BER test method uses real data

Joshua Beaudet   - February 26, 2015

Today's methods for BER (bit-error-rate) testing of high-speed serial links such as PCIe and SATA rely on predetermined patterns that don't represent real-world situations. These patterns use a defined amount of jitter sent through a precise channel from a DUT's transmitter to its receiver; the DUT runs in a loopback state. Obtaining a valid BER measurement requires at least a trillion bits. The BER measurement is difficult to perform because of the test equipment needed and the setup to properly run the test.

A different method, developed at the UNH-IOL, uses real-world data. While more work needs to be done, the test method proposed here shows promise for using real-world data for testing digital communications systems such as PCIe and SATA. The advantage of the UNH-IOL-developed method is that it uses readily available equipment without the need for a pattern generator or BERT (bit-error-rate tester). It also eliminates the need for loopback testing and manually aligning clock domains.

Transceivers have different types of loopbacks that can affect clock recovery between the measurement device and DUT, producing mixed results. Unless the clocks are correctly synchronized, false errors can occur. Furthermore, test patterns don't represent realistic traffic. These issues compound to make an untrustworthy test.

The UNH-IOL set out to develop a test procedure that lets two PCIe and SATA devices send traffic to each other—thus removing loopback—while still injecting jitter into the system. Experiments show that the proposed method proposed can yield the same BER estimates to existing methods, but with real-world traffic and a less expensive test setup.

To verify the validity of the proposed test method, engineers at UNH-IOL calibrated each test setup to the specific signal settings. The existing methods give a straight BER measurement. The proposed method will deduce a BER from the CRC (cyclic-redundancy check) errors counted. Using the proposed test, we sent more than trillion bits of traffic between two devices. A jitter generator adds a controlled amount of jitter. The test channel used in this method has the same effects as those used in the traditional BERT-based methods.

**Background**

Engineers perform BER testing to guarantee with some certainty that a DUT will have a known BER when put in a real system. Having a higher BER means the SNR (signal-to-noise ratio) is higher and by Shannon Capacity Law, this means the system capacity goes down.

All serial-communication links have jitter, which affects BER. A stressed receiver eye test is the main way to establish a BER associated with some amount of jitter. The current test procedure involves receiving a jittered repeating test pattern from a test instrument, thus creating the stressed signal,
then retransmitting the recovered pattern by being in a loopback state back to the test station. Having a test pattern with added jitter at the receiver is important in making the guarantee of the certain BER. At data rates greater than 1 Gbit/s, a small change in jitter can cause a large change in BER.

Because BER goes hand in hand with jitter, the received signal must contain specific amounts of jitter to guarantee BER. Jitter can be decomposed into subcategories. TJ (total jitter) breaks into DJ (deterministic jitter) and RJ (random jitter), which is generally caused by thermal noise. Deterministic jitter is further broken down into BUJ (bounded uncorrelated jitter), DDJ (data-dependent jitter), and PJ (periodic jitter). BUJ is caused by coupling and crosstalk. PJ is the periodic variations in edge position over time, often caused by switching power supplies. Data-dependent jitter is further broken down into DCD (duty cycle distortion) and ISI (inter-symbol interference). These last two categories are dependent on the data pattern. Each form of jitter is found in a real system and must be incorporated into testing.

The jitter added to the system shrinks the openings in an eye diagram, thus creating the stressed eye at the receiver. Jitter injection is done by generating timing values based on mathematical models of the jitter components and modulating them onto transmitted edges before sending a signal to the receiver. The added jitter must be calibrated to produce a known amount for the test. Different mathematical models decompose jitter in different ways. RJ is Gaussian and the first part of jitter analyzed. By having a Gaussian jitter curve that fits what is seen in the system, the other aspects of jitter can be determined.

High speed serial technologies such as PCIe and SATA use a CDR (clock data recovery) circuit to recover the clock from the data stream. The circuit must recover an accurate clock from the jittered pattern it receives. If the received data contains too much jitter, the system won’t work. To prevent bit errors (assuming the data is not too jittered), the recovered clock must have a certain phase relationship.

Figure 1 shows a block diagram of an receiver. Because the incoming data is random when looking from the perspective of the receiver, this feat is difficult. Encoding helps with the important clock recovery.

![Figure 1. A receiver must recover a clock from the data, as well as demodulate and decode the data. Source: Modeling and mitigation of jitter in multiGbps source-synchronous I/O links.](image)

Encoding maps bits into a different sequence, which reduces long strings of logic 1s or logic 0s, thus reducing DC levels and improving clock recovery from the data. 8B/10B encoding is often used, with
128B/130B gaining popularity. 8B/10B encoding takes eight bits and maps them to ten bits at the transmitter and decodes the data at the receiver. The encoding insures that there will be a run length of no more than five of the same valued bit in a row. A high transition density keeps the output of the CDR circuit from wandering. Combining encoding with a CRC increases error checker robustness.

CRC codes are used as a check to see if data was corrupted in transmission. A CRC is a bit sequence at the end of a packet that is created using an