BERTEN
Digital Signal Processing

BUM002 v1.0

USER MANUAL

GigaX API for Zynq SoC

A software API for Zynq® PS that Enables
High-speed GigaE-PL Data Transfer & Frames Management

www.bertendsp.com gigax@bertendsp.com +34 942 18 10 11

Table of Contents

Revision History

Date	Version	Revision
14/12/2016	1.0	First version

1 Introduction

GigaX is a lwIP-based API for Xilinx Zynq® SoC that establishes a high-speed communication channel between the GigaE Processing System (PS) port and the Programmable Logic (PL). Running in one of the Zynq® ARM cores, GigaX processes network and transport headers, and manages SDRAM, Ethernet DMA, and AXI interfaces to setup a robust full duplex data link through the PS at 200Mbps.

The software API also implements IP filtering and TCP/UDP headers management to allow using your device as an Ethernet Bridge, Programmable Network Node, Hardware Accelerator, or Network Offloader.

GigaX is easy to install as a library in your SDK project, being able to control both the GigaE peripheral and the AXI DMA interface to enable a direct communication between Ethernet and your VIVADO IP Cores. It implements the following functionalities:

- Full Duplex GigaE-PL Link, with data rates up to 200 Mbps
- TCP/UDP Headers Management and Protocol Conversion
- IP filtering
- Ethernet Data Caching
- Watchdog Timer

2 Functional Description

Figure 1 shows a high-level diagram of the GigaX functionality and interfaces. Based on the open source lwIP stack, GigaX controls the Zynq® GigaE peripheral to send/receive Ethernet frames to/from SDRAM via the AMBA Interconnect bus. After performing IP filtering, Ethernet data is sent to PL through a High Performance AXI Port, which is also used to send processed data back to the PS.

PS-PL transfer uses an AXI DMA implemented in the PL and controlled by GigaX through a General Purpose AXI4-Lite Port. Network headers can be kept or removed before reaching the PL. GigaX also allows the generation or modification of IP headers of the data received from PL.

The complete full duplex communication based on DMA and AXI ports allows high-speed data transfer without overloading the ARM processor. AXI4-Stream interfaces shall be used to transfer Ethernet data to/from your IP cores.

The GigaX Data Caching system is capable to manage data peaks over the maximum transfer rate, ensuring stable communications of variable data flows. The API also implements a software watchdog timer to recover from unexpected situations.

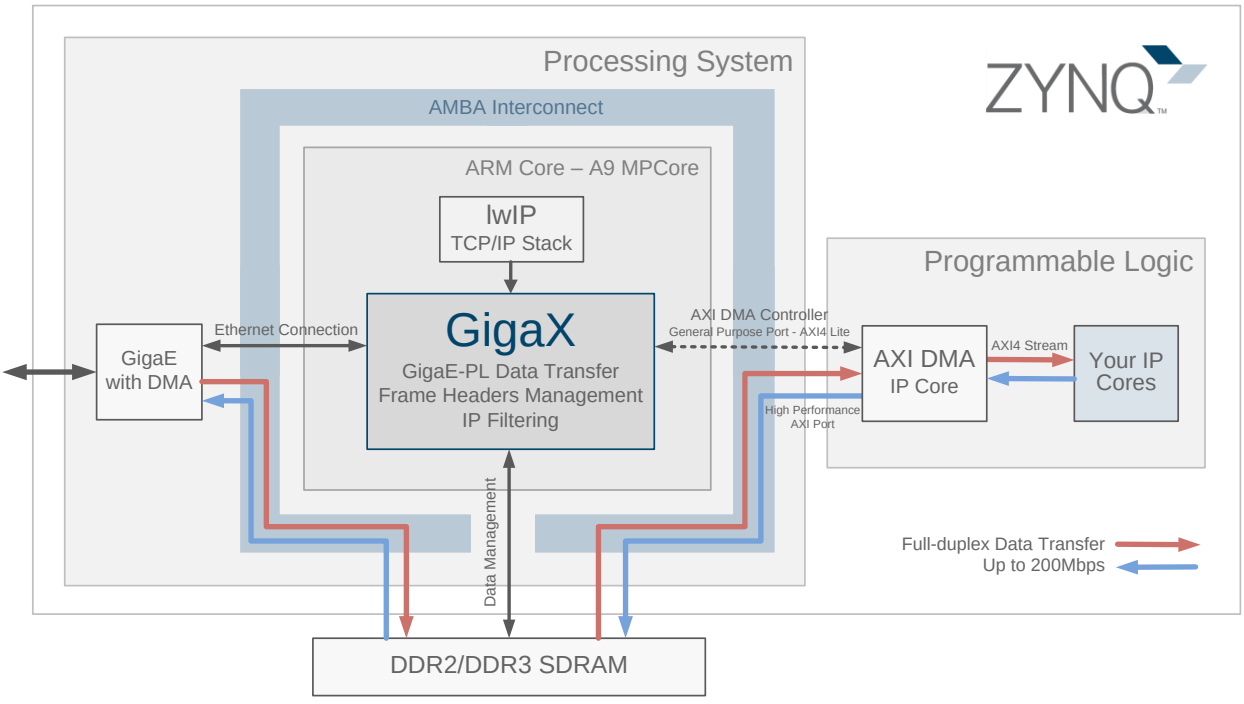

Figure 1. GigaX Functionality and Interfaces

2.1 Full-duplex Data Transfer

GigaX is a C-function executed in a standalone operating system of the Zynq PS. It stablishes two communication channels to setup the full-duplex data transfer with the PL.

The lwIP library is used to implement the connection between the Zynq and the device connected to the Ethernet peripheral. It uses a MAC Controller, a FIFO and the Ethernet DMA controller. The FIFO provides storage for transmitting and receiving packets, which are transferred by Ethernet DMA in scatter-gather mode. DMA works with a buffer descriptor architecture, with separate lists for transmission and reception. Each descriptor defines a fixed area in PS DDR memory. lwIP configuration is detailed in Section 3.3.3, including recommendations for DDR RAM resources allocation.

PS to PL communication is implemented using a PL AXI DMA core controlled from the GigaX API. Data between DDR RAM and the logic is transferred via a High performance AXI port. The communication is based on an architecture specifically designed to attend AXI DMA interrupts.

2.2 Network functionalities

GigaX implements a complete set of network functionalities based on a flexible architecture that are configured by the parameters in Section 3.2.

- Network configuration

The network parameters of the Zynq can be configured, defining device and gateway IP addresses, network mask, and MAC address.

- Headers removal, modification and generation

The API can be configured to send complete packets to PL, or to remove network and transport headers (IP and UDP/TCP) before sending the data to the logic. In the later case, headers' information of PL-to-PS frames is lost and must be re-defined via function parameters when coming back from PL to PS. Depending on the input arguments, the new headers can be regenerated or modified when frames arrive to PS.

Headers modification allows setting new port numbers and IP addresses for the source and destination hosts. GigaX implements frame length modification and TCP-to-UDP transport protocol conversion. TCP headers re-generation is not supported.

- IP Filtering

Ethernet data may be filtered at PS, sending to PL only those frames with source and destination IP within the predefined address range. Frames with other addresses are not sent to PL, but can be processed for other purposes.

IP filtering can be disabled setting *src_ip_min* and *dst_ip_min* to "0.0.0.0", and *src_ip_max* and *dst_ip_max* to "255.255.255.255" (see Table 1).

2.3 Watchdog Timer

GigaX watchdog timer allows the PS to recover after a non-expected critical failure. It consists of a configurable timeout counter that is restarted periodically before reaching zero.

The watchdog also checks PS-PL communication. If the PL AXI DMA is not accepting data during a period equal to the watchdog timeout, the system is reset. Watchdog timeout may be configured according target PL design to avoid undesirable resets. It can be disabled or set to a value between 0 and 12s.

2.4 Ethernet Data Caching

Up to 22MB of RAM space can be assigned to store incoming Ethernet data, being possible to manage communication peaks without losing frames. This feature is especially useful to transfer non-constant data streaming arriving from the GigaE port at a rate over the GigaX maximum communication rate.

As shown in Figure 2, data caching size is configured in SDK (Board Support Packet Settings > lwIP_memory_options > mem_size). It is not recommended to increase this parameter over the maximum value shown in this configuration screen.

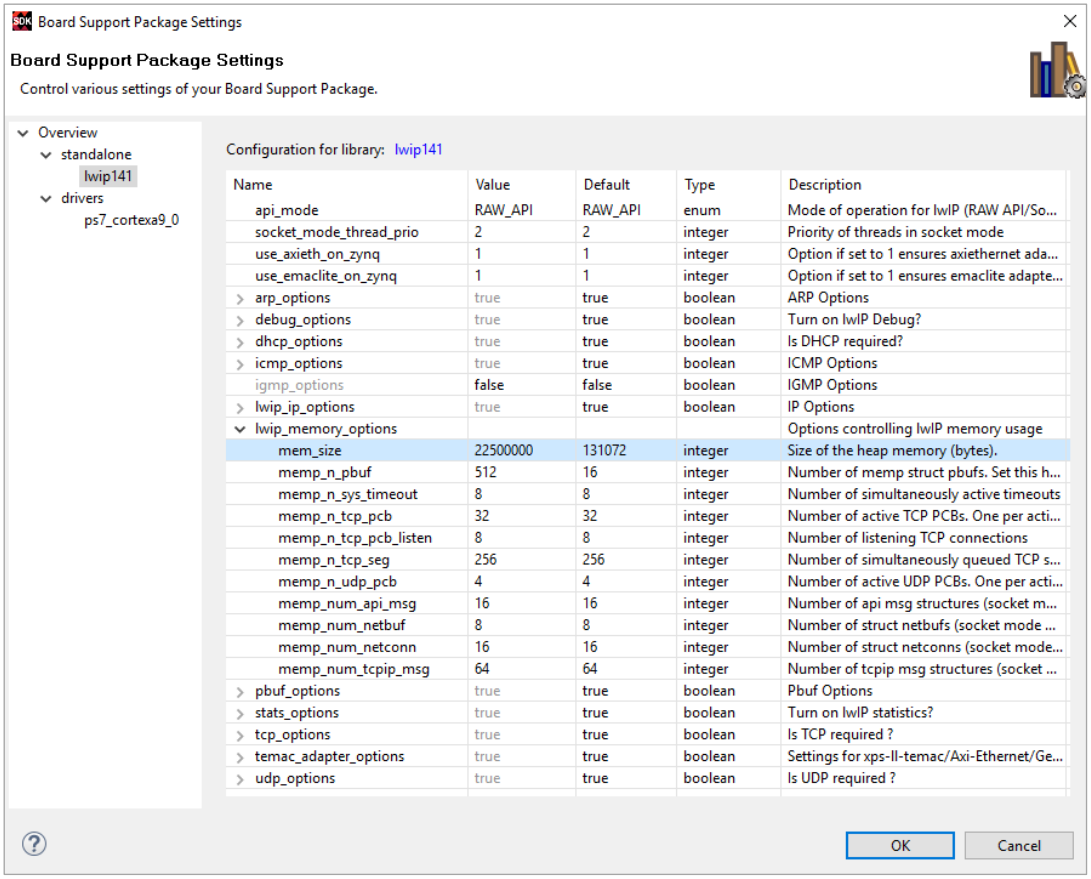

Figure 2: Configuration of the Data Caching Memory Size

3 Operational Guide

3.1 System Requirements

The GigaX is designed to work in any Zynq AP SoC device implementing a standalone operating system. The target board shall include DDR2/DDR3 RAM Memory at the PS to allow full-duplex data transfer between the Gigabit Ethernet interface and the PL.

Hardware

- Zynq® AP SoC
- DDR2/DDR3 RAM Memory
- Gigabit Ethernet Interface

Software

- VIVADO/SDK Design Suite (v2016.2 or later)
- lwIP library v1.4.1 or later
- Stand-alone Operating System (v5.5 or later)

3.2 Using the GigaX function

GigaX API is used with a C function call to be included in your SDK project. The function prototype is the following:

```
int GigaX_pspl_transfer(char IP_zynq[], char MAC_zynq[17], char netmask[], char IP_comp[], int wtd, int headers, char ps2pl_proto[], char pl2ps_proto[], int TCP_port, char src_IP_min[], char src_IP_max[], char dst_IP_min[], char dst_IP_max[], char pl2ps_src_ip[], char pl2ps_dst_ip[], int pl2ps_dst_port, int length_pl2ps, char product_key[]);
```

A usage example is provided in the following code template, which can be used as baselined for modification and integration in the user project. Table 1 shows a detailed description of GigaX inputs arguments.

```
int main(){

    // Ethernet communication
    char ip[]           = "192.168.3.1";
    char mac[17]        = "00:0A:35:00:01:02";
    char nmask[]        = "255.255.255.0";
    char gw[]           = "192.168.3.10";

    // System configuration
    int watchdog        = 3;
    int headers         = 1;
    char ps2pl_proto[]  = "0";
    char pl2ps_proto[]  = "UDP";
    int TCP_port        = 1234;

    // IP filtering
    char src_ip_min[]   = "192.168.0.0";
    char src_ip_max[]   = "192.180.0.0";
    char dst_ip_min[]   = "192.168.3.0";
    char dst_ip_max[]   = "192.168.3.200";

    // Data reformatting
    char pl2ps_src_ip[]  = "-1";
    char pl2ps_dst_ip[]  = "192.168.3.21";
    int pl2ps_dst_port   = 1234;

    // Length of PL-to-PS frames
    int length_pl2ps     = 1460;

    // Licence code
    char product_key[]   = "YOUR-LICE-CODE-NUMB";

    // Call to the initialization function with IP, netmask and gateway
    GigaX_pspl_transfer(ip, mac, nmask, gw, watchdog, headers, ps2pl_proto, pl2ps_proto, TCP_port, src_ip_min, src_ip_max, dst_ip_min, dst_ip_max, pl2ps_src_ip, pl2ps_dst_ip, pl2ps_dst_port, length_pl2ps, product_key);
    return 0;
}
```

Name	Description	Example	Range
ip	Zynq IP address	“192.168.3.1”	
mac	Zynq MAC address	“00:0A:35:00:01:02” “00-0A-35-00-01-02”	
nmask	Network mask	“255.255.255.0”	
gw	Gateway IP address	“192.168.3.10”	
watchdog	Watchdog’s timeout in seconds. When this parameter is 0, watchdog is disabled	3	[0,12]
headers	Remove or keep transport and network headers. If set to 0, headers are removed. If set to 1 headers are kept. (1) (2) (3) (4) (5)	1	[0,1]
ps2pl_proto	Transport protocol of PS-to-PL frames. If set to “0”, frames of both protocols are processed. (2)	“TCP”, “UDP” “0”	
pl2ps_proto	Transport protocol applied to PL-to-PS frames when headers are removed. Always set to “UDP”. (3)	“UDP”	
TCP_port	Port number at which GigaX listens for input TCP data (2)	1234	[1,65535]
src_ip_min	Lower limit of source IP address for Ethernet input frames	“192.168.0.0”	
src_ip_max	Higher limit of source IP address for Ethernet input frames	“192.180.0.0”	
dst_ip_min	Lower limit of destination IP address for Ethernet input frames	“192.168.3.0”	
dst_ip_max	Higher limit of destination IP address for Ethernet input frames	“192.168.3.200”	
pl2ps_src_ip	Source IP address for PL-to-PS frames (4). If set to “0”, it is not modified. If set to “-1”, it is changed to Zynq IP.	“192.168.3.20” “0”, “-1”	
pl2ps_dst_ip	Destination IP address for PL-to-PS frames (5). If set to “0”, it is not modified. If set to “-1”, it is changed to source host IP.	“192.168.3.21” “0”, “-1”	
pl2ps_dst_port	Destination port number for PL-to-PS frames (4). If set to “0”, it is not modified. If set to “-1”, it is changed to source host port number	1234 0, -1	[1,65535]
length_pl2ps	Payload length (in bytes) of the frames generated at Zynq device. Only used when headers are removed.	1460	[18,1472]
product_key	Licence Code	“YOUR-LICE-CODE-NUMB” “YOURLICECODENUMB"	

(1) Ethernet headers are always removed.
(2) When headers are kept or *ps2pl_proto* is UDP, *TCP_port* is ignored.
(3) When headers are kept, *pl2ps_proto* is ignored, and both TCP and UDP packets are processed.
(4) When headers are removed, *pl2ps_src_ip* cannot be zero.
(5) When headers are removed, *pl2ps_dst_ip* and *pl2ps_dst_port* requires an IP address and port number.

Table 1: GigaX Input Arguments

3.3 Software Installation

The following steps describe how to create an SDK project with lwIP and the GigaX API.

3.3.1. Create a lwIP project

1. Open Xilinx SDK.
2. Select **File** → **New** → **Application Project**.
3. Select a name and a location for the application.
4. Select **Standalone** as **OS platform**.
5. In the **Target hardware** section, click **New** and select the hdf-file exported by Vivado (probably to the *projectname.sdk* folder in Vivado project). Section 3.4 shows how to create a basic hardware platform for GigaX and how to export its hdf-file.
6. In the Target Software section, choose **C** language and create a new Board Support Package (BSP).
7. Click **Next**.
8. Select **lwIP Echo Server** template. It adds lwIP to the BSP and source code with a simple demo.
9. Click **Finish**.

Three folders are created in the workspace: the application project, the board support package and the hardware platform. They contain the source code, the operating system and drivers, and the hardware description and FPGA programming file (.bit), respectively.

3.3.2. Add GigaX to the project

1. Remove the code generated by SDK. Expand the *src* folder in the application project. Select all the files, except *lscript.ld* and *Xilinx.spec* and delete them.
2. Right-click in *src* folder, select **New** → **Source file**. Name it **main.c** and select **<None>** as template
3. Write the call to *GigaX_pspl_transfer* function (see Section 3.2).
4. Right-click in the application project and select **Build Configurations** → **Set Active** → **1 Debug**
5. Right-click in the application project and select **Properties**.
6. Go to **C/C++ Build** → **Settings** → **Tool Settings** → **ARM v7 gcc linker** → **Libraries**.
7. Click **Add…** in **Libraries** window and write **GigaX**.
8. Click **Add…** in **Library search path** window and select the folder where libGigaX.a is located. Close the window.
9. Right-click in the application project and select **Build Configurations** → **Set Active** → **2 Release**
10. Repeat steps 5, 6,7 and 8, but now for the *Release* configuration set in step 9.
11. In **Tool Settings** window, go to **Optimization** and set **Optimization level** to **None (-O0).**

3.3.3. Configure lwIP

1. Right-click in the BSP and select **Board Support Package Settings**.
2. Select **Overview** → **Standalone** → **lwip141**.
3. Set the parameters in Table 2.

Section	Name	Value
lwIP_memory_options	mem_size	22500000
	memp_n_pbuf	512
pbuf_options	pbuf_pool_bufsize	7000
	pbuf_pool_size	65535
tcp_options	tcp_queue_ooseq	0
	tcp_wnd	32768

Table 2: lwIP Configuration Parameters

4. Click **OK** to regenerate the BSP.
5. Right-click in the application project and select **Generate Linker Script**.
6. Set the parameters in Table 3 to assign DDR3 resources.
7. Click **Generate**.

Name	Value
Heap size	5242880
Stack size	5242880

Table 3: Linker Script Configuration Parameters

3.4 Hardware Platform Setup

A hardware platform is required to execute the GigaX API. It can be setup by using the provided Tcl-file or manually creating a Vivado block design.

3.4.1. Using a Tcl-file

The following steps configure the hardware platform using the provided Tcl-file:

1. Create a Vivado project.
2. Create a block design.
3. Write **source <path_to_Tcl_file>** in the Tcl console of Vivado.
4. If **Zynq PS** core has not been automatically configured by Vivado, fix bank voltages and DDR parameters. It can be done manually or using a *preset* provided by the board designer.
5. In the **Sources** tab of the block design, right click on the *.bd* file and select **Create HDL wrapper.** Mark **Let Vivado manage wrapper and auto-update** and click **OK**.
6. Synthesize, implement, and generate the bitstream.
7. Select **File** → **Export** → **Export hardware**. Mark **Include bitstream** and select the folder where the results are exported. Finally, click **OK**.

3.4.2. Manual Creation of a Block Design

The hardware platform can be created manually following these steps:

1. Create a Vivado project.
2. Create a block design.
3. Add **Zynq7 Processing System**.
4. Click **Run Block Automation** (Figure 3) to connect I/O peripherals and DDR to the processing system. If a preset file is provided by the board designer, it can be used in the processing system by marking **Apply Board Preset**. Otherwise, bank voltages and memory configuration must be set manually. Finally, click **OK**.

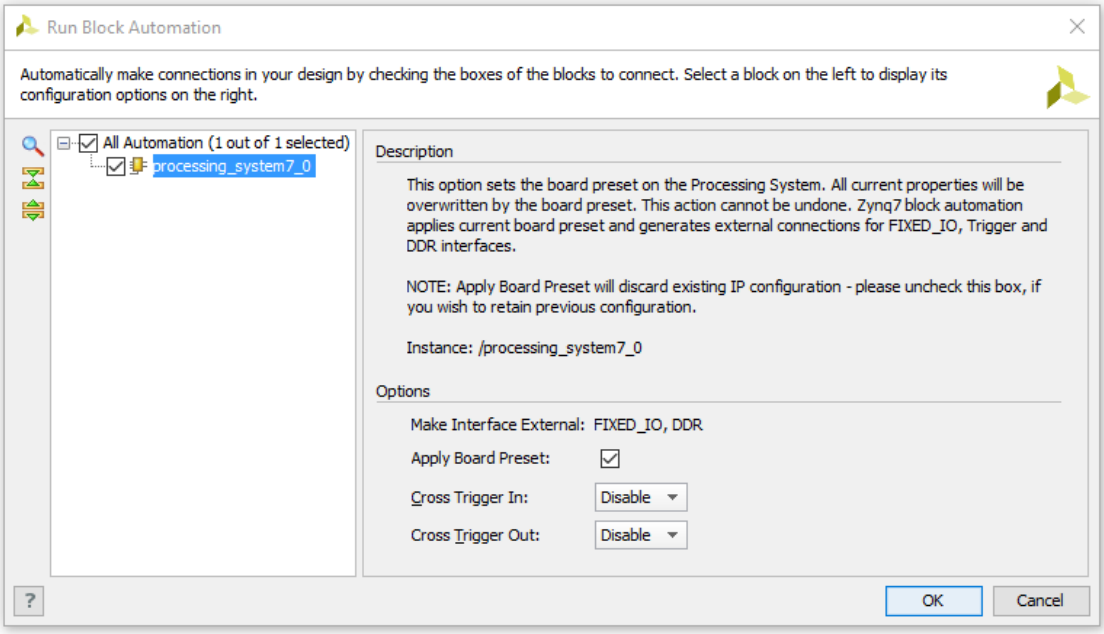

Figure 3: Run Block Automation

5. In **Clock Configuration** (Figure 4), enable the PL clock (e.g. 100MHz), which at least must be used for the AXI DMA IP core of Figure 1.

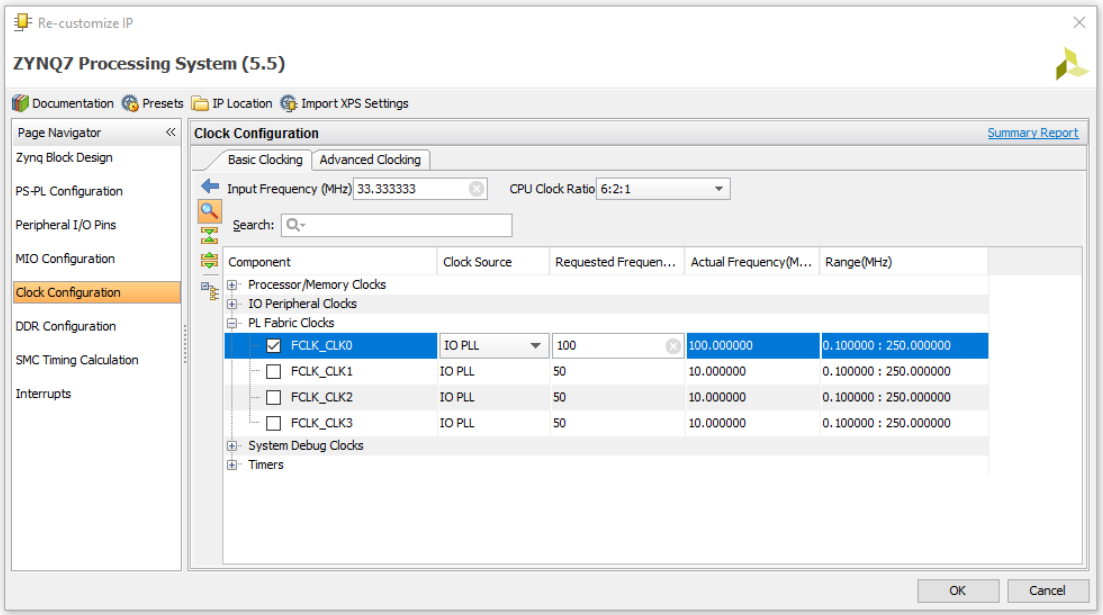

Figure 4: Clock Configuration

6. In **PS-PL Configuration** (Figure 5), add a Slave High Performance AXI Port.

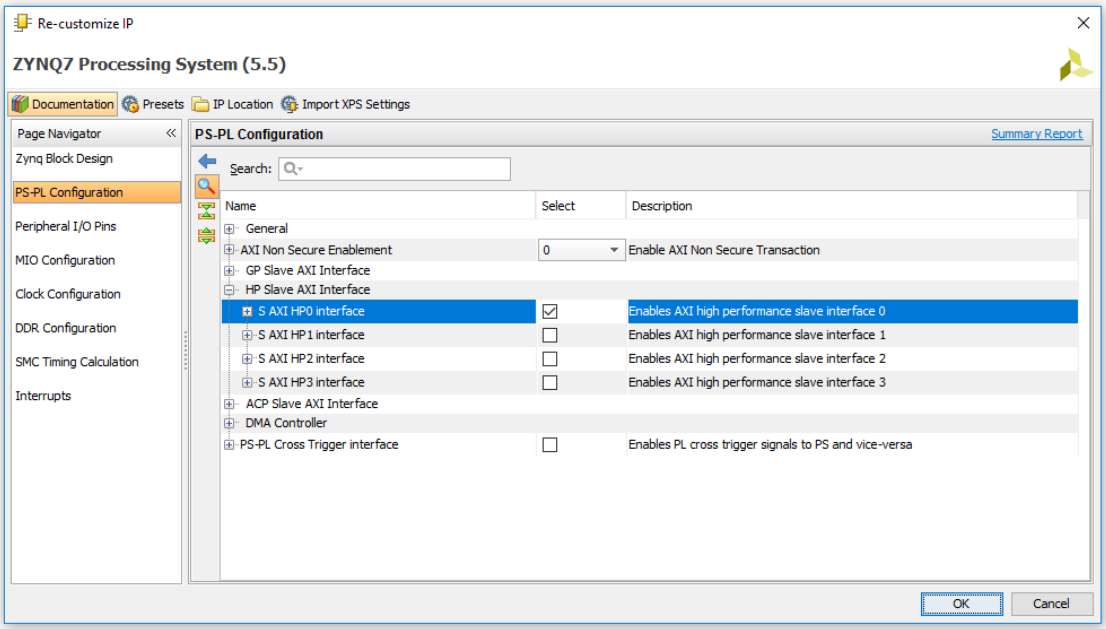

Figure 5: PS-PL Configuration

7. In **Peripheral I/O Pins** (Figure 6), add the Ethernet0 peripheral using MDIO interface and UART1. Other peripherals can be added if needed.

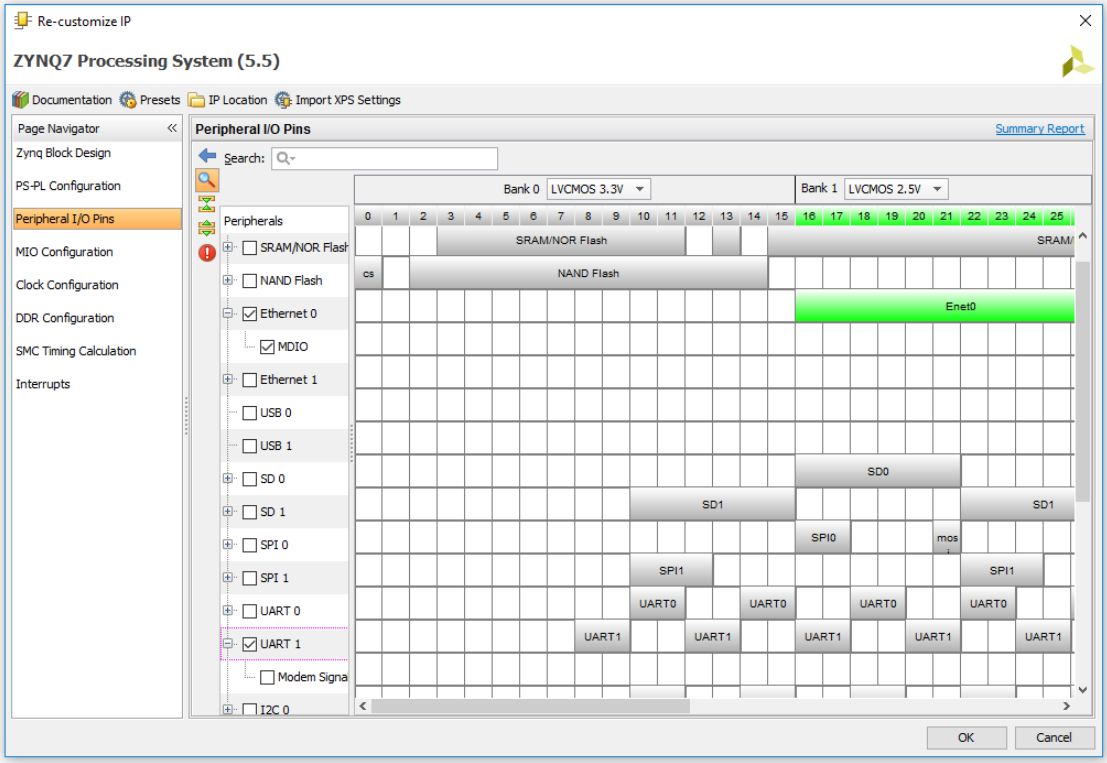

Figure 6: Peripheral I/O Pins

8. In **MIO configuration** (Figure 7), allocate and configure the MIO pins for Ethernet and UART peripherals. An example is provided in the following figure.

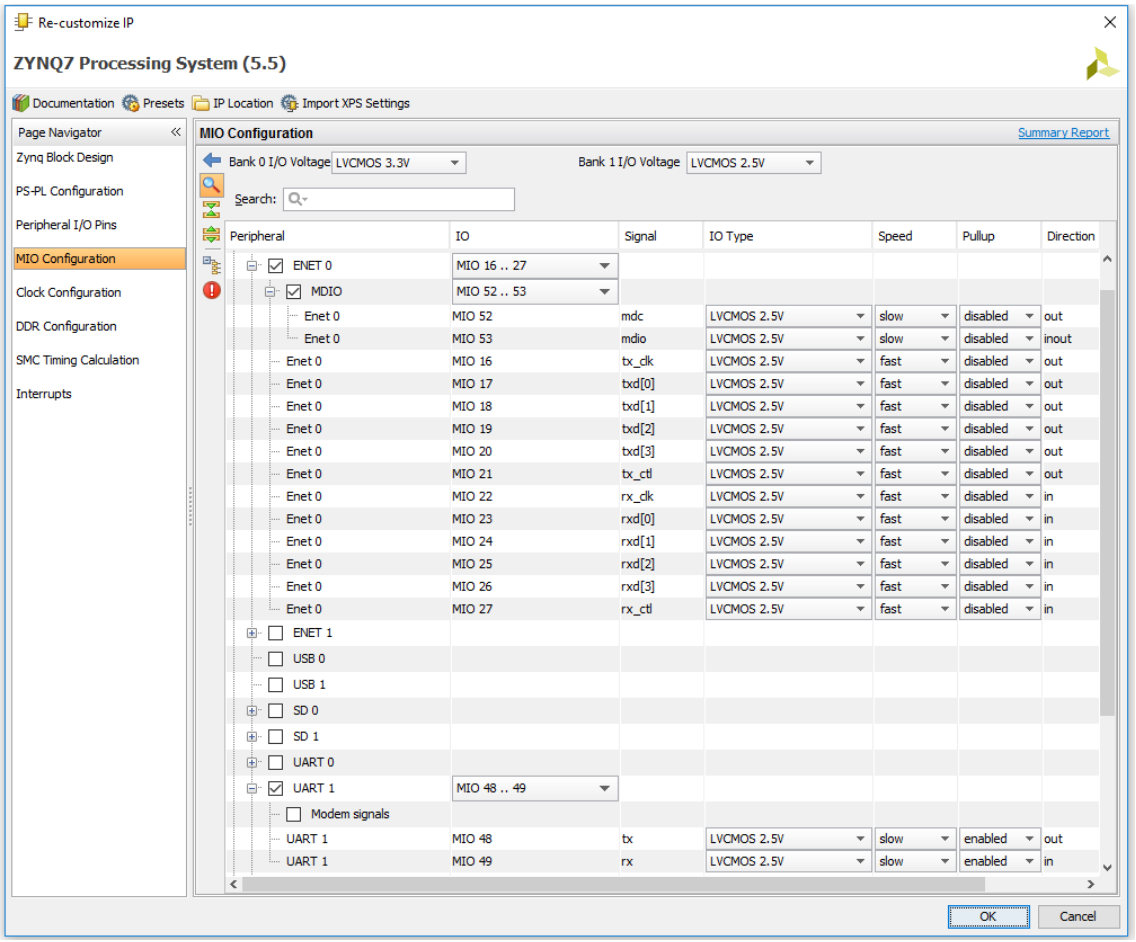

Figure 7: MIO Configuration

9. Select **Interrupts** (Figure 8), and enable IRQ_F2P under PL-PS Interrupt Ports.

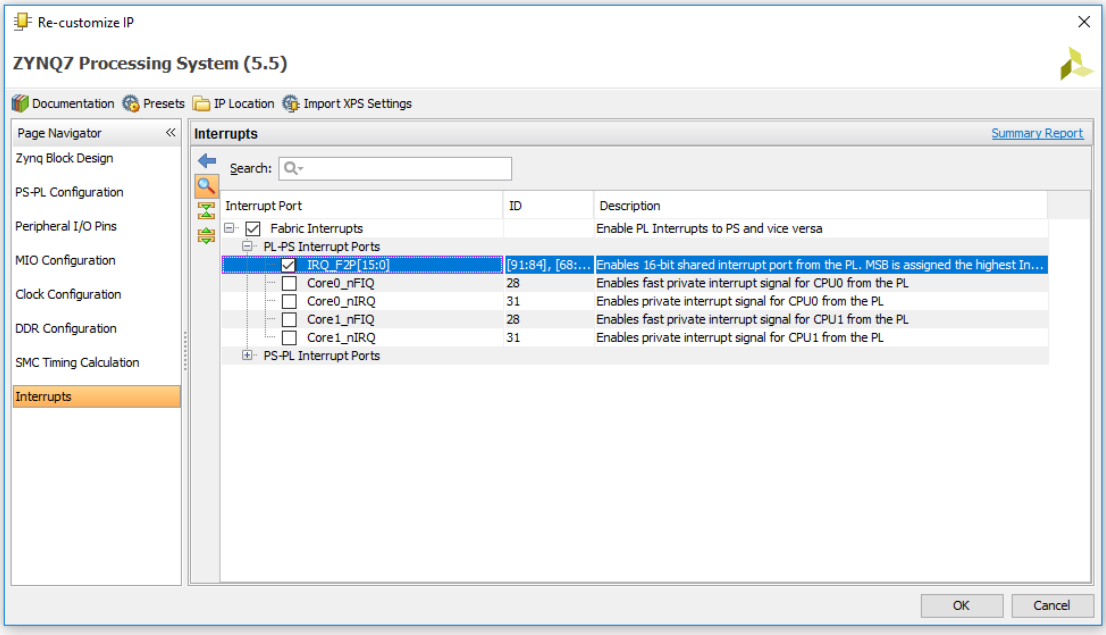

Figure 8: Interrupts Configuration

10. Click **OK** to apply the configurations to the IP core.
11. Add **AXI Direct Memory Access**.
12. In AXI DMA customization window, unmark **Enable Control/Status stream**.
13. Click **Run Block Automation** and select the following connections:
    - S_AXI_LITE (AXI DMA) to M_AXI_GP0 (Zynq PS)
    - S_AXI_HP0 (Zynq PS) to M_AXI_MM2S (AXI DMA)

    It will connect the ports and create other auxiliary IP cores.
14. Click again **Run Block automation** and select the following connections:
    - M_AXI_S2MM (AXI DMA) to S_AXI_HP0 (Zynq PS)
    - M_AXI_SG (AXI DMA) to S_AXI_HP0 (Zynq PS)
15. Add **AXI4-Stream Data FIFO.**
16. Connect manually the FIFO to the others IP cores with the following links:
    - S_AXIS (FIFO) to M_AXIS_MM2S (AXI DMA)
    - M_AXIS (FIFO) to S_AXIS_S2MM (AXI DMA)
    - s_axis_aresetn (FIFO) to peripheral_aresetn (Processor System Reset)
    - s_axis_aclk (FIFO) to FCLK_CLK0 (Zynq PS)
17. Add **Concat**.
18. Connect manually Concat to the others IP cores with the following links:
    - dout (Concat) to IRQ_F2P (Zynq PS)
    - mm2s_introut (AXI DMA) to In0 (Concat)
    - s2mm_introut (AXI DMA) to In1 (Concat)
19. In the **Sources** tab of the block design, right click on the *.bd* file and select **Create HDL wrapper.** Mark **Let Vivado manage wrapper and auto-update** and click **OK**.
20. Synthesize, implement and generate bitstream.
21. Select **File** → **Export** → **Export hardware**. Mark **Include bitstream** and select the folder where the results are exported. Finally, click **OK**.

4 Performance

GigaX features up to 200Mbps full duplex data transfer between Ethernet and PL. Figure 9 shows the frame loss ratio as a function of throughput for UDP traffic, and the maximum transfer rate as a function of the frame length. Shorter frames involve processing more headers, losing communication efficiency.

These figures are valid for UDP communication. TCP connections achieve maximum data rates of 190Mbps.

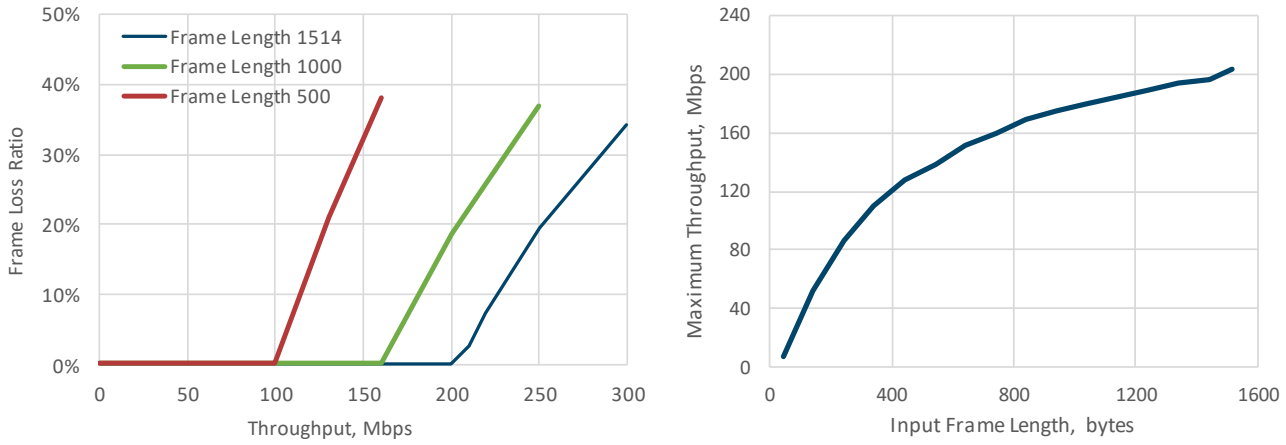

Figure 9: Frame Loss Ratio and Maximum Throughput as a function of UDP frame length

Figure 10 shows the link behavior for an ad-hoc hardware accelerator (Figure 13), when transmitting a data flow at 250 Mbps (dotted line) with a frame length of 1514 bytes. The system is only capable to manage and send back data at 200 Mbps, causing a 20% packet lost.

Figure 11 show the test result for a board working as network offloader. Two computers send data to the GigaX device, which are sent back to a target host. Two UDP flows at 90Mbps are received, being able to re-transmit them at 180 Mbps (solid line) without losses. The dotted line corresponds to the data sent by one of the devices, time interval in which both sources are sending data to the GigaX board simultaneously.

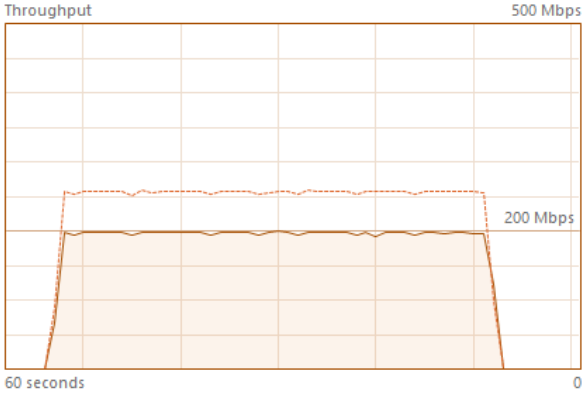

Figure 10: Throughput Limited by Zynq to 200Mbps

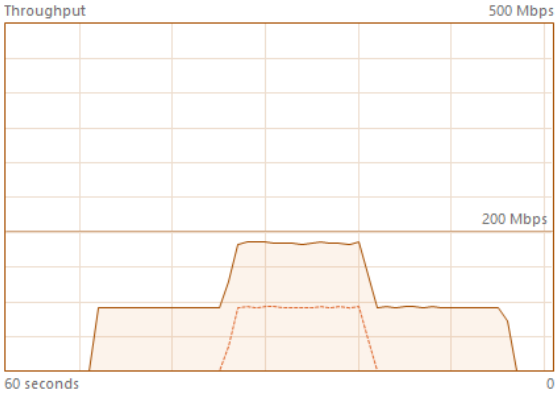

Figure 11: UDP traffic from two devices

www.bertendsp.com gigax@bertendsp.com +34 942 18 10 11

5 Typical Applications

5.1 Ethernet Bridge

In the configuration of Figure 12, the Zynq devices powered by GigaX implement a bridge between networks. UDP frames are sent to PL removing Ethernet headers without changing network and transport layers. At the other end, frames received from PL are sent to the target host in the new network after regenerating Ethernet headers.

The Data Interface can be any wired or wireless communication link that interfaces with the Zynq programmable logic. The configuration parameters for the terminals in Figure 12 are the following:

```
Zynq @LAN1: GigaX_pspl_transfer("192.168.3.10", "00:0A:35:00:01:02", "255.255.255.0",
                "192.168.3.1", 3, 1, "UDP", "UDP", 0, "192.168.0.1", "192.168.5.255",
                "192.175.6.1", "192.175.6.20", "0", "0", 0, 0, "YOUR-LICE-CODE-NUMB")

Zynq @LAN2: GigaX_pspl_transfer("192.175.6.15", "00:0A:35:00:01:03", "255.255.255.0",
                "192.175.6.28", 3, 1, "UDP", "UDP", 0, "192.175.6.1", "192.175.6.255",
                "192.168.3.1", "192.168.3.80", "0", "0", 0, 0, "YOUR-LICE-CODE-NUMB")
```

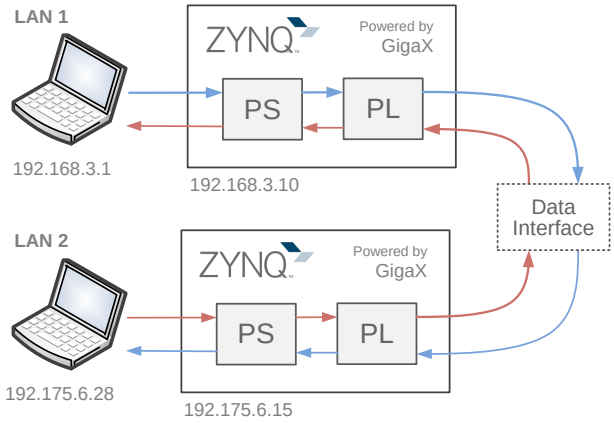

Figure 12: Ethernet Bridge

5.2 Ad-hoc Hardware Acceleration

In the setup of Figure 13, the Zynq is used as hardware accelerator of a computer connected ad-hoc via Ethernet. The high-speed communication link between the computer and the PL allows intensive processing of Ethernet data in the programmable logic.

In this example, packets are sent to PL removing Ethernet headers without changing network and transport layers. The IP addresses and port number of the headers are modified before sending PL data to the computer. Configuration examples for both UDP and TCP packets are provided.

```
UDP Data: GigaX_pspl_transfer("192.168.3.10", "00:0A:35:00:01:02", "255.255.255.0",
                "192.168.3.1", 3, 1, "UDP", "UDP", 0, "192.168.0.1", "192.168.4.255",
                "192.168.0.1", "192.168.4.255", "-1", "-1", -1, 0, "YOUR-LICE-CODE-NUMB")

TCP Data: GigaX_pspl_transfer("192.168.3.10", "00:0A:35:00:01:02", "255.255.255.0",
                "192.168.3.1", 3, 1, "TCP", "UDP", 1234, "192.168.0.1", "192.168.4.255",
                "192.168.0.1", "192.168.4.255", "-1", "-1", -1, 0, "YOUR-LICE-CODE-NUMB")
```

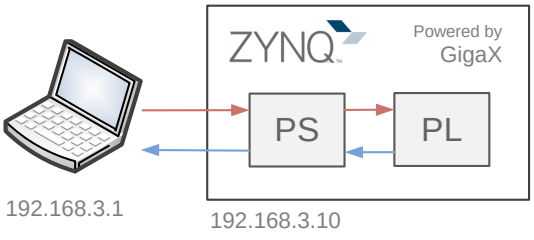

Figure 13: Ad-hoc Hardware Acceleration

5.3 Network offloader

As network offloader, a Zynq board powered by the GigaX API can receive data from several devices connected to a network, process them and send the resulting frames back to the source of each data flow or to a selected device.

In Figure 14, UDP frames are received via Ethernet and sent to PL without modifying transport and network layers. The source and destination IP addresses and port numbers are exchanged when generating PL-to-PS headers to send the data back to each source host.

Figure 15 is a similar case, but all the frames are sent back to the device specified by the *pl2ps_dst_ip* and *pl2ps_dst_port* parameters.

```
Data back to the source: GigaX_pspl_transfer("192.168.3.10", "00:0A:35:00:01:02",
    "255.255.255.0", "192.168.3.5", 3, 1, "UDP", "UDP", 0, "192.168.0.1", "192.168.4.255",
    "192.168.0.1", "192.168.4.255", "-1", "-1", -1, 0, "YOUR-LICE-CODE-NUMB")

Data back to target device: GigaX_pspl_transfer("192.168.3.10", "00:0A:35:00:01:02",
    "255.255.255.0", "192.168.3.5", 3, 1, "UDP", "UDP", 0, "192.168.0.1", "192.168.5.254",
    "192.168.1.1", "192.168.1.255", "-1", "192.168.3.3", 1234, 0, "YOUR-LICE-CODE-NUMB")
```

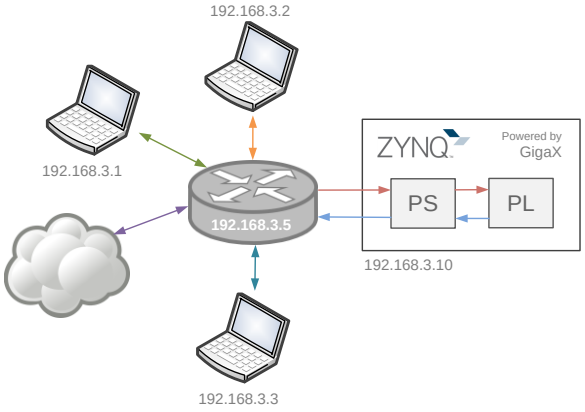

Figure 14: Network Offloader – Data send back to each source

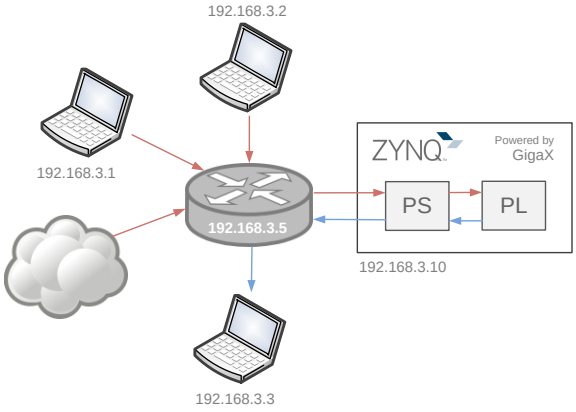

Figure 15: Network Offloader – Data send back to a selected device

6 Known Issues

Issue	Description	Workaround
GigaX loss Ethernet packets at reception in heavy traffic situations when frame length is small	Ethernet peripheral "EmacPS" has a hardware bug (SI# 692601) which causes that the reception path becomes unresponsive when reception throughput is high (> 80 Mbps) and the frame length is small (< 150 bytes of payload). The shorter the frame is, the lower is the reception throughput needed for blocking Ethernet peripheral, because the number of frames processed per second increases.	Use a bigger frame length or a lower rate.

7 Licensing

The software package includes a perpetual licence for all the projects in your company, two years of software upgrades, and unlimited support for bug fixing.

The GigaX Software Licence Agreement details the terms and conditions for the software utilisation. A unique product key is provided, allowing the licensee to use the software according to this agreement. The key is added as a parameter of the GigaX function call (see Table 2).

8 Contact and Support

Technical support: gigax@bertendsp.com
Contract & Sales: sales@bertendsp.com
Direct Phone: +34 942 18 10 11