BitifEye
Digital Test Solutions

ValiFrame N5990A Option 005

Integrated BER Counter Interface Support

Product Description

Content

IBerReader Interface

The IBerReader interface is a .NET software interface which allows to communicate with properitary external tools to perform automated tests with ValiFrame. It is available for MIPI D-Phy, M-Phy, and SATA Rx,, HDMI, DisplayPort, MHL, PCIe, USB testing. It contains methods, which will be called by ValiFrame during test execution to configure the device under test (DUT) and request the pass/fail information from the DUT. A dll will be loaded at run time and a class will be instantiated, which supports the IBerReader interface. The main function for getting the pass/fail information is the method GetCounter(out double bitCounter, out double errorCounter). In the following chapter the definition is shown.

BerReader Interface Definition

```
using System;
using System.Collections.Generic;
using System.Text;

namespace BerReader
{
    public interface IBerReader
    {
        /// <summary>
        /// This method is called to connect to your error reader.
        /// </summary>
        /// <param name="address">The address string can be used by your implementation
        /// to configure the connection to the IBerReader interface</param>
        void Connect(string address);

        /// <summary>
        /// This method is called to close the connection
        /// </summary>
        void Disconnect();

        /// <summary>
        /// This method is called prior to individual tests to select the channels under
        /// test and the test mode. It can be used to load pre-defined settings.
        /// </summary>
```

```
        /// <param name="mode"> Defines the test mode during the test. The modes are
        /// Clock_HS, DataX_HS, DataX_LP, DataX_ULP with X: 0-<number of
    /// data lines -1></param>
        void Init(string mode);

        /// <summary>
        /// Is called at the beginning of the error measurement and allows
        /// a reset for the DUT to be implemented.
        /// </summary>
        void ResetDut();

        /// <summary>
        /// Starts the counters. This method MUST reset all counters!
        /// </summary>
        void Start();

        /// <summary>
        /// Stop the DUT to read out the counters (see
        /// GetReadCounterWithoutStopSupported()).
        /// </summary>
        void Stop();

/// <summary>
        /// This method returns counters, the 1st counting the bits/frames/lines
        /// or bursts and the 2nd one counting the errors detected by the MipiBerReader.
        /// The automation software will compute the BER using the following
        /// equation BER=errorCounter/bitCounter. In the case bitCounter = 0 even when
        /// the stimulus is sending data, this is also interpreted as fail.
        /// </summary>
        /// <param name="bitCounter"> Contains the number of bits which are received
        /// by the DUT. If it is not possible to count bits the value can also contain
        /// frames, or bursts. It is just a matter of the value defined as target BER.
        /// If it is not possible to get the number of bits/frames/bursts then the
        /// method can return a value of -1 and the automation software can compute
        /// the number of bits from the data rate and the runtime.</param>
        /// <param name="errorCounter"> Total number of errors since the last start.
        /// </param>
```

```
        void GetCounter(out double bitCounter, out double errorCounter);

        /// <summary>
        /// This method returns a Boolean value indicating whether the device
        /// supports reading the counters while it is running. If this method
        /// returns false, the device needs to be stopped to read the counters.
        /// In this case the automation software will stop data transmission
        /// before calling the GetCounter() function, and re-start data transmission
    /// again after reading the counter values.
        /// </summary>
        /// <returns> false if device needs to be stopped before reading the counters,
        /// true if the counters can be read on the fly.</returns>
        bool GetReadCounterWithoutStopSupported();
        /// <summary>
        /// This property returns a number to multiply the value delivered by the
        /// bitCounter in the GetCounter() function.
        /// </summary>
        Double NumberOfBitsPerFrame {set; get;};

        /// This property returns the number of payload
        /// bits in a frame used for the detection of the BER.
        /// If i.e. the errorCounter in the GetCounter() function is just the
        /// checksum error then this parameter is the number of the payload.
        /// </summary>
        double NumberOfCountedBitsPerFrame {set; get;};

    }
}
```

IBerReader Usage by ValiFrame

Each of the methods and properties of this interface will be used during test execution. To understand the meaning and duty of the functionality it is required to understand at which point of a test these functions will be called. A helpful source for understanding the meaning are the comments above of each methods in the previous chapter “IBerReader Interface Definition”.

Integration

Copy your compiled version, which is a dll, into the ValiFrame program files folder. Each application (MIPI, MPhy, SATA) has a separate program files folder which is named like the application name. By name of the dll (MIPI D-Phy: MipiCustomBerReader.dll, MIPI M-Phy: MPhyCustomBerReader.dll, SATA: SataCustomBerReader.dll) it will be identified and loaded. If the loading is successful, a new entry in the BerReader list of the Configure DUT dialog is visible. After selection of the “Custom BER Reader” the address field will be editable, and the text of the address field will be used as argument for the Init(string address) method.

Connect/Disconnect

If the operator presses the start button in the ValiFrame user interface, which is the start of the execution of the test list the Connect(string address) will be called, before the execution of the first test. If this fails with an exception, the test automation aborts the execution. In it is needed to check why the connect function did not work. Check the functionality by using the Test GUI with the same text in the address field. At the end of each run the Disconnect() method will be called. A run is the time between the press on the start button and the final completed dialog, which is shown at the end. In case that the repetition parameter is >0 the Disconnect() will be called after the last repetition.

Init

Init(string mode) will be called once at the beginning of each test. The default parameters are set for the signalling and it is expected that the DUT can handle these default levels and timings at the RX side.

The content of the argument string “mode” is application dependent and needs to be requested from the application specific documentation. A more direct way to find out the exact string which is given as mode is to put a break point in the init method, and see what ValiFrame is setting for each test. In some applications like M-Phy the argument of the mode string is available via the property grid and can even be modified for each test individually.

ResetDUT and GetCounter

ResetDut() will be called just before each test point. It should be used to reset the bit and error counter and re-initialize the DUT to be ready for testing. A RX test procedure requires in the most cases several test points. For example, during a voltage sensitivity test several voltages will be set and for each test point a **ResetDUT()** will be called. After the **ResetDUT()** the test automation will wait with the **GetCounter(out double bitCounter, out double errorCounter)**

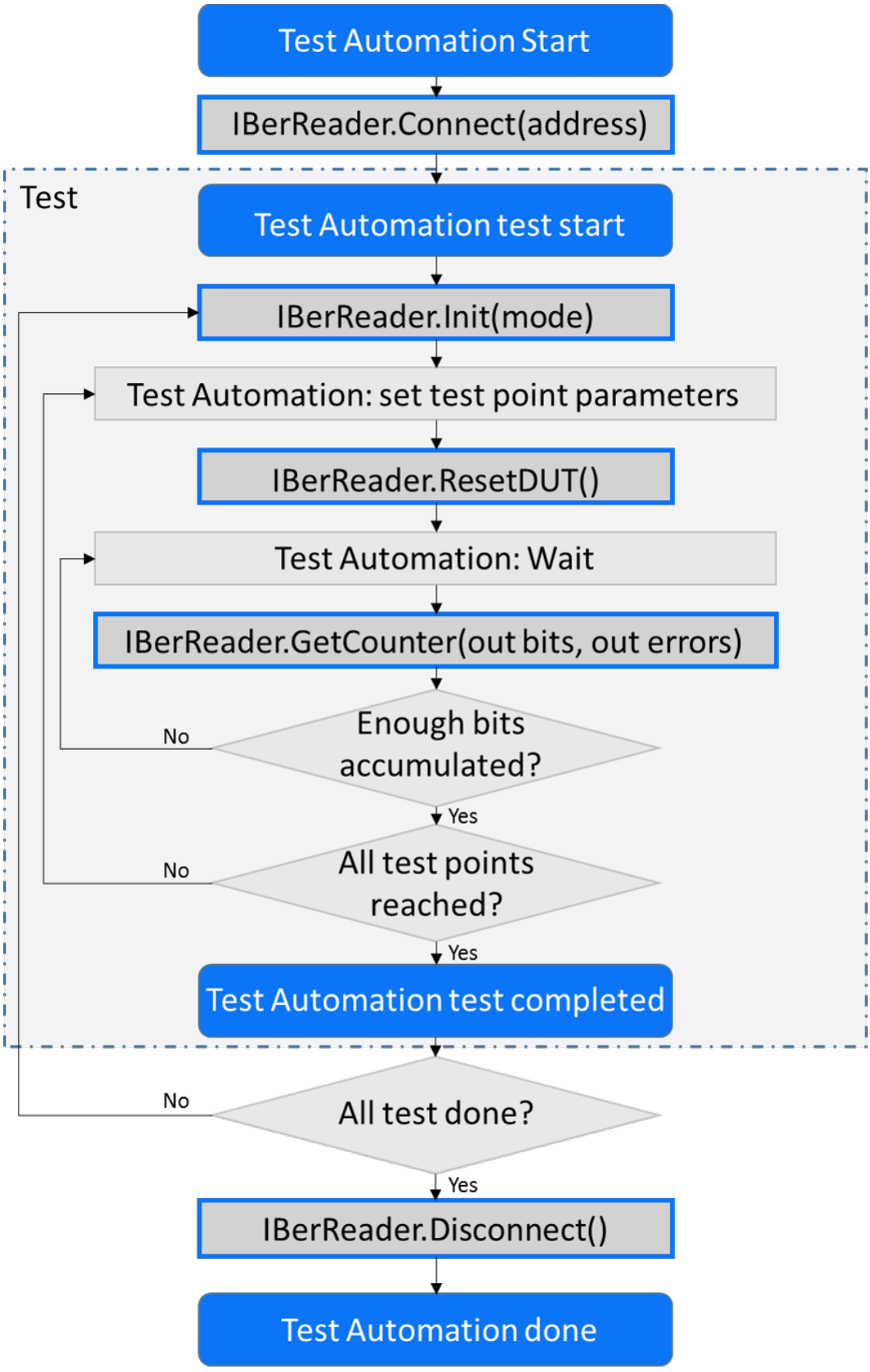
until the expected number of bits are transmitted. The number of bits are depending on the target bit error rate and the confidence level which needs to be reached. As a simple calculation the number of bits which needs to be compared is about three times of the target BER. Example target BER: 1e-10, Data Rate 1 GBit/s → 3e10 bits are required → 30 seconds test time is needed for each test point.

For a target BER below 1e-9 it may be suitable to request the bit and error counter before the theoretical integration time is reached to speed up the testing in case that errors are already visible. In this case the test automation will call the GetCounter() method several times for each test point. It is expected that the bit and the error counters will not be reseted by these calls.

Configuration and Conditions

GetReadCounterWithoutStopSupported() will be called before of each test point. If the return value is true, then no Stop() and Start() method will be called. For some implementation it is necessary to stop the execution of the bit comparison (see the Logic Analyser Example), before it is possible to read out the counter. **NumberOfBitsPerFrame** and **NumberOfCountedBitsPerFrame** will be used to compute the test time and BER. By these properties it is possible to give a frame counter instead of a bit counter in the **GetCounter()** function. The BER will be calculated in this case by multiplying the bit counter with the **NumberOfCountedBitsPerFrame**. In applications in which the data stream contains blanking periods, or bits which will not be taken into account for the bit comparison, the **NumberOfBitsPerFrame** and **NumberOfCountedBitsPerFrame** will be different, and the test time will be extended. The values for these parameters depend on the pattern which is used for testing. For some applications the test pattern is well defined, but the test automation allows to use another one. In this case the **IBerReader** should provide suitable values for these parameters.

Figure 1: Flow chart of the IBerReader usage in a test run



Example Code Description

The SDK for implementing the IBerReader interface comes with several example Visual Studio Projects and a project containing a test user interface, to test the implementation beforehand. The projects are:

- IBerReaderTestGui: project for building the IBerReader Test GUI. Via this GUI the functionality of the own implementation can be tested.

1. LogicAnalyserBerReader: implementation using a logic analyser as configuration tool and error detector.

- MipiCustomBerReader: example code with empty method implementations, which can be used as start for the own implementation.
- OfflineBerReader: example code for a offline BER reader. Instead of a direct access to a tool that configures the DUT, dialogs are shown to let the user do the configuration, and finally ask, if the DUT is working properly.

All projects require two references, one to VFBase and the other to VFInstruments, which are libraries of ValiFrame. They can be added by do a right-clicking on the References folder of the Visual Studio Project and select “Add References...”. By doing so a dialog pops up in which the user can select the “Browse...” tab and select VFBase.dll and VFInstruments.dll from the ValiFrame Program Files folder.

IBerReader Test GUI

The Test GUI allows to test the own implementation of a CustomBerReader. The GUI allows to execute all methods of the own IBerReader implementation without having them integrated into ValiFrame alreadey to separate the development and debugging from the ValiFrame integration. Since the source code is available debugging can be done via this user interface.

VisaInstrumentBerReader

The *VisaInstrumentBerReader* project is a project containing an empty CustomBerReader class. All necessary methods and properties are available but just contains a throw new Exception(“Not implemented.”) call. This project can be used as template for implementation.

OfflineBerReader

The OfflineBerReader project contains the implementation of an “offline” version of a IBerReader supporting class. All methods will show a dialog to the operator instead of doing a

call to the DUT or tools directly. It can be used to see when and how the IBerReader methods are used inside of the test automation.

LogicAnalyserBerReader

The LogicAnalyserBerReader IBerReader implementation is using the ValiFrame Logic Analyser driver for controlling an Agilent 16900A Logic Analyser. ValiFrame is using a .Net Remoting Server running on the LA for accessing instrument settings, stopping and starting the analyser and generator modules, and accessing the trigger counter. It is the same implementation which will be used in MIPI D-Phy for controlling the DUT via the PPI interface. In this application, the LA is configuring the device via the pattern generator modules, and the analyser modules are connected to the parallel interface of the DUT. Via this parallel interface the same data which are received via the high speed D-Phy interface are sampled, and via a special trigger setup the pattern is compared. The counter of the trigger are used for gaining the number of received bursts, and the number of errors. The configuration of the DUT is done by loading and executing a LA pattern generator setup file, and the setup for the analyser contains the necessary port assignment for the PPI. It is in the hand of the customer, to create a suitable setup for the actual DUT.

DpCustomBerReader

The *DpCustomBerReader* is designed for DisplayPort devices. Because of the extended communication requirements during the DisplayPort link training procedure, this CustomBerReader is a bit more advanced. For detailed information see the document, which is shipped together with the example code of the *DpCustomBerReader* .

Debugging

Debugging using the Test GUI

The easiest way for debugging is using the Test GUI (see chapter IBerReader Test GUI). The source code is available and therefore it is possible to set break points in each method which is using the IBerReader functionality. If the developer puts the project of the IBerReader implementation in the same Visual Studio solution, then it will also follow the source lines of the IBerReader class in case of stepwise running the debugger (Key F10, step over, and F11, step into). To get a reasonable signal at the inputs of the DUT the test automation, a frame generator, or any other signal source, which can generate specification conform signals can be used. If the test automation is used, then either the offline BER reader can be used to stop the execution of the test at each IBerReader call, or the compiled dll. If the compiled dll is used it

makes more sense to use the test automation as startup application for debug than the Test GUI. Using the frame generator as controller for the signal sources is the most flexible way, because by the frame generator all parameters can be set in the optimum range to test the error free behavior of the DUT, and setting one parameter at the edge of the DUT capability, the error counter functionality can be tested. As minimum requirement the VFBase.dll and VFInstrument.dll needs to be copied from the ValiFrame program files folder and referenced.

Debugging using ValiFrame

The second way is to use ValiFrame (N5990A, ValiFrame.exe) as startup application (debug tab in the BerReader project properties). In this case the compiled dll needs to be copied into the ValiFrame program files folder, and the Custom BER Reader needs to be selected in the Configure DUT dialog of ValiFrame, to get calls into the BerReader class methods. If breakpoints are set into the method implementation, the arguments and timings of each of the methods can be monitored. The disadvantage of using this method is that it always requires a full functional ValiFrame installation and license. For development, which is done very often on a separate PC, without ValiFrame installation.