N2101B
PXIT Bit Error Ratio Tester Module

User's Guide

Agilent Technologies

Manual Part Number
N2101-90002

Edition
October 2007
Printed in USA

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

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.

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

ISM1-A This text denotes the instrument is an Industrial Scientific and Medical Group 1 Class A product.

ICES/NMB-001 This is a marking to indicate product compliance with the Canadian Interference-Causing Equipment Standard.

This product complies with the WEEE Directive (2002/96/EC) marking requirements. The affixed label indicates that you must not discard this electrical/electronic product in domestic household waste. Product Category: With reference to the equipment types in the WEEE Directive Annex I, this product is classed as a "Monitoring and Control instrumentation" product. Do not dispose in domestic household. To return unwanted products, contact your local Agilent office, or refer to www.agilent.com for more information.

N10149 The C-Tick mark is a registered trademark of the Australian spectrum Management Agency.

This symbol indicates the Environmental Protection Use Period (EPUP) for the product's toxic substances for the China RoHS requirements.

Contents

1

General Information

Introduction

The N2101B BERT implements a full function Bit Error Ratio testing system in a triple-wide 3U PXI module. It consists of a high accuracy clock source, data pattern generator and error detector.

Main Features

The N2101B provides the following features:

- Differential data generation and analysis
- Standard data patterns PRBS 2^n-1, n = 7, 9, 11, 15, 23, 31; K28.5, K28.7, CRPAT
- User specified patterns up to 128 kilobits (128 x 1024); typical maximum user length is 2048 bits. Refer to "To use your own patterns" on page 3-7 for more information.
- Internal clock rates for the following time bases: GBE, Fiber Channel, Infiniband, and SONET for rates up to 8.5 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 with external clock input: 155 Mb/s to 10.3125Gb/s.
- Bit error measurements with internal clock: 155.52 Mb/s, 622.08 Mb/s, 1.0625 Gb/s, 1.25 Gb/s, 2.125 Gb/s, 2.48832 Gb/s, 2.50 Gb/s, 3.125 Gb/s, 4.25 Gb/s, 5.00 Gb/s, 6.25 Gb/s, and 8.50 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
- Eye opening measurement at defined error rates
- Display of bathtub curve in standard and fast modes

External Clock

The N2101B can operate in external mode from 500 Mb/s to 10.3125Gb/s.

- The External re-timed option should be used from 500 Mb/s to 8.5 Gb/s.
- The External Direct should be used from 8.5 Gb/s to 10.3125Gb/s.

Since the clock signal has a minimum operational frequency of 500 MHz, the N2101B can operate down to 155 Mb/s by using a clock frequency of four times the data bit rate. This is called Quad Rate Clock Mode and is automatically entered when you select any bit rate below 500 Mb/s, whether the clock signal is generated internally or through the external inputs. Quad Rate Clock Mode functions by maintaining each data bit for four consecutive clock cycles before transitioning to the next data bit. When selecting an externally generated clock with a desired bit rate of less than 500 Mb/s, the Configuration dialog field below that input will display "Quad Rate Clock Mode," indicating that the you must supply a clock signal at quadruple the bit rate entered to the appropriate clock input(s). For example, a bit rate of 155.52 Mb/s requires a 622.08 MHz clock signal to the Tx Clk In connector.

NOTE As a protection mechanism for the N2101B, if an external signal is input below 8.5 Gb/s, the instrument automatically sets itself to external re-timed.

NOTE **N2101A.** For the N2101A, bit rates greater than 2.5 Gb/s up to 3.3 Gb/s use a “double clocking mode” for the external clock. Double clocking requires a synchronous clock input with a frequency of twice the desired bit rate. For example, for a 2.702 Gb/s bit rate, you must enter 2702 into the External Rate field of the Configuration Panel and connect a 5.404 GHz clock signal to the Tx Clk In input. The Configuration panel shows 2700.000 (Mbps) “double Rate clock Mode” and the Control Panel will then display 5404.00 MHz ‘Double Rate Mode.’

Clock Trigger

When the transmitter is internally clocked, the clock trigger output (Clk Trig Out connector) can be set to the clock rate divided by 1, 2, 4, or 8 as shown in Table 1-1. (N2101A modules can be set to clock divide rates of 1, 2, 4, 8, or 9.) When externally clocked, these values are derived from the rate applied to the Tx Clk In connector.

Table 1-1. Clock Trigger Output Frequency Options[a]

Internal Bit Rate Clock	Div 2	Div 4	Div 8	Div 1
155.52 Mb/s	311.04 MHz	155.52 MHz	77.76 MHz	622.08 MHz
622.08 Mb/s	311.04 MHz	155.52 MHz	77.76 MHz	622.08 MHz
1.0625 Gb/s	531 MHz	265 MHz	132.81 MHz	1.0625 GHz
1.25 Gb/s	625 MHz	312 MHz	156.25 MHz	1.250 GHz
2.125 Gb/s	1.06 GHz	531 MHz	265.63 MHz	2.125 GHz
2.48832 Gb/s	1.24 GHz	622 MHz	311.04 MHz	2.48832 GHz
2.50 Gb/s	1.25 GHz	625 MHz	312.50 MHz	2.50 GHz
3.125 Gb/s	1.56 GHz	780 MHz	390.62 MHz	3.125 GHz
4.25 Gb/s	2.13 GHz	1.06 GHz	531.25 MHz	4.25 GHz
5.00 Gb/s	2.5 GHz	1.25 GHz	625.00 MHz	5.0 GHz
6.25 Gb/s	3.125 GHz	1.56 GHz	781.25 MHz	6.25 GHz
8.50 Gb/s	4.25 GHz	2.13 GHz	1.063 GHz	8.50 GHz

a. This table is only accurate for the N2101B. The N2101A provides Div 8, Div 9, and Ref Clk (106.25 MHz, 155.52 MHz, and 156.25 MHz). The only bit rates valid for the N2101A: 1.0625 Gb/s, 1.25 Gb/s, 2.125 Gb/s, 2.48832 Gb/s, 2.50 Gb/s, 4.25 Gb/s, and 5.00 Gb/s.

Trig In Connector

The front-panel Trig In connector is reserved for future use and is not active.

General Safety Considerations

This product has been designed and tested in accordance with IEC Publication 1010, Safety Requirements for Electronic Measuring Apparatus, and has been supplied in a safe condition. The instruction documentation contains information and warnings which must be followed by the user to ensure safe operation and to maintain the product in a safe condition.

Install the plug-in module according to the enclosure protection provided and placing filler panels in empty slots. This instrument does not protect against the ingress of water. This instrument protects against finger access to hazardous parts within the enclosure.

WARNING **Laser Safety Notice. The N2101B is used to control optical transceivers. 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.**

WARNING **If this product is not used as specified, the protection provided by the equipment could be impaired. This product must be used in a normal condition (in which all means for protection are intact) only.**

WARNING **No operator serviceable parts inside. Refer servicing to qualified service personnel. To prevent electrical shock do not remove covers.**

WARNING **Use a dry cloth or one slightly dampened with water to clean the external case parts. Do not attempt to clean internally.**

CAUTION This product is designed for use in Installation Category II and Pollution Degree 2 per IEC 1010 and 664 respectively.

CAUTION Electrical channel input circuits 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.

CAUTION The N2101B 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 **Use appropriate caution when using Agilent products for testing lasers.**

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. Purchase acceptable ESD accessories from your local supplier.

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

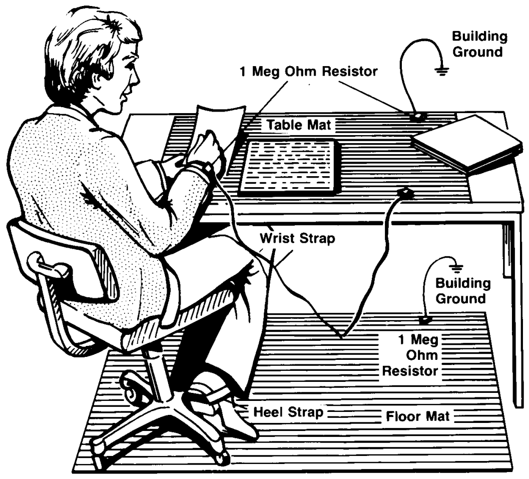


Figure 1-1. Static-safe Work Station

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.

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.**

Connector Care

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

Electrical Connectors

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

Returning the N2101B 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 contact your local Agilent Call Center.

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:
   - Insert the product in an anti-static bag.
   - 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 the embedded PXI controller or an external PC using a PCI - cPCI/PXI remote bridge (such as the NI MXI-4 product).

NOTE You can control the BERT with Agilent Vee only through ActiveX.

If an external PC is used, the PC must meet the following specifications:

- Windows 2000 or XP operating system
- 1 GB RAM
- Pentium, 133 MHz or greater
- NI-VISA

NOTE When using an external PC as the controller, you must follow the installation steps in the sequence described for the PC BIOS to locate the instruments in the PXI chassis.

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 US, call 1 (800) 829-4444. In other regions, visit http://www.agilent.com/find/assist. Before returning an instrument for service, you must first contact your local Agilent Call Center.
3. Verify that the following environmental conditions exist before proceeding with the installation procedure.

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 N2101B provides adequate earth grounding.**

WARNING **Ensure that the air supply to the chassis is working correctly. The N2101B requires an optimal air flow within the chassis. It is recommended that you regularly change filters on the PXI Chassis.**

Step 2. Install the Instrument Driver Software

NOTE This procedure assumes that NI-VISA has already been installed.

1. Log onto the PC with administrator privileges so that you can install the software.
2. Go to the Agilent website: www.agilent.com/find/pxit.
3. Click on the Technical Support link and then the Drivers link.
4. Download the latest version of the following driver:

   `Agilent N2101B BERT Driver`

5. Once the download is complete, run the N2101B installation file.
   a. During the installation, enter the user name and organization.
   b. Select the **all users** option to ensure the software is available to all users of the PC. Click **Next**.
6. When the installation process is finished, click **Finish**. The N2101B control software is now installed.
7. To ensure that the PC BIOS will be able to locate the instruments in the PXI chassis, follow this step according to the controller that you are using:
   - If you are using an external PC and remote bridge, turn off the PXI chassis and PC power.
   - If you are using an embedded controller, turn off the PXI chassis and PC power and remove all N2101B modules from the chassis.

NOTE You can control the BERT with Agilent Vee only through ActiveX.

NOTE As part of the installation, the N2101B User’s Guide is made available on the Windows Start menu.

Step 3. Install the N2101B

1. With the PC and chassis powered off, install the N2101B 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 the software is correctly installed, you will see an indication that new hardware is detected and that the system is attempting to locate the associated software driver. When this process is complete, a notification appears indicating that the hardware is ready for use.

   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 N2101B instrument.

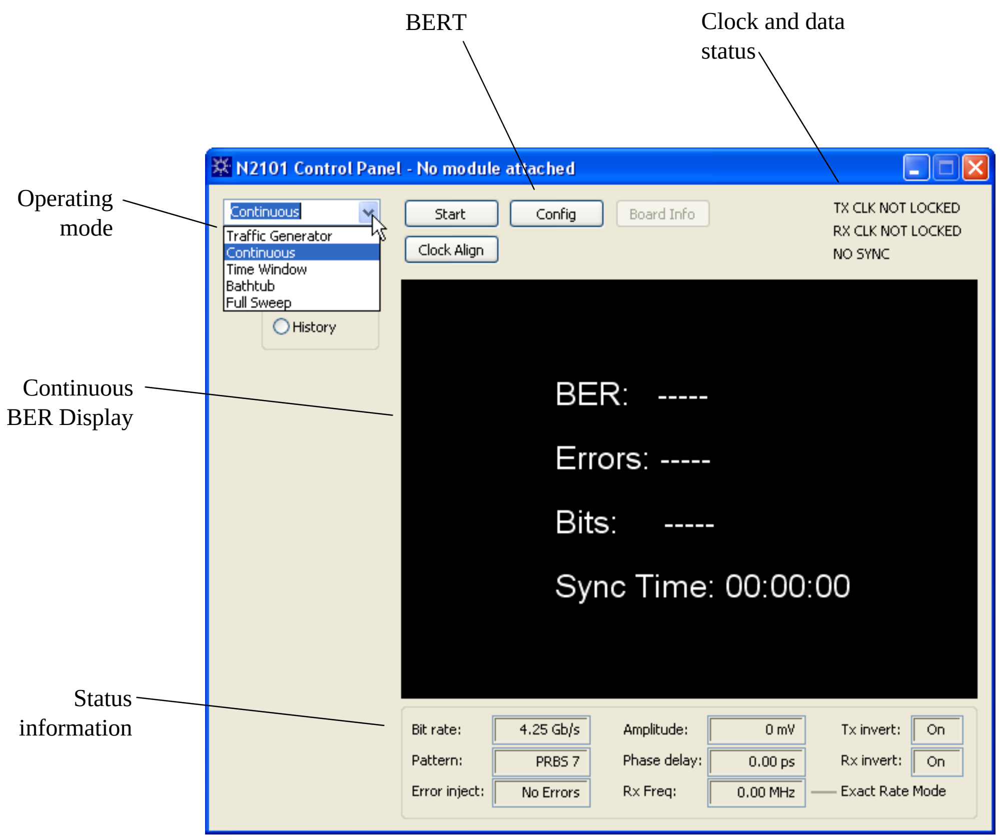
NOTE The N2101B User’s Guide is provided as a PDF file. To locate and view the user’s guide, go to the Windows **Start** menu and select **All Programs > Agilent > N2101B > Documents > User’s Guide**.

3

Using the Control Panel

Introduction

This chapter describes how to start and configure the Windows Control Panel application. All aspects of the N2101B can be controlled through the Control Panel. Use the top row of buttons to configure the module, I/O, and to connect to the module. Use the list box in the top-left corner of the main form to control the N2101B mode of operation.



TX CLK NOT LOCKED
RX CLK NOT LOCKED
NO SYNC

BER: -----
Errors: -----
Bits: -----
Sync Time: 00:00:00

Bit rate: 4.25 Gb/s
Pattern: PRBS 7
Error inject: No Errors
Amplitude: 0 mV
Phase delay: 0.00 ps
Rx Freq: 0.00 MHz
Tx invert: On
Rx invert: On
Exact Rate Mode

Figure 3-1. Control Panel

To open the Control Panel

1 Click the Windows **Start** menu.

2 Click **All Programs** > **Agilent** > **N2101B**.

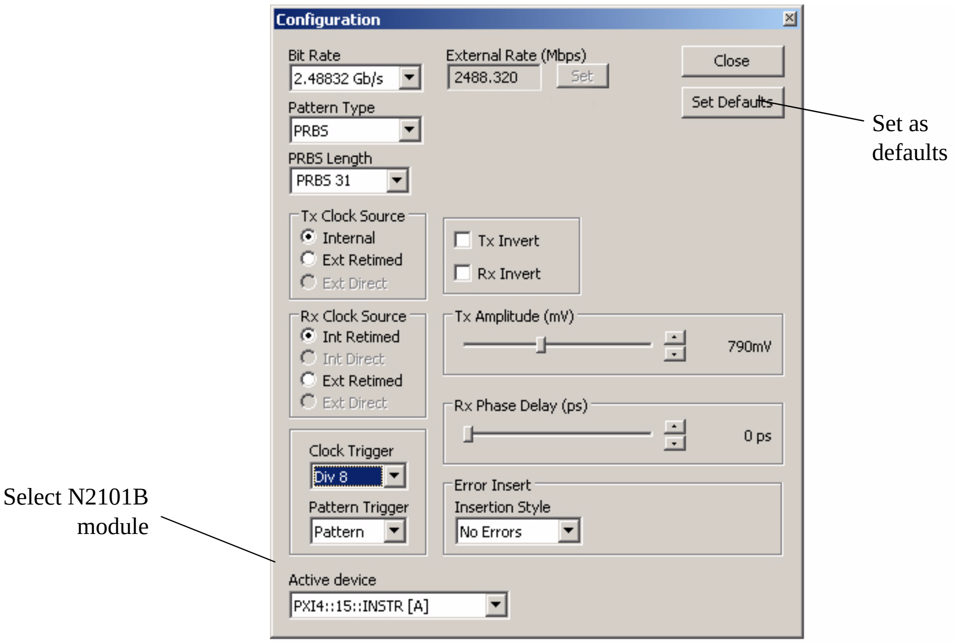
3 Click **N2101B Control Panel**.

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 (at the top of the control panel) to view the Configuration dialog box.



Bit Rate: 2.48832 Gb/s
External Rate (Mbps): 2488.320
Pattern Type: PRBS
PRBS Length: PRBS 31
Tx Clock Source: Internal (selected), Ext Retimed, Ext Direct
Rx Clock Source: Int Retimed (selected), Int Direct, Ext Retimed, Ext Direct
Tx Amplitude (mV): 790mV
Rx Phase Delay (ps): 0 ps
Clock Trigger: Div 8
Pattern Trigger: Pattern
Error Insert — Insertion Style: No Errors
Active device: PXI4::15::INSTR [A]

Figure 3-2. Configuration Dialog Box

3 Click the **Set Defaults** button to restore the N2101B to its factory default settings listed in Table 3-1.

Table 3-1. Default Settings

Parameter	Setting
Bit Rate	4.25 Gb/s
Pattern	PRBS
PRBS	2^7-1
Tx Clock Source	Internal
Rx Clock Source	Internal Retimed
Tx Invert	Transmit signal polarity is not inverted
Rx Invert	Receive signal polarity is not inverted
Tx Amplitude	250 mV
Phase Delay	0 ps
Error Insert	No Errors
Clock Trigger	Divide by 8
Pattern Trigger	Pattern

4 Select **Active device.** The software automatically selects the instrument and the Active device field will indicate the instrument description or address.

5 Connect a cable from the transmit to the receive side of the N2101B.

6 Select the desired settings from the following possibilities:

Bit Rate. 155.52 Mb/s, 622.08 Mb/s, 1.0625Gb/s, 1.25Gb/s, 2.125Gb/s, 2.48832Gb/s, 2.5Gb/s, 3.125 Gb/s, 4.25Gb/s (Default), 5.0Gb/s, 6.25 Gb/s, and 8.50 Gb/s.

Pattern. 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, and 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 set to internal (default), external retimed, or external direct. Internal uses the BERT internal oscillators to drive the PLL timing circuitry. External retimed uses a clock connected from an external source to the Tx Clk In to drive the PLL timing circuitry. External direct uses a clock connected from an external source to the BERT, bypassing the PLL timing circuitry and connecting it directly to the data control hardware. Operating the transmit clock source at 155.52 Mb/s will function in quadruple clocking mode in the N2101B. Note that an *N2101A BERT* may be required to operate in “Double Clocking Mode.” The external clock will have to be 2x the desired bit rate clock (for example, 5.4 GHz for 2.7 Gb/s rate).

Rx Clock Source. Can be set to internal retimed (default), internal direct, external retimed, or external direct. Internal retimed uses the currently selected transmit clock to drive the PLL timing circuitry with a phase detector to provide at least two bit periods of phase sweep at any rate (155.52 Mb/s to 8.5 Gb/s). Internal direct uses the currently selected transmit clock, bypassing the PLL and using a delay line to provide at least 200 ps of phase sweep for operation greater than 8.5 Gb/s. External retimed uses a clock connected from an external source to the Rx Clk In connector to the PLL timing circuitry with a phase detector to provide at least two bit periods worth of phase sweep at any rate (155.52 Mb/s to 8.5 Gb/s). External direct uses a clock connected from an external source to the Rx Clk In, bypassing the PLL and using a delay line to provide at least 200 ps of phase sweep for operation greater than 8.5 Gb/s.

External Rate. Specified the desired bit rate clock. This field is only active when either Tx or Rx is set to External.

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.)

Tx Amplitude (mV). Specifies the amplitude of transmit signal specified in millivolts. (default = 250 mV).

Phase Delay. This option specifies a phase delay between the signal and receiver clocks. The range is dependent upon the Rx Clock Source setting as described above. The increment is in 1-ps steps. When retimed (either internal or external) the phase sweep is greater than 2 bit periods at the current rate. When direct (either internal or external) there is at least 200 ps.

Error Insert. Allows you to inject errors into the data pattern: Single burst of K errors where you specify K, Continuous (Rates of 10^{-n} for n = 3 to 10), or None (Default).

Clock Trigger. The Clock Trigger is a divided down version of the transmit clock. You can set it to either of the following: Div1, Div2, Div4, Div8 (default), Refer to “Clock Trigger Output Frequency Options” on page 1-4.

Pattern Trigger. Specifies the pattern trigger: Pattern (generates a trigger pulse synchronous with the transmitted pattern: default), or 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.)

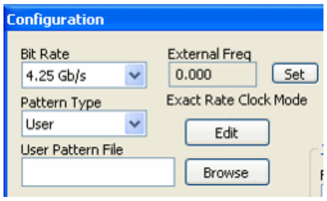
7 Click **Close** in the Configuration dialog box and **Start** in the Control Panel to begin the BER measurement. When the measurement is complete, the Control Panel displays the error rate.

To use your own patterns

You can use your own user-defined patterns with the N2101B. The control panel application includes a user pattern editor that allows you to save and import text pattern-files to and from the N2101B.

1 Open the Configuration dialog box

2 Select **User** from the **Pattern Type** list. The N2101B immediately begins generating and synchronizing to any specified pattern.


Configuration

Bit Rate: 4.25 Gb/s
External Freq: 0.000
Pattern Type: User
Exact Rate Clock Mode
User Pattern File:

Figure 3-3. Pattern Type Set to User

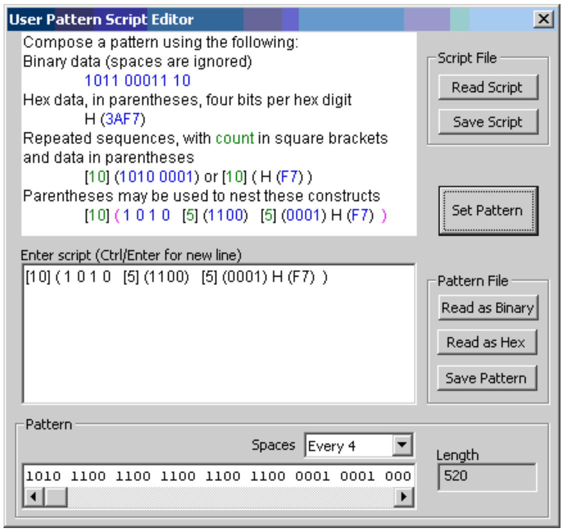
3 Click **Edit** to view the User Pattern Script Editor shown in Figure 3-4. The editor uses an abbreviated script syntax, which is explained in the editor.

4 Enter the syntax for the bit pattern in the Script field.

5 Click **Set Pattern** to verify the syntax and display the resulting pattern of bits in the Pattern field.

User-defined 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})



Compose a pattern using the following:
Binary data (spaces are ignored)
1011 00011 10
Hex data, in parentheses, four bits per hex digit
H (3AF7)
Repeated sequences, with count in square brackets and data in parentheses
[10] (1010 0001) or [10] (H (F7))
Parentheses may be used to nest these constructs
[10] (1 0 1 0 [5] (1100) [5] (0001) H (F7))

Enter script (Ctrl/Enter for new line)

```
[10] ( 1 0 1 0  [5] (1100)  [5] (0001) H (F7)  )
```

Spaces: Every 4

```
1010 1100 1100 1100 1100 1100 0001 0001 000
```

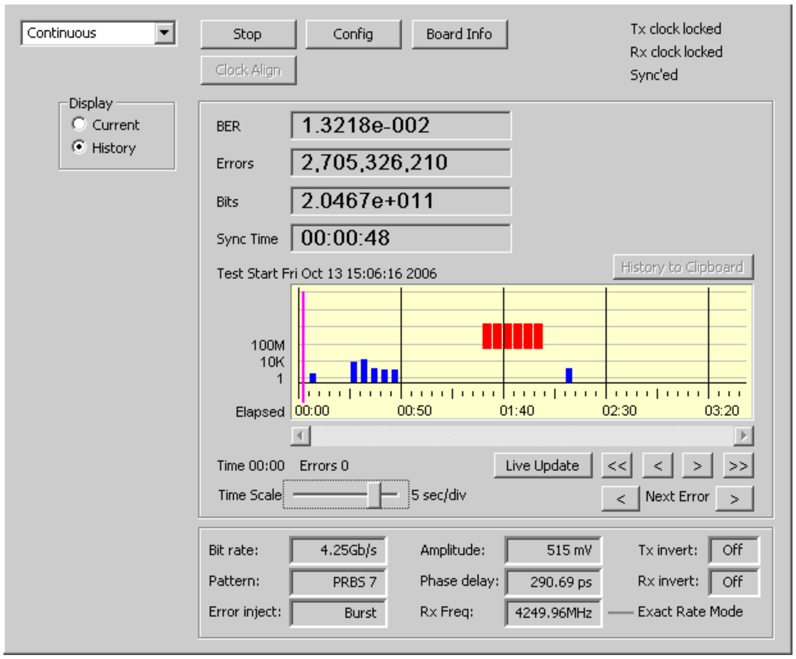
Length: 520

Figure 3-4. User Pattern Editor

To use Continuous test mode

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. In Continuous test mode, you can use the following controls:

- Select **Current** to display the continuous BER results, as shown in Figure 3-1.
- Select **History** to access a scrolling display of bit errors and out-of-sync conditions that is updated in real-time, as shown in Figure 3-5.
- Click the available controls to manually scroll the bit errors, or scale the time base of the display without interrupting the current test.
- Click **Live Update** to resume the automatic real-time scrolling.



Display: Current [✓ History]

BER 1.3218e-002
Errors 2,705,326,210
Bits 2.0467e+011
Sync Time 00:00:48
Test Start Fri Oct 13 15:06:16 2006

Time 00:00 Errors 0
Time Scale 5 sec/div

Bit rate:	4.25Gb/s	Amplitude:	515 mV	Tx invert:	Off
Pattern:	PRBS 7	Phase delay:	290.69 ps	Rx invert:	Off
Error inject:	Burst	Rx Freq:	4249.96MHz		Exact Rate Mode

Figure 3-5. Historical Display of Bit Errors

To save test data to an ASCII file

- 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 2007
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
```

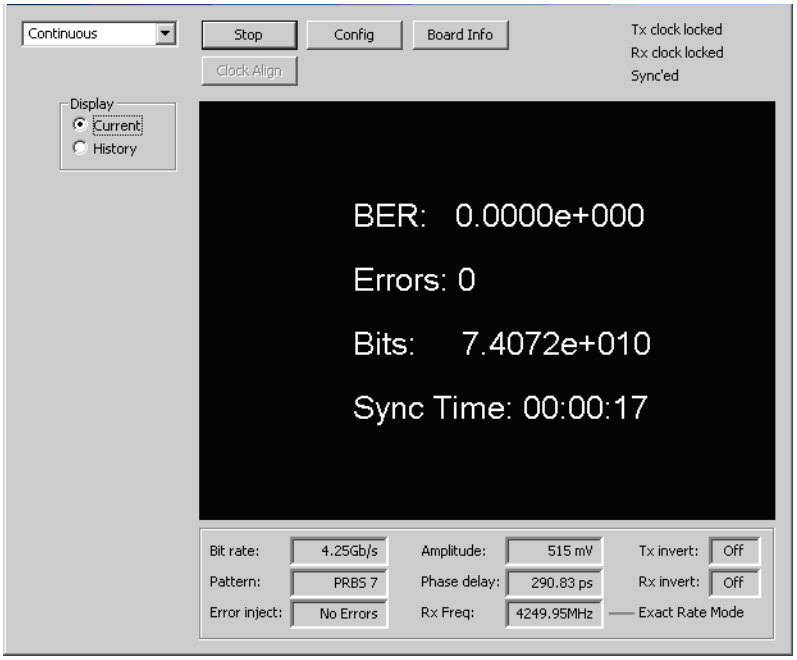
To power off the instrument

1. Close the control panel.
2. Power down the PC.
3. Switch off the power to the PXI chassis.

Quick Confidence Check

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

1. In the Control Panel, select **Time Window** or **Continuous** from the drop-down list.
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. Select **Internal** for the Tx Clock Source and **Int Retimed** for the Rx Clock Source.
5. Close the dialog box.
6. Install loopback cables between the transmitter and receiver connectors with either of the following methods:
   - Use both differential signal pairs.
   - Make a single Tx to Rx connection and terminate the unused connectors with a 50-ohm cap.
7. Click **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 changing.
8. Click **Start**. The form shown in Figure 3-6 on page 3-12 appears.
9. Check that no errors are being reported. The top-right of the Control Panel is a status display. There should be an indication that both the Tx clock and the Rx clock are locked. When the clock alignment is completed, there should be an indication that the state is Sync’d.



Tx clock locked
Rx clock locked
Sync'ed

BER: 0.0000e+000
Errors: 0
Bits: 7.4072e+010
Sync Time: 00:00:17

Bit rate: 4.25Gb/s
Pattern: PRBS 7
Error inject: No Errors
Amplitude: 515 mV
Phase delay: 290.83 ps
Rx Freq: 4249.95MHz
Tx invert: Off
Rx invert: Off
Exact Rate Mode

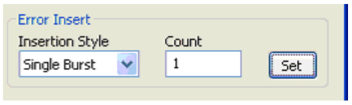
Figure 3-6. BER Screen with Zero Errors

10 Click the **Config** button.

11 Inject a single error as follows.

a Select single burst **Error Insert** from the configuration panel.

b Set the Count to **1**.



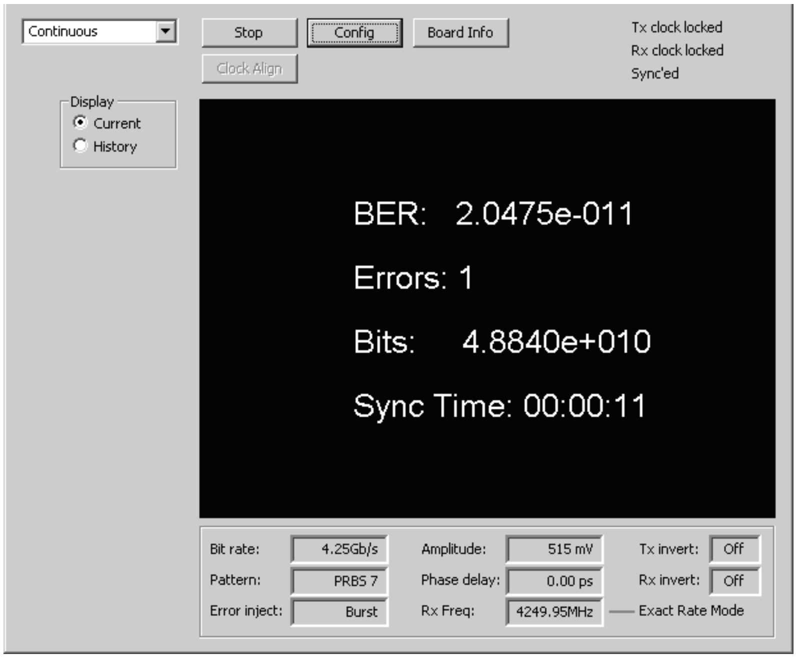
Insertion Style: Single Burst
Count: 1

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

12 Click **Set** once.

13 Close the dialog box.

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



BER: 2.0475e-011
Errors: 1
Bits: 4.8840e+010
Sync Time: 00:00:11

Bit rate: 4.25Gb/s
Pattern: PRBS 7
Error inject: Burst
Amplitude: 515 mV
Phase delay: 0.00 ps
Rx Freq: 4249.95MHz
Tx invert: Off
Rx invert: Off
Exact Rate Mode

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

When the BERT is in the Quad Rate Clock Mode, you will see the following indicators:

- The Control Panel shows "Quad Rate Mode" at the bottom right next to the Rx Frequency display.
- The Rx Frequency display shows the actual frequency of the 4x clock. (For example, a 155.52 Mb/s data rate will display an operating Rx clock frequency of 622.08 MHz.)
- For all data rates 500 Mb/s or greater, you will also see the following:
  - The identifier on the Configuration dialog shows "Exact Rate Clock Mode."
  - The identifier on the Control Panel shows "Exact Rate Mode."
  - The Rx frequency display matches the selected data rate (XXX Mb/s ≥ XXX MHz).

Upgrading the Instrument's Firmware

As part of the installation, the module's firmware is automatically updated; you do *not* need to perform the steps in this section. However, if you have an *N2101A* module, perform the following steps to upgrade the firmware:

1. Close any open Control Panels.
2. On the Windows **Start** menu, click **All Programs > Agilent > N2101B > N2101A 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 FPGA file located in the installation directory (C:\Program Files\Agilent\N2101B), and click **Start**.

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

4

Programming

Introduction

This chapter outlines the general steps for making measurements that are controlled through the API and then continues with describing the details associated with the API structure.

Controlling measurements through the API

The API is provided in the form of a Windows based DLL and an ActiveX control. The API allows the host application to perform the following tasks:

- Create and delete communication channels to the N2101B.
- Receive notification for status change.
- Perform data acquisitions.
- Configure the N2101B.
- Perform measurements.
- Perform eye mask testing.

A Dynamic Link Library (DLL) provides an API for developing custom applications. An ActiveX control wrapper for the DLL is also provided to extend programmability to any ActiveX container application or COM compliant programming environment. These environments include Visual C++, Visual Basic, C#, LabView and many others. Refer to “DLL API Reference” on page 4-20.

Performing a bit error measurement

Follow these general steps to perform a bit error measurement:

1. Configure the traffic generator. Refer to “Step 1. Configure the Traffic Generator” on page 4-4.
2. Configure the receiver. Refer to “Step 2. Configure the Signal Receiver” on page 4-8.
3. Configure the trigger outputs. Refer to “Step 3. Configure the Trigger Outputs” on page 4-11
4. Select the measurement. Refer to “Step 4. Select a Measurement” on page 4-12.

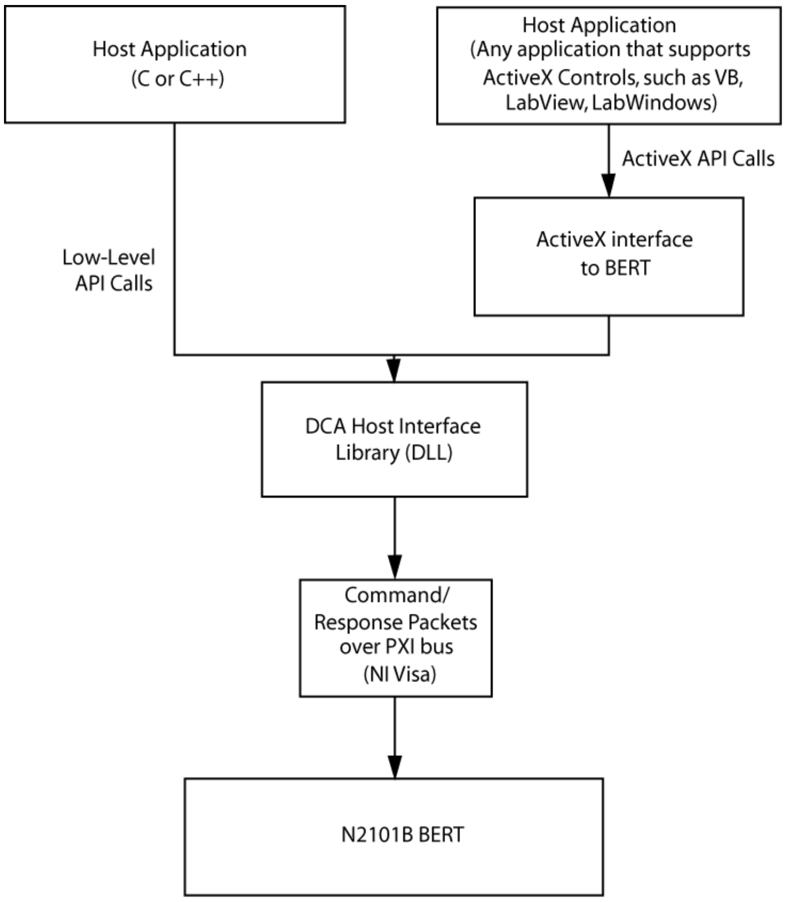


Figure 4-1. Host Library Overview

Step 1. Configure 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 N2101B VISA Handle” on page 4-18. The complete configuration of the traffic generator is passed as the second parameter to this function, with the following options:

Bit Rate Specifies one of the following bit rates of the generated signal:

- 155.52 Mb/s
- 622.08 Mb/s
- 1.0625 Gb/s
- 1.25 Gb/s
- 2.125 Gb/s
- 2.48832 Gb/s
- 3.125 Gb/s
- 2.50 Gb/s
- 4.25 Gb/s
- 5.00 Gb/s
- 6.25 Gb/s
- 8.50 Gb/s

Pattern Specifies one of the following data patterns:

- PRBS:
  - 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:
  - K28.5
  - K28.7
  - CRPAT
- User

A user pattern is defined by a handle to the file that contains the pattern and the pattern length.

Tx Clock Source and Frequency The transmit clock source can be set to internal (default), external retimed, or external direct. Internal uses the BERT internal oscillators to drive the PLL timing circuitry. External retimed uses a clock connected from an external source to the Tx Clk In to drive the PLL timing circuitry. External direct uses a clock connected from an external source to the BERT, bypasses the PLL timing circuitry and connects it directly to the data control hardware.

Operating the transmit clock source at 155.52 Mb/s will function in quadruple clocking mode in the N2101B.

NOTE *The N2101A BERT* may be required to operate in “Double Clocking Mode.” The external clock will have to be 2x the desired bit rate clock (for example, 5.4 GHz for 2.7 Gb/s rate).

Signal Amplitude Specifies the output signal amplitude in mV. The actual output range of the N2101B is from 250 mV to 1000 mV. Requesting the output signal amplitude to be set below the minimum value, results in the output signal being set to 250 mV. Requesting a value above the output amplitude range, results in the output amplitude being set to 1000 mV. The resolution of the output signal is 5 mV.

Injected Errors Specifies the type of errors to inject, if any.

Data Invert Sets whether the data is inverted on the transmit signal.

Transmitter External Clocking Configuration Example

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

main () {
   ConfigureTheTransmitter();
}


void ConfigureTheTransmitter(void) {
   ViSession instHandle; /* BERT VISA handle*/
   BertTxConfigExt txConfig; /* BERT transmitter configuration.*/
   /* Get the VISA instrument handle to the first BERT module found in
   the system. */
   PXIT336GetInstHandle(&instHandle, 0);

   if (instHandle != NULL) {
      txConfigExt.ClkSrc = 1; /* select clock source */
      txConfigExt.ClkRef = 2500.0;
      txConfigExt.Pattern.Type = DC;
      /* select data pattern */
      txConfigExt.Pattern.DcPattern = K28dot7;
      txConfigExt.DataInvert = 1; /* data inversion */
      txConfigExt.Amplitude = 500; /* in millivolts */
      /* Instruct the BERT to use the new transmitter parameters */
      PXIT336ConfigureTxExt(instHandle, &txConfig);
   }
}
```

Internal Clocking 4.25Gb/s PRBS-7 Configuration Example

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

main () {
   GeneratePRBS7();
}

void GeneratePRBS7(void) {
   ViSession instHandle; /* BERT VISA handle*/
   BertTxConfigExt txConfig; /* BERT transmitter configuration.*/
   /* Get the VISA instrument handle to the first BERT module found in
   the system. */
   PXIT336GetInstHandle(&instHandle, 0);

   if (instHandle != NULL) {
      PXIT336Reset(instHandle); /* Reset the BERT module. */
      PXIT336GetTxConfigExt(instHandle, &txConfig); /* Get the current
      BERT transmit configuration. */
      txConfig.BitRate = _4dot25_Gbps;
      txConfig.Pattern.Type = PRBS;
      /* Set the BERT configuration to generate a PRBS 7 pattern. */
      txConfig.Pattern.PrbsLength = PRBS_7;
      /* Instruct the BERT to use the new transmit parameters. */
      PXIT336ConfigureTxExt(instHandle, &txConfig);
   }
}
```

Step 2. Configure the Signal Receiver

Calling the function PXIT336ConfigureRx configures the BERT receiver. The first parameter is a VISA handle identifying the BERT module to be configured. Refer to "Obtaining the N2101B VISA Handle" on page 4-18. A pointer to an instance of the BertRxConfig structure is passed as the second parameter. The content of the BertRxConfig structure fully defines the BERT receiver configuration with the following parameters:

Bit Rate Specifies one of the following bit rates of the generated signal:

- 155.52 Mb/s
- 622.08 Mb/s
- 1.0625 Gb/s
- 1.25 Gb/s
- 2.125 Gb/s
- 2.48832 Gb/s
- 3.125 Gb/s
- 2.50 Gb/s
- 4.25 Gb/s
- 5.00 Gb/s
- 6.25 Gb/s
- 8.50 Gb/s

Pattern Specifies one of the following data patterns. Prior to setting the external frequency below 8 GHz, set the N2101B to internal re-timed.

- PRBS:
  - 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:
  - K28.5
  - K28.7
  - CRPAT
- User

A user pattern is defined by a handle to the file that contains the pattern and the pattern length. Refer to “BertUserPattern” on page 4-29.

Rx Clock Source Can be set to internal retimed (default), internal direct, external retimed, or external direct. Internal retimed uses the currently selected transmit clock to drive the PLL timing circuitry with a phase detector to provide at least two bit periods worth of phase sweep at any rate (155.52 Mb/s to 8.5 Gb/s). Internal direct uses the currently selected transmit clock, bypasses the PLL and uses a delay line to provide at least 200 ps of phase sweep for operation > 8.5 Gbps. External retimed uses a clock connected from an external source to the Rx Clk In to the PLL timing circuitry with a phase detector to provide at least two bit periods worth of phase sweep at any rate (155.52 Mb/s to 8.5 Gb/s). External direct uses a clock connected from an external source to the Rx Clk In, bypasses the PLL and uses a delay line to provide at least 200 ps of phase sweep for operation > 8.5 Gbps.

Data Invert Sets whether the data is inverted on the received signal.

Phase Delay This option specifies a phase delay between the signal and receiver clocks. The range is dependent upon the Rx Clock Source setting as described above. The increment is in 1-ps steps. When retimed (either internal or external) the phase sweep is greater than 2 bit periods at the current rate. When direct (either internal or external) there is at least 200 ps. Refer to “PXIT336SetRxPhaseDelay” on page 4-24 for more detail.

Receiver Configuration Example

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

main () {
   ConfigureTheReceiver();
}

void ConfigureTheReceiver(void) {
   ViSession instHandle; /* BERT VISA handle*/
   BertRxConfigExt rxConfig; /* BERT receiver configuration.*/
   /* Get the VISA instrument handle to the first BERT module found in
   the system. */
   PXIT336GetInstHandle(&instHandle, 0);

   if (instHandle != NULL) {
      /* select the receiver's clock source. */
      rxConfig.UseExternalClkSrc = Internal;
      rxConfig.BitRate = _1dot25_Gbps; /* select receiver's bit rate */
      rxConfig.Pattern.Type = DC;
      /* select receiver pattern */
      rxConfig.Pattern.DcPattern = K28dot7;
      rxConfig.DataInvert = 0; /* no data inversion */
      rxConfig.PhaseDelay = 0; /* no phase delay */
      /* Instruct the BERT to use the new receiver parameters. */
      PXIT336ConfigureRxExt(instHandle, &rxConfig);
    }
}
```

Step 3. Configure the Trigger Outputs

There are two trigger outputs on the N2101B:

- Clock Trigger

  The clock trigger output can be a divided down version of the transmit clock.

NOTE The transmit clock will be four times the bit rate when in quadruple clocking mode. (For example, 155.52 Mb/s Div 1 yields a 622.08 MHz clock trigger output.) Refer to “PXIT336SetClockTrigger” on page 4-24.

  The clock trigger can be placed in one of the following settings:

  DIV = 1 (transmit clock, as set)

  DIV = 2 (transmit clock divided by 2)

  DIV = 4 (transmit clock divided by 4)

  DIV = 8 (transmit clock divided by 8)

- Pattern Trigger

  You can program the pattern trigger to generate a pulse at the start of the transmit pattern, or toggle every 128 bits. For PRBS patterns, a trigger is generated every 64th pattern. Refer to “PXIT336SetPatternTrigger” on page 4-26.

NOTE The front-panel Trig In connector is reserved for future use and is not active.

Step 4. Select a Measurement

The measurements outlined in this section describe the necessary parameters and the expected results for the following:

- Bit Error Rate Measurement
- Eye Opening Measurement
- Bathtub Measurement
- Time Window Measurement

Bit Error Ratio Measurement

The bit error ratio (BER), also referred to as bit error rate, 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 PXIT336ConfigureRxExt().
3. Select the type of measurement you want to make by calling the function PXIT336StartMeasure().
4. Ensure that the measurement is complete by using one of the following:
   - Poll the BERT's status, using PXIT336GetStatus().
   - Set one of the parameters to the PXIT336StartMeasure() function as a call back function when the measurement is complete. The measurement of Continuous BER does not complete by itself and does not issue callbacks.
5. Read the measurement results:
   - If only one bit error count result is expected, call the function PXIT336GetBitErrorCount().
   - If a series of bit error count results is expected, inspect the result data structure passed as a parameter to PXIT336StartMeasure().

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

Eye Opening Measurement

Measures the eye opening for a specified BER value by running a bathtub measurement and analyzing the data.

You can also run a fast version of the bathtub measurement (by setting the parameter bFastBathtub to true) where the bit errors are counted from the start of the X-axis and continue until there are no longer any errors measured. The measurement then continues at the end of the X-axis and moves toward the start of the X-axis until there are no more errors measured. The jitter bathtub displays the corresponding eye width, total jitter, and deterministic jitter measurements.

Parameters This measurement only takes two parameters:

- The bit error rate (BER) value for which to determine the eye opening.
- A pointer to a function that will be called once the eye opening has been determined.

Results A structure of the type BertEyeOpening gives the eye opening result in picoseconds.

Example

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

void EyeOpeningMeasureExample(void);
/* Variable to store TIA fit eye width result */
Double eyeOpeningResult;
/* Variable to store TIA fit jitter result */
Double eyeOpeningTJ;
/* Variable to store MJSQ fit eye width result */
Double eyeOpeningResult2;
/* Variable to store TIA fit total jitter result */
Double eyeOpeningTJ2;
/* Variable to store MJSQ fit deterministic jitter result */
Double eyeOpeningDJ2;

main () {
   EyeOpeningMeasureExample();
   printf("Eye opening: %d ps", eyeOpeningResult);
}

void EyeOpeningMeasureExample(void) {
   ViSession instHandle; /* BERT VISA handle*/
   /* Retrieve instrument handle */
   PXIT336GetInstHandle(&instHandle, 0);
   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;
    eyeOpeningParamers.pTJ = &eyeOpeningTJ;
    eyeOpeningParamers.pEyeOpening2 = &eyeOpeningResult2;
    eyeOpeningParamers.pTJ2 = &eyeOpeningTJ2;
    eyeOpeningParamers.pDJ2 = &eyeOpeningTD2;
    eyeOpeningParamers.bFastBathtub = false;
    /* Initiate the eye opening measurement. */
    PXIT336StartMeasure(instHandle, EyeOpening, &eyeOpeningParameters);
}
```

Bathtub Measurement

This measurement generates a Bathtub diagram with the phase delay on the X-axis and the bit error rate on the Y-axis. You can configure the phase resolution and the number of measured points.

Parameters The input parameters for this measurement are as follows:

- Phase increment. This is the number of phase delay increments between each individual measurement point. 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 interval. 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 one of the following functions called to ensure the completion of the Bathtub measurement.
  - Poll the BERT's status, using PXIT336GetStatus().
  - Set one of the parameters to the PXIT336StartMeasure() function as a call back function when the measurement is complete.
- Number of measurement points. This is the total number of measurement points to be included in the Bathtub curve.

Results An array of structures of type BertMeasurePoint contains the number of error bits and the total number of measurement points. Please note that it is the caller's responsibility to allocate the memory for this array. The recommended array size is 8192 as defined by the macro MAX_BER_MEASURE_TABLE_ENTRIES in pxit336.h.

Time Window Measurement

This measurement reports the bit error rate in a sliding time window. To receive the report describing the number of bit errors measured during the last elapsed time window, use the call PXIT336GetBitErrorCount().

Parameters This measurement only takes two parameters.

- The window size specified in microseconds (0 to 221 seconds).
- 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 The results are in 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

You can inject errors in the traffic generator output by calling PXIT336InsertError() with a pointer to a structure of type BertErrorInsert. By setting the style to NoErrors, you can switch off the continuous rate injection. Each call with style of singleBurst injects count bit errors.

Configuring the options

Style Specifies one of the following styles of error insertion:

- NoErrors
- SingleBurst
- Continuous

Rate Specifies the rate at which errors will be inserted.

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

Obtaining the N2101B VISA Handle

All calls to the BERT API take a VISA handle as the first parameter. This handle identifies the module to be accessed by the function call. You can obtain the handle by using either of the following methods:

- Using the VISA API, directly
- Calling the function PXIT336GetInstHandle()

The first parameter to the PXIT336GetInstHandle()function call is a pointer to the location where the VISA handle will be stored.

The second parameter enumerates the BERT modules present in the system. By setting this second parameter to 0, the VISA handle is returned for the first BERT module detected in the system. By incrementing this parameter in subsequent calls, the VISA handle is returned for other BERT modules detected in the system.

To to obtain a VISA handle for each BERT module present in the system, follow these steps:

1. Call PXIT336GetInstHandle in a loop to enumerate all BERT modules present in the system.
2. Increment the value of the second parameter until the VISA handle returned is NULL.

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. */
   /* Retrieve instrument handle */
   PXIT336GetInstHandle(&instHandle, 0);
   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 describes 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 to compare to one of the major error codes described in this table. Bits 16-31 of the errors codes may contain information about the cause of the error.

NOTE The N2101B does not support the following N2101A functions: PXIT336SetJitterFrequency and PXIT336GetJitterFrequency.

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 recognized as a valid handle. Make sure there is at least one PATTERN GENERATOR present in the system, the PXI chassis is powered up, and 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 N2101B board registers. Power cycle the system. Contact support if the problem persists.
BOARD_DEVICE_ERROR	0x00060000	Cannot access one of the devices present on the board. This 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 that is detected in the system. This handle is stored in the location that is passed as a 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 will be stored for the first BERT module detected in the system.

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.

PXIT336GetTxConfigExt

Retrieve the current transmit portion of the BERT configuration.

Function Prototype int PXIT336GetTxConfigExt(ViSession inst, BertTxConfigExt *txConfig);

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

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

PXIT336GetRxConfigExt

Retrieve the current receiver portion of the BERT configuration.

Function Prototype int PXIT336GetRxConfigExt(ViSession inst, BertRxConfigExt *rxConfig);

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

rxConfig Pointer to an instance of BertRxConfigExt where the current receiver configuration will be stored.

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 *txConfig);

inst VISA handle of the BERT module to configure.

txConfig Pointer to an instance of BertTxConfigExt containing the new transmit configuration.

PXIT336SetTxAmplitude

Specifies the amplitude of the pattern generator output 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 switch off the 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.

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.

PXIT336ConfigureRxExt

Configures the receive block.

Function Prototype int PXIT336ConfigureRxExt(ViSession inst, BertRxConfigExt *rxConfig);

inst VISA handle of the BERT module to configure.

rxConfig Pointer to an instance of BertRxConfigExt containing the receive configuration.

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.

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 1, 2, 4, or 8 version of the transmit clock. (Divide by 1 is the same rate as 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	Div 2	Div 4	Div 8	Div 1
155.52 Mb/s	311.04 MHz	155.52 MHz	77.76 MHz	622.08 MHz
622.08 Mb/s	311.04 MHz	155.52 MHz	77.76 MHz	622.08 MHz
1.0625 Gb/s	531 MHz	265 MHz	132.81 MHz	1.0625 GHz
1.25 Gb/s	625 MHz	312 MHz	156.25 MHz	1.250 GHz
2.125 Gb/s	1.06 GHz	531 MHz	265.63 MHz	2.125 GHz
2.48832 Gb/s	1.24 GHz	622 MHz	311.04 MHz	2.48832 GHz
2.50 Gb/s	1.25 GHz	625 MHz	312.50 MHz	2.50 GHz
3.125 Gb/s	1.56 GHz	780 MHz	390.62 MHz	3.125 GHz
4.25 Gb/s	2.13 GHz	1.06 GHz	531.25 MHz	4.25 GHz
5.00 Gb/s	2.5 GHz	1.25 GHz	625.00 MHz	5.0 GHz
6.25 Gb/s	3.125 GHz	1.56 GHz	781.25 MHz	6.25 GHz
8.50 Gb/s	4.25 GHz	2.13 GHz	1.063 Ghz	8.50 GHz

Since the clock signal has a minimum operational frequency of 500 MHz, the N2101B can operate down to 155 Mb/s by using a clock frequency of four times the data bit rate. This is called Quad Rate Clock Mode and is automatically entered when you select any bit rate below 500 Mb/s, whether the clock signal is generated internally or through the external inputs. Quad Rate Clock Mode functions by maintaining each data bit for four consecutive clock cycles before transitioning to the next data bit.

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

inst VISA handle of the N2101B module to configure.

clkTrigCfg Specifies the divider ratio for the Clock Trigger output. This ratio can be div1, div2, div4, div8, 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 connector on the N2101B module can output a single bit pulse synchronized 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 dependent 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 the 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 to 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 from which to read the bit error count.

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

Data Types and Structures

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

BertBitRate

This enumeration defines the different bit rates supported.

NOTE The values starting at _Ext_Base are for Agilent calibration use only and are not to be used for the TxConfigExt.BitRate or RxConfigExt.BitRate fields.

```
typedef enum
{
    _1dot0625_Gbps = 0,
    _1dot25_Gbps, // 1
    _2dot125_Gbps, // 2
    _2dot48832_Gbps, // 3
    _2dot66_Gbps, // 4
    _4dot25_Gbps, // 5
    _2dot500_Gbps, // 6
    _5dot0_Gbps, // 7
    _dot62208_Gbps, // 8
    _dot15552_Gbps, // 9
    _3dot125_Gbps, // 10
    _6dot25_Gbps, // 11
    _8dot50_Gbps, // 12
    _Ext_Base,
    _Ext_8dot50_Gbps = _Ext_Base,
    _Ext_9dot95328_Gbps,
    _Ext_10dot3125_Gbps,
    _Ext_10dot709_Gbps,
    _Ext_End = _Ext_10dot709_Gbps,
    _CustomerBR = 255,
} BertBitRate;
BertClockSources

typedef enum _PXITBERT_CLKSRCS
{
    CLKSRC_INTERNAL = 0,
    CLKSRC_EXTERNAL_LS,
    CLKSRC_EXTERNAL_HS,
    CLKSRC_INTERNAL_HS,
} PXITBERT_CLKSRCS;

CLKSRC_INTERNALInternal Retimed
CLKSRC_EXTERNAL_LS External Retimed
CLKSRC_EXTERNAL_HS External Direct
CLKSRC_INTERNAL_HS Internal Direct
See page 3-4 for descriptions.
```

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 configurations.

```
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;
```

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 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. Prior to setting the external frequency below 8 GHz, set the N2101B to internal re-timed.

BertRxConfig

This structure fully describes the receiver configuration. This data structure is typically used as a parameter for the PXIT336ConfigureRxExt call.

```
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.
ClkSrc	Selects the clock source. Must be set to CLKSRC_INTERNAL (0) for internal clocking and the ClkRef desired bit rate using an external clock source.

BertErrorInsert

This structure fully decribes the errors to be injected into the traffic generator 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 specifies the rate at which errors will be injected into the traffic generator output when the BertErrorInjectStyle is set to Continuous.

```
typedef enum
{
    TenMinus3 = 0,
    TenMinus4,
    TenMinus5,
    TenMinus6,
    TenMinus7,
    TenMinus8,
    TenMinus9,
    TenMinus10
} BertErrorRate;
```

BertErrorCount

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

```
typedef unsigned long BertErrorCount;
```

BertMeasureType

This structure specifies the type of measurement you want 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 storing 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 specifies 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	The maximum window 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, because 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 measurements 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 instance when a time window has elapsed. It can 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 to store the plot.
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 point. 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 specifies the parameters of an eye opening measurement. The bathtubParam is defined above, and the targetBER is the desired bit error rate. The TIA fit calculation eye width result in picoseconds is returned in *pEyeOpening, with the total jitter returned in *pTJ. The MJSQ fit calculated eye width, total jitter, and deterministic jitter are returned in *pEyeOpening2, *pTJ2, and *pDJ2, respectively.

```
typedef struct
{
BertBathtubParam BathtubParam;
   double TargetBER;
   double * pEyeOpening;
   double * pTJ;
   double * pEyeOpening2;
   double * pDJ2;
   double * pTJ2;
   bool bFastBathtub;
} BertEyeOpeningParam;
```

BertBitErrorCount

This structure reports 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 updates 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                CalibrationRequired;
    bool                TxClockLocked;
    bool                RxClockLocked;
    bool                RxSynched;
} BertStatus;
```

ActiveX API Properties

This section describes the available properties of the ActiveX API for controlling the N2101B BERT. Attempting to change a property value while a measurement is running might cause the measurement to fail. For example, changing the bit rate while making a continuous BER measurement or plotting a bathtub. The BERT software is designed to ignore attempts at changing properties while a measurement is running.

TxBitRate

Specifies the transmitter's bit rate. Prior to setting the external frequency below 8 GHz, set the N2101B to internal re-timed.

Type	LONG
Allowed Values	0: 1.0625 Gb/s
	1: 1.25 Gb/s
	2: 2.125 Gb/s
	3: 2.488 Gb/s
	5: 4.25 Gb/s.
	6: 2.5 Gb/s
	7: 5.0 Gb/s
	8: 622.08 Mb/s
	9: 155.52 Mb/s
	10: 3.125 Gb/s
	11: 6.25 Gb/s
	12: 8.50 Gb/s

RxBitRate

Specifies the receiver's bit rate.

Type	LONG

Allowed Values	0: 1.0625 Gb/s
	1: 1.25 Gb/s
	2: 2.125 Gb/s
	3: 2.488 Gb/s
	5: 4.25 Gb/s.
	6: 2.5 Gb/s
	7: 5.0 Gb/s
	8: 622.08 Mb/s
	9: 155.52 Mb/s
	10: 3.125 Gb/s
	11: 6.25 Gb/s
	12: 8.50 Gb/s

TxClockSource

Specifies the transmitter's clock source.

Type	LONG
Allowed Values	0: Internal clock
	1: External retimed clock
	2: External direct clock

RxClockSource

Specifies the receiver's clock source.

Type	LONG
Allowed Values	0: Internal retimed clock
	1: External retimed clock
	2: External direct clock
	3: Internal direct clock

TxExtClockRate

Specifies the frequency of the transmit clock when the transmit clock source (TxClockSource property) is set to External

Type FLOAT

Allowed Values A floating-point value greater than 125 and less than 11000, that specifies the desired bit rate in Mb/s.

RxExtClockRate

Specifies the frequency of the receive clock when the receive clock source (RxClockSource property) is set to External.

Type FLOAT

Allowed Values A floating-point value greater than 125 and less than 11000, that specifies the desired bit rate in Mb/s.

TxPattern

Specifies the transmitter's bit pattern.

Type LONG

Allowed Values 0: PRBS 7

1: PRBS 9

2: PRBS 11

3: PRBS 15

4: PRBS 23

5: PRBS 31

6: K28.5

7: K28.7

8: CRPAT

9: User Pattern. Set by a call to LoadUserPattern method.

RxPattern

Specifies the receiver's bit pattern.

Type LONG

Allowed Values 0: PRBS 7

1: PRBS 9

2: PRBS 11

3: PRBS 15

4: PRBS 23

5: PRBS 31

6: K28.5

7: K28.7

8: CRPAT

9: User Pattern. Set by a call to LoadUserPattern method.

TxDataInvert

Specifies the transmiter's data inversion.

Type LONG

Allowed Values 0 (Data not inverted) and 1 (Data inverted)

RxDataInvert

Specifies the receiver's data inversion.

Type LONG

Allowed Values 0 (Data not inverted) and 1 (Data inverted)

TxAmplitude

Specifies the transmit signal amplitude in millivolts.

Type LONG

Allowed Values 250 to 1000

RxPhaseDelay

Specifies the receiver's phase delay in picoseconds.

Type LONG

Allowed Values 0 to the maximum for the receiver's selected bit rate. The method GetMaxRx-PhaseDelay returns the maximum available delay at the currently selected rate.

ClockTrigger

Specifies the clock trigger output divider.

Type LONG

Allowed Values 0 (Bit clock divided by 8), 1 (Bit clock divided by 4), 2 (Ref Clk divided by 2), or 3 (Bit clock rate).

PatternTrigger

Specifies the pattern trigger output source.

Type LONG

Allowed Values 0 (Pattern trigger) or 1 (Bit clock divided by 128).

ContinuousErrorRate

Specifies the continuous error rate to be applied to the transmit bit pattern. See also GenerateErrorBurst method for insertion of errors under program control.

Type LONG

Allowed Values 0: No errors

1: 10-3

2: 10-4

3: 10-5

4: 10-6

5: 10-7

6: 10-8

7: 10-9

8: 10-10

DisplayEnable

Enables or disables the BERT Active-X control’s display window. When enabled, the window presents a display similar to that of the BERT control panel application.

Type LONG

Allowed Values 0 (The window is not displayed) and 1 (The display is enabled and the window is displayed).

DisplayTextSize

Selects one of three text sizes for the display window.

Type LONG

Allowed Values 0 (smallest), 1 (medium), or 2 (largest).

SyncStatus (read-only)

Indicates that the receiver is synchronized with the received pattern.

Type LONG

Allowed Values 0 (No pattern sync) or 1 (Pattern sync).

AllStatus (read-only)

Returns a word where the individual bits indicate status conditions.

Type LONG

Allowed Values Bit 0 [0x01]: Pattern sync

Bit 1 [0x02]: Loss-of-sync since the AllStatus property was last read

Bit 2 {0x04]: The receiver phase-locked-loop is in lock

Bit 3 [0x08]: The transmitter phase-locked-loop is in lock

Bit 4 [0x10]: The module calibration status is invalid

ClockBoardSerialNumber

Retrieves the clock board's serial number. (ReadOnly)

Type BSTR

Allowed Values NA

ClockBoardRev

Retrieves the clock board's revision. (ReadOnly)

Type LONG

Allowed Values NA

ClockBoardType

Retrieves the clock board’s product code. (ReadOnly)

Type	BSTR
Allowed Values	NA

DataGenSerialNumber

Retrieves the data generator module’s serial number. (ReadOnly)

Type	BSTR
Allowed Values	NA

DataGenRev

Retrieves the data generator module’s module revision. (ReadOnly)

Type	LONG
Allowed Values	NA

ActiveX API Methods

This section describes the available methods of ActiveX API for controlling the N2101B BERT.

NOTE To avoid a deadlock condition, do not place a call to StartClockAlign, StartContinuousBER, StartTimeWindow, StartFullSweep, StartBathtub, StartFastBathtub, StartEyeOpening, or StopMeasure within the event handler.

NOTE Do not place a call to a function that can create a window or perform GUI operations from within the event handlers. Instead, use the event handler to send a (user-defined) message to that window, and let the message handler perform the GUI operations.

Module Connection Methods

OpenDevice

CHAR OpenDevice(BSTR pszDevice);

This method is used to open an instance to a BERT generator. This method must be invoked prior to invoking any other BERT generator methods.

Parameters BSTR pszDevice – Name of the device in NI Visa format. For example, PXI3::12::INSTR

Returns TRUE if the device was successfully opened, otherwise false.

CloseDevice

void CloseDevice(void);

This method closes a previously open instance to a BERT.

Parameters NA

Returns NA

Configuration Methods

GenerateErrorBurst

LONG GenerateErrorBurst(LONG ulErrorCount);

Generates a burst of bit errors. See the ContinuousErrorRate property for continuous insertion of errors at a specified BER. ContinuousErrorRate must be set to No Errors when calling GenerateErrorBurst.

Parameters LONG ulErrorCount

Specifies the number of bit errors to generate.

Returns Refer to “Methods Error Codes” on page 4-56.

LoadUserPattern

void

LoadUserPattern(

VARIANT* pvPatternData,

LONG ulBitCount);

Set the length and data for a user-specified pattern.

Parameters VARIANT* pvPatternData

Pointer to a VARIANT that contains the pattern data. The VARIANT must contain an array of LONG and the binary data must be left-justified within the array. For example, a pattern of 40 bits would occupy the first word of the array and the left-most eight bits of the second word; a pattern of 10 bits would occupy the leftmost ten bits of the first word only.

LONG ulBitCount

LONG value that specifies the length, in bits, of the user pattern.

NOTE Programming environments that cannot pass pattern data in a VARIANT can use the functions InitUserPattern, SetUserPatternData / SetUserPatternDataLong, and StartUserPattern to perform this function.

Returns NA

InitUserPattern

void

InitUserPattern(

LONG ulBitCount

);

Sets the length for a user-specified pattern and starts the byte-by-byte transfer of pattern data to the BERT.

Parameters LONG ulBitCount

LONG value that specifies the length, in bits, of the user pattern.

Returns NA

SetUserPatternData, SetUserPatternDataLong

void

SetUserPatternData(

BYTE ucDataByte

);

void

SetUserPatternDataLong(

LONG ulDataByte

);

These calls transfer up to 8 bits of pattern data to the BERT.

Each call to one of these methods transfers a byte of data. Bytes must be transferred in order, and the transfer must begin with a call to InitUserPattern. After transferring the last byte, call StartUserPattern to begin the BERT operation with the newly transferred pattern.

NOTE SetUserPatternDataLong is provided to accommodate programming environments that cannot work with 8-bit data. SetUserPatternDataLong cannot transfer more than eight bits of data with each call.

Parameters BYTE ucDataByte (SetUserPatternData)
An 8-bit unsigned value that specifies a byte of the user pattern.
LONG ulDataByte (SetUserPatternDataLong)
LONG value that specifies a byte of the user pattern.

Returns NA

StartUserPattern

void

StartUserPattern();

Initiates BERT operation with the user pattern data transferred by InitUserPattern and SetUserPatternData / SetUserPatternDataLong.

Parameters NA

Returns NA

Status Methods

IsMeasureComplete

LONG IsMeasureComplete(void);

Returns a value that indicates whether the most recently initiated measurement operation has finished.

Parameters NA

Returns An integer value, as follows:

0: The measurement has not yet finished.

1: The measurement is complete.

GetLastError

LONG GetLastError(void);

Returns the error status from the most recent property set or method call.

Parameters NA

Returns A value as described in the section, “Methods Error Codes” on page 4-56“.

Measurement Methods

GetMaxRxPhaseDelay

LONG GetMaxRxPhaseDelay(void);

Returns the maximum available Rx phase delay value for the current Rx rate setting.

Parameters None

Returns A LONG value representing the maximum delay in picoseconds.

GetRxFrequency

FLOAT GetRxFrequency(void);

Returns the measured frequency of the signal at the receiver input.

Parameters NA

Returns A FLOAT value representing the frequency in MHz.

StartClockAlign

LONG StartClockAlign(DOUBLE dTargetBER);

Initiates Clock Data Alignment.

Parameters DOUBLE dTargetBER:

Indicates the bit error rate at which to perform the clock data alignment.

Returns Refer to “Methods Error Codes” on page 4-56.

GetClockAlignResult

void GetClockAlignResult(DOUBLE* pdPhaseDelay);

Returns the phase delay resulting from Clock Data Alignment.

Parameters DOUBLE* pdPhaseDelay:

Pointer to a double variable where phase delay resulting from the clock align will be stored.

Returns NA

StartContinuousBER

LONG StartContinuousBER();

Starts a continuous BER measurement. Call GetContinuousBERResult to retrieve the cumulative results from the measurement.

Parameters NA

Returns Refer to “Methods Error Codes” on page 4-56.

GetContinuousBERResult

LONG GetContinuousBERResult(LONG* pResult);

Returns the bit count and error count since the start of a continuous BER measurement.

Parameters LONG* pResult:

Pointer to an array of four (unsigned) LONG words that receive the result, as follows:

[0] Bit count, high-order 32 bits

[1] Bit count, low-order 32 bits

[2] Error count, high-order 32 bits

[3] Error count, low-order 32 bits

Returns Refer to “Methods Error Codes” on page 4-56.

StartTimeWindow

LONG

StartTimeWindow(

LONG nWindowLength,

LONG nMaxErrors,

LONG nNbOfIterations

);

Initiates a time window bit error rate measurement. Call GetTimeWindowResult after each window interval to retrieve the results from that interval.

Parameters LONG nWindowLength

Specifies the length of the time window in microseconds.

LONG nMaxErrors

Maximum number of errors after which to abort the time window and report a bit error rate. A value of 0 for this parameter indicates that the full length of the time window must expire before reporting a BER, regardless of the number of bit errors.

LONG nNbOfIterations

Number of iterations of the time window. This indicates how many times to perform a BER measurement. A value of 0 indicates an infinite number of time window measurements.

Returns Refer to “Methods Error Codes” on page 4-56.

GetTimeWindowResult

void GetTimeWindowResult(DOUBLE* pdBER);

Returns a time window bit error rate measurement result.

Parameters DOUBLE* pdBER

Pointer to a double variable where the result of the most recent iteration of the time windowed BER measurement will be stored.

Returns NA

StartBathtub

LONG

StartBathtub(

LONG nMaxTimePerPoint,

LONG nMaxNbErrorPerPoint,

LONG nPhaseResolution);

Initiates a bathtub measurement. Call GetBathtubResult to retrieve the results of the bathtub measurement.

Parameters ULONG nMaxTimePerPoint

Specifies the maximum length of time to spend measuring errors for each point of the bathtub chart. This time is specified in microseconds.

ULONG nMaxNbErrorPerPoint

Specifies the maximum number of bit errors to measure for each point of the bathtub chart.

ULONG nPhaseResolution

Delay resolution in picoseconds of the bathtub measurement.

Returns Refer to “Methods Error Codes” on page 4-56.

StartFastBathtub

LONG

StartFastBathtub(

LONG nMaxTimePerPoint,

LONG nMaxNbErrorPerPoint,

LONG nPhaseResolution);

Initiates a fast bathtub measurement. Call GetBathtubResult to retrieve the results of the fast bathtub measurement.

Parameters ULONG nMaxTimePerPoint

Specifies the maximum length of time to spend measuring errors for each point of the bathtub chart. This time is specified in microseconds.

ULONG nMaxNbErrorPerPoint

Specifies the maximum number of bit errors to measure for each point of the bathtub chart.

ULONG nPhaseResolution

Delay resolution in picoseconds of the fast bathtub measurement.

Returns Refer to “Methods Error Codes” on page 4-56.

StartFullSweep

LONG

StartFullSweep();

Initiates a full sweep measurement. Call GetBathtubResult to retrieve the results of the full sweep measurement.

Parameters NA

Returns Refer to “Methods Error Codes” on page 4-56.

GetBathtubResult

void GetBathtubResult(VARIANT* pData, LONG* pNbItems);

Returns the bathtub plot points resulting from a bathtub, fast bathtub, full sweep, or eye opening measurement.

Parameters VARIANT* pData

Pointer to a VARIANT variable. This variant will contain a 2 dimensional safearray containing the the list of points making up the bathtub plot. Array items [0][i] will contain the bit error rate and array items [1][I] will contain the phase delay for each point i.

LONG* pNbItems

Pointer to a variable where the number of points making up the bathtub plot will be stored.

Returns None.

GetBathtubNbOfPoints

LONG GetBathtubNbOfPoints(void);

Returns the number of points that make up the bathtub plot points. This is part of an alternative to using the GetBathtubResult method.

Parameters NA

Returns Returns the number of points that make up the bathtub plot.

GetBathtubPoint

void

GetBathtubPoint(

LONG nPointIdx,

DOUBLE* dPhaseDelay,

DOUBLE* dBER);

Returns an individual point of the bathtub plot. This is part of an alternative to using the GetBathtubResult method.

Parameters LONG nPointIdx

Index of the bathtub point to retrieve.

DOUBLE* dPhaseDelay

Pointer to a variable where the phase delay for the point identified by nPointIdx will be stored.

DOUBLE* dBER

Pointer to a variable where the bit error rate for the point identified by nPointIdx will be stored.

Returns NA

StartEyeOpening

LONG

StartEyeOpening(

LONG nMaxTimePerPoint,

LONG nMaxNbErrorPerPoint,

LONG nPhaseResolution,

DOUBLE dTargetBER);

Initiates an eye opening measurement. Call GetEyeOpeningResult to retrieve the result of the measurement; call GetBathtubResult to retrieve the results of the sweep made during the measurement.

Parameters ULONG nMaxTimePerPoint

Specifies the maximum length of time to spend measuring errors for each point of the bathtub chart. This time is specified in microseconds.

ULONG nMaxNbErrorPerPoint

Specifies the maximum number of bit errors to measure at each point in the bathtub chart.

ULONG nPhaseResolution

Delay resolution in picoseconds of the fast bathtub measurement.

DOUBLE dTargetBER

BER level at which the eye opening is to be measured.

Returns Refer to “Methods Error Codes” on page 4-56.

GetEyeOpeningResult

void GetEyeOpeningResult(DOUBLE *pEyeWidth);

Returns the measured eye width.

Parameters DOUBLE* pEyeWidth

Pointer to a variable where the measured eye width will be stored.

Returns A floating-point value representing the measured eye width in picoseconds calculated using TIA fit.

NOTE The value –1 is returned when there is insufficient BER data to make the eye width measurement. For example, this can happen when the measurement time per point is too short to acquire data at low BER values.

GetEyeOpeningTJ

void GetEyeOpeningTJ(DOUBLE *pTJ);

Returns the measured total jitter.

Parameters DOUBLE* pTJ

Pointer to a variable where the measured total jitter will be stored.

Returns A floating-point value representing the measured total jitter in picoseconds calculated using TIA fit.

NOTE The value –1 is returned when there is insufficient BER data to make the eye width measurement. For example, this can happen when the measurement time per point is too short to acquire data at low BER values.

GetEyeOpeningResult_MJSQ

void GetEyeOpeningResult_MJSQ(DOUBLE *pEyeWidth2);

Returns the measured eye width.

Parameters DOUBLE* pEyeWidth2

Pointer to a variable where the measured eye width will be stored.

Returns A floating-point value representing the measured eye width in picoseconds using MJSQ fit.

NOTE The value –1 is returned when there is insufficient BER data to make the eye width measurement. For example, this can happen when the measurement time per point is too short to acquire data at low BER values.

GetEyeOpeningTJ_MJSQ

void GetEyeOpeningTJ_MJSQ(DOUBLE *pTJ2);

Returns the measured total jitter.

Parameters DOUBLE* pTJ2

Pointer to a variable where the measured total jitter will be stored.

Returns A floating-point value representing the measured total jitter in picoseconds calculated using MJSQ fit.

NOTE The value –1 is returned when there is insufficient BER data to make the eye width measurement. For example, this can happen when the measurement time per point is too short to acquire data at low BER values.

GetEyeOpeningDJ_MJSQ

void GetEyeOpeningDJ_MJSQ(DOUBLE *pDJ2);

Returns the measured deterministic jitter.

Parameters DOUBLE* pDJ2

Pointer to a variable where the measured deterministic jitter will be stored.

Returns A floating-point value representing the measured deterministic jitter in picoseconds calculated using MJSQ fit.

NOTE The value –1 is returned when there is insufficient BER data to make the eye width measurement. For example, this can happen when the measurement time per point is too short to acquire data at low BER values.

StopCurrentMeasure

LONG

StopCurrentMeasure(void);

Stops the current measurement.

Returns Refer to “Methods Error Codes” on page 4-56.

Methods Error Codes

The table below specifies bits 16 to 31 of the methods return values. You should mask off the lowest 16 bits of the return values to then compare them to one of the major error codes described in this table. Bits 16-31 of the error code may contain information about the cause of the error.

Table 4-3.

Mnemonic	Value	Description
SUCCESS	0x00000000	Operation completed successfully.
UNEXPECTED_ERROR	0x00010000	Unexpected error. Contact PXIT support with a description of how this error occured.
MEASURE_IN_PROGRESS_ERROR	0x00020000	The requested operation cannot be completed because there is a measurement currently running. Try calling the StopCurrent-Measure() method before retrying.
UNKNOWN_DEVICE	0x00030000	The instrument handle used is not recognised as a valid handle. Make sure there is at least one BERT present in the system, the PXI chassis is powered up, and 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 PXIT BERT board registers. Power cycle the system. Contact PXIT 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 reponding or if there is no connection between the data generator board and clock board. Bits 0-7 of the error code contain the device causing the error. Bits 8-15 of the error code contain the bus number of the device causing the error.

Events

MeasureStep

This event is generated every time a measurement step is completed. For example, this event occurs with either of the following conditions:

- At the end of each time window interval.
- When a new bathtub point has been measured.

Parameter

LONG nMeasureType

Indicates the type of measurement that generated the event.

1: Clock Align

3: Time Window

4: Bathtub

MeasureCompletion

This event is generated when a measurement intiated by a call to StartClockAlign, StartTimeWindow, StartFullSweep, StartBathtub, StartFastBathtub, or StartEyeO-pening completes.

Parameter

LONG nMeasureType

Indicates the type of measurement that generated the event.

1: Clock Align

3: Time Window

4: Bathtub

5

Specifications

Specifications

The distinction between specifications and characteristics is described as follows:

- Specifications describe warranted performance. All specifications apply after the instrument is turned on for one (1) hour.
- *Characteristics* provide useful information by giving functional, but nonwarranted, performance parameters. *Characteristics are printed in italics.*

Table 5-1. Environmental Specifications

Use	Indoor
Maximum power consumption	24W
Dimensions	Three-slot wide 3U PXI module

Table 5-2. Transmit (Tx) Specifications

Range of operation	155 Mb/s to 10.3125Gb/s
Range of Operation with External Clock	*155 Mb/s to 10.3125Gb/s (characteristic)*
Internal clock frequencies[a]	155.52 MHz, 622.08 MHz, 1.0625 GHz, 1.25 GHz, 2.125 GHz, 2.48832 GHz, 2.50 GHz, 3.125 GHz, 4.25 GHz, 5.00 GHz, 6.25 GHz, and 8.50 GHz
Line rate accuracy for internal generated clock	± *0.01% (characteristic)*
External retimed clock operation range. Exact rate clocking[b]	1 Gb/s $\leq$ f $\leq$ 8.5 Gb/s
External retimed clock operation range. Quadruple rate clocking.	155 Mb/s $\leq$ f $\leq$ 500 Mb/s with a 4x clock frequency input of 500 MHz $\leq$ f $\leq$ 2 GHz
External direct range	8.5 Gb/s $\leq$ f $\leq$ 10.3125Gb/s
Output jitter	2.5 ps rms (Max) *1.5 ps RMS (characteristic)*
Rise/Fall times (20-80%)	25 ps (Max) *22 ps (characteristic)*
Jitter	2.5 ps (Max) *1.5 ps (characteristic)*
Output voltage range (single-ended)	250 mV to 1V
Amplitude accuracy	± *10% (characteristic)*
Amplitude resolution	5 mV
Bit error insertion	Single and multiple bit bursts, Continuous BER of 10^{-n} (n = 3,4,5,6,7,8,9,10)
Connector type	SMA

a. Option dependent.
b. f = input frequency

Table 5-3. Receive (Rx) Specifications

Supported data patterns	PRBS 7, PRBS 9, PRBS 11, PRBS 15, PRBS 23, PRBS 23, PRBS 31, K28.5, K28.7, CRPat, and user pattern
Input range (differential)	50 mV to 2V
Input sensitivity (differential)	50 mV
Input impedance (single-ended)	*50 Ohm (characteristic)*
Input impedance (differential)	*100 Ohm (characteristic)*
Phase sweep range (< 8 Gb/s)	> 2 UI
Phase sweep range (> 8 Gb/s)	200 ps
Connector type	SMA

Table 5-4. Clock Trigger Output Specifications

Divided clock rate outputs	Bit clock rate divided by 1, 2, 4, 8, & 128 (output on separate port[a] see Table 5-5).
Pattern trigger	Triggers every 64th pattern for PRBS
Pattern trigger / clock output voltage	*1 V pp (characteristic)*
Clock voltage output (dc coupled)	*400 mV to 1.1 V (characteristic)*
External clock input voltage range	*500 mV to 1 V (characteristic)*
External clock input frequency range	500 MHz to 10.3125GHz
Connector type	SMA

a. The clock trigger output at 155 Mb/s has four options that are based on the quadruple rate clock of 622 MHz: 622/8, 622/4, 622/2, and 622.

Table 5-5. Clock Trigger Output Frequency Options

Internal Bit Rate Clock	Div 2	Div 4	Div 8	Div 1
155.52 Mb/s	311.04 MHz	155.52 MHz	77.76 MHz	622.08 MHz
622.08 Mb/s	311.04 MHz	155.52 MHz	77.76 MHz	622.08 MHz
1.0625 Gb/s	531 MHz	265 MHz	132.81 MHz	1.0625 GHz
1.25 Gb/s	625 MHz	312 MHz	156.25 MHz	1.250 GHz
2.125 Gb/s	1.06 GHz	531 MHz	265.63 MHz	2.125 GHz
2.48832 Gb/s	1.24 GHz	622 MHz	311.04 MHz	2.48832 GHz
2.50 Gb/s	1.25 GHz	625 MHz	312.50 MHz	2.50 GHz
3.125 Gb/s	1.56 GHz	780 MHz	390.62 MHz	3.125 GHz
4.25 Gb/s	2.13 GHz	1.06 GHz	531.25 MHz	4.25 GHz
5.00 Gb/s	2.5 GHz	1.25 GHz	625.00 MHz	5.0 GHz
6.25 Gb/s	3.125 GHz	1.56 GHz	781.25 MHz	6.25 GHz
8.50 Gb/s	4.25 GHz	2.13 GHz	1.063 Ghz	8.50 GHz

Specifications

Index