Strmá 3001/11B
616 00 Brno
Czech Republic

Ethernet Example Design

Reference manual

Strmá 3001/11B
616 00 Brno
Czech Republic

Document Information:

Document content : Reference Manual
Project Title : Ethernet Example
Author : Marek Kváš
Version : 1.0
Date : 9 May 2017

Document Revision History:

Version	Date	Author	Comments
1.0	2017-05-09	MK	Document created

Strmá 3001/11B
616 00 Brno
Czech Republic

IP core revision history:

Version	Date	Description
1.0	2017-05-09	Design released for public use.

Strmá 3001/11B
616 00 Brno
Czech Republic

Contents

Strmá 3001/11B
616 00 Brno
Czech Republic

1 Overview

This Ethernet example design is primarily developed to demonstrate six speed metallic Ethernet functionality of the Xenie 1.0 module attached to Xenie 1.0 BB baseboard and UDP/IPv4 for 10 G Ethernet IP core. Xenie module is equipped with Marvell PHY that supports six standards for metallic Ethernet:

- 10GBASE-T
- 5GBASE-T
- 2.5GBASE-T
- 1000BASE-T
- 100BASE-TX
- 10BASE-Te

This means it can be used in Ethernet metallic network at one of six speeds – 10 Gbps, 5 Gbps, 2.5 Gbps, 1 Gbps, 100 Mbps, 10 Mbps.

The UDP/IPv4 core (developed and maintained by DFC Design, s.r.o.) is available at no cost from XXXXXX and can be freely used for any kind of project.

Functionality of the example design is fairly basic. UDP protocol is used for all communication. The design provides three services at three UDP ports.

The first service is responding to “Xenie discovery” packet – that enables to find out whether any Xenie with this design is available in any connected network and to discover its network settings details (MAC, IP, netmask).

The second service is UDP loopback. Any packet sent to UDP port of the service is sent back to original sender. UDP payload is not altered.

The last service is similar to UDP loopback, but inserts certain statistical information into the looped back packet. This can be used to test achievable bandwidth or error rate.

The User Datagram Protocol (UDP) is one of the core members of the Internet protocol suite. As such, it is the standard part of network stack implementations available on probably all platforms equipped with Ethernet interface.

UDP/IP core (and so this design) doesn’t implement supporting protocols as Address Resolution Protocol (ARP – translating IP addresses to MAC addresses), Dynamic Host Configuration Protocol (DHCP – often use to assign IP addresses dynamically) or Internet Control Message Protocol (ICMP – services like ping). Services that are commonly provided by these protocols must be replaced by user defined mechanisms.

!!!!!!!!!!!!!!!!!!!!! WARNING !!!!!!!!!!!!!!!!!!!!!

This Xenie Ethernet Example design together with XenieEthExample application generates high bandwidth network traffic which - if accidentally routed to corporate/public network segment - can saturate network infrastructure and effectively prevent network from correct function. Some devices may even start to behave unpredictably than.

It is highly recommended to use this example on dedicated private network segments only. If you are not sure about network infrastructure, consult situation with your network administrator.

!!!!!!!!!!!!!!!!!!!!! WARNING !!!!!!!!!!!!!!!!!!!!!

2 Project directory structure

Directory name	Content description
doc	Folder with this documentation
ip_repo	Vivado IP repository that must be references in the project.
axi_mdio	Auxiliary core implementing MDIO cluase 45
udp_ip_10g	UDP/IPv4 for 10 G Ethernet core
mb_fw	Firmware for MicroBlaze. Folder used as workspace for Vivado SDK.
drivers	Customized (bug fixed) Vivado SDK peripheral divers replacing the original ones.
xenie_eth_test_womtd	Application project for MicroBlaze.
outputs	Folder with scripts that help generate final bitstreams/images
res	Directory where resulting bitstreams or images are placed.
src_data	Folder for binary sources needed to create final bitstream/ image – both generated or basic.
src	Mostly manually written source files.
bd	Folder for Vivado blcok diagram (regenerated from script at the first build).
constr	Folder for constraint files XDC.
hdl	VHDL manually coded source files.
ip	IP parametrization files (XCI).
rxaui_0	Adjusted Xilinx RXAUI core. Distributed as generated core as adjustments make XCI file only distribution hard to manage. BUFM core replaced by BUFG.
udp_ip_10g_0	Parametrization for UDP/IPv4 core.
sw	Folder for software projects related to the example project.
XenieEthExample	Windows based software project testing functionality of Xenie Ethernet Example design.
tcl	TCL scripts/batch files helping to build whole project.
vivado	Folder where Vivado project is created.

3 Xenie Ethernet test internals

Figure 1 shows simplified structure of the whole assembly together with design internals. Xenie BaseBoard provides power supply and connectors for the Xenie module that carries more complex circuits – FPGA, DDR3 memories, Marvell PHY, oscillators, clock generator, configuration FLASH, UID EEPROM and others that are not used by this design.

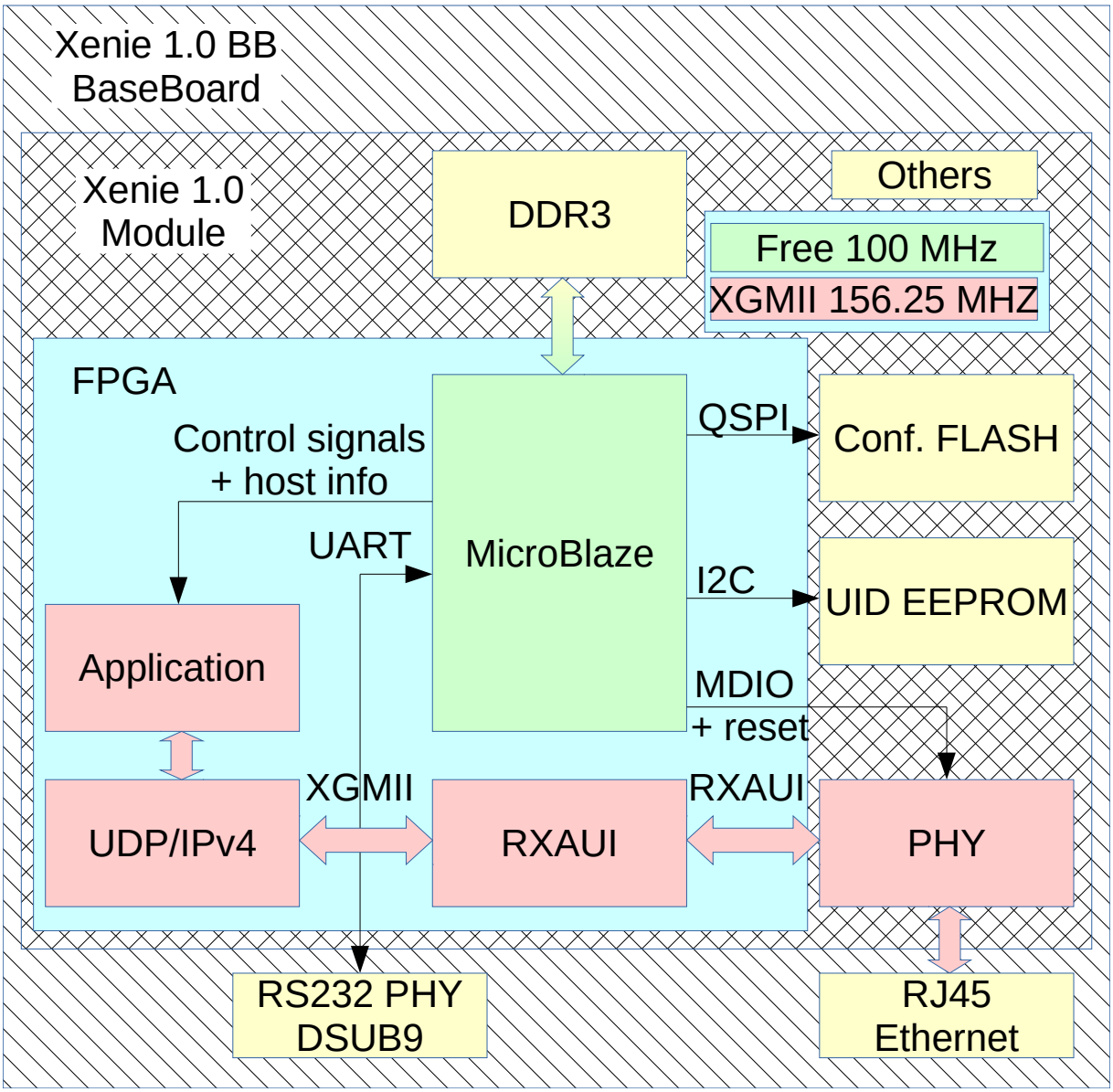

Figure 1: Ethernet Example block diagram

FPGA Design itself consists of two parts – MicroBlaze soft-core processor and Ethernet data path.

MicroBlaze takes care of proper reset and initialization of the PHY via MDIO interface and reset and initialization of the Ethernet path inside the FPGA design.

Ethernet data path consists of three building blocks – Application, UDP/IP4 core and Xilinx RXAUI core. Logic of services that Xenie with this design provides to other network nodes is implemented in the Application block. The UDP/IPv4 block is freely available IP core providing UDP/IP stack. Xilinx RXAUI core is also freely available, but only as a part of Vivado design suite and it function is to transforms internal parallel XGMII interface to transceiver based high speed serial RXAUI interface connected to the PHY.

3.1 Clocking

Two oscillators placed on Xenie are used to provide clocks for FPGA.

Oscillator running at 200 MHz is used for system purposes. It goes to MicroBlaze block where it is transformed in MMCM block to clocks for MicroBlaze processor, DDR3 controller and other peripherals.

Oscillator running at 156.25 MHz is connected to reference clock input of GTX transceivers, that are used to implement RXAUI interface.

Ethernet data path (red color in the block diagram) clock is taken from RXAUI core(XGMII clock). By default RXAUI clock uses BUFM buffer for this clock. Depending of amount of logic fed with this clock, it may not suffice (e.g. almost always when ILA core is inserted for debugging). For this reason RXAUI core was adjusted to use BUFG buffer in this example. If this is not needed for real application, standard RXAUI core can be used.

3.2 MicroBlaze and xenie_eth_test_womtd application

The MicroBlaze block is implemented as Vivado IP Integrator block diagram. It consists of:

- Clock generator – produces various clocks needed (DDR3, AXI, etc.)
- Block RAM memories – user application storage
- DDR3 controller - used as RAM
- QSPI controller - configuration SPI Flash memory access
- I2C controller – unique identifier EEPROM access
- MDIO controller – PHY configuration
- GPIO cores – used to control resets, configure host network settings
- UART lite core – to provide debugging print via RS232 interface

Application that is started after MicroBlaze reset is stored in the block RAM and so it is carried in FPGA bitstream. By default it is xenie_eth_test_womtd application.

The application performs these steps:

1. All internal peripherals are initialized.
2. Version and compile date of bitstream is read and printed together with other version info.

3. UID used later as MAC address is read form EEPROM
4. Check whether PHY needs reset (based on currently running FW). If not skip next two steps.
5. Check valid FW is in the Flash at address 0x00800000 and read it to DDR RAM.
6. Load FW stored in DDR now to PHY and start it.
7. Setup PHY - RXAUI coding and LED behaviour.
8. Limit auto-negotiation to full-duplex modes – half duplex is not supported by UDP/IPv4 core.
9. Setup host addresses for UDP/IPv4 core and take it out of reset to start it.
10. Monitor link status and speed and print any changes.

It is important to note, that configuration flash must contain FW for PHY that is provided by Marvell as binary file. In order to simplify handling this file in configuration flash, header must be prepended. It is done in TCL script.

Process of loading FW into PHY and several other procedures are done by Marvell MTD library. Source code of the library is provided by Marvell under NDA and so cannot be provided as part of the project. It is wrapped into marvell_eth_phy_lib static library that is provided in binary form. If any customization of PHY configuration process is needed, either contact DFC Design for help, or ask Marvell for NDA and sources/documentation for PHY.

3.3 Application core

Application layer that operates above UDP/IPv4 core is implemented in VHDL in udp_ip_10g_test_app.vhd file.

Application listens on three UDP ports. As described in the overview each one provides different service.

3.3.1 Xenie discovery service

Port number of this service is determined by g_net_info_port generic of the application core. By default it is set to 0xDFCC.

When any datagram is received on this port, network datagram with network information is sent to its sender. Datagram structure can be expressed in C as follows:

```
struct xenie_net_info_pkt_s {
      uint64_t tstmp;
      unsigned char mac_addr[6];
      unsigned char pad[2];
      IN_ADDR ip_addr;
      IN_ADDR net_mask;
};
```

As Xenie accepts broadcasts, it can be conveniently used to discover whether there is any Xenie connected to network and create static ARP record to the ARP table based on retrieved information.

3.3.2 Loopback service

Port number of this service is determined by g_loopback_port generic of the application core. By default it is set to 0xDFC0.

Each datagram received on this port is sent back to sender without any changes. Please not that application core works above UDP layer, so only UDP payload wont be changed. Protocol headers are updated as necessary.

3.3.3 Test service

Port number of this service is determined by g_test_port generic of the application core. By default it is set to 0xDFC1.

When datagram is received on this port (request), it is first check for its format. It must contain correct magic number at the beginning and it must be long enough to contain response data. If this conditions are not fulfilled, it is discarded without response.

This service is used by test application XenieEthExample.

The response is formatted to be compatible with following structures:

```
struct test_counters {
	uint32_t loopback_pkt_cnt;
	uint32_t test_pkt_cnt;
	uint32_t unknown_port_pkt_cnt;
	uint32_t test_pkt_accepted_cnt;
};

#pragma pack(push)
struct testPkt {
	uint32_t magic;
	uint32_t seqNo;
	uint64_t tstmp;
	struct test_counters cnts;
	char data[MAX_PKT_DATA_LENGTH];
};
#pragma pack(pop)
```

Structure testPkt defines structure of both request and response datagram. The request must contain correct magic number (0xDFCDFC01) and be large enough to contain first four members of the testPkt structure.

Member seqNo is copied from request to response by the application core and provides user to match requests and responses easily.

Member tstp is 64 bit wide time stamp counter value. The counter runs at XGMII clock 156.25 MHz. The value is inserted by the application core when datagram is being sent. Even though it doesn't mark exactly neither moment of reception of the request packet nor moment of transmission of response packet it can be used to track traffic level very precisely.

Counters that are members of structure test_counters provide statistical information about amount of datagrams handled by the core:

- loopback_pkt_cnt – number of datagrams processed by loopback service.
- test_pkt_cnt – number of datagrams received by test service
- test_pkt_accepted_cnt – number of datagrams received by test service, that fulfill al requirements and were responded to
- unknown_port_pkt_cnt – number of datagrams that were received and discarded on unserveced UDP ports

4 Test setup

In order to enable quick and easy test of Xenie with Ethernet test design, simple MS Windows based testing application XenieEthExample was implemented in C language (MS Visual studio project attached).

In order to test Xenie at the top speed, computer equipped with 10GBASE-T interface is needed, but any lower speed is supported as well.

Xenie can be either directly connected to the computer or it can be connected via any Ethernet infrastructure (switches).

It is recommended that both Xenie and test computer is in one Ethernet segment – i.e. there are no routers in the way. In general you can use any network topology, but XenieEthExample application wont be able to discover Xenie and will fail – different discovery method is needed in such a case.

Default network settings are:

IP address: 192.168.10.96

Net mask: 255.255.255.0

MAC address: Globally unique

5 XenieEthExample application

The application is written in MS Visual studio 2010 in plain C. It is tested on MS Windows 7, but should be compatible with other Windows version. It uses standard Windows Sockets API for network access.

As the first step, presence of any Xenie with correct running design is verified using Network info discovery packet. This packet is broadcasted to all interfaces. When Xenie receives it, it responds with unicast response containing its MAC, IPv4 address and netmask. Because Xenie example design doesn't have ARP, static ARP record is created in order to enable unicast communication.

Static ARP record may - if not well understood and managed - cause weird behaviour of the network. Static ARP record is not permanent, so it disappears after system reboot, or ARP table reset.

If Xenie is found on the network, number of transmitter / receiver thread pairs is created. The number of this pairs is given by SESSIONS_COUNT macro.

Each transmitter thread sends UDP test traffic as fast as possible to Xenie. Xenie captures this packets, inserts some statistical data into each datagram and sends them back. This "looped back" datagrams are captured by the corresponding receiver thread. It checks the statistics inserted by Xenie and determines whether any datagrams were lost. The lost count and some other information is stored in the context of each receiver.

The main thread periodically reads the statistics from all the receiver threads, sums them up (if applicable) and prints info to the console.

Integrity of the received frames is protected by 32 bit CRC at the Ethernet layer so it is not necessary to check for datagram payload correctness. Corrupted packets are simply discarded by NIC.

Timestamps from Xenie are used to measure bandwidth. Calculated number should match number given by Windows task manager. Be careful about interpretation of graphs shown by the task manager as it sums up TX and RX traffic. So if 50 % of bandwidth is really used, graphs show 100 % already.

In order to generate maximum traffic, maximum frame size is used. MTU macro can be modified to limit packet size. By default 9000 bytes is used - it is standard maximum value for jumbo frames. When this macro is set to value higher than real MTU used by machine running this application, datagrams are fragmented at the IP layer. As Xenie cannot handle fragmented IP packets, the test wont work as expected - high but not 100 % packet loss is most likely result. If host machine has MTU high enough, but there is any network switch on the path to Xenie that doesn't support set MTU, the traffic will most likely be lost completely. Check your NIC settings for the MTU before you run this application.

!!!!!!!!!!!!!!!!!!!!! WARNING !!!!!!!!!!!!!!!!!!!!!

This Xenie Ethernet Example design together with XenieEthExample application generates high bandwidth network traffic which - if accidentally routed to corporate/public network segment - can saturate network infrastructure and effectively prevent network from correct function. Some devices may even start to behave unpredictably than.

It is highly recommended to use this example on dedicated private network segments only. If you are not sure about network infrastructure, consult situation with your network administrator.

!!!!!!!!!!!!!!!!!!!!! WARNING !!!!!!!!!!!!!!!!!!!!!

6 How to build project

Project is targeted to and tested on Vivado design suite 2016.4. It may not be built on other versions and manual adjustments to scripts may be necessary.

In order to ensure correct working of the scripts below, PATH must be set for both Vivado and SDK executables. It usually means to add these folders to the PATH environment variable:

C:\Xilinx\Vivado\2016.4\bin;

C:\Xilinx\Vivado\2016.4\lib\win64.o

C:\Xilinx\SDK\2016.4\bin

If you are building the project for the first time:

1) Go to project_folder/src/tcl

2) Run either build_all.bat, or build_all.tcl. This starts following actions

- Block diagram and Vivado project are recreated (recreate_project.tcl, recreate_bd.tcl)
- Synthesis and implementation are started (build_project.tcl)
- HW project is exported for usage in SDK(build_project.tcl)
- SDK workspace is created (xsdk_prepare_workspace.tcl)
  - HW platform is created
  - Repository reference is added
  - Board Support Package is created
  - Microblaze application project xenie_eth_test_womtd is imported to the workspace
  - All projects are built
- Flash image (MCS file) containing FW (binary) for Marvell PHY is created (outputs/create_mcs_marvell_only.tcl)
- All generated data that are needed to generate image with Microblaze with preloaded application are concentrated to outputs/src_data (outputs/update_src_data.tcl)
- Flash image (MCS file) with FPGA configuration bitstream is created (outputs/create_mcs_microblaze.tcl)

3) Wait until build process ends

If you need to rebuild project after any changes, you can open project in Vivado folder and use GUI to do it. Or call aforementioned scripts according to your needs. You cannot use build_all scripts as it tries to recreate block diagram and project in existing folders and fails.

7 How to load Flash content to Xenie

There are two ways how to load all necessary images to Xenie configuration flash.

You can either use TCL script:

1. Connect JTAG cable to Xenie BaseBoard
2. Run either program_xenie.bat or program_xenie.tcl. This step performs this
   - Erase Flash in ranges needed for images
   - Program Marvell FW image and FPGA bitstream image
   - Verify content of Flash
3. Power-cycle Xenie when process is finished

Or you can open Vivado project and program images manually in HW manager. In that case you need to respect this flash memory map:

0x00000000 – FPGA bitstream

0x00800000 – Marvell PHY FW with header prepended