MGT SFP Example Design

Application Note
# Table of Contents

Revision History

Disclaimer

Assumptions

Acronyms

Reference documents

Support

Introduction

Compatibility

KRM-3510 MGT Sub-Carrier configuration

System Block diagram

File Structure
  - SRC Folder

Building the Example Design

Aurora Master Block

Aurora Slave Block

Virtual Button
Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Document revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 6th 2015</td>
<td>1.0</td>
<td>Initial document</td>
</tr>
</tbody>
</table>

Disclaimer

Copyright © Knowledge Resources GmbH. All rights reserved.

All provided data is for information purposes only and not guaranteed for legal purposes.

Information has been carefully checked and is believed to be accurate;

However, no responsibility is assumed for inaccuracies.

Referenced brand and product names are trademarks or registered trademarks of their respective owners.

Specifications are subject to change without notice.

Trademark Acknowledgement:

Brand and product names are trademarks or registered trademarks of their respective owners.
Assumptions

The reader is familiar with Xilinx FPGA and SoC components and the related terminology in common use.

Acronyms

KR: Knowledge Resources GmbH
MGT: Multi Gigabit Transceiver
NA: Not Applicable
PL: Programmable Logic
PS: Processing Subsystem
SFP: Small Form-factor Pluggable
SoC: System on Chip

Reference documents

ZYNQ all programmable SoC, Xilinx, www.xilinx.com
DDR3 SDRAM, Micron, www.micron.com

Support

KR will provide free of charge to qualified customers:

- Schematic and PCB libraries with Module and carrier board design components (Altium)
- 3D STEP models of the module and heat spreader plate
- LINUX BSP and LINUX port (Plug and Boot ready)
- Reference schematics of the evaluation boards (Altium native and PDF)
- Reference designs for on-board PL memory use
- Constraints files (pinning) to accelerate design starts

Further support to aid in customer specific design in's is available at competitive rates, please contact KR for details: + 41 61 545 2080 or mail to office@knowres.com
Introduction

This example design contains two Xilinx Aurora 64b/66b IP Cores which are connected to SFP A and B on the KRM3510 MGT Sub-Carrier. Both Aurora Cores are operating at 8 Gb/s. It can be used to test the MGT connections or as a base for a custom design using MGT communications.

Compatibility

This example design was developed for and tested on a KRM-3Z30 Module, a KRM-3500 Carrier and a KRM-3510 MGT Sub-Carrier. The design was built using Vivado 2014.4. The build script may work with newer and older versions of Vivado if they use the same PS7 IP Block revision as 2014.4. The .bat script expect the Xilinx Tools to be installed in C:\Xilinx\Vivado, if they are installed in a different location the path needs to be modified in the .bat script.

KRM-3510 MGT Sub-Carrier configuration

It is assumed that the clock generator on the KRM-3510 Sub-Carrier is generating a 200MHz reference clock on its CLK0 (MGTB) pins and a 50MHz clock on its CLK3 (FABA) pins. This is the default configuration of the clock generator chip. If you need to generate different frequencies, please contact KR. User configurability is being worked on and will be added in the future.

System Block diagram
**File Structure**

In order to reduce the file size the example Design is distributed as a script which generates the Vivado project. The Project folder contains a source folder as well as two build scripts:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build.tcl</td>
<td>TCL script which creates the Vivado Project including all the necessary sources. This can be launched using the “source” command in the TCL command line in Vivado</td>
</tr>
<tr>
<td>KRM3Z**.bat</td>
<td>Windows batch script which launches the build.tcl script using the correct settings for the corresponding module. Double click it to generate the Vivado project.</td>
</tr>
<tr>
<td>src</td>
<td>Folder containing all source files (VHDL, as well as Xilinx XCI and XDC files)</td>
</tr>
</tbody>
</table>

**SRC Folder**

Inside the src folder the various source types are separated into folders:

<table>
<thead>
<tr>
<th>Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>bd</td>
<td>Vivado IP integrator block diagrams in TCL script form.</td>
</tr>
<tr>
<td>constraints</td>
<td>Xilinx constraint files for pin assignment, setting IO standard etc.</td>
</tr>
<tr>
<td>hdl</td>
<td>HDL source files for all user-written modules as well as top level modules.</td>
</tr>
<tr>
<td>ip</td>
<td>XCI files of the Xilinx IP blocks used in the design.</td>
</tr>
<tr>
<td>presets</td>
<td>Presets for the Zynq processing system for specific modules and PL-DDR pinout constraints.</td>
</tr>
</tbody>
</table>

**Building the Example Design**

In order to build the example design, double click the .bat script corresponding to the module you wish to use. Please note that this design only works for modules that feature MGTs. The script will ask which version of Vivado you wish to use and what the project should be called. It will then call the script build.tcl which will create a new folder including all the needed files.

When the process has finished, open the Vivado project in the new folder. You can either directly start the Bitstream generation, which will launch the whole design flow (Synthesis, Implementation, Bitstream Generation) or do it step by step. The first time Synthesis is run will take a while because all the used IP blocks need to be generated as well.
After the Bitstream generation, open the Hardware Manager and select File -> Export -> Export Hardware and check ‘Include Bitstream’. Then select File -> Launch SDK using default settings.

In SDK select File -> New -> Application Project and create a simple “Hello World” application.
Program the FPGA using Xilinx Tools -> Program FPGA.

Right-Click on the application project and select Run As -> Launch on Hardware

In Vivado open the Hardware Manager and Auto Connect to the device
The Hardware manager will find two integrated logic analyzers (ILA) and one virtual input output (VIO)

Open the master aurora ILA tab and drag the m_axi_rx_tvalid signal into the Trigger Setup Window. Set it up to trigger when the signal is high as shown in the screenshot. Press the Run button. Do the same for the slave aurora block.

Switch to the VIO tab, drag the start_m and start_s signals into the I/O Probes window. Change their values from 0 to 1 using the drop-down menu. This should trigger the ILAs we set up in the previous step.
You should be able to see the data received by the aurora core in the now filled wave window.
The Aurora Master Block contains four sub-components: the Aurora 64b/66b IP core itself, a test pattern generator, a ChipScope Integrated Logic Analyzer (ILA) and a Reset Generator. It is called the master block because the Aurora 64b/66b IP block contains additional shared logic. This includes the transceiver quad PLL, the transceiver refclk buffer, clocking and reset logic. The signals generated in these included shared logic blocks (User CLK, MMCM Lock, etc.) are then passed to the Aurora Slave Block’s Aurora 64b/66b IP core which does not have the additional blocks.

The pattern generator contains a ROM implemented in Block RAM and a simple readout logic block. It starts sending data to the Aurora core via the AXI Stream signals as soon as the Aurora core is ready and the start signal is asserted.

The Reset Generator asserts and de-asserts the Reset PB and PMA Init signals to ensure proper startup of the Aurora Core.

The ILA is used to monitor incoming data.

The Aurora Slave Block contains four sub-components: the Aurora 64b/66b IP core itself, a test pattern generator, a ChipScope Integrated Logic Analyzer (ILA) and a Reset Generator. It is called the slave block because the Aurora 64b/66b IP block receives clocking and reset signals from the Aurora Master Block. The pattern generator contains a ROM implemented in Block RAM and a simple readout logic block. It starts sending data to the Aurora core via the AXI Stream signals as soon as the Aurora core is ready and the start signal is asserted.

The Reset Generator asserts and de-asserts the Reset PB and PMA Init signals to ensure proper startup of the Aurora Core.

The ILA is used to monitor incoming data.
Virtual Button

The Virtual I/O is used as a virtual button to start the pattern generators. Once the system is programmed, they can be accessed through Vivado’s Hardware Manager and a JTAG connection.