XAPP1082 (v2.0) August 5, 2013

PS and PL Ethernet Performance and Jumbo Frame Support with PL Ethernet in the Zynq-7000 AP SoC

Authors: Srinivasa Attili, Sunita Jain, Sumanranjan Mitra

Summary

The focus of this application note is on Ethernet peripherals in the Zynq®-7000 All Programmable (AP) SoC. This application note describes using the processing system (PS) based gigabit Ethernet MAC (GEM) through the extended multiplexed I/O (EMIO) interface with the 1000BASE-X physical interface using high-speed serial transceivers in programmable logic (PL). This application note also describes the implementation of PL-based Ethernet supporting jumbo frames.

The designs provided with this application note enable the use of multiple Ethernet ports, and provide kernel-mode Linux device drivers. In addition, this document includes Ethernet performance measurements with and without checksum offload support enabled.

Introduction

The Zynq-7000 AP SoC device is based on the Xilinx® All Programmable SoC architecture. These products integrate a dual core ARM® Cortex™-A9 MPCore™ based PS and PL in a single device.

The PL includes the programmable logic, configuration logic, and associated embedded devices. The PS comprises the processor unit, on-chip memory, external memory interfaces, and peripheral connectivity interfaces including two gigabit ethernet controllers (GEM), which access PL signals through the extended multiplexed I/O (EMIO) interface to connect different physical interfaces.

In the designs provided with this application note, the PS-GEM0 is connected to the Marvell PHY through the reduced gigabit media independent interface (RGMII), which is the default setup for the ZC706 board. The focus of this application note is the design of additional Ethernet ports.The designs described in this application note are:

- PS Ethernet (GEM1) that is connected to a 1000BASE-X physical interface in PL through an EMIO interface
- PL Ethernet implemented as soft logic in PL and connected to the 1000BASE-X physical interface in PL

© Copyright 2013 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. AMBA, AMBA Designer, ARM, ARM1176JZ-S are trademarks of ARM in the EU and other countries. HDMI, HDMI logo, and High-Definition Multimedia Interface are trademarks of HDMI Licensing LLC. All other trademarks are the property of their respective owners.

Figure 1 shows the various Ethernet implementations on the ZC706 board.

Note: The three Ethernet links cannot be active at the same time because the ZC706 board offers only one SFP cage for the 1000BASE-X PHY. The PS-GEM0 is always tied to the RGMII Marvell PHY. The PS-GEM1 and the PL Ethernet share the 1000 BASE-X PHY so only two Ethernet Links can be active at a given time.

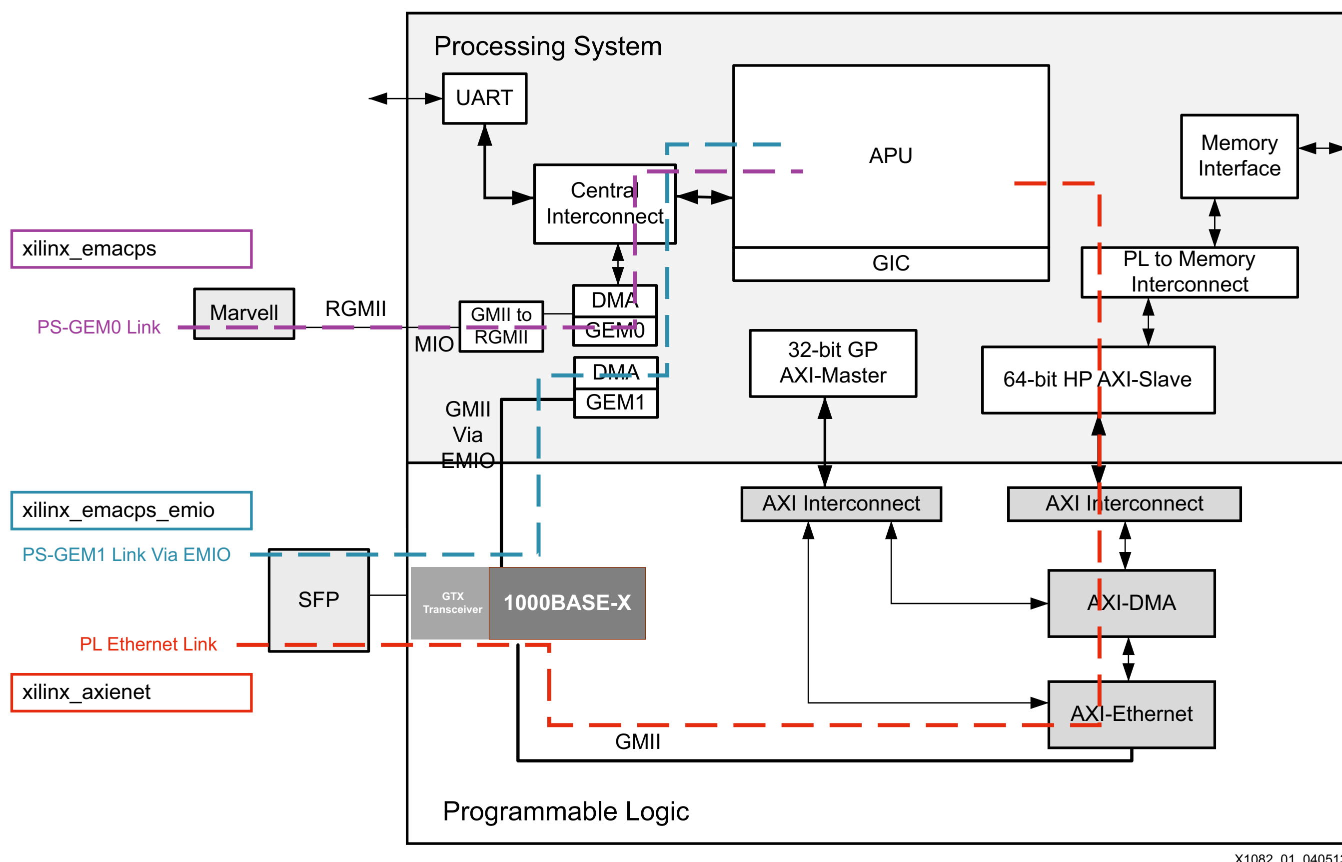

Figure 1: **Zynq-7000 AP SoC Ethernet Interface**

Reference Clock Generation

The design uses the GTX transceiver X0Y10 on the Zynq-7000 AP SoC connected to the SFP cage on the ZC706 board for 1000BASE-X transceivers. The GTX transceiver reference clock (125 MHz differential) is generated from the Si5324 jitter attenuator on the ZC706 board. The clock divider values are adjusted to generate 125 MHz from the 114.285 MHz crystal connected to the Si5324.

The Si5324 driver programs the device over the I2C interface to generate the required clock value. This driver initializes the Si5324 *once* at boot time. At boot time, the driver probe function is invoked by the I2C framework. The probe function fetches the client address from the device tree and programs the hardware registers with the relevant values. See [Ref 1] for details on Si5324.

Using PS GEM Through EMIO

This section describes how to use the PS Ethernet block GEM1 with the PL PHY through the EMIO interface. The PS Ethernet block is exposed to the PL through the EMIO, GMII, and the management data input/output (MDIO) interfaces. The 1000BASE-X PCS/PMA core is used as Ethernet physical media, and uses the high-speed serial transceivers to access the SFP cage on the ZC706 board. The connection between the SFP cage to a standard Ethernet LAN is through an SFP-to-RJ45 converter module.

Hardware Design

Figure 2 shows the design block diagram. The GMII interface connects the PHY and PS EMAC through the EMIO pins. The GEM1 block is enabled while generating the hardware system. See the *Checksum Offloading* section in the *Gigabit Ethernet Controller* chapter in [Ref 2] for information on checksum offloading in PS_GEM. See the chapter on using 1000BASE-X PHY with Zynq-7000 AP SoC in [Ref 3] for more information.

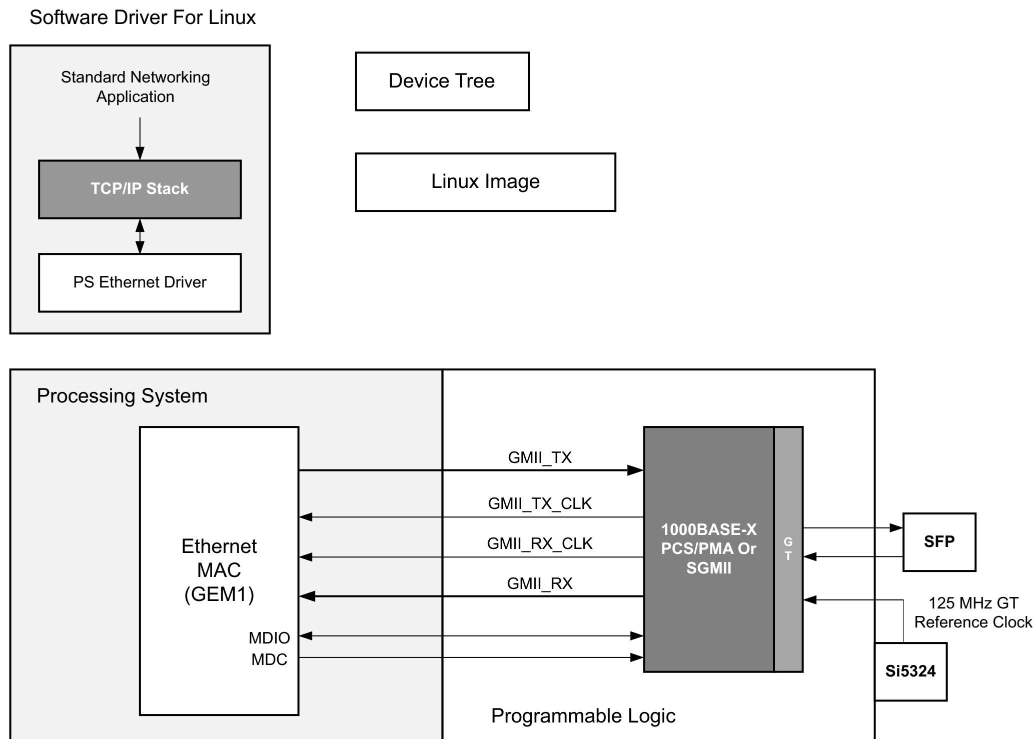

Figure 2: **Design Block Diagram**

Software Design

The design uses the `xilinx_emacps_emio.c` driver code (included with the reference design zip file), which is based on the PS GEM driver `xilinx_emacps.c`.

To enable GEM1 through the EMIO interface, specific registers must be programmed. This is part of the PS configuration data used by the Zynq-7000 AP SoC first stage bootloader (FSBL). On system generation with the EMIO enabled for the second GEM, the `ps7_init.tcl` file that is available on SDK export of the hardware design, includes the register settings by default, which are:

- To select the EMIO as the source of receive clock, data, and control signals:

  Set SLCR.GEM1_RCLK_CTRL[SRCSEL] bit to 1
- To select the EMIO as the source to generate reference clock:

  Set SLCR.GEM1_CLK_CTRL[SRCSEL] bit to 3'b1xx

  where 'x' is don't care (can be either 1 or 0)

The xilinx_emacps_emio driver uses the DMA controller attached to the GEM Ethernet controller in the PS. This driver is responsible for several functions, including DMA descriptor rings setup, allocation, and recycling. The interrupt handling is done only for the PS GEM events, as the interrupt status implicitly reflects the DMA events as well. Additionally, the device tree is updated to include PS-GEM1 with relevant parameters.

Note: To support other PL physical interfaces, such as TBI, the hardware design and device tree must be edited. The PHY specific initialization is handled in the phylib subsystem in the Linux driver and information regarding the PHY can be provided in the device tree. To use the phylib subsystem for PHY programming, the phylib subsystem must support the PHY initialization routine for the desired PHY. If not, the PHY initialization routine should be implemented in the driver.

Linux Driver

A monolithic Linux device driver is provided for this design. Figure 3 shows the software architecture for the PS Ethernet interfaces. Appendix A includes the device tree for this design.

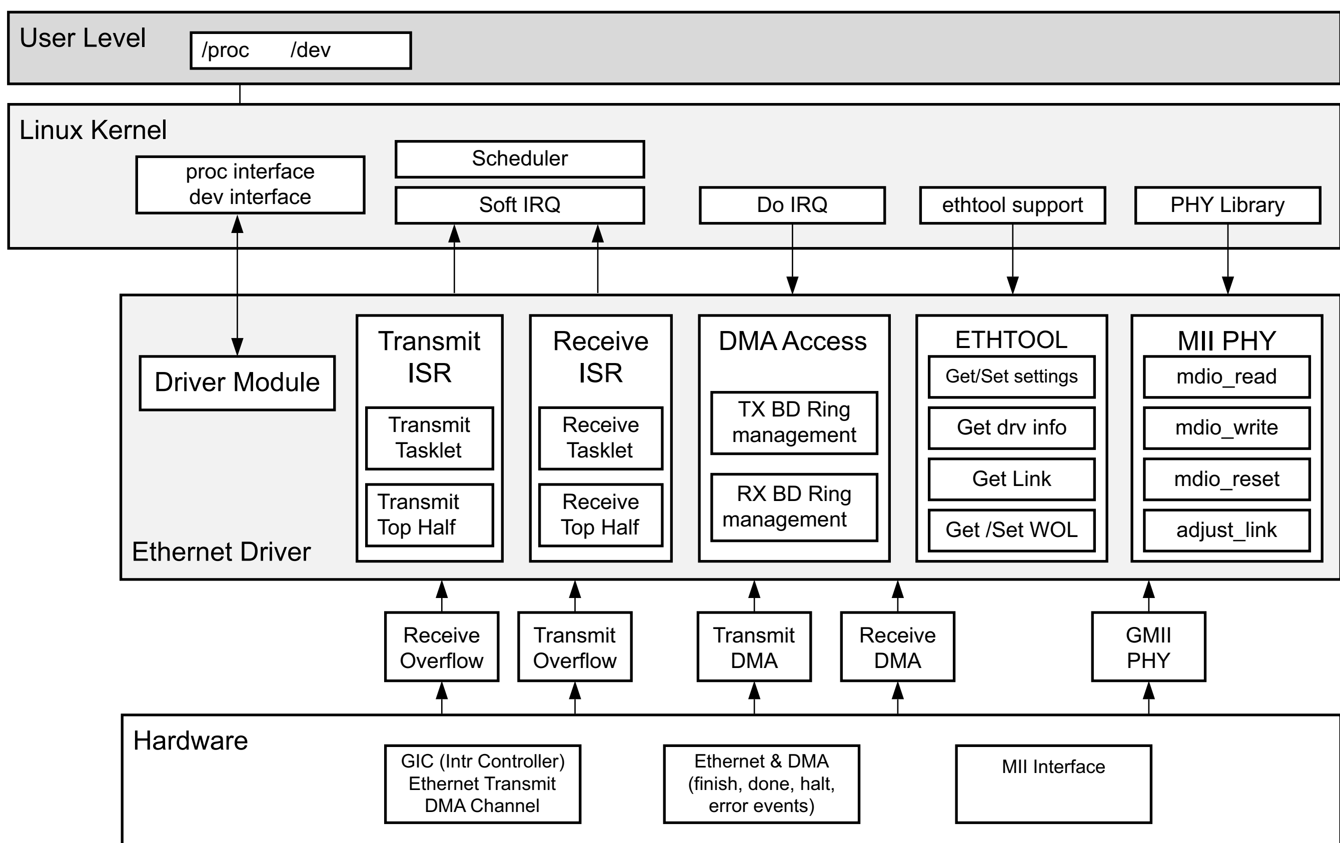

Figure 3: **Software Architecture PS Ethernet Interfaces**

Using PL Ethernet

This section describes a PL implementation of Ethernet. The design consists of the AXI Ethernet, AXI DMA, and AXI Interconnect IP cores. The AXI Ethernet IP is connected to the 1000BASE-X PHY. The design uses the high performance (HP) port for fast access to the PS-DDR memory, however, the general purpose slave port can also be used if the HP port is occupied with other peripherals.

Hardware Design

Figure 4 shows the block diagram for the Ethernet implementation in PL.

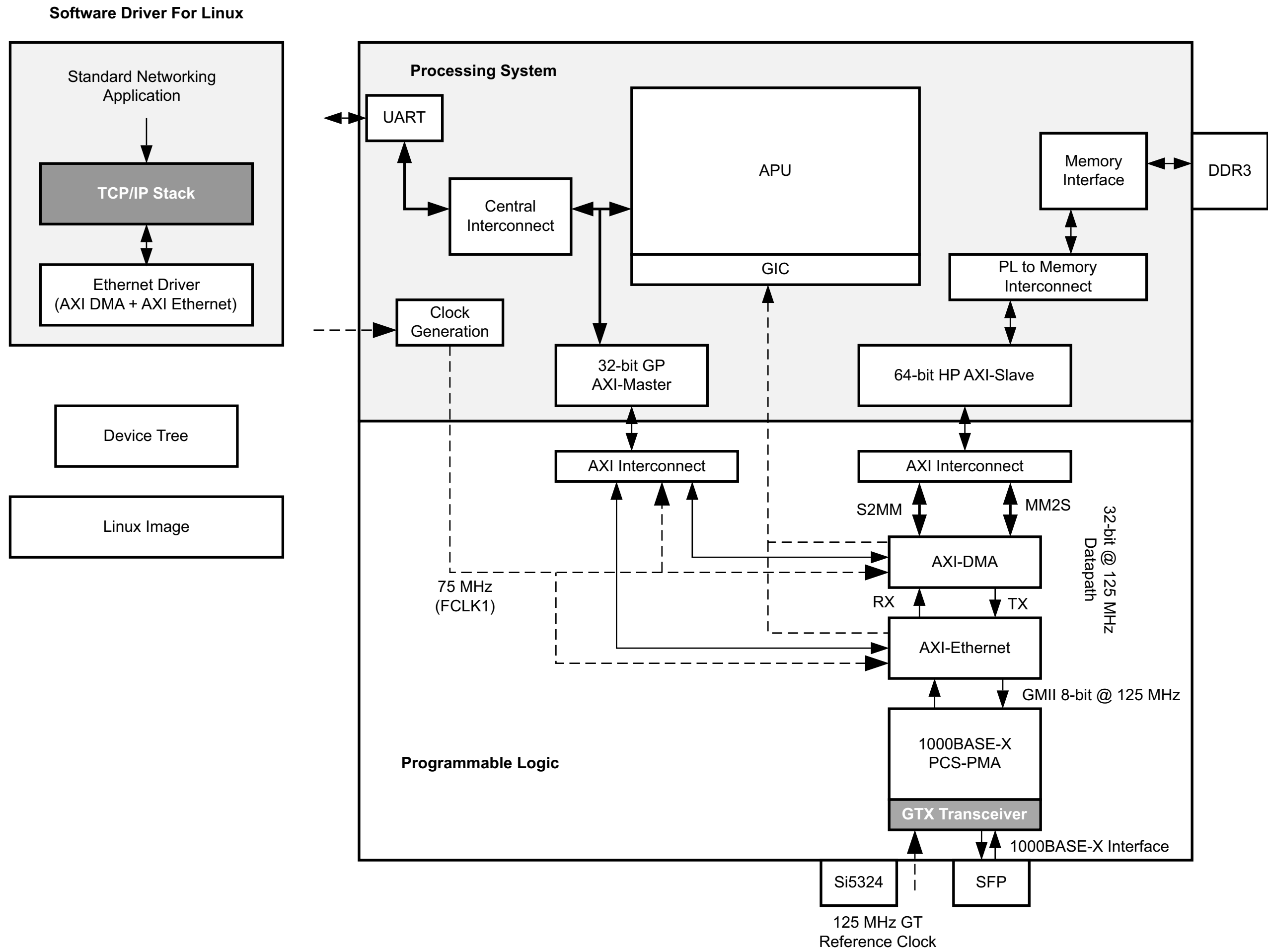

Figure 4: **PL Ethernet Design Block Diagram**

The HP port is used for fast data transfers between the PL and the PS DDR3 memory. It connects to the AXI DMA scatter-gather, stream to memory mapped (S2MM), and memory mapped to stream (MM2S) interfaces through the AXI interconnect. This interconnect also performs data-width conversion to connect the 64-bit HP port to the 32-bit interfaces of AXI DMA. In the AXI DMA, both the scatter-gather option and data realignment engine are enabled for the S2MM and MM2S paths.

The streaming interface of the AXI DMA is connected to the AXI Ethernet IP. The AXI Ethernet IP has full checksum offloading (CSO) enabled and has FIFO depths of 32K to support jumbo frame transfers.

The AXI Ethernet IP is enabled in GMII mode, with the automatic I/O inclusion disabled. The automatic I/O inclusion in the AXI Ethernet IP is disabled to prevent I/O buffer inclusion so that another IP can be connected to these GMII ports within the FPGA. The AXI Ethernet IP is connected to the 1000BASE-X IP through this GMII interface.

For the control interface, a general-purpose AXI master port is enabled in the PS. This port connects to the AXI DMA and AXI Ethernet IP cores. The 1000BASE-X PHY registers are accessed using the MDIO interface provided through the AXI Ethernet IP.

The interrupt ports from the AXI DMA and the AXI Ethernet IPs are connected to the general interrupt controller (GIC) in the PS.

Note: For further details on the IP cores, see [Ref 3], [Ref 4], and [Ref 5].

Software Design

This section describes the software aspects of the design.

The monolithic Linux driver code facilitates the functionality listed here:

- PL Ethernet MAC accesses
- AXI DMA transfers
- Physical media initialization for 1000BASE-X interface

Linux Driver

Figure 5 shows the software architecture for the design. A monolithic Linux device driver is provided.

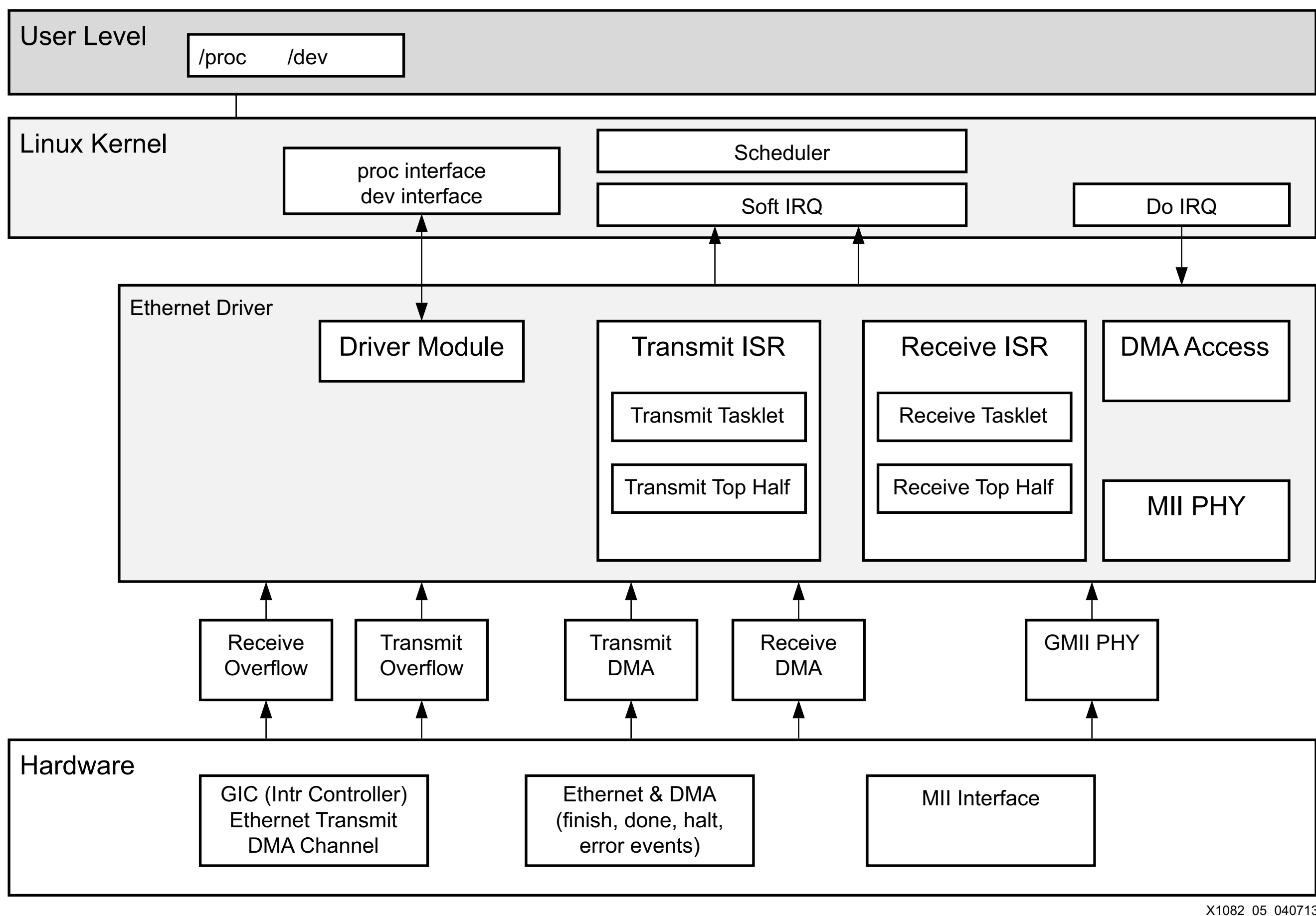

Figure 5: **Driver Architecture for PL Ethernet**

The driver is divided into these sections (see Appendix C for more information):

- Initialization
- MAC driver hooks
- PHY timer
- Interrupt service routines

About Device Trees

A device tree contains the kernel and driver configuration settings. These settings are parsed by the drivers at the time of loading and parameters are set as defined in the device tree.The Linux drivers' device trees consist of:

- PS Ethernet MAC EMIO-specific: PS GEM1 section, containing PS MAC parameters.
- PL Ethernet-specific:
  - DMA section, containing AXI DMA parameters.
  - Ethernet section, containing the AXI Ethernet MAC parameters.
- I2C section, containing Si5324 parameters. The Si5324 device is the reference clock generator for the 1000BASE-X PHY transceivers.

Hardware Requirements

Testing the design requires:

- Standard PC, preferably running the Linux OS
- Ethernet port supporting 1000 Mb/s
- Netperf tool [Ref 6]
- Zynq-7000 AP SoC ZC706 board with an SFP-to-RJ45 adapter module for testing

Figure 6 shows the board setup. Jumper J17 should be set to enable transmission through the SFP. The design was tested with the HP 378928-B21 Cisco Gigabit Ethernet RJ45 SFP Module. Linux PC with Fedora Core 16 (kernel 3.4) with netperf v2.6 is used for testing here.

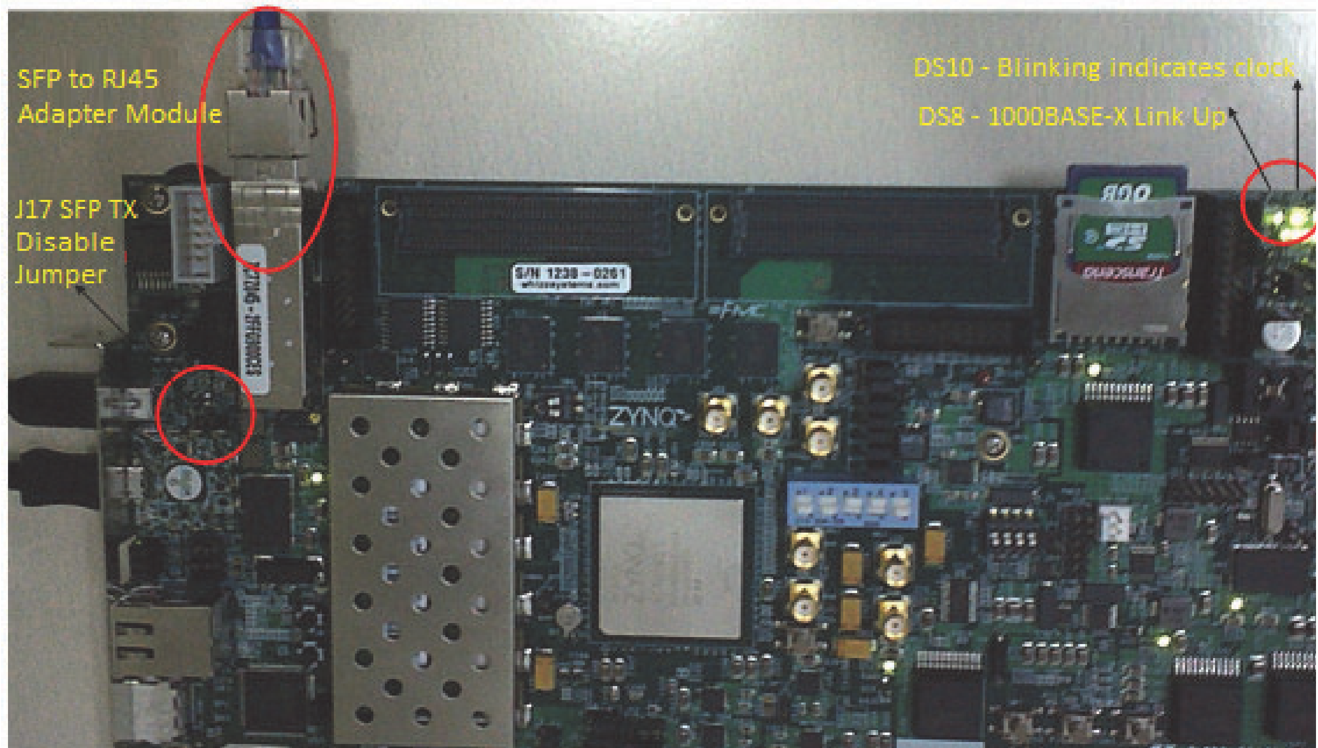

Figure 6: **Board Setup**

Ethernet Performance

This section presents a summary of Ethernet throughput associated with the designs.

The performance of various Ethernet applications at different layers is less than the throughput of the software driver and the Ethernet interface. This is due to the various headers and trailers inserted in each packet by the various layers of the networking stack. Ethernet is used as a medium to carry traffic. Various protocols, such as TCP/UDP, implement protocol specific header/trailer formats.

CPU Affinity Considerations

In a multi-processor environment, CPU affinity is the ability of an OS scheduler to bind a certain process to a given processor. The OS scheduler tries to schedule a process on the same processor where it last executed. If the processor is not available, the process is scheduled on a different processor.

Binding a process to a processor ensures that the process is always scheduled on the same processor. The primary benefit of binding a process with a processor is optimal cache performance, as it circumvents the invalidating of cache that is necessary each time a process is scheduled on a different processor.

The CPU affinity of a process can be altered with the taskset program in Linux. For all benchmarking results provided, netserver or netperf was bound to CPU2 using taskset. In this example, binding netserver and netperf results in significant performance improvement:

```
zynq>  taskset 2 ./netserver
zynq> taskset 2 ./netperf -H <peer IP address>
```

Note: Zynq Linux has netperf and netserver available on it. Additional netperf options used are explained in sections where results are presented.

Test Methodology

Netperf v2.6 [Ref 6] is used as the testbench for measuring Ethernet performance. It is a data transfer application running on top of the TCP/IP stack and operates on a client-server model.

Netperf works with a concept of message size. Message size indicates TCP payload size. The actual frame size on line includes overheads (TCP and IP headers and Ethernet headers) in addition to the message size. The numbers provided are run with TCP_NODELAY (-D option) enabled. This option disables the Nagle algorithm which coalesces the TCP segments to improve performance for smaller size frames.

The command for message size variation is- netperf -H <ip> -- -m <msg_size> -D

Note: Due to networking stack implementation changes in different kernel versions, results are expected to vary when tests are run on various platforms.

PS Ethernet through the EMIO: Throughput Observation

Figure 7 shows the throughput variation for PS-GEM (netperf run with -s 16384 option). It is seen that throughput improves with increasing message size. The dip in performance for the 1,494B message size is due to the splitting of the frame into two - one with a 1,448B message size and another with a 46B message size as MTU=1,500B. This gives rise to the traffic of a larger size packet followed by a smaller size packet on wire.

Note: Performance is the same between GEM0 using RGMII through MIO and GEM1 using 1000BASE-X through the EMIO because the MAC hardware and driver are the same and the only difference is in the PHY initialization.

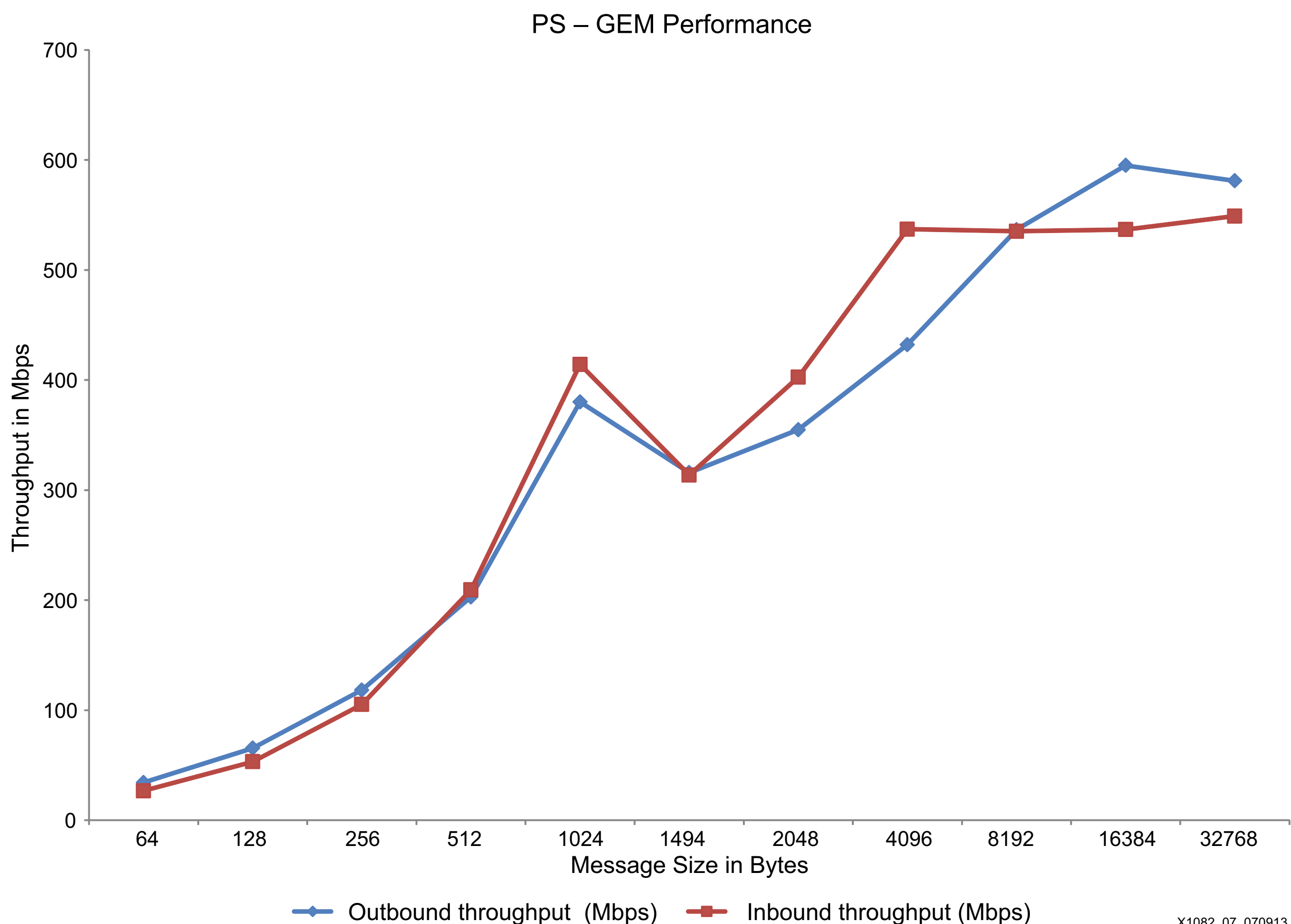

Figure 7: **PS GEM Throughput Observation**

PL Ethernet: Throughput Observation

Figure 8 shows a graph of PL EThernet throughput with varying message size (netperf run with -s 65536 option). As can be seen, throughput improves with increasing message size.

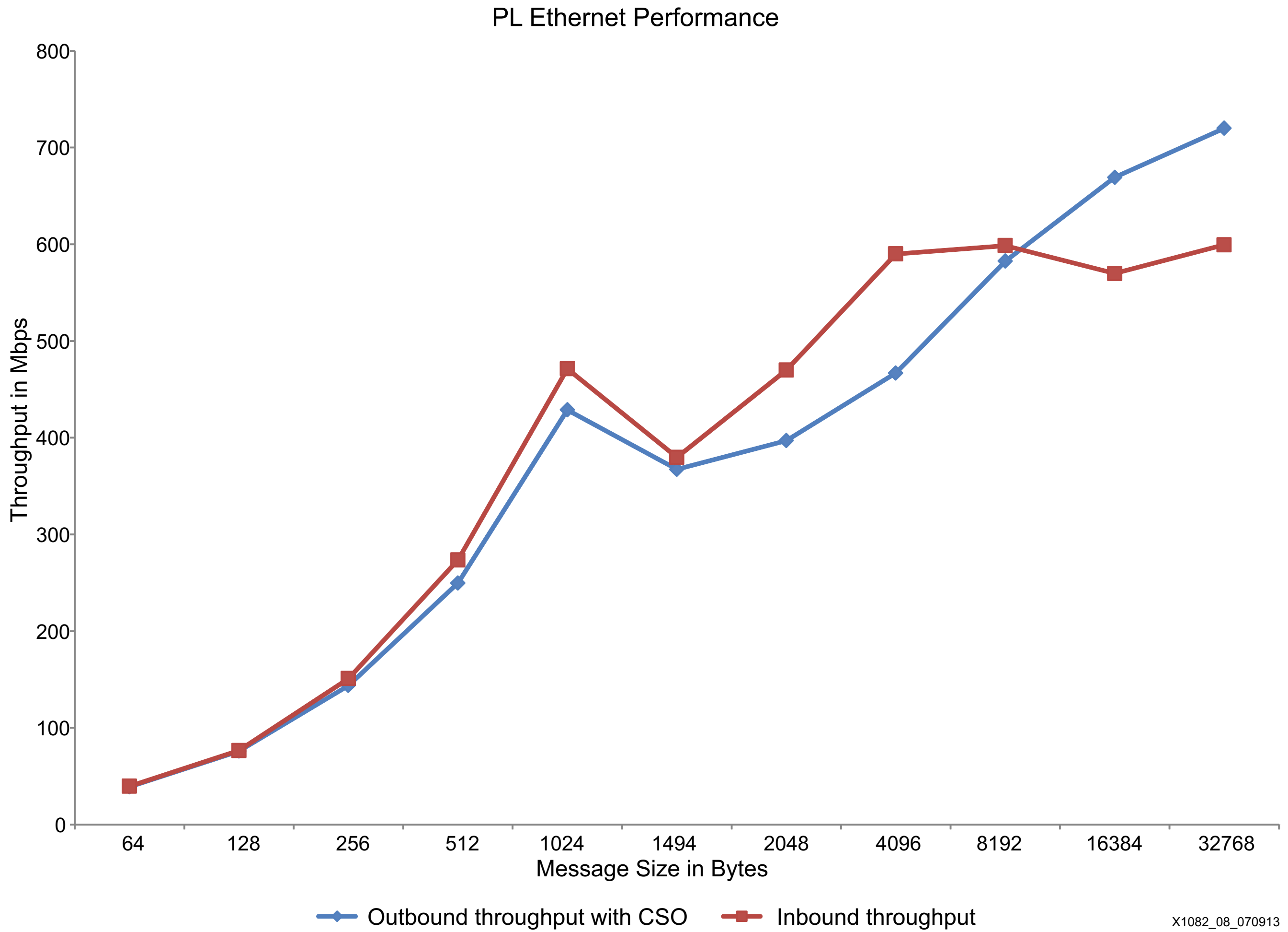

Figure 8: **PL Ethernet Throughput**

Figure 9 shows how Checksum Offload (CSO) improves throughput and CPU utilization. The test here was run with Nagle's Algorithm enabled to ensure the maximum possible performance achievable. Checksum calculation involves byte operations that are costly in terms of CPU cycles. Offloading the checksum calculation to the hardware frees up CPU cycles.

The enabling or disabling of CSO is determined by the driver from device tree (see Appendix B).

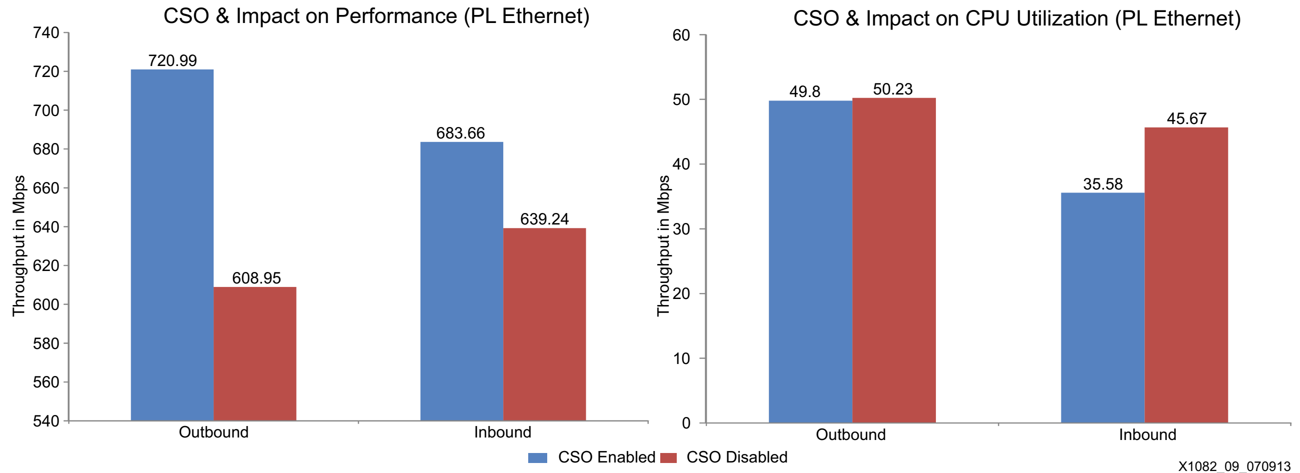

Figure 9: **Impact of CSO on Throughput and CPU Utilization**

Jumbo Frame Performance

The AXI Ethernet IP supports jumbo frames. Jumbo frames are defined as Ethernet frames greater than 1514B.

Maximum Transmission Unit (MTU) is the largest amount of data a protocol packet can carry. Figure 10 illustrates throughput variation with MTU.

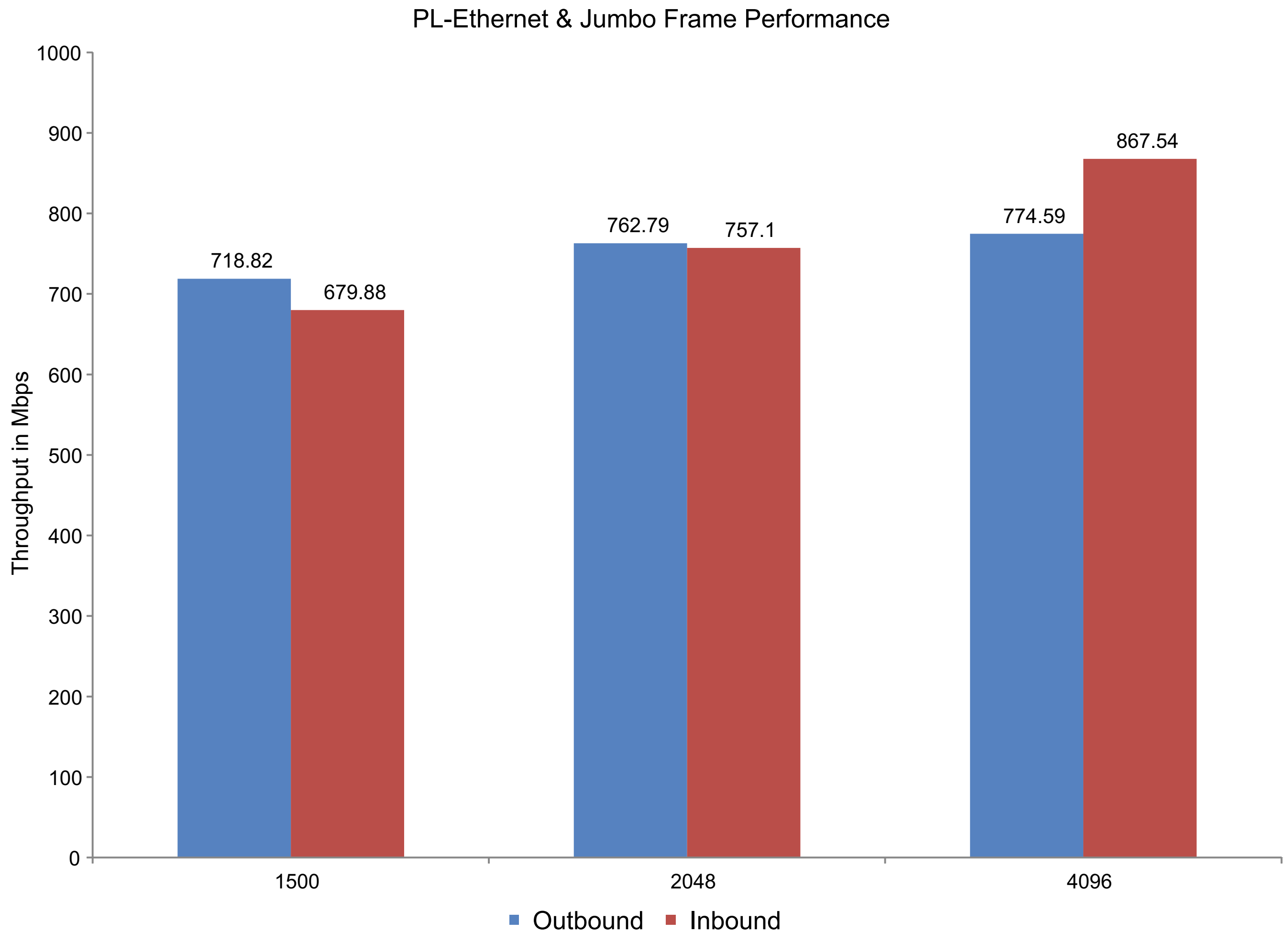

Figure 10: **Jumbo Frame Performance in PL Ethernet**

This example shows how to use the ifconfig utility to change MTU values. The Ethernet interface is eth1 in this example:

```
zynq> ifconfig eth1 down
zynq> ifconfig eth1 mtu <2048|4096>
zynq> ifconfig eth1 up <IP address>
```

Conclusion

This application note provides designs for implementing the PS Ethernet through the EMIO with PHY and Ethernet implementation in the PL to support multiple Ethernet links and jumbo frames. Performance benchmarking results for the designs are included in this application note.

The test results show a trend of throughput improvement with increasing packet size, and the impact of CSO on both throughput and CPU utilization.

Reference Design

The reference design files for this application note can be downloaded from:

https://secure.xilinx.com/webreg/clickthrough.do?cid=203511

Follow the instructions in the readme for building hardware and software code.

Table 1 shows the reference design matrix.

Table 1: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Xilinx
Target devices (stepping level, ES, production, speed grades)	Zynq-7000 AP SoC
Source code provided	Yes
Source code format	Verilog, C
Design uses code/IP from existing Xilinx application note/reference designs, CORE Generator™ software, or third-party	Yes
Simulation	
Functional simulation performed	No
Timing simulation performed	No
Test bench used for functional and timing simulations	No
Test bench format	N/A
Simulator software/version	N/A
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/version	ISE® 14.6
Implementation software tools/versions used	PlanAhead™ 14.6
Static timing analysis performed?	No
Hardware Verification	
Hardware verified?	Yes
Hardware platform used for verification	ZC706 board

Appendix A

PS EMIO Ethernet Device Tree

This appendix presents Linux driver device tree details for the PS EMIO Ethernet, and I2C for the Si5324 clock generator device on ZC706.

PS EMIO Ethernet

```
ps7_ethernet_1: ps7-ethernet1@e000c000 {
                local-mac-address = [ 00 0a 35 00 00 20 ];
                        compatible = "xlnx,ps7-ethernet-emio-1.00.a";
                        reg = <0xe000c000 0x1000>;
                        interrupts = <0 45 4>;
                        interrupt-parent = <&gic>;
                        phy-handle = <&phy1>;
                        xlnx,ptp-enet-clock = <133000000>;
                        xlnx,slcr-div0-1000Mbps = <8>;
                        xlnx,slcr-div0-100Mbps = <8>;
                        xlnx,slcr-div0-10Mbps = <8>;
                        xlnx,slcr-div1-1000Mbps = <1>;
                        xlnx,slcr-div1-100Mbps = <5>;
                        xlnx,slcr-div1-10Mbps = <50>;
                        #address-cells = <0x1>;
                        #size-cells = <0x0>;
                        mdio {
                                #address-cells = <1>;
                                #size-cells = <0>;
                                phy1: phy@6 {
                                        compatible = "vitesse,v82111";
                                        device_type = "ethernet-phy";
                                        reg = <6>;
                                };
                        };
                };
```

I2C for Si5324

```
i2c@4 {
     #address-cells = <1>;
     #size-cells = <0>;
     reg = <4>;
     rtc@54 {
           compatible = "nxp,pcf8563";
           reg = <0x51>;
     };
     si5324@68 {
           compatible = "si5324";
           reg = <0x68>;
     };
 };
```

Appendix B PL Ethernet Device Tree

This appendix presents Linux driver device tree details for the AXI DMA, AXI Ethernet, and I2C for the Si5324 clock generator device on the ZC706 board.

AXI DMA

```
                        axi_dma_0: axi-dma@40400000 {
                          axistream-connected = <&axi_ethernet_0>;
                          compatible = "xlnx,axi-dma-6.03.a",
"xlnx,axi-dma-1.00.a";
                          interrupt-parent = <&gic>;
                          interrupts = < 0 59 4 0 58 4 >;
                          reg = < 0x40400000 0x10000 >;
                          xlnx,dlytmr-resolution = <0x7d>;
                          xlnx,enable-multi-channel = <0x0>;
                          xlnx,family = "zynq";
                          xlnx,generic = <0x0>;
                          xlnx,include-mm2s = <0x1>;
                          xlnx,include-mm2s-dre = <0x1>;
                          xlnx,include-mm2s-sf = <0x1>;
                          xlnx,include-s2mm = <0x1>;
                          xlnx,include-s2mm-dre = <0x1>;
                          xlnx,include-s2mm-sf = <0x1>;
                          xlnx,include-sg = <0x1>;
                          xlnx,instance = "axi_dma_0";
                          xlnx,mm2s-burst-size = <0x100>;
                          xlnx,num-mm2s-channels = <0x1>;
                          xlnx,num-s2mm-channels = <0x1>;
                          xlnx,prmry-is-aclk-async = <0x1>;
                          xlnx,s2mm-burst-size = <0x100>;
                          xlnx,sg-include-desc-queue = <0x1>;
                          xlnx,sg-include-stscntrl-strm = <0x1>;
                          xlnx,sg-length-width = <0xe>;
                          xlnx,sg-use-stsapp-length = <0x1>;
                  } ;
```

AXI Ethernet

```
 axi_ethernet_0: axi-ethernet@40440000 {
                          axistream-connected = <&axi_dma_0>;
                          clock-frequency = <76923080>;
                          compatible = "xlnx,axi-ethernet-3.01.a",
"xlnx,axi-ethernet-1.00.a";
                          device_type = "network";
                          interrupt-parent = <&gic>;
                          interrupts = < 0 57 4 >;
                          local-mac-address = [ 00 0a 35 01 02 03 ];
                          phy-handle = <&phy1>;
                          reg = < 0x40440000 0x40000 >;
                          xlnx,avb = <0x0>;
                          xlnx,halfdup = <0x0>;
                          xlnx,include-io = <0x0>;
                          xlnx,mcast-extend = <0x0>;
                          xlnx,phy-type = <5>;
                          xlnx,phyaddr = <0x3>;
                          xlnx,rxcsum = <0x0>;
                          xlnx,rxmem = <0x8000>;
                          xlnx,temac-type = <5>;
                          xlnx,rxvlan-strp = <0x0>;
                          xlnx,rxvlan-tag = <0x0>;
                          xlnx,rxvlan-tran = <0x0>;
                          xlnx,stats = <0x1>;
```

```
                xlnx,txcsum = <0x0>;
                xlnx,txmem = <0x8000>;
                xlnx,txvlan-strp = <0x0>;
                xlnx,txvlan-tag = <0x0>;
                xlnx,txvlan-tran = <0x0>;
                xlnx,type = <0x1>;
                mdio {
                        #address-cells = <1>;
                        #size-cells = <0>;
                        phy1: phy@3 {
                                compatible = "marvell,88e1111";
                                device_type = "ethernet-phy";
                                reg = <3>;
                        } ;
                } ;
        } ;
```

I2C for Si5324

```
i2c@4 {
     #address-cells = <1>;
     #size-cells = <0>;
     reg = <4>;
     rtc@54 {
           compatible = "nxp,pcf8563";
           reg = <0x51>;
     };
     si5324@68 {
           compatible = "si5324";
           reg = <0x68>;
     };
 };
```

Appendix C

PL Ethernet Linux Device Driver

This appendix provides information for the Linux PL Ethernet driver.

Initialization

When the driver is inserted into the kernel (using the insmod tool), the entry function is:

```
static int __init axienet_init(void)
{
        platform_driver_register(&axienet_of_driver);
}
```

This in turn invokes the function:

```
static int __devinit axienet_of_probe(struct platform_device *op)
```

The probe function is the actual initialization function that performs these tasks:

- Create the Ethernet driver structure alloc_etherdev().
- Set up the Ethernet driver structure.
- Map the physical device register address space into the kernel address space of_iomap().
- Read the driver configuration properties from the device structure and set the driver flags accordingly. The properties handled are:
    - TX CSO: xlnx, txcsum
    - RX CSO: xlnx, rxcsum
    - RX memory: xlnx, rxmem

- MAC type: xlnx, temac-type
- PHY type: xlnx, phy-type
- DMA node: axistrea-connected
- MAC address: local-mac-address

- Map the D99MA register address space (physical) into Kernel address space of_iomap().
- Get TX IRQ and RX IRQ numbers
- Set MAC address
- MDIO setup
- Setup tasklets

MAC Driver Hooks

The MAC driver supports these handles to interface to upper layers:

- *Open*: this is the driver open routine. It invokes PHY start, allocates ISRs, and enables the interrupts and ISR handling. It also resets the AXI_DMA core and its buffer descriptors are initialized. Additionally, it starts the network interface queues, PHY timer for poll routine.
- *Stop*: this is the driver stop routine. It stops the PHY, removes the interrupt handlers, disable interrupts. AXI_DMA (RX and TX) is stopped and the descriptors are release. Also the DMA tasklet is disabled, network interface queues are stopped, PHY timer is removed.
- *Start_xmit*: this routine is invoked from upper layers to initiate transmission of a packet. It fetches next available descriptor, populate their fields, start transmission, by starting the DMA transfer. It also considers the transmit CSO setting and accordingly populates transmit descriptor user application fields.
- *Change_mtu*: this hook is called to change the MTU size dynamically. It is used to support jumbo frames.
- *Set_mac_address*: this function changes the MAC address of the Ethernet core.

PHY Timer

This driver architecture does not use phylib to manage 1000BASE-X PHY, rather using timer based PHY polling instead. PHY timer is a system timer that invokes a handle for every two ticks. Its handle, poll_gmii, serves two purposes:

- Keep the Ethernet MAC's duplex setting in sync with the PHY setting.
- Update the system with state of the link.

This routine accesses the PHY registers to get the current status of the duplex and link. If it finds that the link is down, it immediately stops the interface and vice versa.

Interrupt Service Routines

The Linux driver has two interrupt service routines (ISR) as follows:

- *Receive ISR*: this ISR handles AXI DMA receive interrupts. It checks for the RX status; if the status is OK, it processes the descriptors and passes them to interface for further process.
- *Transmit ISR*: this ISR handles AXI DMA transmit interrupt. It checks for the TX status; if the status is OK, it clears the descriptors and unmaps corresponding buffers so that CPU can regain ownership of the same. At the end, it invokes interface queue wake-up, so that transmission can resume.

In case of an error status, ISR schedules the tasklet to reset the DMA and Ethernet services, and reconfigures all transmit and receive descriptors.

References

This document uses the following references:

1. Si5324 Data Sheet, www.silabs.com/Support%20Documents/TechnicalDocs/Si5324.pdf
2. *Zynq 7000 All Programmable SoC Technical Reference Manual* (UG585)
3. *LogiCORE IP Ethernet 1000BASE-X PCS/PMA or SGMII v11.5* (PG047)
4. *LogiCORE IP AXI DMA v6.03a* (PG021)
5. *LogiCORE IP AXI Ethernet v3.01a* (DS759)
6. Netperf, www.netperf.org
7. *ZC706 Evaluation Board for the* Zynq-7000 AP SoC *XC7Z045 All Programmable SoC User Guide* (UG954)
8. *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476)

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
04/09/2013	1.0	Initial Xilinx release.
08/05/2013	2.0	Added last sentence under Hardware Requirements. Updated example and note under CPU Affinity Considerations. Updated CPU Affinity Considerations and Figure 7. Deleted Figure 8, "Impact of CSO on PS GEM CPU Utilization". Updated PL Ethernet: Throughput Observation and Figure 8 and Figure 9. Updated Figure 10. Updated ISE and PlanAhead versions from 14.4 to 14.6.

Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapps.