Introduction

You can use Altera® Arria® II GX and Stratix IV GX devices, which have transceivers, to implement a number of different multi-gigabit high-speed interfaces. This application note describes some common causes of transceiver interfaces failing to synchronize and some steps to debug a design’s power supplies, resets, and PLLs, using methodical fault finding techniques. The application note includes an example design and testbench, which you can implement on an Altera Transceiver Signal Integrity Development Kit, Stratix IV GX Edition.

For more information on the Transceiver Signal Integrity Development Kit, Stratix IV GX Edition, refer to the Transceiver Signal Integrity Development Kit, Stratix IV GX Edition web page.

System Requirements

This application note uses the following system requirements for debugging:

- The Quartus® II software version 9.1—including the SignalTap® II logic analyzer to monitor and verify design internal signals
- The ModelSim-Altera simulator version 6.5b—to simulate and verify the functionality of the design
- Transceiver Signal Integrity Development Kit, Stratix IV GX Edition—to demonstrate debugging
- An oscilloscope—to measure the power supplies associated with the transceiver design

Run a Simulation

Before a design is debugged in hardware, ensure that it works in simulation to rule out any RTL issues.

Figure 1 shows a simulation of an example design that implements a one channel receiver and transmitter design with the receiver configured in automatic lock mode. Figure 1 does not show the full reset sequence as the required phase-locking delay means the image is unreadable.
Figure 2 shows the full reset sequence including the phase locking period required for hardware.

Completion of a full reset sequence and correct transmission of control characters according to the protocol implemented allows the receiver to correctly synchronize to the incoming data.

Figure 3 shows that after the reset sequence is finished, the receiver reports errors until it detects the next occurrence of a synchronization control character denoted by the assertion of the rx_ctrldetect, and rx_patterndetect signals. The receiver then asserts its rx_syncstatus signal indicating that it is synchronized and word aligned to the incoming data.
When the design functions correctly in simulation, you can debug the design in hardware.

**Check Power Supply Integrity**

As part of any board debugging procedure, you should ensure that the power supply integrity is sufficient for the design and it meets the initial design requirements in terms of nominal voltage, current supply, and ripple.

Power supply integrity is important for all high-speed transceiver and communications designs as they rely on PLLs that contain voltage controlled oscillators (VCOs). Noisy power supplies may generate jitter.

Always refer to the Pin Connections Guidelines and handbook for the specific device in use, for recommendations for the transceiver supplies. In general, Altera recommends the use of linear regulators for supplying power to the transceivers.

For more information, refer to Device Pin Connection Guidelines and the Stratix IV Device Handbook.

Altera recommends linear regulators, because they typically offer much better noise performance than switched regulators—even when filtered with inductors, which is becoming an increasingly common cost-saving measure.

Typically, you should target less than 3% peak-to-peak voltage ripple for transceiver supplies. For example, a 1.1-V nominal supply equates to a 33-mV maximum requirement for peak-to-peak noise.

For guidelines on allowable voltage ripple for Stratix IV GX supplies, refer to the Power Delivery Network (PDN) Tool for Stratix IV Devices and the Device-Specific Power Delivery Network (PDN) Tool User Guide. You should use the PDN tool when designing your PDN.
Noisy power supplies may affect the transceivers in the following ways:

■ Either the transmitter or receiver PLLs fail to lock
■ Excessive jitter on the transmitted data
■ Receiver synchronization failure
■ Data corruption
■ Intermittent behavior

Measuring Power Supplies

If you observe any PLL locking, jitter, or intermittent issues, measure the power supplies.

You should measure the noise on any power supplies under full expected load conditions. Potential noise increases with current load. Measuring the supplies when the transceivers are not in operation or perhaps only one of 36 are operational is not representative of the worst case noise that may be present on the supply.

To measure the power supplies, follow these steps:

1. Set an oscilloscope to DC coupling.
2. Set the oscilloscope to have a DC offset equal to the nominal voltage.
3. Set the oscilloscope voltage sensitivity so that any noise fills 90% of the screen.
4. Initially set the trigger point to the nominal voltage of the supply.
5. Apply the probe, and then slowly increase the trigger voltage from nominal so that the scope triggers only on the highest amplitude voltage peaks that may be present.

Avoid leaving the scope trigger voltage on the nominal voltage, which may cause the scope to trigger on edges that do not result in the highest peaks. Hence a worst case measurement may not be achieved.

You should keep ground leads and measurement loops to a minimum, to increase the accuracy of any measurements and to avoid making the supplies seem noisier than they are.

Figure 4 shows a measurement of a 1.1-V nominal supply that is too noisy if you use it for $V_{CCD}$, $V_{CCR}$, or $V_{CCL}$ on a Stratix IV GX design.
Repeat steps 4 and 5, but decrease the trigger point from the nominal value to search for voltage troughs.

The true peak-to-peak noise of the power supply is the maximum peak measured minus the minimum trough.

Table 1 shows the transceiver power supply ripple specifications. Any design should target these specifications.

### Table 1. Transceiver Power Supply Ripple Specifications

<table>
<thead>
<tr>
<th>Supply Name</th>
<th>Nominal Voltage (V)</th>
<th>Ripple (%)</th>
<th>Ripple (mV)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>0.9</td>
<td>3.3</td>
<td>29.7</td>
<td>PLD core</td>
</tr>
<tr>
<td>VCCT</td>
<td>1.1</td>
<td>3</td>
<td>33</td>
<td>Transceiver transmitter</td>
</tr>
<tr>
<td>VCCR</td>
<td>1.1</td>
<td>3</td>
<td>33</td>
<td>Transceiver receiver</td>
</tr>
<tr>
<td>VCCL</td>
<td>1.1</td>
<td>3</td>
<td>33</td>
<td>Transceiver clocking</td>
</tr>
<tr>
<td>VCCH</td>
<td>1.4</td>
<td>3</td>
<td>42</td>
<td>Transceiver I/O buffer</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td></td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>VCCA</td>
<td>2.5</td>
<td>5</td>
<td>125</td>
<td>Transceiver high V power</td>
</tr>
<tr>
<td></td>
<td>3.0</td>
<td></td>
<td>150</td>
<td></td>
</tr>
</tbody>
</table>

**What to Do with Noisy Power Supplies**

If the transceiver and PLL supplies are noisy or out of specification, you should clean them up by any available method to determine whether the noise observed is affecting the integrity of the transceiver interface.
Using series inductors as filters for transceiver supplies can provide useful access points and good debugging aids. They can be removed, thus isolating a potentially noisy voltage source from the transceiver load. They also allow the introduction of alternative cleaner power sources.

**PCB Stackup**

An important part of any power supply design is the PCB stackup, layout, and routing. Ideally, you should review and check PCB stackup, layout, and routing, before PCB sign-off. There is little that you can do to rectify any PCB oversights or mistakes after a PCB has been manufactured, but you should still include it in any debug activities and it is a good idea to review a PCB if power supplies are found to be noisy.

Altera gives the following recommendations

- Supply power to the FPGA through solid or split-plane structures.
- Place power and ground planes on adjacent planes that have thin dielectric to provide inter-plane capacitance and thus aid high frequency transient current supply to the FPGA.
- Place power and ground plane pairs close to the surface of the PCB stackup on the same side as the FPGA.
- Place decoupling capacitors on the side of the PCB that is closest to the power and ground layer positions in the stackup.
- Connect decoupling capacitors and BGA power and ground directly to the power and ground planes that they are coupling using their own individual vias.

Adjacent capacitors or FPGA pins should never share power or ground vias. Figure 5 shows two examples of bad power or ground via sharing practices. This practice introduces mutual inductance and increases simultaneous switching noise (SSN).

**Figure 5. Examples of Bad Power and GND Via Sharing**

For further power distribution network (PDN) recommendations, refer to the Board Design Resource Center.

If you overlook any of the Altera PCB recommendations, assess the impact on the PDN and the operation of the transceiver and correct where possible.
Add Decoupling

Local decoupling capacitors are essential to the integrity of the FPGA power supply when measured at the load. The quantity and value of any decoupling scheme is highly dependent upon the PCB design and also the FPGA design.

You can use the Altera Power Distribution Network Tool to help design your transceiver power supplies according to your specific requirements. The tool may also be used as a debugging aid to assess the effectiveness of a PDN.

For more information, refer to the The Altera Power Distribution Network Tool User Guide.

If after assessing all of the design PDN and decoupling schemes, you find that there is insufficient decoupling, add some more by one of the following methods:

- Try replacing larger bulk capacitors with different values.
- If you require more decoupling on a board that is already designed and manufactured, solder small 0402 or 0201 capacitors onto the back of the ball-grid array (BGA) breakout vias.

Piggy backing extra capacitors on to existing capacitors is not very effective, because via sharing is taking place.

Although the above methods are not ideal for high volume production, you must clean the supplies sufficiently to determine the cause of the problem and hence debug the design.

Replace or Short-Circuit Series Filter or Inductor

Altera recommends you do not use switch-mode regulators and series inductors to provide a clean and cheap supply, instead of using linear regulators where recommended.

Switch-mode regulators typically have an order of magnitude more noise than a linear regulator. In addition, if the series inductor is incorrectly chosen, and insufficient downstream decoupling capacitance is applied, the inductor can starve the FPGA supply pins of instantaneous current, cause the voltage to drop, and increase supply noise that the FPGA sees.

A simple and effective experiment is to replace the series inductor with a zero-ohm resistor or direct short circuit. You should ensure that the power capability of any resistor is sufficient. Implementing a short circuit ensures that any instantaneous current required by the transceiver can be sourced by the regulator quicker than if you use an inductor.

Replacing the series inductor with a 0-Ω resistor may also increase noise as the filtering effect of the inductor is removed.
Bench Supply
If access to each of the individual transceiver supplies has been provided by way of series filtering or zero-ohm isolation resistors, a good debugging process is to try and feed all of the individual transceiver supplies from a typical bench supply. Bench supplies can be cleaner than typical PCB mounted switch mode regulators. A bench power supply should also have a plentiful supply of current.

As bench supplies typically have a wide output voltage range, ensure you set the voltage correctly to avoid damage to the FPGA.

Linear Supply
If access is available to each of the transceiver supplies, you can source them using linear regulators, which typically have an order of magnitude less peak-to-peak ripple than switched mode regulators.

Altexa recommends the use of linear regulators for transceiver supplies.

Power Up, Reset, and PLL Locking
When you have debugged the power supplies, you can now confirm that the interface has initialized correctly. For the purpose of this document, the term initialized refers to power-up, reset, and PLL locking.

The Initialization Sequence
A well-defined power up and reset sequence is required to reliably initialize a transceiver design. Figure 6 shows a typical power up, reset, and PLL locking sequence for a Stratix IV GX transceiver interface using the optional gxb_powerdown signal. The required power-up, reset, and PLL locking sequence may differ depending on the desired protocol.
The paragraph numbers refer to the numbers in Figure 6.

1. After configuration the FPGA begins offset cancellation, which is denoted by the assertion of the busy signal.

2. Only when offset cancellation is complete can the gxb_powerdown and pll_powerdown signals be asserted.

3. De-assertion of the gxb_powerdown signal allows the transceiver to power up and the transmitter PLL to begin locking to the refclk.

4. The transmitter PLL locks and asserts the pll_locked signal.

5. The tx_digitalreset and rx_analogreset signals may be de-asserted after the assertion of pll_locked. After the de-assertion of the tx_digitalreset, the transceiver begins transmitting data present on its tx_datain port in accordance with the configuration of the ALTGX megafunction.

6. De-assertion of the rx_analogreset signal allows the clock recovery unit (CRU) and receiver PLL to begin locking to the refclk. When the receiver PLL locks to the refclk, it asserts the rx_pll_locked signal.

Figure 6 does not show rx_pll_locked.
For this example, it is assumed that data is present on the receiver \texttt{rx\_datain} pins and that the LTD and lock-to-reset (LTR) controller is operating in automatic mode.

7. When the receiver PLL is locked, the CRU begins comparing the clock recovered from the incoming data with the \texttt{refclk}. When the parts per million (PPM) detector detects that the difference is within that specified in the ALTGX instance it switches from the \texttt{refclk} to the recovered clock and asserts the \texttt{rx\_freqlocked} signal. At this point the recovered clock is frequency locked but not necessarily phase locked.

8. A phase locking delay of 4 \( \mu s \) after the assertion of the \texttt{rx\_freqlocked} signal is required for the CRU to be phase locked. At this point the \texttt{rx\_digitalreset} signal may be deasserted and data can be received.

### SignalTap II Analyzer

In addition to confirming that the design functions correctly in simulation, you can debug the design using the SignalTap II analyzer. Figure 7 shows the equivalent reset sequence as in Figure 1 but implemented in hardware. PLL locking times are different to those seen in simulation but the reset sequence remains the same. The sample clock frequency in Figure 7 is 5 MHz.

Figure 7. Example Reset Sequence In the SignalTap II Analyzer

<table>
<thead>
<tr>
<th>Type</th>
<th>Alias</th>
<th>Name</th>
<th>16</th>
<th>6</th>
<th>15</th>
<th>52</th>
<th>48</th>
<th>46</th>
<th>89</th>
<th>98</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTGReset</td>
<td>ALTGReset_reset_in</td>
<td>\texttt{reset_in}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALTGReset</td>
<td>ALTGReset_reset_in_powerdown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALTGReset</td>
<td>ALTGReset_reset_in_powerdown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALTGReset</td>
<td>ALTGReset_reset_in_powerdown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALTGReset</td>
<td>ALTGReset_reset_in_powerdown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALTGReset</td>
<td>ALTGReset_reset_in_powerdown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALTGReset</td>
<td>ALTGReset_reset_in_powerdown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALTGReset</td>
<td>ALTGReset_reset_in_powerdown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALTGReset</td>
<td>ALTGReset_reset_in_powerdown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8 shows the full reset sequence. Figure 8 also shows the transceiver status signals to demonstrate the receiver has correctly detected the control words, is synchronized and working correctly, which is indicated by the asserted \texttt{rx\_syncstatus} signal. The sample clock frequency in Figure 8 is 20 MHz.

Figure 8. Full Example Reset Sequence In the SignalTap II Analyzer

| Type | Alias | Name | 2 | 1024 | 2048 | 32768 | 65536 | 1048576 | 16777216 |
|------|-------|------|----|------|-------|---------|---------|---------|
| ALTGReset | ALTGReset\_reset\_in | \texttt{reset\_in} |   |   |   |   |   |   |   |
| ALTGReset | ALTGReset\_reset\_in | \texttt{reset\_in} |   |   |   |   |   |   |   |
| ALTGReset | ALTGReset\_reset\_in | \texttt{reset\_in} |   |   |   |   |   |   |   |
| ALTGReset | ALTGReset\_reset\_in | \texttt{reset\_in} |   |   |   |   |   |   |   |
| ALTGReset | ALTGReset\_reset\_in | \texttt{reset\_in} |   |   |   |   |   |   |   |
| ALTGReset | ALTGReset\_reset\_in | \texttt{reset\_in} |   |   |   |   |   |   |   |
| ALTGReset | ALTGReset\_reset\_in | \texttt{reset\_in} |   |   |   |   |   |   |   |
| ALTGReset | ALTGReset\_reset\_in | \texttt{reset\_in} |   |   |   |   |   |   |   |
| ALTGReset | ALTGReset\_reset\_in | \texttt{reset\_in} |   |   |   |   |   |   |   |

© December 2009 Altera Corporation
Implement a Simple Design in Hardware

You should measure the power supply integrity under maximum load, to get a worst case reference. However, you can implement a design in its most basic form to confirm that the transceiver is functioning as intended. This example implements an ALTGX instance in basic ×1 mode.

The design should implement a valid reset sequence and also supply the ALTGX instance with meaningful data that allows simple debugging. A count allows manual verification but a more realistic test is a pseudo random binary sequence (PRBS) code.

The design should also correctly send synchronization control characters to allow the receiver state machine to synchronize.

The design may either be implemented using the ALTGX loopback feature, or an external loopback may be implemented on board for a more comprehensive test that includes PCB traces and connectors.

Example Design

The example design included with this application note targets the Stratix IV GX signal integrity board and implements a single transmitter-receiver transceiver that may be looped back internally.

The project contains a simple reset state machine, which waits until offset cancellation is complete and monitors the assertion of the respective `pll_locked`, `rx_pll_locked`, and `rx_freqlocked` signals before releasing resets and transmitting and receiving data.

The project contains a simple data generator that can present either an 8-bit count value or a PRBS onto the `tx_datain` port of the transmitter, but also inserts a synchronization word (0xBC) every 256 words, which allows the receiver to synchronize.

The project also contains a SignalTap II file that allows the transceiver status signals in the design to be monitored when it is running on the device.

After successful reset initialization, `pll_locked` and `rx_freqlocked` are high. `rx_pll_locked` may toggle, which is OK as the receiver is in lock-to-data mode. The `rx_syncstatus` signal is asserted and periodic `rx_patterndetect` and `rx_ctrldetect` assertions on receiving a control character (0xBC) every 256 words may be observed.

Observing the 8 bit `tx_datain` and `rx_dataout` busses allows you to confirm that data is correct. The SignalTap II file in the example design uses `rx_clkout` as its sample clock, which is asynchronous to `tx_clkout` that presents data to the `tx_datain` port. You may see occasional discrepancies between `tx_datain` and `rx_dataout` in the SignalTap II analyzer because of this asynchronicity.

If no errors are detected, `rx_disperr` and `rx_errdetect` remain de-asserted.

The project also contains an RTL testbench, which you may use with the ModelSim-Altera simulator to verify this design. Run the simulation for approximately 10 µs to see correct data and synchronization.
With modification, this design can provide verification on hardware that the power supplies, clocks, and resets are sufficiently stable for a basic single-channel transceiver design. This design only targets a single channel and does not reflect a full design, but you can scale up this design.

The design is for use on the Stratix IV GX signal integrity board, but you may modify it for use on other suitable boards.

**Example Design Simulation**

The example design contains a testbench that demonstrates the transceiver reset and synchronization sequence.

To run the testbench, follow these steps:

1. Unzip the .zip file to a convenient folder.
2. Open the ModelSim simulator
3. Change directory to `Basic\Simulation`.
4. In the ModelSim Command window type `run.do` and press return.

**Example Design Board Settings**


Before downloading the debug design to the Stratix IV GX signal integrity board, ensure that the following board settings are correct:

- Set \( V_{CC} \) to 1.4 V by connecting pins 1 and 2 of J11
- Provide loopback on the following SMA connectors:
  - \( GXB2\_TX7P \) \( J31 \) to \( GXB\_RX7P = J35 \)
  - \( GXB2\_TX7N \) \( J33 \) to \( GXB\_RX7N = J37 \)

For more information, refer to the *Transceiver Signal Integrity Development Kit, Stratix IV GX Edition Getting Started User Guide*.

**User Controls**

You may program the design through the USB connector. You can use the SignalTap II analyzer through the USB connector. To reset the design, press the USER PB0 button.

The design allows you to force a loss of synchronization, to demonstrate cause and effect of transmitting incorrect control character data.

Table 2 shows the Stratix IV GX signal integrity board USER dual in-line package (DIP) switch definitions and functionality for the example design.
The following scenarios use the example design (see “Example Design” on page 11), to demonstrate some realistic debug situations. In all cases, the design was implemented on the Stratix IV GX signal integrity board and loopback provided using SMA cables or internal loopback. The transceiver was configured in basic mode for a 2.5-Gbps line rate with a 156.25-MHz reference clock. The interface to the programmable logic device (PLD) fabric is 8 bits.
**Desired Synchronized State**

Figure 9 shows the moment of synchronization after the de-assertion of the `rx_digitalreset` signal. After the de-assertion of the `rx_digitalreset` signal, the receiver begins searching for the word alignment control character. Before receiving the first word alignment character, the receiver likely outputs erroneous data on `rx_dataout` and also asserts `rx_errdetect` and `rx_disperr`. The deasserted `rx_syncstatus` signal indicates that the receiver is neither word aligned nor synchronized.

On reception of a valid word alignment character, the receiver asserts `rx_ctrldetect` and `rx_patterndetect` for one `rx_clkout` clock cycle. Assuming the control character is correct, the receiver asserts `rx_syncstatus` to indicate that the receiver is synchronized and word aligned.

**Figure 9. Synchronization**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assuming that the transmitter is sending regular synchronization characters, the receiver continues to detect the control characters and maintain synchronization. In Figure 10, the data transmitted is a hexadecimal count interspersed with a `0xBC` control character. The receiver detects the control character 23 `rx_clkout` clock cycles later. Transceiver latency depends on the configuration of the transceiver.

**Figure 10. Synchronized Receiver (Zoomed In)**

**Rx_freqlocked is Deasserted**

Figure 11 shows a SignalTap II plot taken from the example design with a deliberate fault to cause a failure. Figure 11 shows `rx_freqlocked` deasserting and `rx_pll_locked` toggling.
Lock to data (LTD) mode is denoted by the assertion of \texttt{rx\_freqlocked}. The \texttt{rx\_pll\_locked} signal may toggle when the clock data recovery (CDR) is in lock to data (LTD) mode. The \texttt{rx\_freqlocked} signal is deasserted when the CDR cannot recover the embedded clock from the incoming data, because of either an excessive PPM difference to the \texttt{refclk} or perhaps a signal threshold issue.

Close inspection of the associated PLL, reset, and other transceiver status signals should allow you to accurately diagnose the problem. For example:

- When \texttt{gxb\_powerdown} is de-asserted and \texttt{pll\_locked} is asserted, the transceiver is powered up, the \texttt{refclk} is stable and the transmitter PLL is locked.
- \texttt{tx\_digitalreset} is deasserted, data is on \texttt{tx\_datain}, and there are periodic assertions of the \texttt{tx\_ctrlenable} input.
- When \texttt{rx\_freqlocked} is deasserted, the CDR cannot recover the clock from the incoming data, which indicates the received signal should be investigated.
- The \texttt{rx\_errdetect} and \texttt{rx\_disperr} signals are asserted, which denotes an 8B10B and or disparity errors.
- As \texttt{rx\_ctrlidetect} and \texttt{rx\_patterndetect} are not asserted, no control characters are detected.
- The \texttt{rx\_dataout} signal is incorrect. All of this information indicates that something is wrong with the data that should be present on \texttt{rx\_datain}.

Inspection of the loopback cables showed that the SMA connector was not connected on the \texttt{n} half of the differential pair. Correct connection provided the receiver with the signal it requires for correct operation.

### No Synchronization—No \texttt{tx\_ctrlenable}

Figure 12 shows a SignalTap II plot taken from the example design with a deliberate fault to cause a failure.

In this scenario, both the transmitter and receiver PLLs are operating correctly and the CDR is in the LTD mode. Hence data is received. The data transmitted is a simple hexadecimal count, but in this scenario the data received appears corrupt and errors are detected.
In this scenario, you may be tempted to focus on the signals that are toggling—rx_errdetect and rx_disperr—which indicate 8B10B and or disparity errors. Do not overlook signals that are not toggling. You should observe that the rx_ctrldetect and rx_patterndetect signals are not toggling and that rx_syncstatus is deasserted, which indicates that the receiver has not detected a control character or correct word alignment pattern and is not synchronized.

In basic mode, for the receiver to correctly synchronize it must detect a valid control character (0xBC in this scenario) in the received datastream. Figure 12 shows that no control characters or word alignment patterns are detected by the receiver, despite the transmitter interrupting the count data with a 0xBC word.

In this scenario, the receiver cannot detect the transmitted 0xBC control character, because the data generator logic is not instructing the transmitter to encode the 0xBC character as a control code by asserting the tx_ctrlenable signal and so is transmitted as a data character.

The example design that demonstrates this scenario has a feature to switch off the necessary tx_ctrlenable assertions. Re-enabling the correct encoding of the 0xBC word as a control character allows the receiver to correctly detect it, word align, and synchronize correctly.

Figure 13 shows tx_ctrlenable asserted at the same time as 0xBC is presented to tx_datain. 23 rx_clkout cycles later, the same word is seen on rx_dataout. The data is correct with no errors. In addition, rx_syncstatus is asserted indicating that the receiver is synchronized.
Figure 14 shows a SignalTap II plot taken from the example design that has a deliberate fault to cause a failure.

In this scenario, both the transmitter and receiver PLLs are operating correctly and the CDR is in the LTD mode. Hence data is received. The data transmitted is a simple hexadecimal count, but in this scenario the data received appears corrupt and errors are detected.

The `rx_errdetect` and `rx_disperr` signals, which indicate 8B10B or disparity errors, are toggling. The `rx_ctrldetect` and `rx_patterndetect` signals are not toggling, which means control characters are not received. The `tx_ctrlenable` signal is asserted, which indicates that the word on `tx_datain` is encoded as a control character. The `rx_syncstatus` signal is deasserted, which indicates that the receiver has not word aligned or synchronized.

Figure 14 shows that no control characters or word alignment patterns are detected by the receiver despite the transmitter apparently transmitting a control synchronization character denoted by the `tx_ctrlenable` signal asserted.

Figure 15 shows that in this scenario the transceiver expects a 10-bit 8B10B encoded character of K28.5—(10'b1100000_0101) for correct word alignment. The 8-bit equivalent of the K28.5—character is 0xBC.
In this scenario, the receiver cannot detect the transmitted synchronization character, because the data generator logic has incorrectly presented an 0xFF word to the tx_datain port.

The example design that demonstrates this scenario has a feature that enables you to send 0xFF as an incorrect control synchronization character. Re-enabling the correct 0xBC word allows the receiver to correctly detect the control character, word align, and synchronize correctly (Figure 15).

Figure 15. MegaWizard Word Alignment Pattern Definition

**Transceiver Status Signal Debug**

If any of the PLL locked or associated CDR output signals are not asserted as expected, there are a number of tests that you can carry out to find the cause.

**pll_locked Does not Assert**

If the pll_locked signal does not assert, perform the following checks:

- Ensure that gxb_powerdown is not asserted.
Measure refclk, and ensure that it is the correct frequency.

Measure the refclk and ensure that the signal complies with the input FPGA device characteristics—VCMP, VOD, VIL, VIH, VMIN, VMAX.

Measure the refclk jitter and ensure that it is within specification.

As PLLs are susceptible to noise, measure the transceiver power supplies to ensure that they are the correct voltage and that any peak-to-peak voltage ripple is lower than 3% of the nominal setting.

Check that the cal_blk_clk is within defined frequency limits. The cal_blk_clk calculates the transceiver on-chip termination.

**rx_pll_locked Does not Assert**

If the rx_pll_locked signal does not assert, perform the following checks:

- Debug the refclk (debug similar to the pll_locked signal). Check frequency, voltage, jitter, transceiver power supply noise.

- If the design uses rx_cruclk, debug similar to the pll_inclk and refclk signal.

  The rx_pll_locked signal may toggle when the CRU is in LTD mode (rx_freqlocked asserted), which is normal behavior.

**rx_freqlocked Does not Assert**

If the rx_freqlocked signal does not assert, perform the following checks:

- Confirm the data received at the pin is the correct data rate as defined in the ALTGX MegaWizard® Plug-In.

- Measure the data eye at the receiver and ensure that it complies with the AC specifications in the device handbook.

- Ensure that the jitter on the incoming data is within those specified in the device handbook.

- Try increasing the PPM setting in the ALTGX MegaWizard Plug-In.

- Ensure that the received data has enough transitions for the CRU to recover the clock. Try 8B10B encoding. If the design does not use 8B10B, try transmitting a constant 0xBC control character.

- Try looping the transmitter back onto the receiver via a cable or one of the on-chip serial loopback modes. This powerful debug tool ensures that the received data is 0 PPM from the transmitter refclk. Using on-chip serial loopback also eliminates possible PCB routing causes.

- Check that the cal_blk_clk is within defined frequency limits. The cal_blk_clk calculates the transceiver on-chip termination.

- Measure the RREFB pin for noise. This pin defines a bias voltage for transceiver on-chip termination.
Received Data has Errors

If erroneous data is received, or the rx_errdetect or rx_dispperr signals are asserted, perform the following checks:

- Check that the reset sequence has been completed and that the pll_locked, rx_pll_locked and rx_freqlocked are asserted as per the device handbook and descriptions in this document.
- Check data is present on the receiver pins.
- Check the data received at the pin is the correct data rate as defined in the ALTGX MegaWizard Plug-In.
- Check the signal integrity at the receiver pins meets the device specification.
- Check the cal_blk_clk is within defined frequency limits. The cal_blk_clk calculates the transceiver on-chip termination.
- Measure the RREFB pin for noise. This pin defines a bias voltage for transceiver on-chip termination.
- Check the receiver is synchronized

Receiver Is Not Synchronizing

If the receiver is not synchronizing, perform the following checks:

- Ensure the correct synchronization control character is transmitted.
- Ensure tx_ctrlenable is asserted when the synchronization control word is presented to the transmitter.
- Ensure the correct word alignment pattern is transmitted.
- Ensure that the word alignment pattern is correctly defined in the ALTGX MegaWizard Plug-In.

Conclusion

Using a combination of standard debugging techniques including simulation, basic laboratory equipment, the SignalTap II analyzer, and basic test designs, you can quickly debug most transceiver designs.

By breaking the debug activities down into logical and manageable portions, you can determine the cause of most issues.

You can modify the test design provided with this application note to suit a custom hardware design and use it to determine that the transceiver interface is functional in its most basic form.
## Revision History

Table 4 shows the revision history for this application note.

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>December 2009</td>
<td>Updated for Stratix IV GX devices.</td>
</tr>
<tr>
<td>1.0</td>
<td>January 2009</td>
<td>First release of this application note.</td>
</tr>
</tbody>
</table>

Table 4. Revision History