxConfig

Retrieve the current configuration of the transmit part of the BERT.

Function Prototype int PXIT336GetTxConfig(ViSession inst, BertTxConfig *txConfig);

inst VISA handle of the BERT module from which to read the configuration.

txConfig Pointer to an instance of BertTxConfig. Points to the location where the transmit configuration will be stored.

PXIT336GetRxConfig

Retrieve the current configuration of the receive part of the BERT.

Function Prototype int PXIT336GetRxConfig(ViSession inst, BertRxConfig *rxConfig);

inst VISA handle of the BERT module from which to read the configuration.

rxConfig Pointer to an instance of BertRxConfig where the current receive configuration will be stored.

PXIT336ConfigureTx

Configures the transmitter, including the bit rate, generated data pattern, and signal amplitude settings.

Function Prototype int PXIT336ConfigureTx(ViSession inst, BertTxConfig *txConfig);

inst VISA handle of the BERT module to configure.

txConfig Pointer to an instance of BertTxConfig containing the new transmitter configuration to be applied.

PXIT336ConfigureTxExt

Configure the transmitter. This function extends the PXIT336ConfigureTx capabilities to support external clock sourcing. For new programs use this API for all transmitter configurations.

Function Prototype int PXIT336ConfigureTxExt(ViSession inst, BertTxConfigExt *txConfigExt);

inst VISA handle of the BERT module to configure.

txConfigExt Pointer to an instance of BertTxConfigExt containing the new transmit configuration to be applied.

PXIT336SetTxAmplitude

Specifies the amplitude of the pattern generator output. The amplitude is specified in millivolts.

Function Prototype int PXIT336SetTxAmplitude(ViSession inst, unsigned long amplitude);

inst VISA handle of the BERT module to configure.

amplitude Output signal amplitude specified in millivolts.

PXIT336InsertError

Inserts errors into the transmitted bit stream. NoErrors, SingleBurst, and Continuous Rate of errors may be specified. Constant bit error rate injection will remain on until this function is called again with a different setting, or the transmitter is reinitialized. For single burst errors, each call to this function will insert the specified number of errors. To turn off continuous bit error rate injections, call this function with the style NoErrors.

Function Prototype int PXIT336InsertError(ViSession inst , BertErrorInsert *pErrorInsert);

inst VISA handle of the BERT module to configure.

pErrorInsert Pointer to an instance of BertErrorInsert structure describing the error insertion to be applied.

PXIT336GetErrorInsertCfg

Retrieves the current error insertion configuration.

Function Prototype int PXIT336GetErrorInsertCfg(ViSession inst, BertErrorInsert *pErrorInsert);

inst VISA handle of the BERT module to configure.

pErrorInsert Pointer to an instance of a BertErrorInsert structure where the current error insertion configuration will be stored when this function returns.

PXIT336ConfigureRx

Configures the receive block.

Function Prototype int PXIT336ConfigureRx(ViSession inst, BertRxConfig *rxConfig);

inst VISA handle of the BERT module to configure.

rxConfig Pointer to an instance of BertRxConfig containing the receive configuration to be applied.

PXIT336SetRxPhaseDelay

Specifies the receive phase delay.

Function Prototype int PXIT336SetRxPhaseDelay(ViSession inst, double dPhaseDelay);

inst VISA handle of the BERT module to configure.

dPhaseDelay Specifies the phase delay in picoseconds to be applied.

PXIT336SetClockTrigger

Selects the configuration of the Clock Trigger output. The Clock Trigger output is a divided version of the transmit clock. It can be a divide by 8 or divide by 9 version of the transmit clock. If RefClk is selected, the output is the internal PLL frequency, which is specified by the following table.

Table 4-2. Clock Trigger Output Frequency Options

Internal Bit Rate Clock (Gbps)	Div 8 (MHz)	Div 9 (MHz)	RefClk (MHz)
1.0625	132.81	118.06	106.25
1.25	156.25	138.89	156.25
2.125	265.63	236.11	106.25
2.48832	311.04	276.48	155.52
2.50	312.50	277.78	156.25
4.25	531.25	472.22	106.25
5.00	625.00	555.56	156.25

Function Prototype int PXIT336SetClockTrigger(ViSession inst, BertClockDivTrigger clkTrigCfg);

inst VISA handle of the N2101A module to configure.

clkTrigCfg Specifies the divider ratio for the Clock Trigger output. This ratio can be div8, div9, or RefClk.

PXIT336GetClockTrigger

Retrieves the current configuration of the clock trigger output.

Function Prototype int PXIT336GetClockTrigger(ViSession inst, BertClockDivTrigger *clkTrigCfg);

inst VISA handle of the BERT module to configure.

clkTrigCfg Pointer to an instance of BertClockDivTrigger where the clock trigger configuration will be stored when this function returns.

PXIT336SetPatternTrigger

Selects the configuration of the Pattern trigger output. The PAT-TRIG-OUT on the PX2000-336 module can output a single bit pulse synchrized with the data pattern. The Pattern Trigger output can also be set to the bit rate clock divided by 128.

Function Prototype int PXIT336SetPatternTrigger(ViSession inst, BertPatternTrigger patTrigCfg);

Inst VISA handle of the BERT module to configure.

patTrigCfg Specifies the type of triggering desired. Value of 0 indicates that the Pattern Trigger output will pulse at the beginning of each repetition of the transmit pattern. The Pattern Trigger's pulse rate is then dependant on the length of the PRBS or user pattern generated on the transmit output. Value of 1 will generate a divide by 128 version of the clock.

PXIT336GetPatternTrigger

Retrieves the current configuration of the pattern trigger output.

Function Prototype int PXIT36GetPatternTrigger(ViSession inst, BertPatternTrigger *patTrigCfg);

inst VISA handle of the BERT module to configure.

patTrigCfg Pointer to a BertPatternTrigger instance where the configuration of the pattern trigger output will be stored when this function returns.

PXIT336StartMeasure

Starts a measurement. The measurement type is defined by the measurement descriptor passed as second parameter.

Function Prototype int PXIT336StartMeasure(ViSession inst, BertMeasureType bertMeasureType, BertMeasure *pMeasureParam);

inst VISA handle of the BERT module to be used.

bertMeasureType Type of measurement to be executed

pMeasureParam Pointer to an instance of BertMeasureParam. Points the structure describing the measurement to be carried out.

PXIT336GetBitErrorCount

Reads the current bit error count. It stores the number of bit errors and the number of compared bits in the structure pointed to by the second parameter. This function is typically used to get the result of a measurement initiated by a call to PXIT336StartMeasure.

Function Prototype int PXIT336GetBitErrorCount(ViSession inst, BertBitErrorCount *errorCount);

inst VISA handle of the BERT module to read the bit error count from.

errorCount Pointer to an instance of the bit error count. Specifies where the bit error count is to be stored.

PXIT336SetJitterFrequency

Sets the frequency of the inserted jitter. Not calibrated.

Function Prototype int PXIT336SetJitterFrequency(ViSession inst, int frequency);

inst VISA handle of the BERT module to configure.

frequency Frequency in Hz of the inserted Jitter. Uncalibrated!

PXIT336GetJitterFrequency

Gets the frequency of the inserted jitter.

Function Prototype int PXIT336GetJitterFrequency(ViSession inst, int *frequency);

inst VISA handle of the BERT module to configure.

frequency Pointer which specifies where the inserted jitter Frequency value is to be stored.

PXIT336SetJitterAmplitude

Sets the amplitude of the inserted jitter. This is feature is not supported by PXIT and is provided "As Is".The percentage of the eye width that is affected by the inserted DJ is uncalibrated. Units are in DAC steps (255 max).

Function Prototype int PXIT336SetJitterAmplitude(ViSession inst, int amplitude);

inst VISA handle of the BERT module to configure.

amplitude Amplitude (in DAC units 0 to 255) of the inserted Jitter. This is uncalibrated.

PXIT336GetJitterAmplitude

Gets the current amplitude setting of the inserted jitter.

Function Prototype int PXIT336GetJitterAmplitude(ViSession inst, int *amplitude);

inst VISA handle of the BERT module to configure.

amplitude Pointer which specifies where the inserted jitter amplitude (DAC units) is to be stored.

Data Types and Structures

This section specifies the data structures used in the API. For the latest information, refer to the include file pxit336.h provided in the release package.

BertBitRate

This enumeration defines the different bit rates supported.

```
typedef enum
{
    _1dot0625_Gbps = 0,
    _1dot25_Gbps,
    _2dot125_Gbps,
    _2dot48832_Gbps,
    _2dot66_Gbps,
    _4dot25_Gbps,
    _2dot500_Gbps,
    _5dot0_Gbps
} BertBitRate;
```

BertPattern

This data structure is used to individually describe transmit and receive patterns. All parameters to define transmit and receive patterns are included in this structure. This structure is typically used to specify the traffic generator and receiver's configuration.

```
typedef struct
{
     BertPatternType     Type;
     Union
     {
     BertPrbsLength  PrbsLength;
     BertDcPattern   DcPattern;
     BertUserPattern UserPattern;
     };
} BertPattern;
```

BertPatternType

This enumeration defines the three possible pattern types supported by the BERT.

```
typedef enum
{
```

```
    PRBS,
    DC,
    User
} BertPatternType;
```

BertPrbsLength

This enumeration defines the PRBS pattern lengths supported.

```
typedef enum
{
    PRBS_7,
    PRBS_9,
    PRBS_11,
    PRBS_15,
    PRBS_23,
    PRBS_31
} BertPrbsLength;
```

BertDcPattern

This enumeration defines the DC patterns supported.

```
typedef enum
{
    K28dot5 = 0,
    K28dot7,
    CRPAT,
    GBETESTPAT
} BertDcPattern;
```

BertUserPattern

This structure is used to define a user pattern. It is used to specify the buffer in which the user pattern is stored and the bit length of that pattern.

```
typedef struct
{
    unsigned long * BitPattern;
    int BitSize;
} BertUserPattern;
```

BertTxConfig

This data structure describes the traffic generator's configuration. It is supported for legacy applications, and is superseded by BertTxConfigExt() below.

```
typedef struct
{
    BertBitRate         BitRate;
    BertPattern         Pattern;
    unsigned long       Amplitude;
    bool                DataInvert;
    bool               UseExternalClkSrc; //internal only (=0)
} PXIT336TxConfig;
```

BertTxConfigExt

This is an extended version of BertTxConfig structure defined above that includes the external clock reference frequency field. It is used as a parameter to the PXIT336ConfigureTxExt() and PXIT336GetTxConfigExt functions and is the recommended method for transmitter configuration in either clocking mode.

```
typedef struct
{
    BertBitRate         BitRate;
    float               fBitRate;
    BertPattern         Pattern;
    unsigned long       Amplitude;
    bool                DataInvert;
    PXITBERT_CLKSRCS    ClkSrc;
    float               ClkRef;
    float               ExtLSBitrates[PXITBERT_NLSREFS];
} BertTxConfigExt;
```

BitRate	Transmitter bit rate.
fBitRate	Transmitter bit rate in MHz
Pattern	Describes the transmitted data pattern. See through above for further details
Amplitude	Output signal amplitude specified in millivolts.
DataInvert	Data pattern signal polarity. TRUE causes the output data pattern to be inverted.
ClkSrc	Selects the clock source. 0 is internal, 1 is external. Other values are reserved for future use.
ClkRef	Specifies the data bit rate (in Mb/s) derived from the external clock input. In exact clocking mode, this is equal to the external reference clock input. In double clocking mode, this will be one-half of the supplied clock (e.g. 2720.0 would required a 5.44 GHz external clock input).

BertRxConfig

This structure is used to fully describe the receiver configuration. This data structure is typically used as a parameter for PXIT336ConfigureRx and PXIT336GetRxConfig calls.

```
typedef struct
{
    BertBitRate         BitRate;
    BertPattern         Pattern;
    bool                DataInvert;
```

```
    double              PhaseDelay;
    bool                UseExternalClkSrc;
} PXIT336RxConfig;
```

BitRate	Received signal expected bit rate. See BertBitRate for possible values.
Pattern	Describes the expected receive pattern. See structures above for further details on how to specify a pattern.
DataInvert	Received signal invert. Setting this to TRUE causes the received signal to be inverted before processing.
PhaseDelay	Phase delay in picoseconds.
UseExternalClockSource	Selects the clock source. Must be set to FALSE for internal clocking.

BertErrorInsert

This structure is used to fully decribe errors to be injected into the traffic generator's output.

```
typedef struct
{
    BertErrorInsertStyle    Style;
    BertErrorRate           Rate;
    BertErrorCount          Count;
} BertErrorInsert;
```

BertErrorInjectStyle

This enumeration defines the style of error injection supported by the BERT.

```
typedef enum
{
    NoErrors,
    SingleBurst,
    Continuous
} BertErrorInsertStyle;
```

BertErrorRate

This type is used to specify the rate at which errors will be injected into the traffic generator's output when the BertErrorInjectStyle is set to Continuous.

```
typedef enum
{
    TenMinus3 = 0,
    TenMinus4,
    TenMinus5,
    TenMinus6,
    TenMinus7,
```

```
    TenMinus8,
    TenMinus9,
    TenMinus10
} BertErrorRate;
```

BertErrorCount

Specify the number of errors to inject during each burst when BertErrorInjectStyle is SingleBurst.

```
typedef unsigned long BertErrorCount;
```

BertMeasureType

This structure is used to specify the type of measurement you wish the BERT to carry out.

```
typedef enum
{
    None = 0,
    ClockDataAlign,
    FullSweep,
    TimeWindow,
    Bathtub,
    FastBathtub,
    EyeOpening,
} BertMeasureType;
```

BertMeasureParam

This structure is used in calling PXIT336StartMeasure and is used to store configuration information specific to the type of measurement requested.

```
typedef union
{
    BertWindowParam         WindowParam;
    BertBathtubParam        BathtubParam;
    BertFullSweepParam      FullSweepParam;
    BertClockDataAlignParam ClockDataAlignParam;
    BertEyeOpeningParam     EyeOpeningParam;
    BertPonParam            PonParam;
} BertMeasureParam;
```

BertWindowParam

This structure is used to specify the parameters of a Window BER measurement.

```
typedef struct
{    unsigned long WindowLength;
    double * BER;
    unsigned long MaxNbErrors;
    unsigned long NbOfIterations;
    void (*StepCallback)(void);
    void (*CompletionCallback)(void);
} BertWindowParam;
```

WindowLength	Maximum length - in microseconds of the time window. The maximum window length is 221 seconds. The length of time spent measuring the BER can be shorter than specified by this parameter if a non-zero value is specified for the "MaxNbError" parameter below. A value of zero specifies an infinite window length. Step and completion callbacks are never called if this parameter is set to 0 since the time window will never expire.
BER	Pointer to a type double were the bit error rate for the last elapsed time window will be stored. It is the caller's responsibility to make sure that this points to a valid memory location allocated by the caller. Can be set to NULL.
MaxNbErrors	Maximum number of bit errors after which the time window will elapse regardless of the time length specified by "WindowLength". This allows shortening the time window in instances where there are a large number of bit errors. Can be set to 0 to leave the time window to run to its maximum length regardless of the number of bit errors.
NbOfIterations	Number of time window BER measures to perform. Can be set to 0 to specify that an infinite number of time window BER measures must be performed in succession.
StepCallback	Pointer to a callback function specified by the caller. The function specified by this pointer will be called every time a time window has elapsed. It can be used to indicate that the variable pointed to by "BER" has been updated. It is the caller's responsibility to ensure that this points to a valid callback function. Can be set to NULL to indicate that the caller does not wish to use a callback function.
CompletionCallback	Pointer to a callback function specified by the caller. The function identified by this pointer will be called when the number of time window iterations specified by "NbOfIterations" has been reached.

BertBathtubParam

This structure is used to specify a Bathtub measurement.

```
typedef struct
{
    BertBERMeasurePoint * pBerTable;
    unsigned long * pNbOfPoints;
    unsigned long PhaseResolution;
    unsigned long WindowLength;
    int MaxNbErrorPerPoint;
    void (*StepCallback)(void);
    void (*CompletionCallback)(void);
```

```
} BertBathtubParam;
```

PBerTable	Pointer to an array of type BertBERMeasurePoint structures where the Bathtub plot points will be stored. It is the caller's responsibility to allocate enough memory for the plot to be stored.
PNbOfPoints	Pointer to a variable containing the number of points making up the Bathtub plot. This variable must contain the size of the array of type BertBERMeasurePoint allocated by the caller and pointed to by pBerTable on calling PXIT336StartMeasure(). This size must be given in number of BertBERMeasurePoint instances that can be stored into the memory pointed to by pBerTable. The variable pointed to by pNbOfPoints is updated before each call to the callback functions (StepCallback and CompletionCallback) to reflect the actual number of points making up the Bathtub plot.
PhaseResolution	Phase resolution in picoseconds of the Bathtub diagram. Specifies the phase difference between each successive measurement points. This is equivalent to the resolution on the X-axis of the Waterfall diagram.
WindowLength	Maximum length in microseconds spent counting errors for each point of the Bathtub plot. The maximum window length is 221 seconds. The length of time spent counting errors can be shorter than specified by this parameter if a non-zero value is specified for the 'MaxNbErrorPerPoint' parameter below.
MaxNbErrorPerPoint	Maximum number of bit errors after which the time spent counting errors at a specific phase delay will elapse regardless of the time length specified by 'WindowLength'. This shortens the time it takes to plot a Bathtub measure by reducing the amount of time spent counting errors for phase delays where there is a high number of errors. Can be set to 0 to leave the time window to run to its maximum length regardless of the number of bit errors.
StepCallback	Pointer to a callback function specified by the caller. The function identified by this pointer will be called every time a new point has been added to the Bathtub plot. It is the caller's responsibility to ensure that this points to a valid callback function. Can be set to NULL to indicate that the caller does not wish to use a callback function.
CompletionCallback	Pointer to a callback function specified by the caller. The function identified by this pointer will be called when the Bathtub plot has been completed. It is the caller's responsibility to ensure that this points to a valid callback function. Can be set to NULL to indicate that the caller does not wish to use a callback function.

BertEyeOpeningParam

This structure is used to specify the parameters of an eye opening measurement. The bathtubParam is defined above, and the targetBER is the desired bit error rate. The result in picoseconds, is returned in *pEyeOpening.

```
typedef struct
{
BertBathtubParam BathtubParam;
   double TargetBER;
   double * pEyeOpening;
} BertEyeOpeningParam;
```

BertBitErrorCount

This structure is used to report a bit error count. The first item ErrorCount contains the number of bit errors. The second item BitCount reports the number of bits over which the number of errors occurred. The bit error rate (BER) can be derived from these two items by dividing ErrorCount with BitCount. This structure is updated in real-time

```
typedef struct
{
    unsigned __int64 ErrorCount;
    unsigned __int64 BitCount;
} BertBitErrorCount;
```

BertMeasureStatus

```
typedef enum
{
    Success,
    Fail,
    Invalid,
    InProgress
} BertMeasureStatus;
```

BertStatus

```
typedef struct
{
    BertMeasureType     Requested;
    BertMeasureStatus   Status;
    bool                TxClockLocked;
    bool                RxClockLocked;
    bool                RxSynched;
} BertStatus;
```

Specifications

Specifications

The distinction between specifications and characteristics is described as follows:

- Specifications describe warranted performance over the temperature range 0° C to +40° C and relative humidity <95% (unless otherwise noted). All specifications apply after the temperature of the probe and the probe adapter has been stabilized after 30 minutes of continuous operation.
- *Characteristics* provide useful information by giving functional, but nonwarranted, performance parameters. *Characteristics are printed in italics.*

Table 5-1. Transmit (Tx) Specifications

Output Jitter	1 ps rms
Rise/Fall Time (20-80%)	30 ps
Frequency Modulation (jitter insertion)	0 to 10 MHz
Output Range	250 mV to 1.6V
Output Resolution	0.5 mV
Internal Clock Frequencies	1.0625 GHz, 1.25 GHz, 2.125 GHz, 2.48832 GHz, 2.50 GHz, 4.25 GHz, 5.00 GHz
External Clock Operation Range. Exact Rate Clocking.[a] [b]	1.0 GHz ≤ f ≤2.5 GHz 4.0 GHz ≤ f ≤6.6 GHz
External Clock Operation Range. Double Rate Clocking.[c]	2.5 GHz ≤f/2 ≤3.3 GHz
Bit Error Insertion	Single and multiple bit bursts, Continuous BER of 10^{-n} (n = 3,4,5,6,7,8,9,10)

a. f = Input frequency
b. Clock range 2.5 GHz < f < 4.0 GHz is not supported.

c. Supply 2x of this rate. f = Input frequency

Table 5-2. Receive (Rx) Specifications

Input Range	25 mV to 2V
Input Sensitivity	25 mV
Internal Clock Frequencies	1.0625 GHz, 1.25 GHz, 2.125 GHz, 2.48832 GHz, 2.50 GHz, 4.25 GHz, 5.00 GHz
External Rate Clock Operation Range[a b c]	1.0 GHz ≤ f ≤1.5Hz 2.0 GHz ≤ f ≤2.5 GHz 4.0 GHz ≤ f ≤6.0 GHz
Delay range with External Clock	200 ps

a. f = Tx external input
b. The Rx Clocking must be set to Internal, and the receive path will track the Tx frequency (internal or external).
c. Clock ranges 1.5 GHz < f < 2.0 GHz and 2.5 GHz < f < 4.0 GHz are not supported.

Table 5-3. Trigger Output Specifications

Trig Clock	Bit clock rate divided by 8, 9, or the internal PLL clock (refer to Table 5-4).
Pattern Trigger	Synchronized to data pattern start, or the bit clock divided by 128.
Output Amplitude DC Coupled	500 mV

Table 5-4. Clock Trigger Output Frequency Options for Trig Clock Specifications

Internal Bit Rate Clock (Gbps)	Div 8 (MHz)	Div 9 (MHz)	RefClk (MHz)
1.0625	132.81	118.06	106.25
1.25	156.25	138.89	156.25
2.125	265.63	236.11	106.25
2.48832	311.04	276.48	155.52
2.50	312.50	277.78	156.25
4.25	531.25	472.22	106.25
5.00	625.00	555.56	156.25

Table 5-5. Environmental Specifications

Use	indoor
Dimensions	Three-slot PXI module