XAPP1082 (v5.0) July 16, 2018

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

Authors: Anil Kumar A V, Radhey Shyam Pandey and Naveen Kumar Gaddipati

Summary

The focus of this application note is on Ethernet peripherals in the Zynq®-7000 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 or SGMII 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.

The reference design files for this application note can be downloaded from the Xilinx website. For detailed information about the design files, see Reference Design.

Introduction

The Zynq-7000 SoC device integrates 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 or SGMII physical interface in PL through an EMIO interface
- PL Ethernet implemented as soft logic in PL and connected to the 1000BASE-X or SGMII physical interface in PL

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 or SGMII PHY. The PS-GEM0 is always tied to the RGMII Marvell PHY. The PS-GEM1 and the PL Ethernet share the 1000 BASE-X or SGMII PHY so only two Ethernet Links can be active at a given time. The 1000BASE-X/SGMII PHY and the GTX transceiver are part of the AXI Ethernet core for PL Ethernet design.

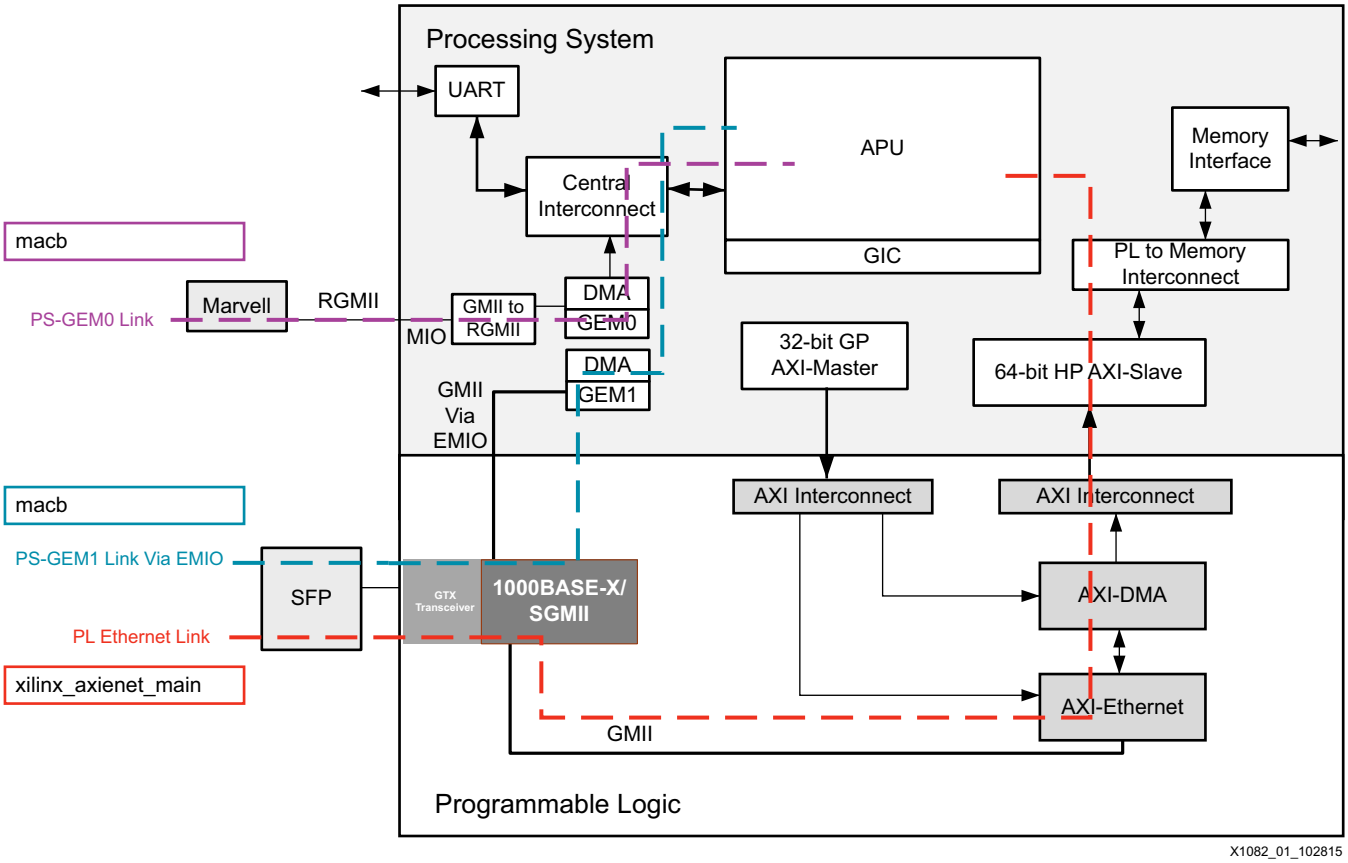

Figure 1: **Zynq-7000 SoC Ethernet Interface**

Reference Clock Generation

The design uses the GTX transceiver X0Y10 on the Zynq-7000 SoC connected to the SFP cage on the ZC706 board for 1000BASE-X or SGMII 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 management data input/output (MDIO) interfaces. The 1G/2.5G Ethernet PCS/PMA or SGMII core is used as Ethernet physical media in 1000BASE-X or SGMII modes, 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 or SGMII PHY with Zynq-7000 SoC in [Ref 3] for more information.

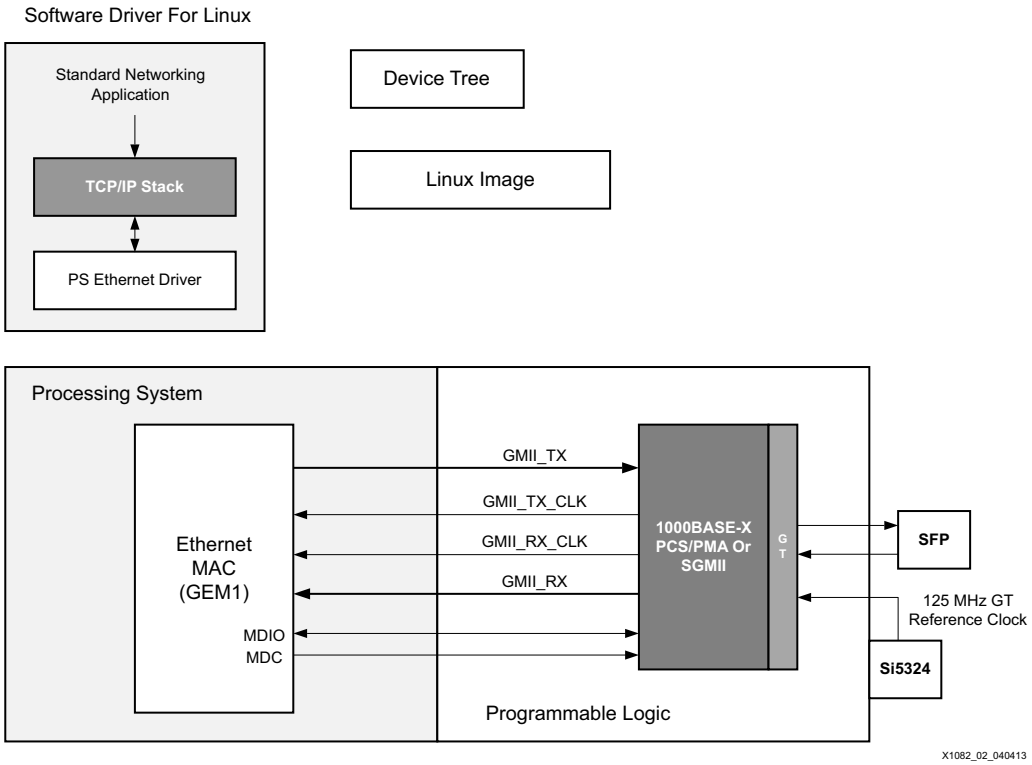

Figure 2: **Design Block Diagram**

Software Design

The design uses the common `macb.c` driver code for the PS- GEM0 and PS-GEM1. 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 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 macb 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 (macb) 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.

Linux Driver

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

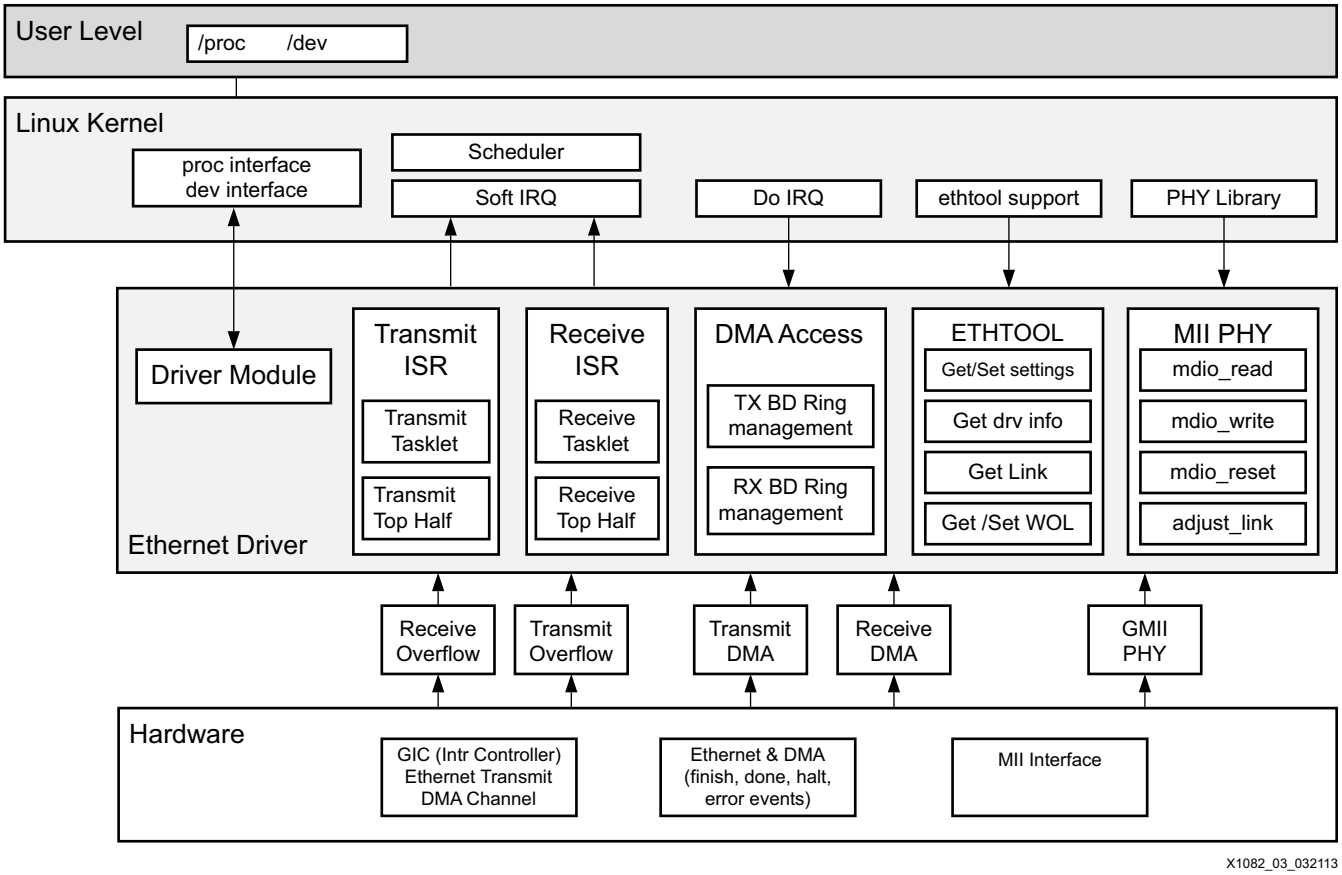

Figure 3: **Software Architecture PS Ethernet Interfaces**

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 or SGMII 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.

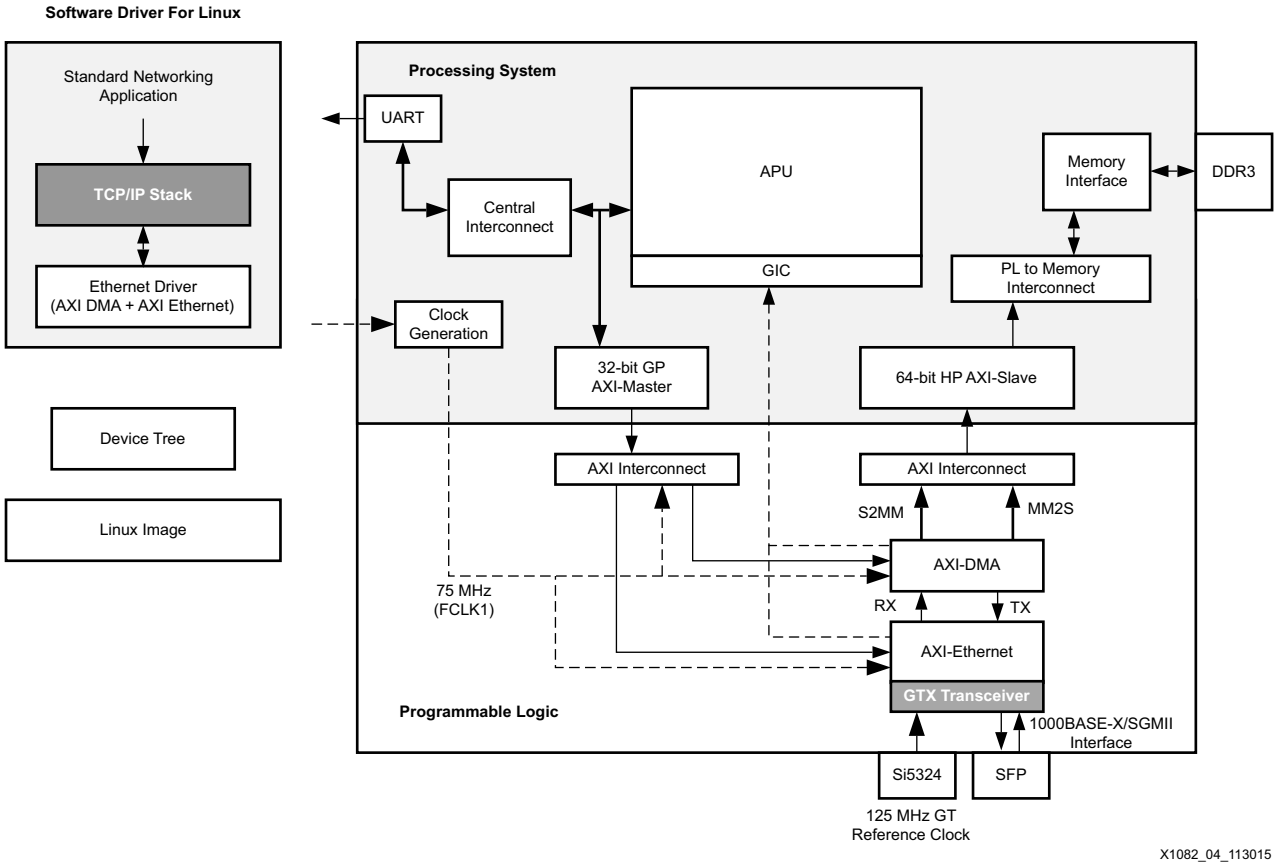

Figure 4: **PL Ethernet Design Block Diagram**

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 subsystem. The AXI Ethernet subsystem has full checksum offloading (CSO) enabled and has FIFO depths of 16K to support jumbo frame transfers.

The AXI Ethernet core implements an Ethernet MAC and supports 1000BASE-X and SGMII PHY interfaces. It connects to the SFP through GTX transceivers through 1000Base-X/SGMII interfaces.

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 cores. The 1000BASE-X or SGMII PHY registers are accessed using the MDIO interface provided through the AXI Ethernet core.

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 or SGMII interface using the phylib subsystem

Linux Driver

Figure 5 shows the software architecture for the design.

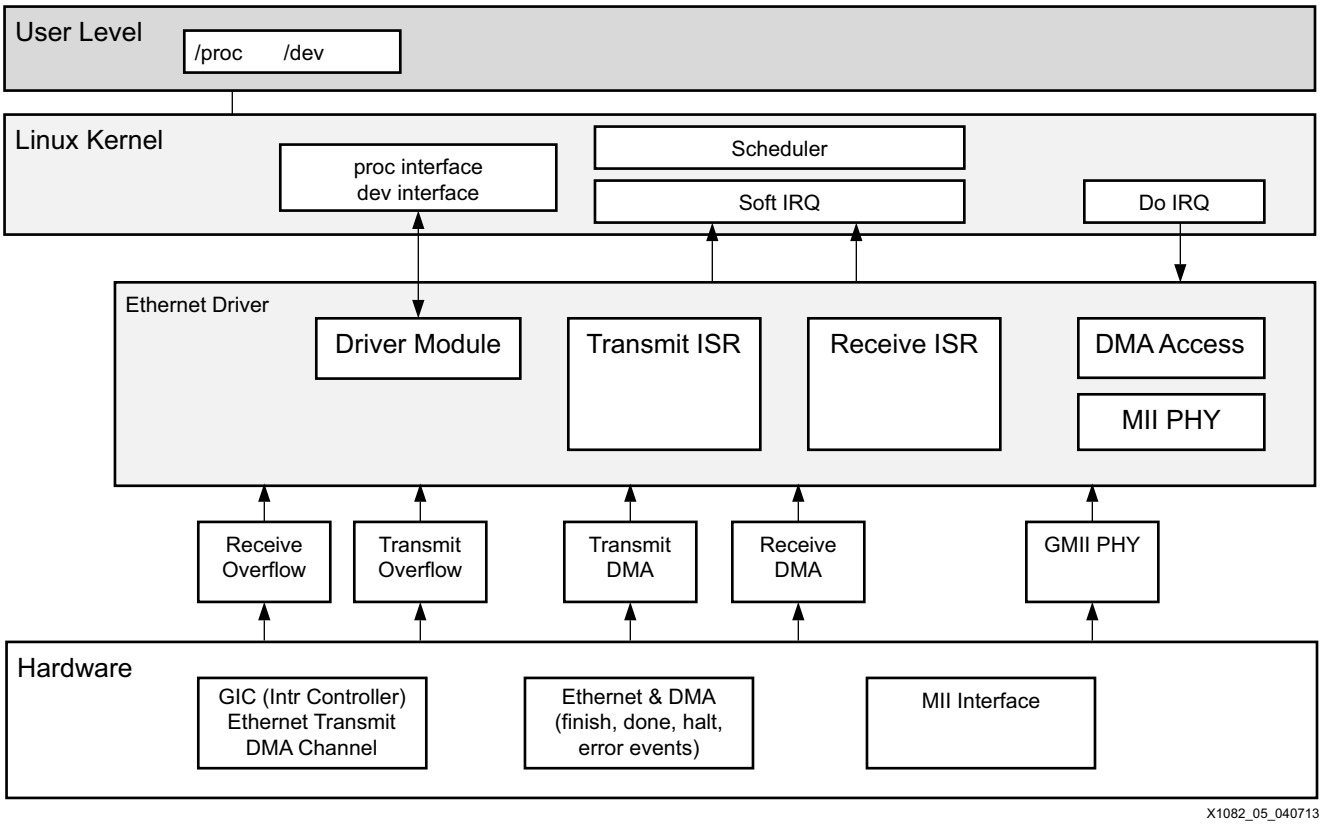

Figure 5: **Driver Architecture for PL Ethernet**

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

- Initialization
- MAC driver hooks
- Interrupt service routines

About Device Trees

The Device Tree is a data structure for describing hardware. Rather than hard coding every detail of a device into an operating system, many aspects of the hardware can be described in a data structure that is passed to the operating system at boot time. 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 or SGMII 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 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.

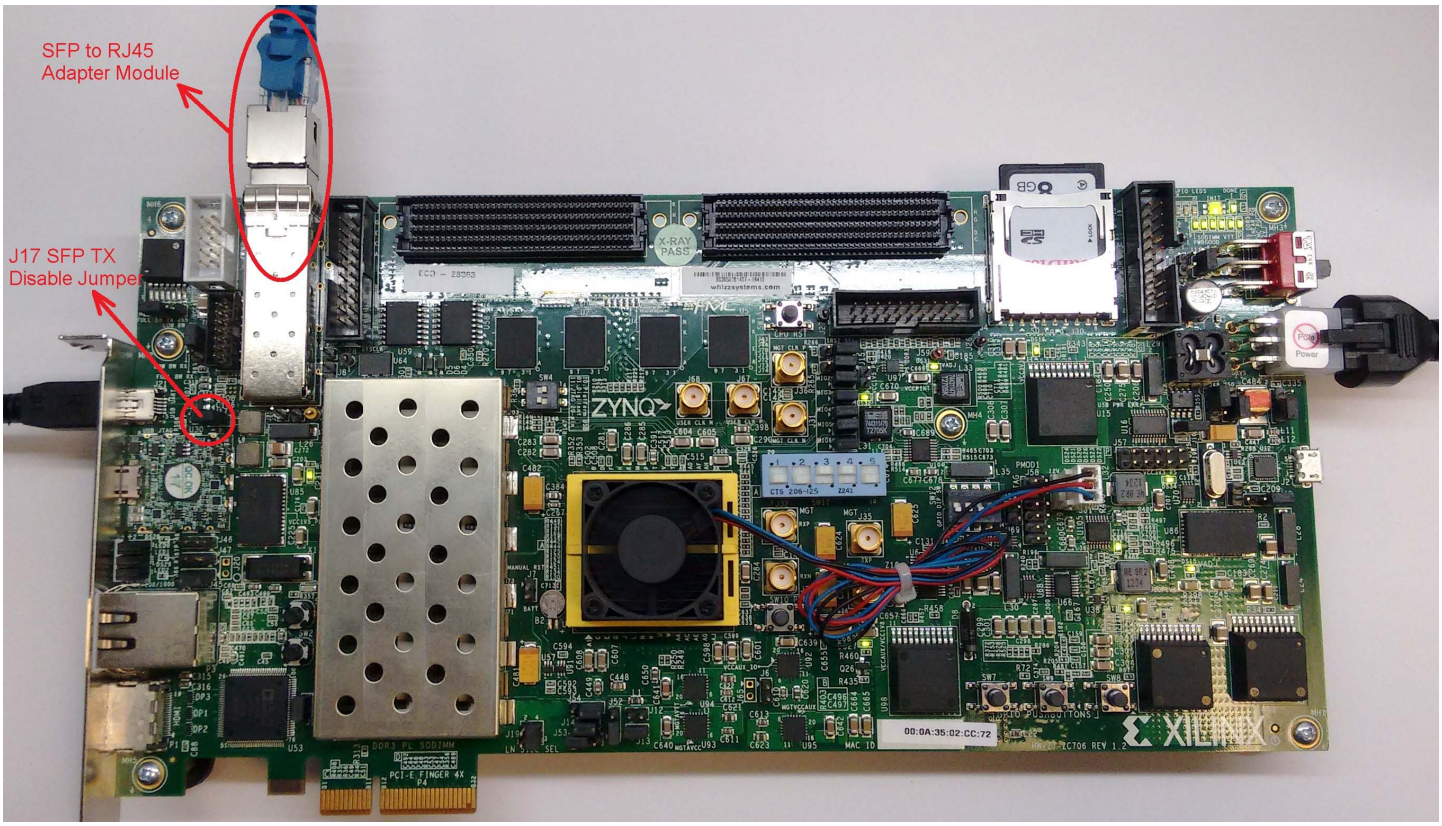

Figure 6: **Board Setup**

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, 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>
```

For test methodology and performance observations, see http://www.wiki.xilinx.com/Zynq+PL+Ethernet.

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 (see http://www.wiki.xilinx.com/Zynq+PL+Ethernet).

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 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	Vivado® tools 2017.4
Implementation software tools/versions used	Vivado tools 2017.4, PetaLinux v2017.4
Static timing analysis performed?	Yes
Hardware Verification	
Hardware verified?	Yes
Hardware platform used for verification	ZC706 board

Appendix A

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:

```
module_platform_driver(axienet_driver);
```

This in turn invokes the function:

```
static int axienet_probe(struct platform_device *pdev)
```

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: axistream-connected
  - MAC address: local-mac-address
  - PHY handle: To handle the PHY device
- Map the DMA register address space (physical) into Kernel address space of_iomap().
- Get TX IRQ and RX IRQ numbers
- Set MAC address
- Get the PHY handle to attach the PHY device
- MDIO setup
- Register to net device

MAC Driver Hooks

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

- *Open*: This driver open routine 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.
- *Stop*: This driver stop routine stops the PHY, removes the interrupt handlers, and disables interrupts. AXI_DMA (RX and TX) is stopped and the descriptors are released. DMA tasklet is disabled and network interface queues are stopped.
- *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.

Interrupt Service Routines

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

- *Receive 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*: 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 TX 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. In-case interface TX queue was stopped due to unavailability of free TX buffers in the transmit path.

Documentation Navigator and Design Hubs

Xilinx Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

This document uses the following references:

1. Si5324 Data Sheet, www.silabs.com/Support%20Documents/TechnicalDocs/Si5324.pdf
2. *Zynq 7000 SoC Technical Reference Manual* (UG585)
3. *1G/2.5G Ethernet PCS/PMA or SGMII v15.0* (PG047)
4. *LogiCORE IP AXI DMA v7.01* (PG021)
5. *AXI 1G/2.5G Ethernet Subsystem v7.0* (PG138)
6. Netperf, www.netperf.org
7. *ZC706 Evaluation Board for the Zynq-7000 XC7Z045 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.
01/07/2015	3.0	Updated Figure 1, Figure 4, and Figure 6. Deleted last sentence preceding Figure 6. Deleted note and performance-related information (subsections "Test Methodology", "PL Ethernet: Throughput Observation", and "Jumbo Frame Performance") from Ethernet Performance and added reference to test methodology and performance observations wiki site. Updated ISE synthesis software from ISE 14.6 to Vivado 2014.4 and PlanAhead version from 14.6 to 2014.4 in Table 1. Deleted Appendix A (PS EMIO Ethernet Device Tree) and Appendix B (PL Ethernet Device Tree).
12/08/2015	4.0.1	Replaced the xilinx_emacps driver with the macb driver. Removed the PHY timer implementation used in the axieth driver and used the phylib interface. Updated Appendix A with the latest axieth driver details. In addition to 1000BASE-X, SGMII PHY interface support is also added to the design. Document is updated to show SGMII mode support. Updated synthesis and implementation software from Vivado 2014.4 to Vivado 2015.4.
07/16/2018	5.0	Updated synthesis and implementation software from Vivado tools 2015.4 to 2017.4.

Please Read: Important Legal Notices