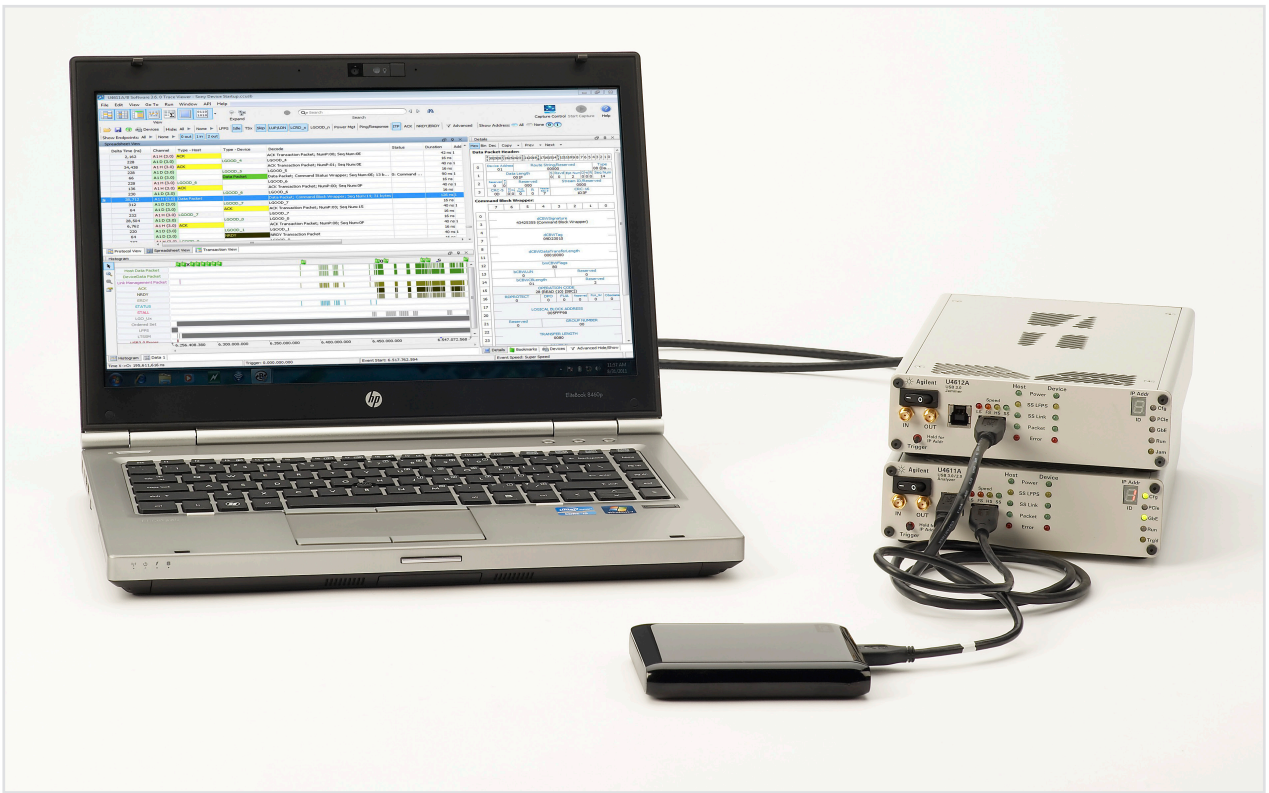
USB 3.0 Protocol Testing with Active Error Insertion

Application Note

Speed up design and verification of USB designs using the U4612A jammer

Discovering the source of errors in today's complex USB 3.0 designs without the right tools is a time consuming task. State-of-the-art designs in mobile computing and high speed digital applications are now adopting SuperSpeed USB 3.0, and require fast and easy insight into conditions that cause design flaws. Traditional debug methods relied on protocol analyzers and difficult error simulation techniques. Errors were created through the lengthy process of writing scripts and simulating errors in post-processed data. This was both a cumbersome and inefficient process, with more time spent on trying to create and insert the error, than studying the effects and the possible causes of an error. Jammers enhance the power of today's protocol analysis tools, by adding active error insertion into the live data stream. This added tool gives engineers the powerful ability to troubleshoot USB designs like never before.



Anticipate ___Accelerate ___Achieve

Agilent Technologies

Active Insertion

Active error insertion allows developers to quickly and easily create error conditions that simulate today's real world challenging design environments. Testing and verifying the error detection and system recovery are greatly enhanced with the addition of a jammer in the design and debug phases. Methods to troubleshoot designs and perform active link testing can be divided into three different methods.

1. Golden device

Common and useful; the golden device method provides the first level of verification that things are working as expected. Golden device simply involves selecting typical device deployment configurations, and then recreating that common scenario. This approach does not allow you to create the error conditions that will occur within the large variety of deployment environments that products will encounter. It simply verifies operation under ideal conditions for a given device configuration.

2. Generator

The generator method involves recreating very tightly controlled test sequences. It allows manual creation of the exact protocol packets for a specific test and applies it to the device under test. The level of control is high, but the time required to develop the test is also high due to the need to create custom scripts for each error condition. It is almost impossible to create a generic test that can be used with all devices. Continuous tuning and optimizing of the test scripts is required. The generator method is useful for testing the link layer operations but cannot be programmed to modify operations in the Transactions, End point, or Device class operation.

3. Active insertion

With Active Insertion, the developer uses a "golden device" to perform all of the normal protocol operations, but also connects a jammer in the middle of the physical connection between the host and device under test (Figure 1). This allows the engineer to make controlled error modifications in real-time and instantly analyze the effect on the device's response. With active insertion, modifications can be created at low levels of communication as well as with the device operation. The possibilities of testing are much greater compared with using a generator that only sends out specific packets.

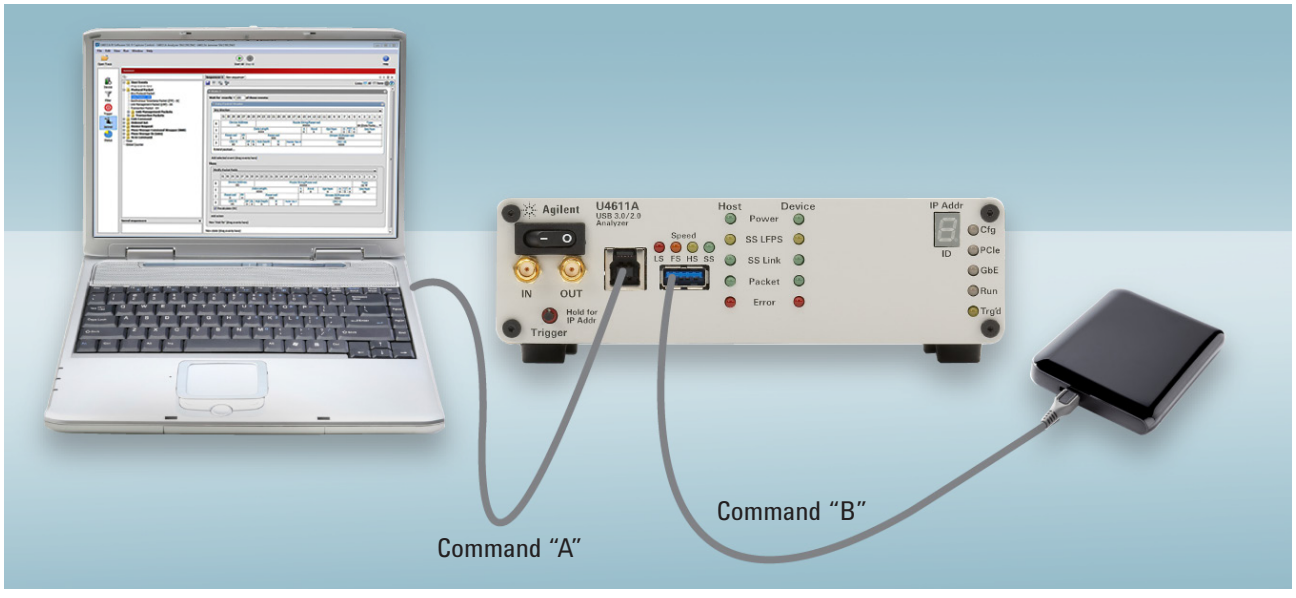


Figure 1. Jammers can modify the commands sent by the host or device in real time to recreate errors or to stress protocol operations.

Advantages of Active Insertion Testing

In normal USB operation a device is connected into an active USB port and both host and device execute flawlessly. Unfortunately the normal setup gives limited visibility into the protocol operation on this link. Therefore the first step is to insert a protocol analyzer into this link to capture and display what is happening on this link. Active insertion takes this process one step further by allowing the developer to insert errors into an active link and change this communication in real time.

Uses of Active Insertion Testing

The ability to quickly change the information on a live link provides the ability to push the design to its limits. Active error insertion is an unprecedented capability. It is revolutionary in that it allows designers to create real-time, user-defined errors, thus replicating a wide range of anticipated real and theoretical fault conditions. This capability is driving improved SuperSpeed design, test criteria, and debug capability, as well shortening design cycles. Those who bring a product to market quickly often maintain a competitive advantage. Losses in productivity directly translate to losses in product cycle time, sales, and market advantage.

For today's USB developers and integrators, verifying design robustness and error recovery can be a challenging task, especially when it is difficult to duplicate test cases and customer issues. New designs require quick, powerful, and easily created user-defined tests that explore a design's outer limits. Pushing a system's design to the full extent with powerful error sequences can quickly uncover future surprises. Creating consistent test sequences helps ensure the confidence needed for product release. The Agilent U4612A USB 3.0 Jammer gives designers the ability to insert a variety of errors into a live data stream to test real-time error handling, system recovery, and duplication of issues seen in the field. The Jammer's error injection allows the creation of random and defined line errors such as CRC or 8b/10b encoding errors, modify or replace frames, frame data and link management packets.

In this application note we will be creating test scripts using the Agilent GUI. It is very similar to the Agilent USB protocol analyzer software to maintain ease of use and without the need to learn new interfaces. The following examples provide just a few of the possibilities for testing.

Creating a Simple Error Injection Script: Verifying 10-bit Error Detection and Link Recovery

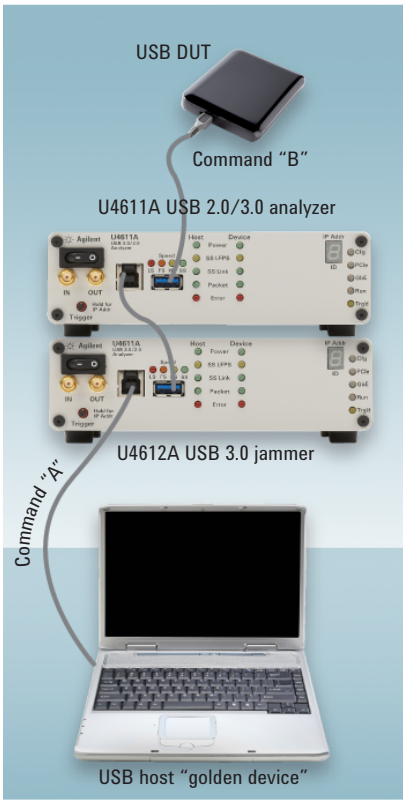


Figure 2. Testing the ability of the device to recover from bit errors on the link

In our first example, we will test the ability of the device to recover from bit errors on the link. Everything about your design works to minimize errors, so let's create a 10-bit error in 3 consecutive data packets, and analyze the design's response. According to the USB 3.0 specification, this should cause our link to go into a recovery state, retrain and then resume communication. Here is a look at our test setup. We connect the Analyzer and Jammer in series with the connection. The order of these connections will affect the visibility of data that is monitored. In this layout, see Figure 2, the analyzer will capture the data that is modified by the Jammer that is going to the DUT (device under test), but not the data that is modified from the DUT to the host. The jammer can modify the traffic in either direction, but the order of the USB connection will affect the data that is visible to our analyzer.

Our test goal is to verify that our device will transition to link recovery if we receive 3 data packets in a row that contain 10-bit errors.

- **Step 1**: Configure the host and DUT to enter a valid data transfer state, by starting a data file copy (or other functions that will transfer data as appropriate).
- **Step 2**: Configure the analyzer to inject a 10-bit error on 3 consecutive data packets and then trigger to stop the test.

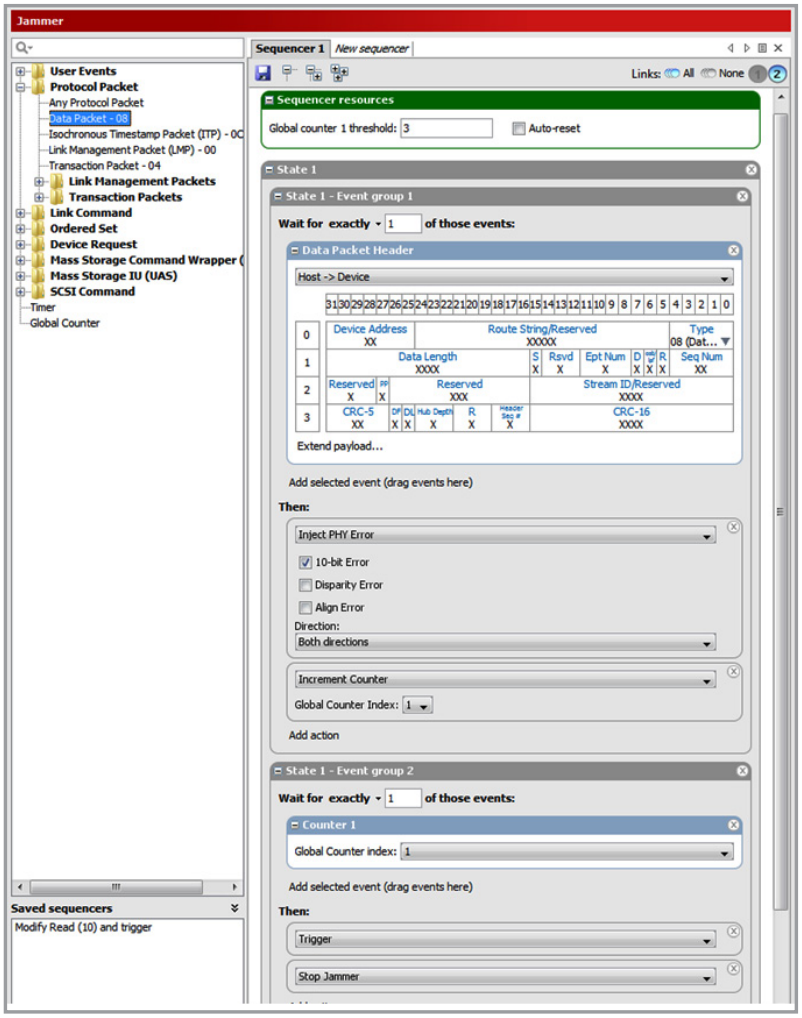


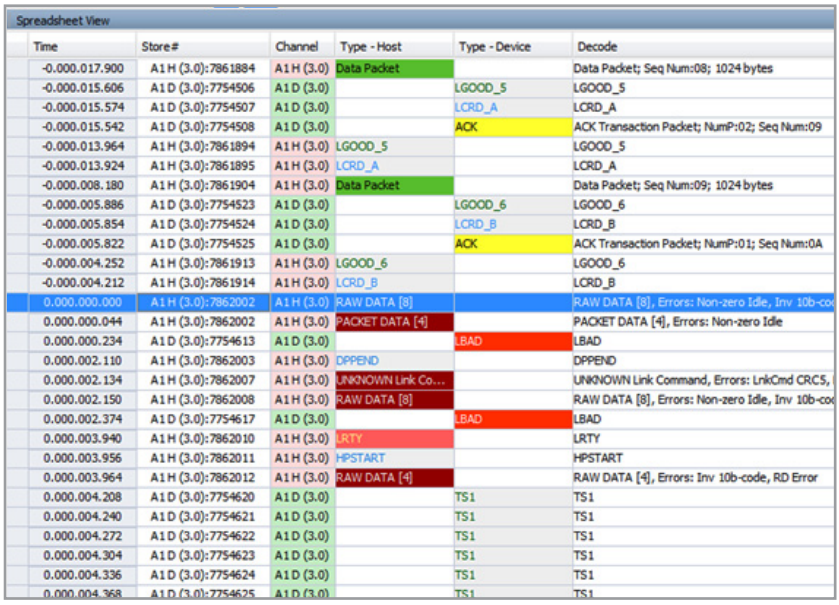
Figure 3. A one-state program that uses a global counter. We tell the analyzer to wait until it detects a data packet sent from the host to the device, then inject a 10-bit error into the packet.

It then increments the global counter and waits for another data packet.

It is also checking to see if the global counter has reached its threshold. If so, then trigger to stop the capture, and stop the jammer.

In the capture buffer, we can see that three errored data packets were indeed sent, and the device correctly reported that there was a problem by sending LBAD (link bad) packets to the host.

The host then (correctly) responded that it would retry the link. However after the three errored packets, the device immediately entered link recovery, as can be seen by the sending of the TS1 training sequence packets. We have now verified that the device is able to correctly respond to 10-bit errors that may occur during data transfers.



Time	Store#	Channel	Type - Host	Type - Device	Decode
-0.000.017.900	A1 H (3.0):7861884	A1 H (3.0)	Data Packet		Data Packet; Seq Num:08; 1024 bytes
-0.000.015.606	A1 D (3.0):7754506	A1 D (3.0)		LGOOD_5	LGOOD_5
-0.000.015.574	A1 D (3.0):7754507	A1 D (3.0)		LCRD_A	LCRD_A
-0.000.015.542	A1 D (3.0):7754508	A1 D (3.0)		ACK	ACK Transaction Packet; NumP:02; Seq Num:09
-0.000.013.964	A1 H (3.0):7861894	A1 H (3.0)	LGOOD_5		LGOOD_5
-0.000.013.924	A1 H (3.0):7861895	A1 H (3.0)	LCRD_A		LCRD_A
-0.000.008.180	A1 H (3.0):7861904	A1 H (3.0)	Data Packet		Data Packet; Seq Num:09; 1024 bytes
-0.000.005.886	A1 D (3.0):7754523	A1 D (3.0)		LGOOD_6	LGOOD_6
-0.000.005.854	A1 D (3.0):7754524	A1 D (3.0)		LCRD_B	LCRD_B
-0.000.005.822	A1 D (3.0):7754525	A1 D (3.0)		ACK	ACK Transaction Packet; NumP:01; Seq Num:0A
-0.000.004.252	A1 H (3.0):7861913	A1 H (3.0)	LGOOD_6		LGOOD_6
-0.000.004.212	A1 H (3.0):7861914	A1 H (3.0)	LCRD_B		LCRD_B
0.000.000.000	A1 H (3.0):7862002	A1 H (3.0)	RAW DATA [8]		RAW DATA [8], Errors: Non-zero Idle, Inv 10b-co
0.000.000.044	A1 H (3.0):7862002	A1 H (3.0)	PACKET DATA [4]		PACKET DATA [4], Errors: Non-zero Idle
0.000.000.234	A1 D (3.0):7754613	A1 D (3.0)		LBAD	LBAD
0.000.002.110	A1 H (3.0):7862003	A1 H (3.0)	DPPEND		DPPEND
0.000.002.134	A1 H (3.0):7862007	A1 H (3.0)	UNKNOWN Link Co...		UNKNOWN Link Command, Errors: LnkCmd CRC5,
0.000.002.150	A1 H (3.0):7862008	A1 H (3.0)	RAW DATA [8]		RAW DATA [8], Errors: Non-zero Idle, Inv 10b-co
0.000.002.374	A1 D (3.0):7754617	A1 D (3.0)		LBAD	LBAD
0.000.003.940	A1 H (3.0):7862010	A1 H (3.0)	LRTY		LRTY
0.000.003.956	A1 H (3.0):7862011	A1 H (3.0)	HPSTART		HPSTART
0.000.003.964	A1 H (3.0):7862012	A1 H (3.0)	RAW DATA [4]		RAW DATA [4], Errors: Inv 10b-code, RD Error
0.000.004.208	A1 D (3.0):7754620	A1 D (3.0)		TS1	TS1
0.000.004.240	A1 D (3.0):7754621	A1 D (3.0)		TS1	TS1
0.000.004.272	A1 D (3.0):7754622	A1 D (3.0)		TS1	TS1
0.000.004.304	A1 D (3.0):7754623	A1 D (3.0)		TS1	TS1
0.000.004.336	A1 D (3.0):7754624	A1 D (3.0)		TS1	TS1
0.000.004.368	A1 D (3.0):7754625	A1 D (3.0)		TS1	TS1

Figure 4. Capture buffer showing errored data packets (device responds LBAD - Link BAD).

Additional simple verification test scenarios can be created that are very similar to this. For example:

- One of the 10-bit errors occurs during a SKP message rather than a data packet.
- Two data packets with a 10-bit error, then one good packet, then two more errored.
- Multiple 10-bit errors in a single data packet.
- Creating additional 10-bit error tests.

All of these conditions can easily be created with simple changes to the test script.

For another example, let's test the effect of a single 10-bit error. The particular device that causes failure is to create a single 10-bit error, but this time we will place the error in the "Set_Configuration" command from the host during device enumeration.

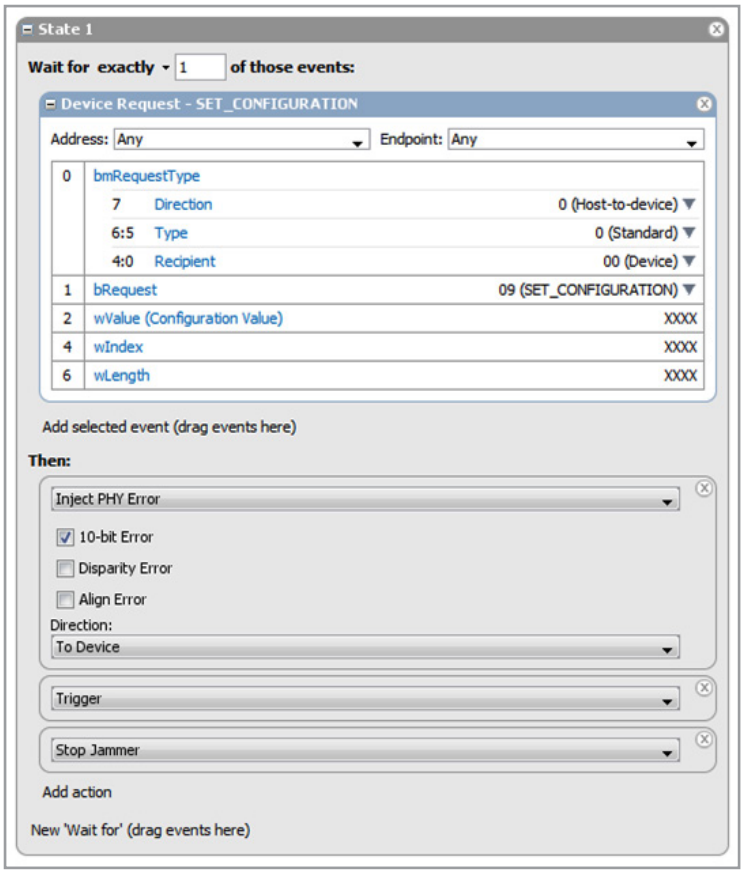


Figure 5. Creating multiple error test scenarios is simple and fast

In the packet capture we see that the set configuration packet was sent, but it has an error. The device sent an ACK response, asking for the same packet again (Seq Num 00), which the host then retransmits, but then checks that status, and immediately takes the device offline with a LGO_U3.

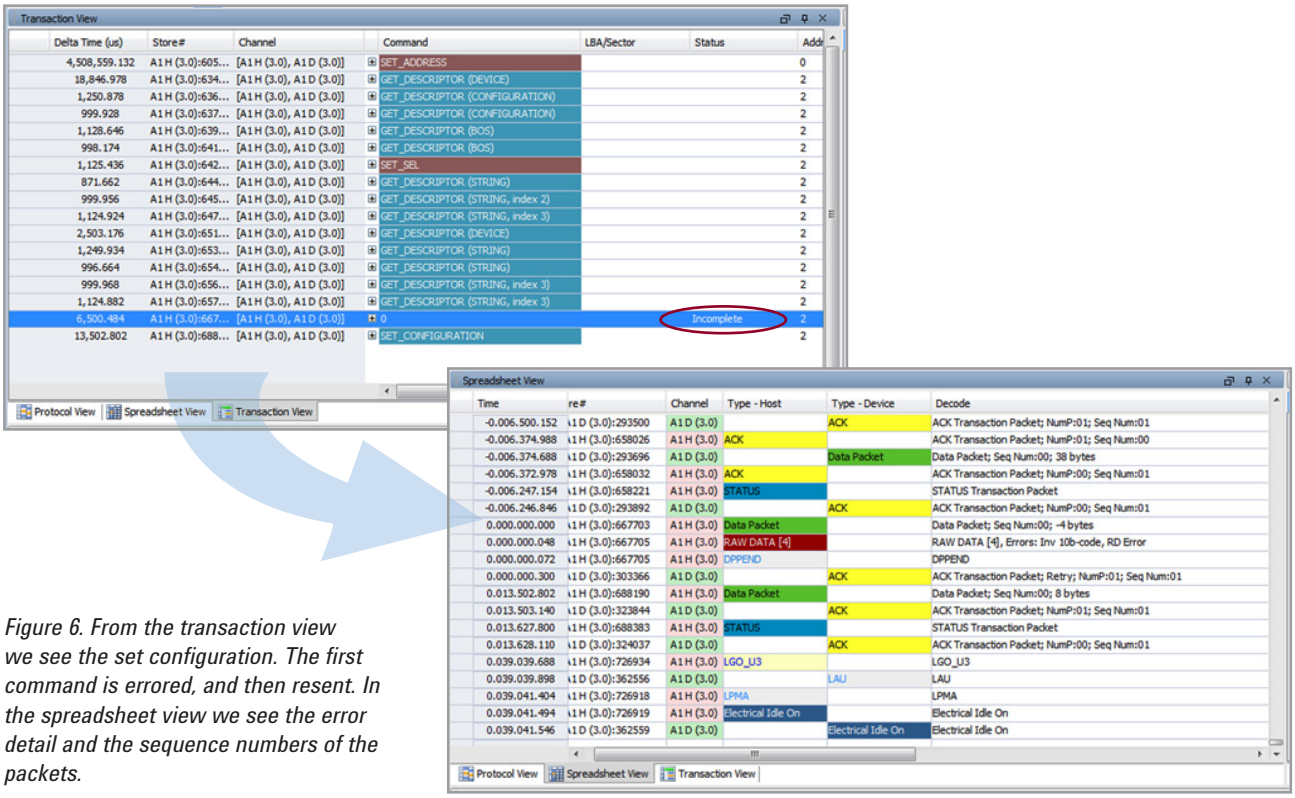


Figure 6. From the transaction view we see the set configuration. The first command is errored, and then resent. In the spreadsheet view we see the error detail and the sequence numbers of the packets.

Inserting 8b/10b errors

Normal operations should have a very low error rate (10-12 BER), but errors do happen. When using active insertion, is it simple to create these errors at any phase of the protocol operation to test and verify the device's response to these errors.

From the USB 3.0 specification: "There are two types of errors when a receiver decodes 8b/10b symbols. One is a disparity error that is declared when the running disparity of the received 8b/10b symbols is not +2, or 0, or –2. The other is a decode error when an unrecognized 8b/10b symbol is received.
Upon receiving notification of an 8b/10b error:

- A port may optionally do the following:
  1. If the link is receiving a header packet, it shall send LBAD.
  2. If the link is receiving a link command, it shall ignore the link command.
  3. If the link is receiving a DPP, it shall drop the DPP.
- The link error count shall remain unchanged."

Testing the Link Layer

Link layer testing begins to focus on the exchange of messages between the logic processes of the host and device. For example, before a device can send data it must receive confirmation that the host has buffers available in which to receive the data. See Link Credit LCRD_x in the USB 3.0 specification.

The link layer commands fall into 4 categories:

- Power management (LGO_Ux commands to change the power state)
- Packet transfer success (LGOOD_X, packet sequencing)
- Link flow control (LCRD_x, signaling available receive buffers)
- Valid U0 link ready (data transfer state, LUP/LDN or other packet every 10 µsec)

The USB Implementers Forum Compliance Committee added link layer testing to the SuperSpeed USB certification program as part of our ongoing effort to ensure comprehensive compliance for the USB 3.0 standard. See **http://www.usb.org/developers/ssusb/testing/ .**

The USB-IF's link layer compliance specification includes 37 test cases that verify hundreds of link layer test assertions from the USB 3.0 base specification. Most of these tests are easily implemented by using the Agilent USB jammer. These tests analyze the proper sequencing, using valid timeouts, and recovery from errors. The following list of the 37 tests comprise link layer compliance test.

Universal Serial Bus 3.0

Link Layer Test Specification (List of Compliance tests)

5.3 Link Layer
- TD.7.1 Link Bring-up Test
- TD.7.2 Link Commands Framings Robustness Test
- TD.7.3 Link Commands CRC-5 Robustness Test
- TD.7.4 Invalid Link Commands Test
- TD.7.5 Header Packet Framing Robustness Test
- TD.7.6 Data Payload Packet Framing Robustness Test
- TD.7.7 RX Header Packet Retransmission Test
- TD.7.8 TX Header Packet Retransmission Test
- TD.7.9 PENDING_HP_TIMER Deadline Test
- TD.7.10 CREDIT_HP_TIMER Deadline Test
- TD.7.11 PENDING_HP_TIMER Timeout Test
- TD.7.12 CREDIT_HP_TIMER Timeout Test
- TD.7.13 Wrong Header Sequence Test
- TD.7.14 Wrong LGOOD_N Sequence Test
- TD.7.15 Wrong LCRD_X Sequence Test
- TD.7.16 Link Command Missing Test (Upstream Port Only)
- TD.7.17 tPortConfiguration Time Timeout Test
- TD.7.18 Low Power initiation for U1 test (Downstream Port Only)
- TD.7.19 Low Power initiation for U2 test (Downstream Port Only)
- TD.7.20 PM_LC_TIMER Deadline Test (Downstream Port Only)
- TD.7.21 PM_LC_TIMER Timeout Test (Downstream Port Only)
- TD.7.22 PM_ENTRY_TIMER Timeout Test (Upstream Port Only)
- TD.7.23 Accepted Power Management Transaction for U1 Test (Upstream Port Only)
- TD.7.24 Accepted Power Management Transaction for U2 Test (Upstream Port Only)
- TD.7.25 Accepted Power Management Transaction for U3 Test (Upstream Port Only)
- TD.7.26 Transition to U0 from Recovery Test
- TD.7.27 Hot Reset Detection in Polling Test (Upstream Port Only)
- TD.7.28 Hot Reset Detection in U0 Test (Upstream Port Only)
- TD.7.29 Hot Reset Initiation in U0 Test (Downstream Port Only)
- TD.7.30 Recovery on three consecutive failed RX Header Packets Test
- TD.7.31 Hot Reset Failure Test (Downstream Port Only)
- TD.7.32 Warm Reset Rx.Detect Timeout Test (Hub Downstream Port Only
- TD.7.33 Exit Compliance Mode Test (Upstream Port Only)
- TD.7.34 Exit Compliance Mode Test (Downstream Port Only)
- TD.7.35 Exit U3 by Reset Test (Downstream Port Only)
- TD.7.36 Exit U3 Test (Host Downstream Port Only)
- TD.7.37 Packet Pending Test (Upstream Port Only)

Figure 7. List of 37 tests comprising the link layer compliance test.

To demonstrate how to test the link layer protocol, consider testing TD 7.14 Wrong LGOOD_X Sequence. This test verifies that the DUT will go to recovery when it receives an incorrect LGOOD_N sequence.

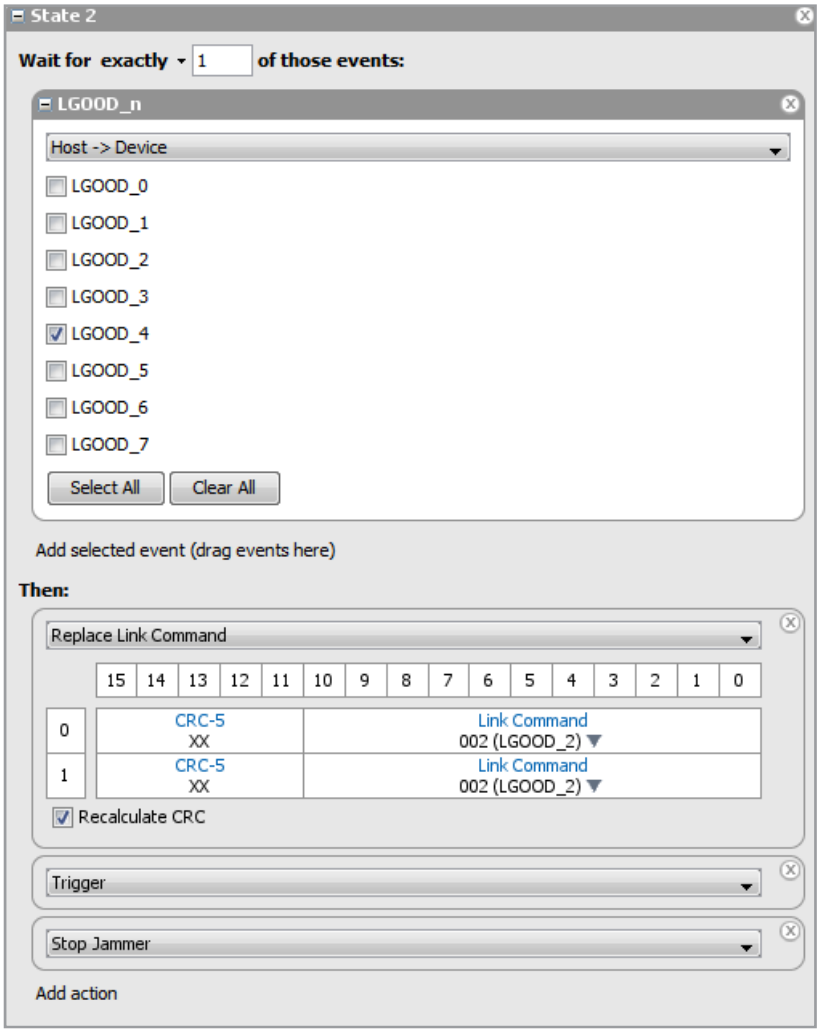


Figure 8. In this test script, we simply wait until there has been a user-defined number of valid sequences completed (state 1 of the script is not displayed), and then our script modifies one of the LGOOD_4 messages to be an LGOOD_2 then it triggers to let us know it finished and where to find the information.

As soon as the test executes we can see in the capture buffer that this device transitioned into recovery when the LGOOD sequence occurred out of order.

These tests work on one specific aspect of the link layer protocol at a time. They test the execution of the recovery processes when errors are detected and the availability of buffer credits, in addition to checking that the timeouts are correctly implemented.

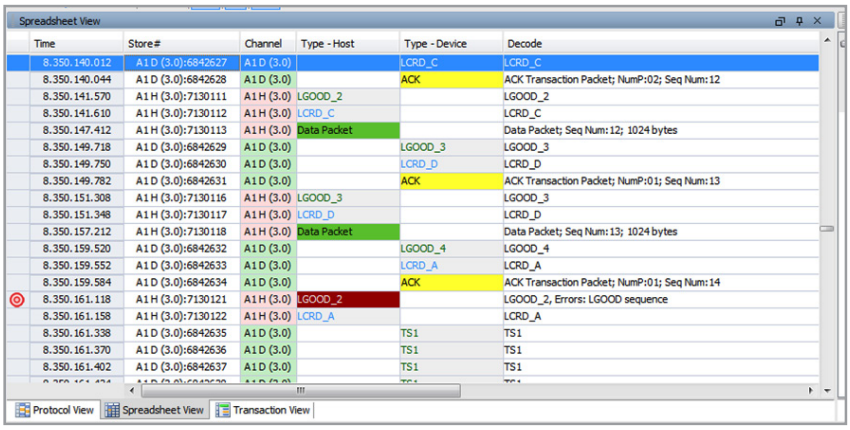


Figure 9. As you can see in the decode display, as soon as the LGOOD_2 message was received by the device (which was received after an LGOOD_3), it executed the recovery sequence (as indicated by the TS1 messages).

Testing Mass Storage Devices

Mass storage devices are the most common application of USB 3.0. The convenience and ease of use makes it the ideal interface to use for backup devices and storage to transport large files. While the USB connection has been in use for some time for this purpose, USB 2.0 devices used a variation of SCSI over USB called bulk only transport (BOT). While USB 3.0 enables greater speed, the BOT protocol is not able to take advantage of performance to match the current generation of devices. The solution to this is a new standard called USB attached SCSI (UAS). UAS uses the same basic command set as BOT, but has improved the efficiency by improving transfer size and reducing overhead, implementing queuing and high performance storage stacks. The USB jammer enables a wide variety of tests of these communication stacks. For example, sometimes the initiator may wish to have the blocks of data read from the medium instead of from the cache memory. The force unit access (FUA) bit is used to indicate that the direct-access device shall access the physical medium. For a write operation, setting FUA to 1 causes the direct-access device to complete the data write to the physical medium before completing the command. For a read operation, setting FUA to 1 causes the logical blocks to be retrieved from the physical medium.

Using the USB jammer allows testing to include utilizing the FUA and DPO flags to force the device to bypass cache and not return until the information is stored and all write operations are completed. On the write function, it forces the device to bypass the cache and not return until the information is physically written to the disc.

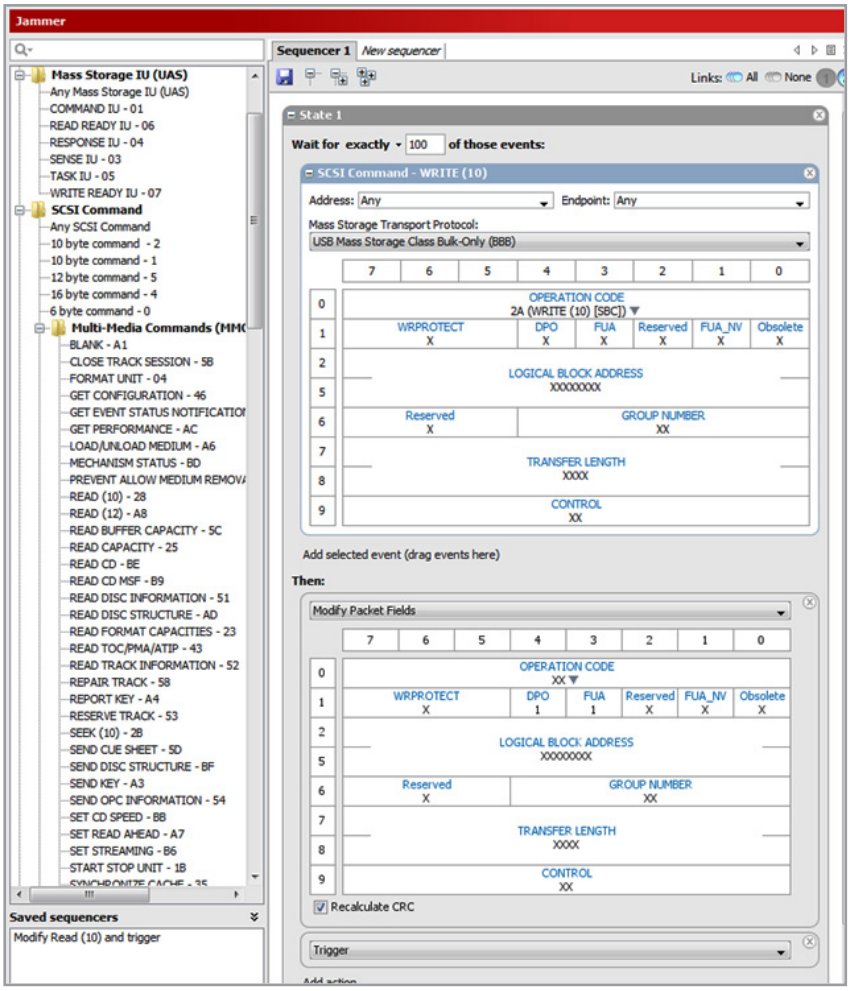


Figure 10. In this example, we are waiting until 100 write commands have executed and then setting the FUA and DPO bits in one write command, making it easy to see the performance hit of a single command.

In the decode display below, you can see that the majority of the write commands execute in approximately 1.4 microseconds. Each command writes 128 kB.

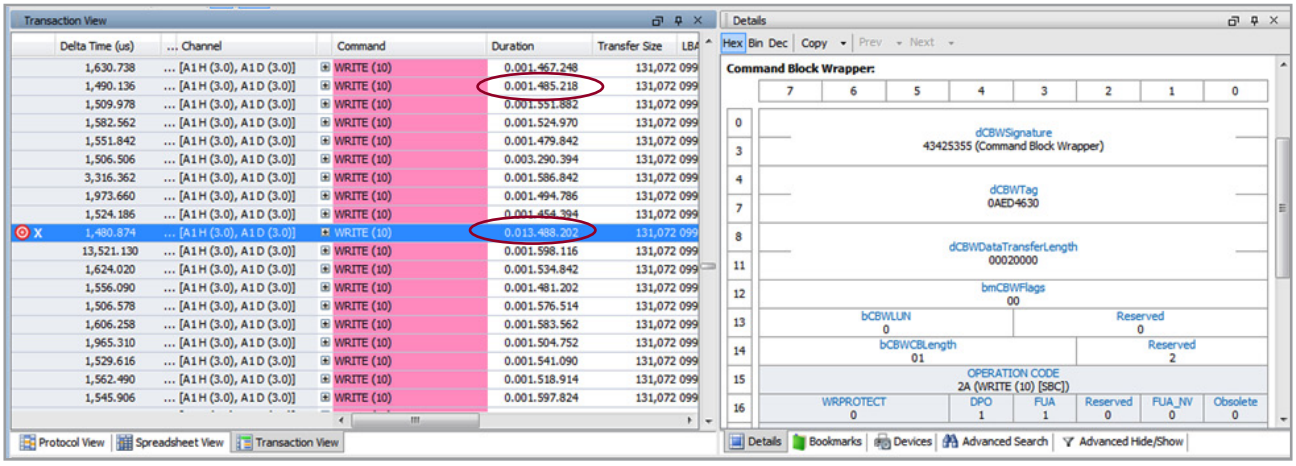


Figure 11. On the selected write command, we can see that the duration of the command is 13.5 microseconds. (Examination of additional transactions with the FUA and DPO bits set have similar duration times.)

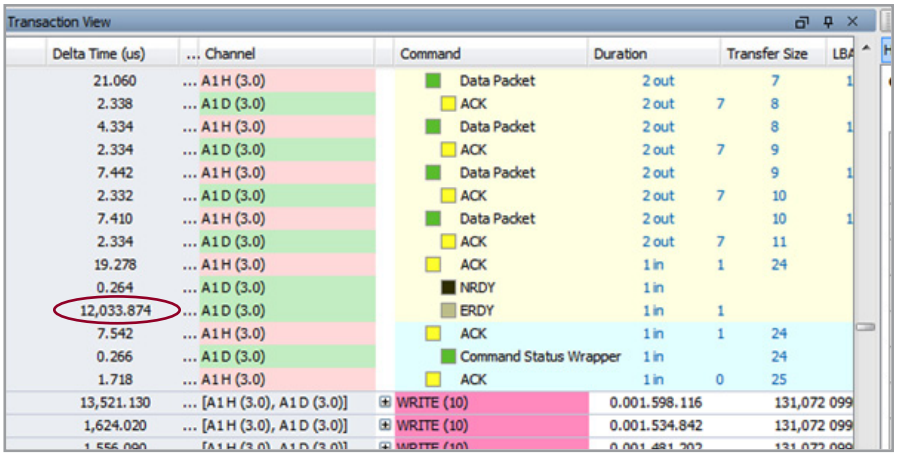


Figure 12. Looking at the detailed packets of the transaction show that the device accepted all of the packets, then sent a NRDY (not ready) message. It was then 12 microseconds later that the device was ready to proceed (as indicated by the ERDY (end device ready) message).

Summary

The Agilent USB jammer has a wide variety of uses that enable testing in ways that have not been available before. The powerful yet easy scripting language allows for the creation of very complex sequences to manipulate the data. The Agilent USB analyzer is the ideal tool to see the results of these tests.

The complete system has an easy-to-use GUI and API (application programming interface) that allows these test sequences to be automated and repeated. After device deployment into the market there are times when problems occur that cannot be replicated with normal device configurations. However, a protocol trace can be captured and delivered to support and R&D for diagnosis. It is critical that after examination of the protocol traces the cause of the failure be replicated in order to verify a resolution. The use of active insertion provides the ability to make a controlled environment for testing.

Figure 13. U4612A USB 3.0 jammer

www.agilent.com

For more information on Agilent Technologies' products, applications or services, please contact your local Agilent office. The complete list is available at:

www.agilent.com/find/contactus

Americas

Canada	(877) 894 4414
Brazil	(11) 4197 3600
Mexico	01800 5064 800
United States	(800) 829 4444

Asia Pacific

Australia	1 800 629 485
China	800 810 0189
Hong Kong	800 938 693
India	1 800 112 929
Japan	0120 (421) 345
Korea	080 769 0800
Malaysia	1 800 888 848
Singapore	1 800 375 8100
Taiwan	0800 047 866
Other AP Countries	(65) 375 8100

Europe & Middle East

Belgium	32 (0) 2 404 93 40
Denmark	45 45 80 12 15
Finland	358 (0) 10 855 2100
France	0825 010 700*
	*0.125 €/minute
Germany	49 (0) 7031 464 6333
Ireland	1890 924 204
Israel	972-3-9288-504/544
Italy	39 02 92 60 8484
Netherlands	31 (0) 20 547 2111
Spain	34 (91) 631 3300
Sweden	0200-88 22 55
United Kingdom	44 (0) 118 927 6201

For other unlisted countries:
www.agilent.com/find/contactus

Revised: January 6, 2012

Product specifications and descriptions in this document subject to change without notice.

Agilent Email Updates

www.agilent.com/find/emailupdates
Get the latest information on the products and applications you select.

Published in USA, March 19, 2012
5991-0097EN

Agilent Technologies