Zentralinstitut für Engineering, Elektronik und
Analytik (ZEA) · Systeme der Elektronik (ZEA-2)

Implementation of UDP communication on a ZYNQ platform for processing clustered data based on multiple Gigabit Ethernet ports for phenoPET

Janani Subraveti

Jül-4406

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
Forschungszentrum

Zentralinstitut für Engineering, Elektronik und Analytik (ZEA) · Systeme der Elektronik (ZEA-2)

Implementation of UDP communication on a ZYNQ platform for processing clustered data based on multiple Gigabit Ethernet ports for phenoPET

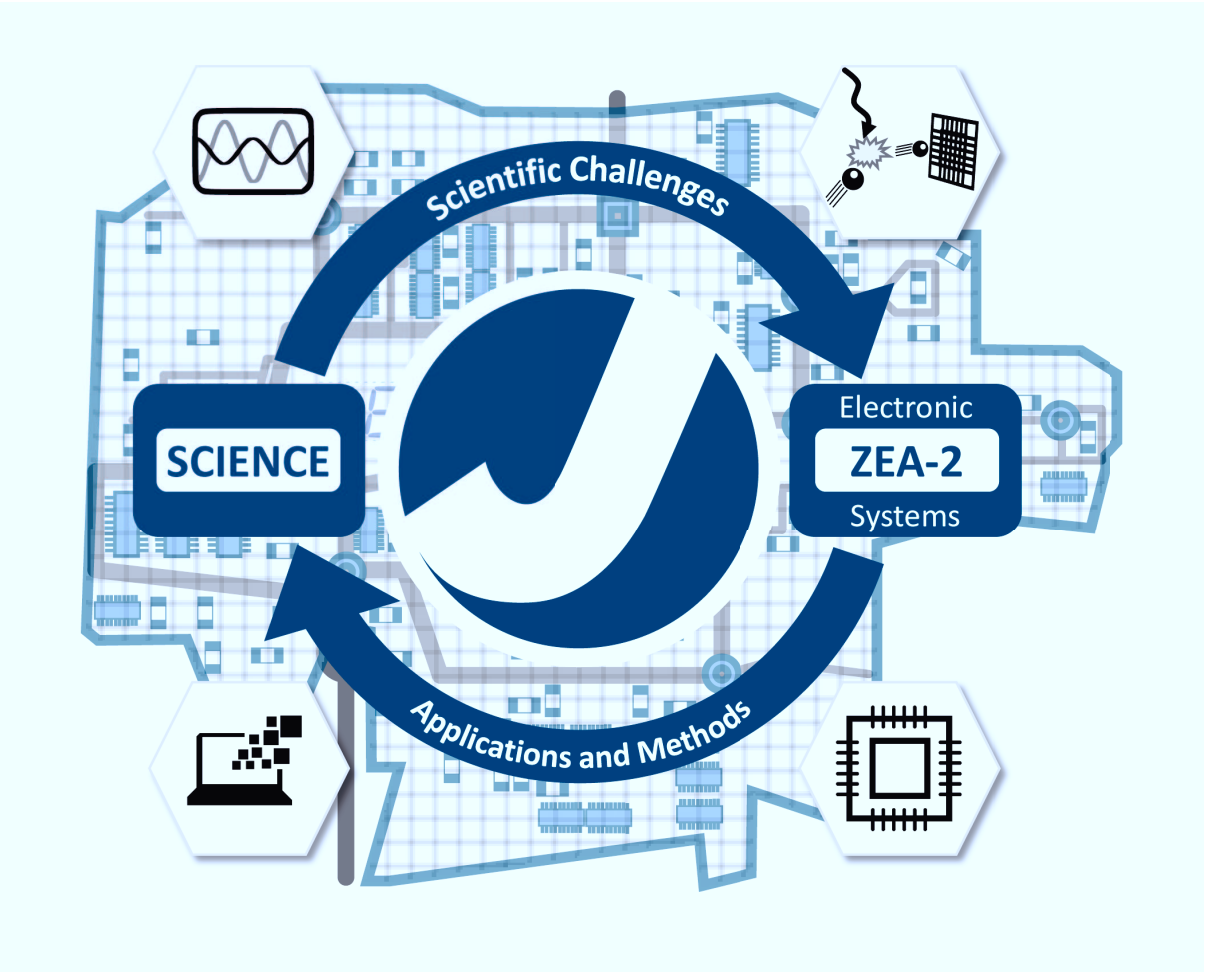
Janani Subraveti

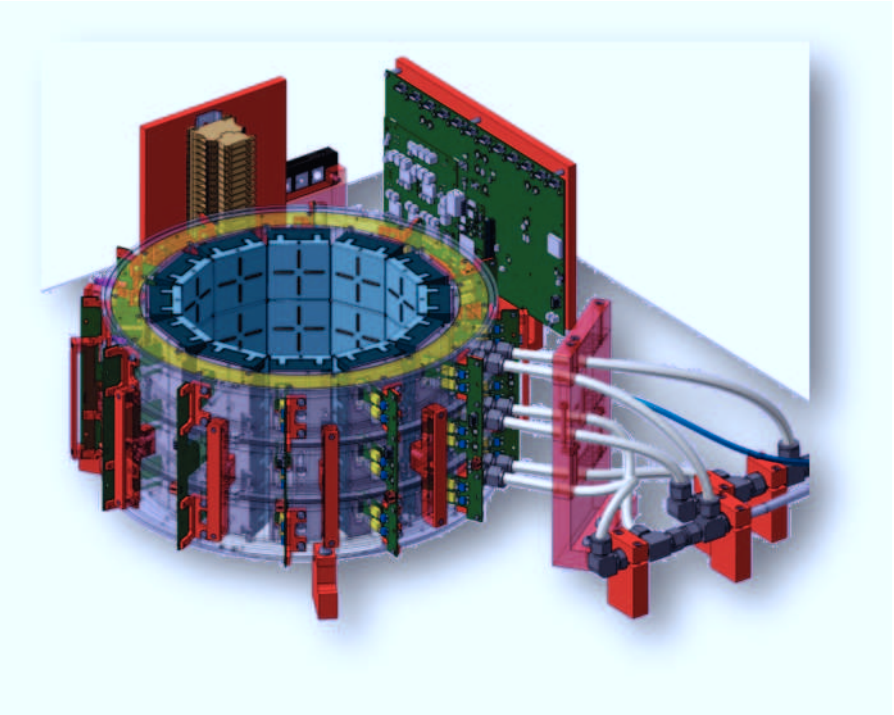
Berichte des Forschungszentrums Jülich
Jül-4406 · ISSN 0944-2952
Zentralinstitut für Engineering, Elektronik und
Analytik (ZEA) · Systeme der Elektronik (ZEA-2)

DE 46a (Master, Hochsch. Bremerhaven, 2017)

Vollständig frei verfügbar über das Publikations-
portal des Forschungszentrums Jülich (JuSER)
unter www.fz-juelich.de/zb/openaccess

Forschungszentrum Jülich GmbH · 52425 Jülich
Zentralbibliothek, Verlag
Tel.: 02461 61-5220 · Fax: 02461 61-6103
zb-publikation@fz-juelich.de
www.fz-juelich.de/zb


Scientific Challenges
SCIENCE
Electronic
ZEA-2
Systems
Applications and Methods



Abstract

As part of biological research carried out on plant phenotyping within the Jülich Plant Phenotyping Centre, a modality to detect the positron emitting radionuclides has been setup. The investigation of transport of short-lived carbon isotope ^{11}C within plants using $^{11}CO_2$ as radiotracer fixed during the photosynthesis dark reactions is the idea behind this research. The Flow and distribution of ^{11}C-labelled photo assimilates within a plant can then be imaged using the PET (Positron Emission Tomography) technology. To this end, a PET imaging system has been developed. This consists of scintillation detectors with scintillation crystals coupled to photodetectors. The radiation, which is emitted after the uptake of the radiotracer, causes light pulses within the scintillation crystals. This light is then converted into electrical signals by the photodetector. The "phenoPET" system is a PET scanner dedicated for plant research that employs digital SiPMs (Silicon Photo Multipliers) as photodetectors organised in 36 detector modules resulting in hit events based on the triggered photon counts fitted in data frames by a central FPGA based unit.

Present study starts with developing a prototype that uses Ethernet FMC module (from Opsero Electronic Design) with four Gigabit Ethernet ports. Concerning illustration based on the pre framework design of data transfer from detector modules, data stream flows from each detector module (consisting of 4 tiles) to the FPGA board (Xilinx Kintex-7 FPGA Mini Module Plus (Avnet)) on LVDS lines. From the FPGA board to the readout computer, USB 3.0 (at 300 MB/s (2.4 Gbps)) is used. For the connection from the readout computer to the storage system (located at air-conditioned place), 10 Gigabit Ethernet is used. Besides, our design is an add-on to the module FPGA, data stream from module FPGA is sent to ZC706 evaluation board (Xilinx Zynq-7000 All Programmable SoC) when the Ethernet FMC module is mounted on FMC (FPGA Mezzanine Card) connector of the ZC706 board. The data is received by four ports over the UDP server application running on the Zynq Processing System. Data reduction technique like clustering on timestamps (when multiple data packets occur in an event of hits) is performed in a time window between 1 - 5 ns. Processed data is sent out from one of the 10 Gigabit Ethernet ports on ZC706 after frame skipping technique being performed on every fifth frame. This study provides a measurement of Ethernet bandwidth utilization versus actual bandwidth from the stress tests performed on the datagrams. It provides information about the utilization of multi processors when the UDP application is running.

Acknowledgements

I would like to express my gratitude to Dipl.-Ing. Mario Schlösser for giving me this opportunity to work with Forschungszentrum Jülich GmbH and for his immense support till the end.

I would like to take the opportunity to thank Prof.Dr.-Ing. Kai Müller from the department of Embedded Systems Design at Hochschule Bremerhaven for kindly accepting the proposal of being my supervisor for my Master's thesis in the Central Institute for Electronics and Analytics (ZEA-2), Forschungszentrum Jülich GmbH, Jülich. I am hugely indebted to my professor for finding out time to reply to my emails.

I am grateful to my supervisor M.Eng. Sebastian Völkel for not only giving the opportunity to accomplish my thesis work in his workgroup but also for all the valuable guidance as being my supervisor at the institute.

I thank Dr. Matthias Streun for his valuable discussions on phenoPET project.

I thank all the people who helped me and supported me throughout the journey of my research study at ZEA-2, Forschungszentrum Jülich GmbH, Jülich.

Special thanks to IT team for all the immediate help in software and equipment support during the tough times in the research study.

I hold immense pleasure to thank my parents for all the faith and my close friend for her moral support throughout my journey till now. Last but not least, I thank my cousin for correcting English grammar and spellings in the thesis.

Table of Contents

List of figures

1. Introduction

To encourage the development of non-invasive technologies to be applied for plant and breeding, the DPPN (German Plant Phenotyping Network) in collaboration with the Forschungszentrum Jülich, the Leibniz Institute of Plant Genetics and Crop Plant Research (IPK) in Gatersleben and Helmholtz Zentrum München (HMGU) is carrying out research. Scanner technology using PET, Positron Emission Tomography technique is such a non-invasive technology. Improvement in plant and agricultural research is important for understanding and solving some difficulties of the future. Keeping in mind, the end goal to ensure the adequate sustenance supply for the world's fast-growing population, there is a need for higher crop yields and plants which are optimized for higher resistance to environmental stresses [1].

With a specific goal to confront these difficulties, the phenotype of a plant (i.e. the plant we really notice or observe – in contrast to the genotype, which is the heritable toolbox of the plant) is the consequence of the interaction between the genetic (hereditary) potential and the effect of environmental factors to which it is and was exposed. DPPN wants to gather quantitative data on the structural and physiological properties of plants and to apply this for primary plant research and breeding. For this reason, new and better techniques to decide, for instance, the size and shape of plants, their resistance to stress or the concentration of important metabolites are being developed [1].

Figure 1.1 Root growth is monitored in so called “Rhizotrones” in which large number of plants can be screened automatically [1].

DPPN had setup frameworks to investigate plants under defined environmental conditions in the laboratory and in the field. For example,

- Root structures will be investigated by magnetic resonance imaging.
- Development rates and photosynthesis intensity and/or efficiency of plants are analysed in automated ways and with robots.
- Water relations of crops are also determined – all for the general aim "to make plants better".

1.1 Methods and Technologies at the Jülich Plant Phenotyping Centre

PET, Positron Emission Tomography is a scanner technology that can be implemented into phenotyping methods which truly provides insight into plant organs. Non-invasive analysis of subterranean structures and functions is possible with this and it has created a basis for in-depth phenotyping. This technique allows to detect positron emitting radionuclides in an object. For this purpose, ^{11}C, positron emitting radionuclide is induced into the plant in the form of $^{11}CO_2$ as radio tracer. During the photosynthetic dark reactions, $^{11}CO_2$ is fixed and the products formed during these reactions are called as photo assimilates. These products formed are labelled with ^{11}C radio isotope and are transported along the plant.

The PET technology can be used to non-invasively identify the flow and distribution of these ^{11}C-labelled photo assimilates by detecting the gamma radiations emitted by positron-emitting radioisotope. An advantage of PET technique can be found from the following example. The photo assimilates formed (or the food synthesised by leaves during these photosynthesis reactions) are moved to all parts including the root tips embedded deep inside the soil. Transport of the carbon over longer distances proceeds through the conducting or vascular tissues (the xylem and the phloem) inside the plants. The cells in these conducting or vascular tissues are under high stress and sensitive to manipulate. Therefore, the transport inside them is hard to be accessed with invasive techniques. The PET technology, a non-invasive form offers interesting chance to detect spatial and temporal distribution and 3D mapping of these radio traces in a plant. Two plant committed PET frameworks have been in operation at Institute of Bio and Geosciences (IBG-2, Plant Sciences) in Jülich Forschungszentrum, they are **PlanTIS** and **phenoPET**.

1.2 PlanTIS

PlanTIS (Plant Tomographic Imaging System) is a custom-built instrument based on Clear PET technology. It is suited for small samples with a maximum field-of-view (FOV) of around 65mm in diameter and a height of 100mm that has been built in 2006 and in operation since 2008 [2].

Figure 1.2 PlanTIS [2]

1.3 phenoPET

phenoPET is also a custom built novel instrument that includes last generation digital photon counter (DPC) technology. The system will provide a field-of-view (FOV) of around 180mm in diameter and 190mm in height and will have much higher detection sensitivity than PlanTIS.

Therefore, phenoPET will contribute to the many plant phenotyping approaches at IBG-2 but only for low throughput applications. The development of this system is a cooperation between three institutes at Forschungszentrum Juelich (ZEA-2: Electronic Systems, ZEA-1: Engineering and Technology and IBG-2: Plant Sciences) and Philips Digital Photon Counting, Aachen, Germany and supported by the Bundesministerium für Bildung und Forschung (BMBF) within the German-Plant-Phenotyping Network (DPPN) [3].

1.4 Project description of pre-framework of phenoPET setup

The development of PET detector for plant phenotyping by Forschungszentrum Jülich together with Philips Digital Photon Counting, Aachen has the scientific goal to study the carbon transport in plants. As discussed in 1.1, the PET technology can be used to non-invasively identify the flow and distribution of these ^{11}C-labelled photo assimilates by detecting the gamma radiations emitted by positron-emitting radioisotope. Gamma rays are emitted when electron-positron annihilation (collision) takes place in the decay of certain radioactive nuclei. This results in creation of gamma ray photon. A ring of digital photon counters to detect the photon pairs recently developed by Philips is being used. For the prototype, for data concentration and for the processing of the coincidences, a Xilinx Kintex evaluation board was used. It is assumed that the necessary data rate from the FPGA to the acquisition computer is about 300 MByte/s. As data link a 10 Gigabit Ethernet link would be preferred, but the evaluation board contains a USB 3.0 interface already, therefore it has been chosen to use in order to reduce the development costs. (See Figure *1.4*) Design overview of pre-framework of phenoPET setup.

Figure 1.3 Pre-framework of phenoPET

The PET framework in its last setup will comprise of three rings of photon detectors. The first version of the PET will have just a single ring, other rings will be added later. Twelve detector modules are present in each ring and each detector is connected to central FPGA board with standard HDMI cables. The data transfer from the detector module to the FPGA board

uses LVDS, Low-voltage differential signalling. From FPGA board to the readout computer, USB 3.0 is used. For the connection from the readout computer to the storage system, (this is not shown in the Figure *1.4*) 10 Gigabit Ethernet will be used [4].

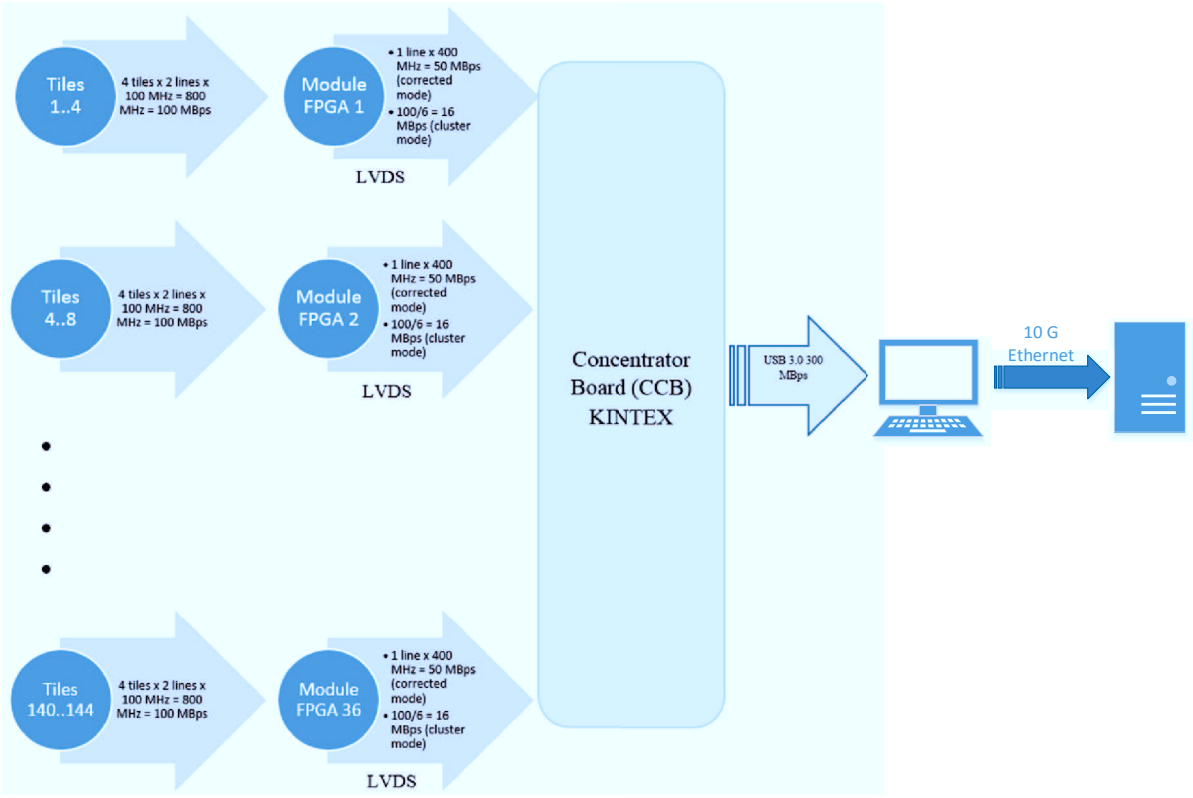


Figure 1.4 Design flow for three rings in the pre framework

As of now, the data acquisition and first pre-processing are done on the FPGAs with in the Detector Module. A central Concentrator Base Board was developed that consists of a Kintex-7 FPGA Mini-Module (Xilinx), 12 Sector Boards connect 3 Modules (one from each ring, 3 modules from 3 rings) and send the data through standard HDMI cables to the Concentrator Base Board. One LVDS pair per Module is used allowing 50 MB/s data rate when transferring data to Concentrator Base Board. Finally, a USB 3.0 connection sends the data to the computer system for storage and reconstruction.

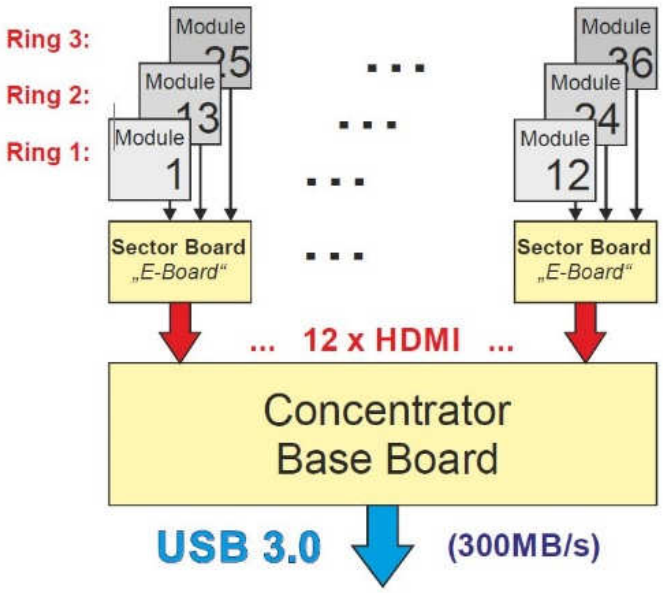


Figure 1.5 Design Architecture in the pre framework [5]

Photon detectors and clustering concept

11C, a short living isotope is chosen for the research study. It needs a scanner with high dynamic range as fast timing and high data rates are important so the Digital Silicon Photomultiplier DCP 3200-22-44 developed by Philips Digital Photon Counting, Aachen were chosen. There are 3200 cells (diodes) present in one pixel of the detector. This is a pure digital device as the number of broken-down cells are counted. One detector module comprises of 4 tiles, each tile has 4*4 dies (See Figure 1.8) and each die has 2*2 pixels. A timestamp (44 ps resolution) and the number of broken-down cells for each pixel are stored for an event of hit. When an event of photon hit occurs, all four pixels of the die are always triggered together. A total of 9216 pixels (3×12×4×16×4) are present in the complete PET with three rings [4].

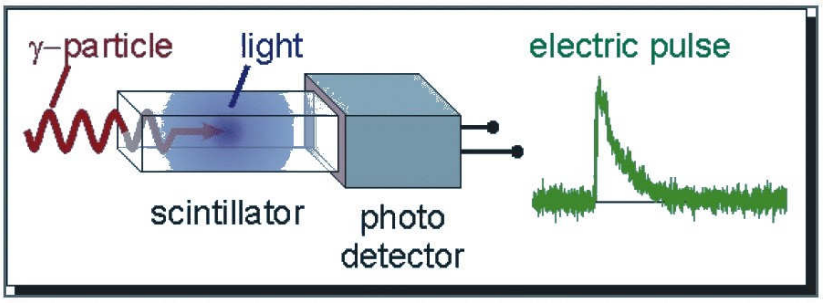


Figure 1.6 Scintillation Detector and photoelectric concept *[6]*

Figure 1.7 Detector module with and without cap [5]

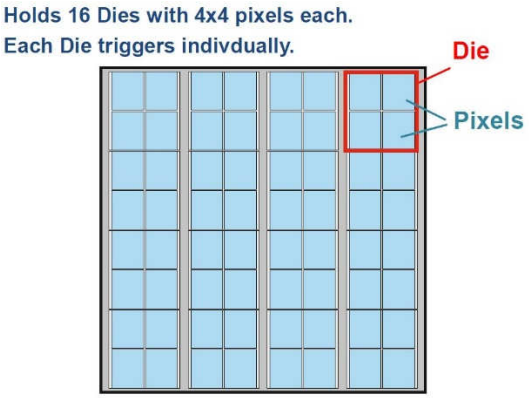


Figure 1.8 Detector Tile *[5]*

Figure 1.9 Scintillator Matrix *[5]*

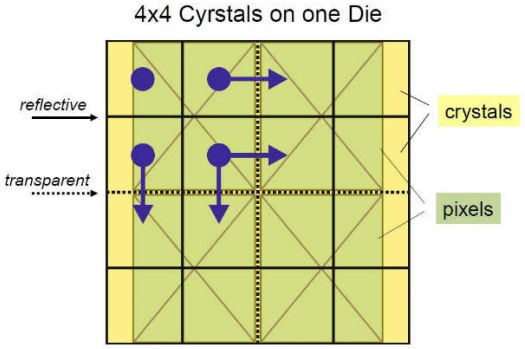


Figure 1.10 Light sharing on Die Pixels *[5]*

Each detector module accommodates four 8×8 pixel Digital Photon Counter devices DPC-3200-22-44 connected to a PCB. A scintillator matrix with 16×16 individual LYSO scintillators is attached to each of this Digital Photon Counter device. (See Figure 1.89) Crystal size is 1.85×1.85×10 mm3. The matrices (Crystal Photonics) are composed with both reflective and transparent contact faces between the crystals in order to optimize crystal identification.

The typical detector for detecting the gammas in a PET scanner is the scintillation detector. The scintillator converts the gamma into a light flash which is detected by a photodetector.

Figure 1.11 A cause for clustering scenario

Ideally, the scintillator idea with the special crystal matrix should keep the light of a single crystal on one Die only (as shown in *Figure 1.11*) so that for each event of hit only one Die should trigger but because of the cross talk often more than one i.e., two or even more Dies trigger on the same event. This can be recognized by identifying the Dies that "group together" i.e. in other words which means "cluster together". This scenario implies that, the timestamps of those events are close and can be in a clustering window of dt_{Clus}. All Dies within this clustering window form a cluster and are expected to belong to the same event.

FPGA Board

The Xilinx Kintex-7 FPGA Mini-Module Plus (Avnet) as central processing unit for data concentration and coincidence logic is chosen because of the limited development time

Its features:

• Enough LVDS-IOs for 12 detector boards
• 256 Mbyte SDRAM
• No 10 Gb-Ethernet
• Instead USB 3.0, fast enough for the expected data rate (300 Mbyte/s).

The boards can be cascaded in a tree structure if more inputs become necessary. 10 Gigabit-Ethernet was the preferred connection with the readout computer, but the Kintex board has only 1Gigabit-Ethernet which is not fast enough. Instead it has a USB 3.0 interface which can be easily used. Figure 1.12 shows the FPGA board with the adapter board for the HDMI cables. (The PCI e connector is not used.). The USB connection of the Kintex board is realised by a Cypress CYUSB3014 chip. This chip is in fact an ARM926EJ processor (32 bit, 200 MHz,

512 k Byte SRAM). The GPIOs are used for FIFO interface with a width of up to 32 bit and running at 100 MHz. This FIFO interface is connected to a DMA engine inside the FPGA which in turn uses the AXI framework to read data from or write data to the DDR3 memory [4].

A Micro Blaze Processor embedded in the FPGA is used for control and diagnostics. Because of the limited length of a USB-3 cable the readout computer should be located near the PET, but the storage system will be connected via 10 Gigabit Ethernet and be located in an air-conditioned place [4].

Figure 1.12 Concentrator Board with Kintex evaluation board and cable adapter *[5]*

2. Motivation and Purpose

This thesis work has started with an aim to co-assist the phenoPET project. As discussed in *Figure 1.4*, in the pre-design framework for data acquisition of phenoPET data, the data transfer from each detector module to the module FPGA and then to the Concentrator Base Board with Kintex FPGA in it uses LVDS, Low-voltage differential signalling. Next from FPGA board to the readout computer USB 3.0 is used.

The data rates at each stage in the pre-design framework are as follows. In the first stage, the event data are transferred via 2 LVDS lines running at 100 MHz to the Module FPGA, giving a bandwidth of 200 Mbit/sec or 25 Mbyte/s.

$$25\,\frac{Mbytes}{s} * 327\ \mu s(= frame\ time) = \sim 8\ Kbytes \quad \textit{Tile frame data.} \tag{1}$$

The data from each tile is transferred on frame base giving 8 Kbytes tile frame data.

$$8 \times 4 = 32\ Kbytes \quad \textit{All tiles frame data.} \tag{2}$$

$$25 \times 4 = 100\,\frac{Mbytes}{s} \quad \textit{Data rate for processing total data stream from 4-tiles.}$$

For 4 tiles (one module), a total of 32 Kbytes, all tiles frame data requires a data transfer rate of 100 Mbyte/s to the Module FPGA.

Next, this data from each module (Module FPGA data) is transferred to the Concentrator Base Board via single LVDS line running at 400 MHz, giving a bandwidth of 50 Mbytes/s.

$$50\,\frac{Mbytes}{s} \times 327\mu s(= frame\ time) = \sim 16\ Kbytes \text{ Module frame data.} \tag{3}$$

$$50 \times 36 = 1800\,\frac{Mbytes}{s} \textit{ Data rate for processing total data stream from 36 modules.} \tag{4}$$

All data is transferred on frame base giving 16 Kbytes Module frame data. For 36 modules, a data transaction rate of 1800 Mbytes/s versus USB3.0 of 300 Mbytes/s has been used.

$$\frac{16\times36}{4} = \frac{576}{4} = 150\,\frac{Mbytes}{s} \textit{ Data rate for processing clustered data stream from 36 modules.} \tag{5}$$

If the data is clustered, in order to process the clustered data out considering a contrast factor of 4, 150 Mbytes/s versus USB3.0 of 300 Mbytes/s has been used. All these data rates and design flow can be seen in *Figure 1.4*.

The purpose of the current thesis is to provide higher data rates or bandwidths during data acquisition and processing. Besides Concentrator Base Board that receives data on LVDS lines, ZC706 evaluation board with Ethernet FMC module on the FMC connector will receive the data from the module FPGAs over 1 Gigabit Ethernet port of Ethernet FMC resulting in a data rate 1000 Mbps or 125 Mbytes/s instead of 50 Mbytes/s (see lines above (3)). The approach to send the data out to the readout computer on 10 Gigabit Ethernet port giving a bandwidth of 1250 Mbytes/s which is close to 1800 Mbytes/s (see lines below (4)) instead of using USB 3.0 with a bandwidth of 300 Mbytes/s is much faster. This ratio gets even better with preprocessed clustered data, which would result in 150 Mbytes/s being send with 1250 Mbytes/s instead of 300 Mbytes/s.

3. Requirement and System Analysis

Based on the motivation and purpose discussed in previous chapter, following requirements are chosen and an analysis is drawn at system level for higher data rates. These Requirements are categorized into hardware and software which are going to be discussed in the following chapter. This analysis is performed at system level in both hardware and software point of views. *Table 1* presents the requirements and respective hardware interfaces chosen.

Requirement	Hardware interfaces chosen
4 Gigabit per second data rate for data acquisition	Ethernet FMC module with four 1 Gigabit Ethernet ports
10 Gigabit per second data rate for sending data out	Board with 10 Gigabit Ethernet support (SFP module)
A board with FMC connectors, FPGA and Processor for data acquisition and processing	Xilinx Evaluation Board with Zynq Processing System (ZC706 and ZCU102 are chosen)
Designing of hardware setup and development of software application running on Procesor to handle the data.	Xilinx Vivado Design Suite and PetaLinux tools

Table 1 Requirement specification and hardware chosen

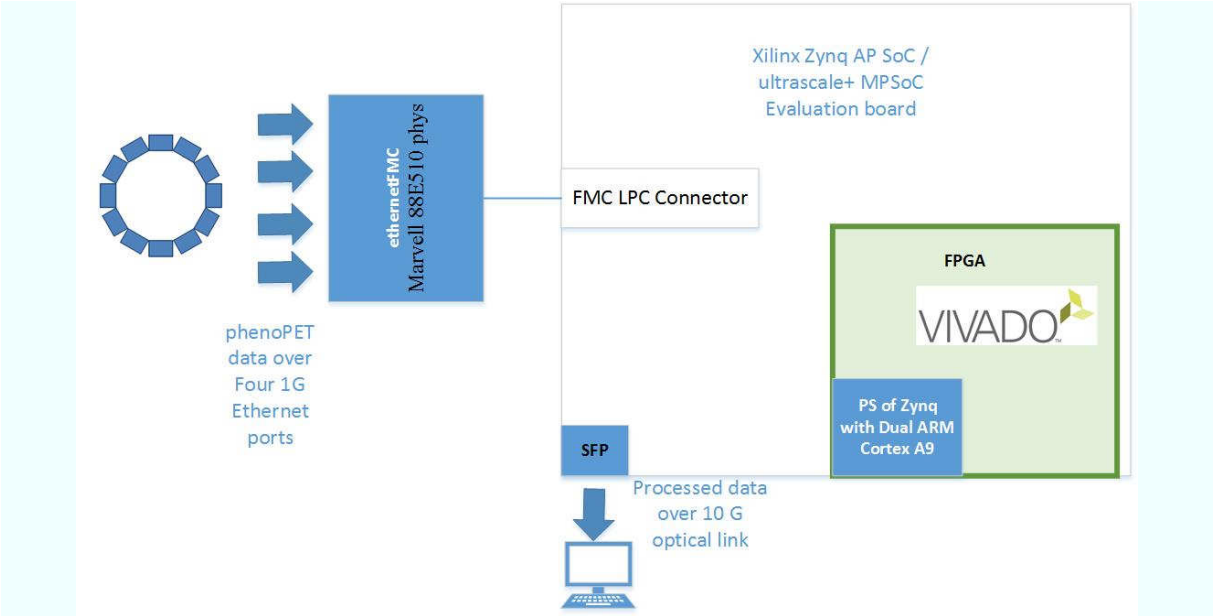


Figure 3.1 System Analysis

3.1 Hardware requirements

Ethernet FMC

Opsero Electronic Design Inc. a design consultancy specialized in FPGA technology has designed a product called Ethernet FMC. The main component in this project is Ethernet FMC module. This is a quad Gigabit Ethernet mezzanine card for FPGA as shown in the Figure 3.2. A mezzanine card is a smaller form of the more familiar Peripheral Component Interface (PCI) or Industry Standard Architecture (ISA) card.

Figure 3.2 Ethernet FMC module or adapter for FPGA

If Ethernet FMC is being used, the Ethernet MAC is inside the FPGA and the PHY is the Marvell 88E1510. The Marvell 88E510 PHY is the default configuration of the Ethernet FMC with auto negotiation for 10 Mbps, 100 Mbps and 1 Gbps capabilities and also the MDI automatic crossover features are supported.

ZedBoard

ZedBoard is an affordable evaluation and development board for the Xilinx Zynq 7000 all programmable SoC (APSoC). ZedBoard contains Xilinx XC7Z020-1CSG484CES Extensible Processing Platform (EPP). It has a dual Cortex-A9 Processing System (PS) with 85,000 Series-7 Programmable Logic (PL) cells. It is based on the Xilinx Zynq-7000 Extensible Processing Platform which can be useful for broad use in many applications. Some of the features like 512 MB of DDR3 along with 256 MB of QSPI Flash, 10/100/1Gigabit Ethernet interface, one LPC (Low Pin Count) FMC (FPGA Mezzanine Card) interface and an SD card interface made us to have ZedBoard to be chosen development board in the current project [7]. But concerning the compulsion for one 10 Gigabit Ethernet interface to send the clustered data out and also for higher DDR3 specification for higher data processing to take place, Zynq-7000 All Programmable SoC ZC706 has been chosen. But in the beginning stages of the project, the data acquisition has been performed on ZedBoard on the four Ethernet ports of Ethernet FMC connected to the LPC FMC interface.

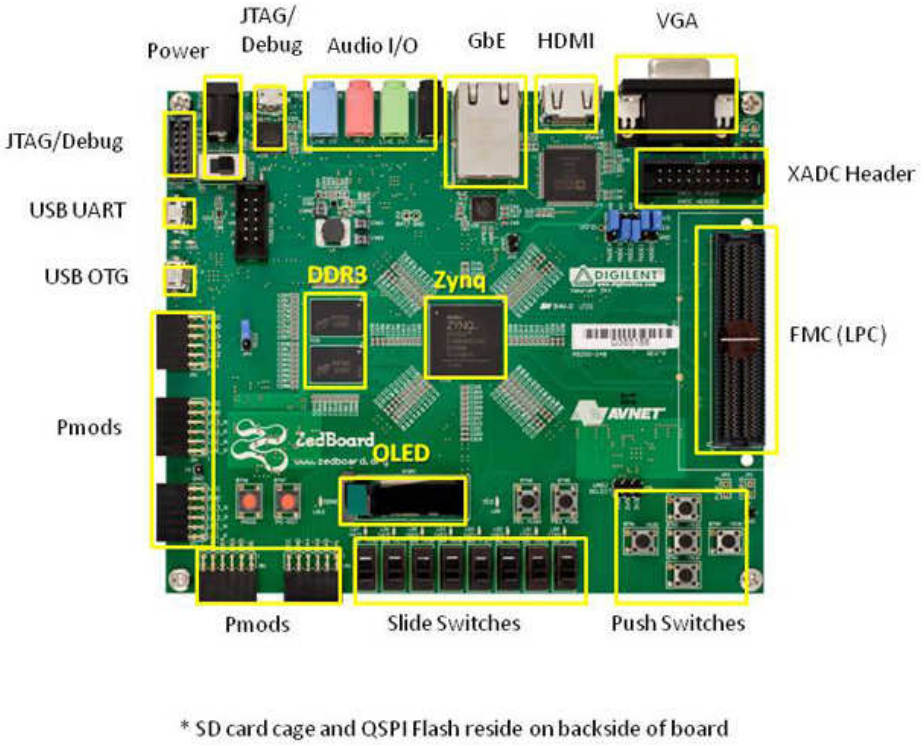


Figure 3.3 Functional overlay of ZedBoard *[8]*

ZC706

For developing and evaluating designs targeting the Zynq-7000 XC7Z045-2FFG900C AP SoC, the ZC706 evaluation board is used. The XC7Z045 AP SoC has an integrated processing system (PS) and programmable logic (PL) on a single piece as shown in *Figure 3.6*.

Figure 3.4 Functional overlay of ZC706 evaluation board *[9]*

Various interfaces for System on Chip with Zynq PS and Kintex PL present on ZC706 evaluation board can be seen in the *Figure 3.5*.

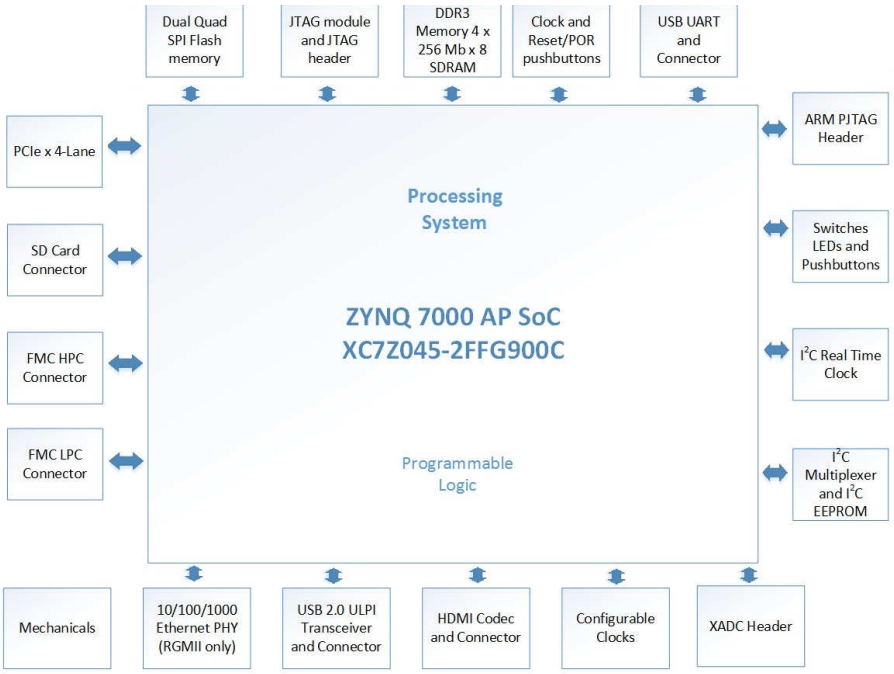


Figure 3.5 ZC706 Evaluation board block diagram *[9]*

A high-level block diagram connecting Programmable Logic (PL) and Processing System (PS) consisting of memory interfaces, Application Processor Unit and IO peripherals on AXI interface is shown in *Figure 3.6*.

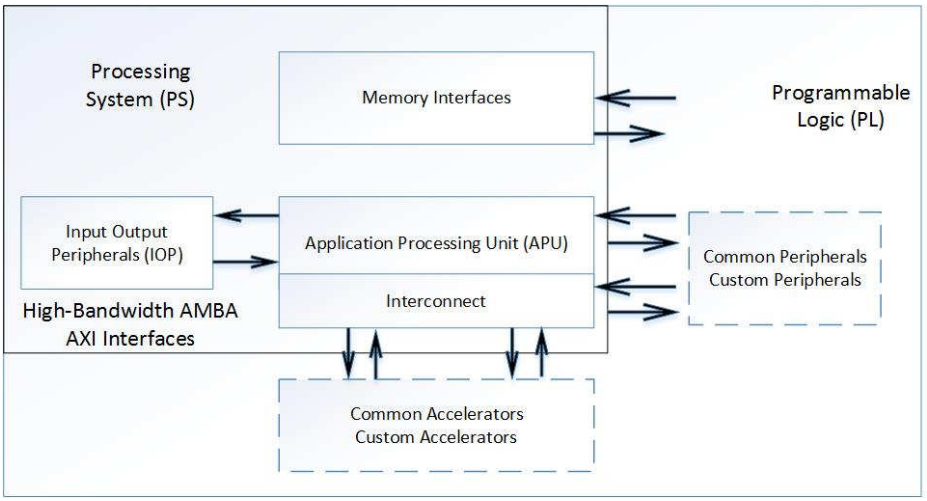


Figure 3.6 High Level Block Diagram *[9]*

Two ARM Cortex – A9 MP core application processors, AMBA interconnect, internal memories, external memory interfaces and peripherals like USB, Ethernet, SPI, SD/SDIO, I2C, CAN, UART and GPIO are integrated inside the Processing System. The PS runs independently of the PL and boots at power-up or reset. A system level block diagram with required components in PS and PL of Zynq 7000 AP SoC is shown in *Figure 3.7* [9].

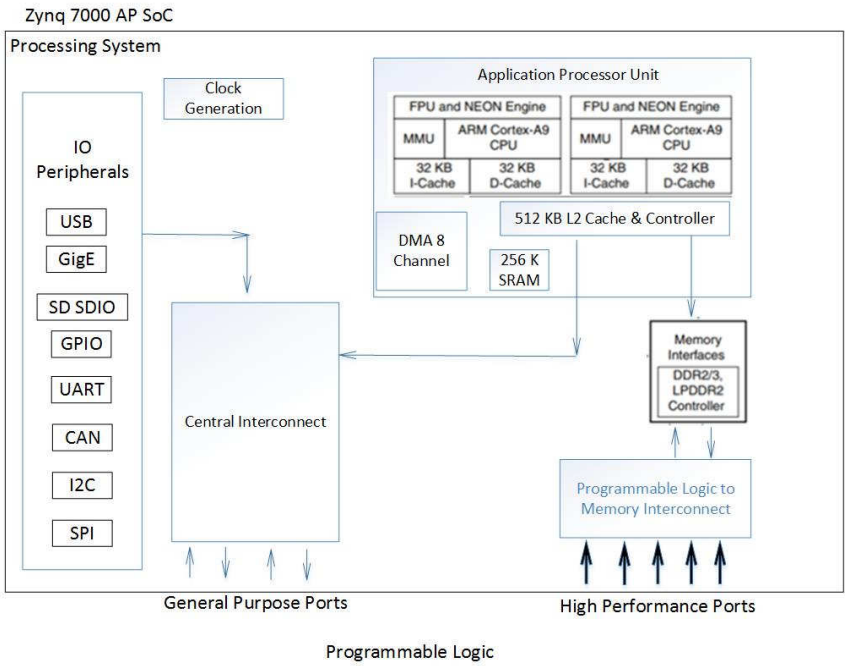


Figure 3.7 Zynq 7000 Block Diagram *[9]*

The actual hardware set up for data acquisition and processing with Ethernet FMC is shown in *Figure 3.8*.

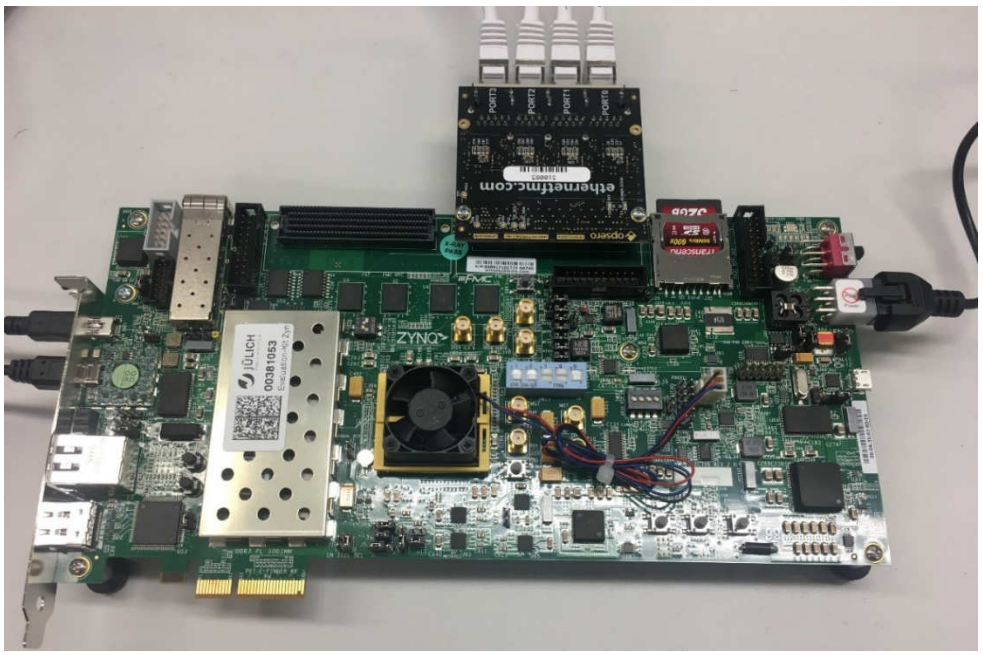


Figure 3.8 Hardware Setup on ZC706 Board in the project

Software to be used

- Xilinx Vivado Design Suite for generating hardware description file of the system under design.
- Xilinx PetaLinux tools for configuring the embedded Linux platform on the ZC706 and implementing the above tasks on the hardware platform.

4. Validation Strategy

Since the project has its purpose for data acquisition and processing at higher data rates, it can be validated in following ways before proceeding to implementation for UDP and clustering.

4.1 Validation Strategy for Data Acquisition

PhenoPET detector has tile frame data of size 8 Kbytes and a total of 32 Kbytes of data from each detector module is transmitted at 1 Gigabit/s rate to the Processing System of Zynq on UDP. A fastest and most simple way for transmitting data to the receiver is possible by UDP. There is no chance for interference in the stream of data under transmission. This prompts an approach for an application to get as close in order to meet the real-time constraints as possible. The four ports receive data from the module FPGA. Since UDP is connectionless, therefore the current system can be validated on UDP data packages of four modules on four Ethernet ports in real time to validate data processing on FPGA as shown in the *Figure 4.2*.

Figure 4.1 Networking framework based on simplified OSI for the DAQ

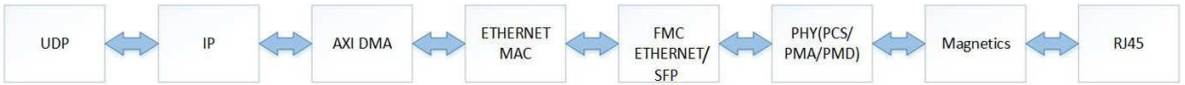


Figure 4.2 Flow of data from physical to transport layer in current project

In digital communication, a message that is sent from one end point to another without any preliminary arrangement is described as “connectionless”. Without first guaranteeing that the recipient at another end is accessible or ready to receive the data, the device at one end of the communication transmits data to another end. Data is to be resent several times if there are

problems with the transmission because the device at one end sending the data simply sends data addressed to the intended recipient. The User Datagram Protocol (UDP) and Internet Protocol (IP) are two such connectionless protocols.

Delivery of all packets is not assured so the connectionless service is an unreliable one. The connectionless service is also called a “best-effort service”, though all attempts to deliver a packet will be made, the unreliability is may be due to hardware faults or exhausted resources and packets may be quietly dropped, duplicated or delayed and may arrive in an improper order.

TCP provides a reliable connection and majority of current Internet applications utilize this protocol. In addition to the features like error checking and correcting, TCP is also in charge of controlling the speed at which the data is sent. One of the qualities of TCP is congestion control in the network where transmission speed will be backed off when congestion occurs. Thus, network is safeguarded from congestion collapse.

TCP would be a preferred protocol to be used because of its features like congestion control. Because of the fact that TCP is a reliable service, delays are introduced at whatever point a bit error or packet loss takes place. Delay is caused because of the retransmission of the broken packet, along with any successive packets that are already been sent can be a large source of jitter. Combined, jitter is raised to an unacceptable level rendering TCP unusable for real-time services. Our current project has the advantage of not requiring a completely reliable transport level. A click or a minor break would only be introduced into the output due to loss of a packet or there is not much effect while further processing the packets.

For these reasons, UDP is being used for the data transmission in our project. UDP is a thin layer on top of IP that inherits all of the properties of IP that TCP attempts to hide. UDP is therefore also a packet based, connectionless, best-effort service. Providing any necessary error checking is up to the application by splitting data into packets.

Since UDP is chosen way for data acquisition of phenoPET detector data on the Processing System of Zynq, iPerf3 can be used to validate the data acquisition over UDP at different bandwidths by running iPerf3 server on the Zynq PS and iPerf3 client that send user datagrams to the server in specific intervals of time.

4.2 Validation Strategy for Data Processing

Our current project aims for the clustering technique to be performed while processing the data.

As discussed Seite 4, when an event of photon hit takes place, only one die has to be triggered for the light of a single crystal but more than one i.e., two or even more Dies trigger on the same hit event. Then the time stamps of those events are close and are in a window which are expected to belong to the same event. So, Clustering technique aggregates all the data that (presumably) belongs to an event. That is, that the time stamps are controlled. As long as the difference in the time stamps of successive data from different dies is less than e.g. 5 ns, they are assumed to originate from the same event. The pixel values of all these then, together with the earliest time mark, form the data packet for this event.

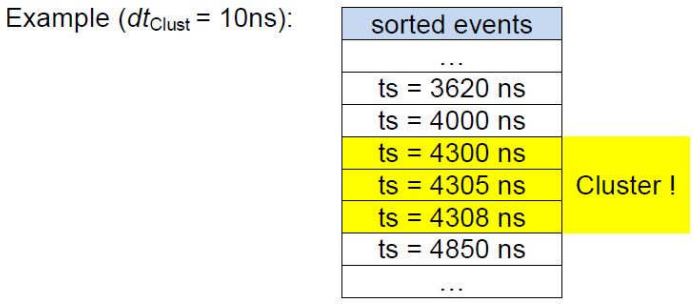


Figure 4.3 Example for Cluster

Clustering is the first thing to do with the data processing in the FPGA. In doing so, all relevant pixels together with the first time mark are tied together into a data packet.

The clustered data can be validated from MATLAB runs performed over the phenoPET detector data frames. These results can be compared to the results obtained when the actual algorithm runs on the Zynq PS. (Compare results in clustered data output results in 11.3 and 11.4 in Appendix)

5. System Concept

Based on the requirement and system analysis discussed in chapter 3, in the current project, the target evaluation board, ZC706 has been used and the Ethernet FMC module has been mounted on the LPC FMC of the board for data acquisition and data processing. This board receives data at a speed of 1 Gigabit/s or 125 MBytes/s on each of the four Ethernet ports. Data is received on the UDP application running on the processing system. Algorithm to identify the clustered events based on the closeness of timestamps in a cluster window is performed on the data acquired on the Zynq Processing System (C programming). And the clustered data is sent out to the readout computer on one of the Ethernet ports at a speed of 10 Gigabit/s or 1250 MBytes/s.

Tasks overview

- *1st step*: To configure the ZC706 evaluation board for network interfaces for the four Ethernet ports of Ethernet FMC module.
- *2nd step*: Boot Linux on the processing system of the board and communicate to the four ports from a host PC on the network interfaces.
- *3rd step*: Send data in UDP packets to the four ports when UDP application program is running on the processing system.
- *4th step*: Run the algorithm on the data from each port in order to identify clusters in cluster window based on time stamp after the data being extracted and send the reduced data out after performing the algorithm at 10 Gigabits/s on another Ethernet port.

A host PC acts like a medium only, to configure the board and setup the network interfaces. In real-time the data is directly sent from the module FPGA to the four Ethernet ports of the Ethernet FMC module. Stress tests are performed on all four Ethernet ports and percentage of packet loss at different bandwidths is calculated. Processor utilization for each task is also known.

Since currently developed prototype uses Ethernet FMC module (from Opsero electronic design) with four Ethernet ports. This Ethernet FMC module is mounted on ZC706 evaluation board via an FMC connector on the board. This design is added to the module FPGA in the previous prototype for data acquisition in phenoPET project as discussed before (under 1.4 and see Figure 1.4). The current design flow including the data rates is shown below as Figure *5.1*.

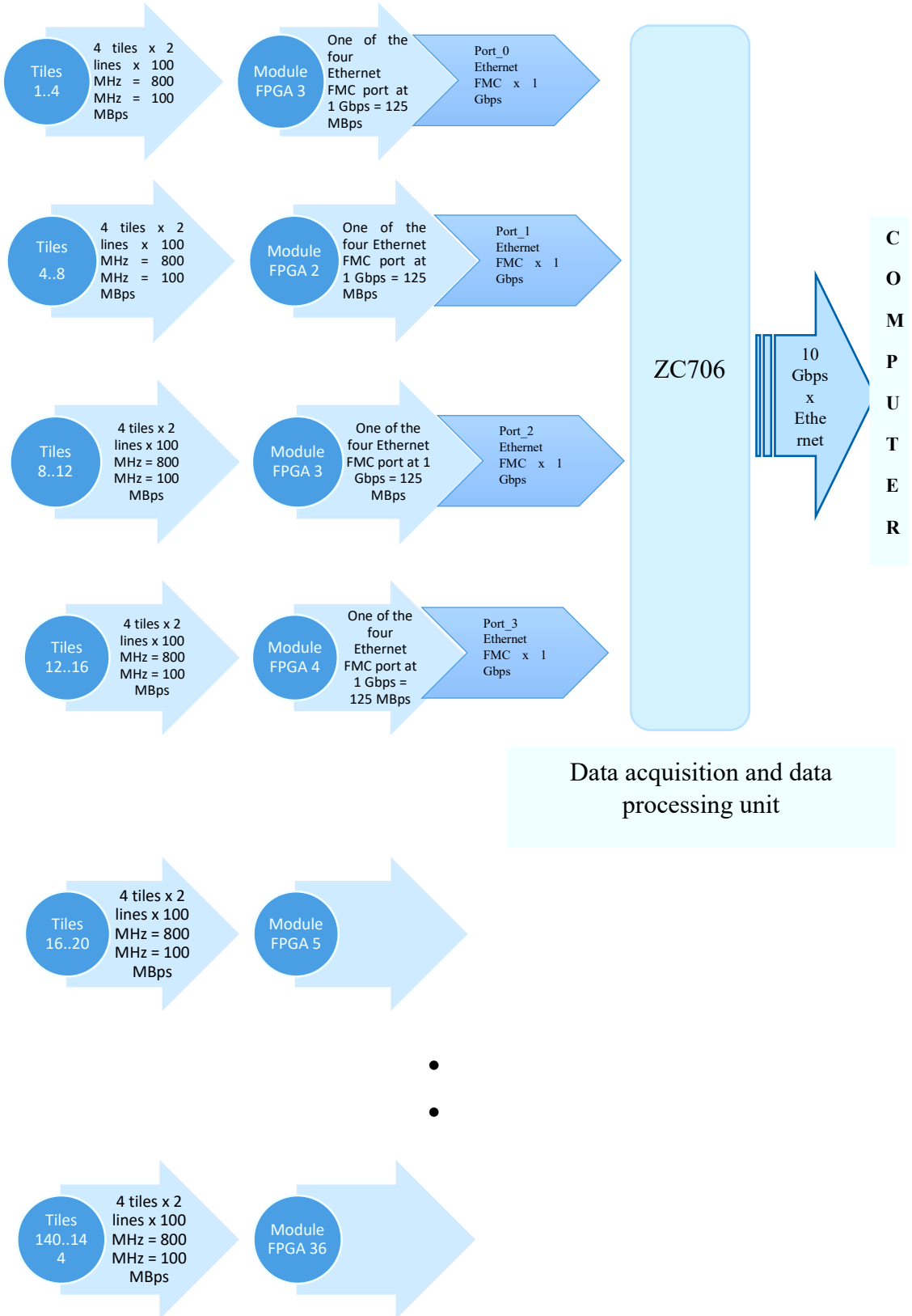


Figure 5.1 Design flow in current framework

The incoming data streams from four module FPGAs now go to UDP server application running on the Zynq processing system on the ZC706 evaluation board through the four ports of Ethernet FMC module that is mounted on FMC connector. Data reduction technique called clustering on timestamps (when multiple data packets occur in an event of hits) is performed in a time window of 5 ns. Processed data will be sent out on the 10 Gigabit Ethernet port on ZC706 after frame skipping technique being performed on every fifth frame.

An idea or the concept before proceeding for implementation on ZC706 board is shown on a Zynq AP SoC (here on ZedBoard) in Figure *5.2*. In implementation point of view, the physical layers implemented by Ethernet FMC are connected through RGMII interface to the MAC layer implemented by IP cores on the FPGA in Vivado design suite and Linux OS is booted that runs on the ARM processor that implements networking support is required.

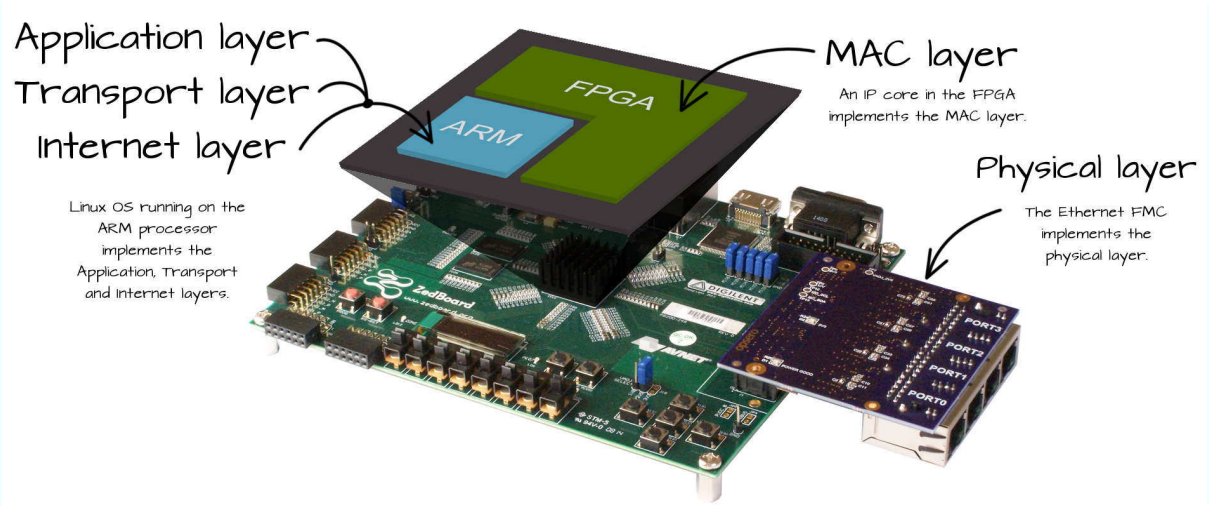


Figure 5.2 System concept at hardware level on an example set up with ZedBoard *[10]*

6. System Implementation

Complete system implementation can be split into two categories, first is design phase of hardware in Xilinx Vivado Design Suite and the second is software development using Xilinx PetaLinux tools.

6.1 Hardware Design in Xilinx Vivado Design Suite

The *Figure 6.1* is a pictorial illustration for the actual hardware design done in Xilinx Vivado Design Suite. It is portrayed below for an easy reference in this section as many of the concepts are to be explained here.

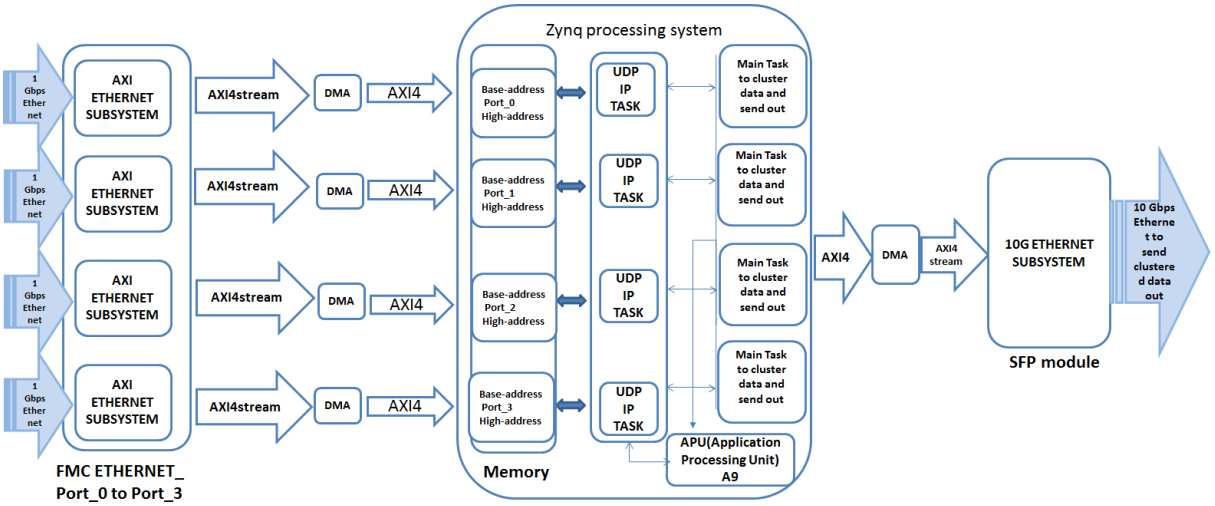


Figure 6.1 System Design Overview

The Vivado Integrated Design Environment (IDE) has an IP-centric design flow in which one can add IP modules to the design from different design sources. A GUI called IP Integrator helps in faster pin connection to IP enabled by an AXI based common user interface. This will greatly decrease the time and efforts for the design. There is also a feature called IP subsystem for multiple IP into one.

In the implementation point of view, the main aim of our project can be described based on *Figure 6.1*. There is IP cores depicting the actual hardware on the PL side and this hardware receives the detector module data of phenoPET when connected to the Ethernet. This data is copied over the AXI interfaces to the DRAM memory and the software we run here will process the data on the ARM host and send the processed or clustered data out on 10 Gigabit port depicted with another IP core on the PL side.

6.2 Implementation of PHYs of Ethernet FMC in Vivado

As shown in the *Figure 6.1*, the design implementation starts with connecting PHYs of Ethernet FMC and MACs of Ethernet Subsystem IP core. During implementation, there are some corrections or skews to be considered in the application data paths. The Ethernet PHY and the Ethernet MAC have a physical connection called RGMII interface between them. A dual data rate (DDR) interface called RGMII interface that consists of a transmit path from FPGA to PHY and a receive path from PHY to FPGA interfaces is present between them. Independent clocks, 4 data signals and a control signal are present in both the paths.

The data and clock are output simultaneously in the RGMII standard specification (i.e. without any skew on the clock), as shown in *Figure 6.2*.

Figure 6.2 Clock skew stages in RGMII interface *[11]*

But if the data signals are to be properly sampled at the receiver side, the delay has to be added to the clock signal, either by the PCB traces or by the receiver itself in the RGMII standard. The clock and data signals after delay in the clock has been added is shown in the *Figure 6.4*.

Figure 6.3 RGMII Interface without clock skew *[11]*

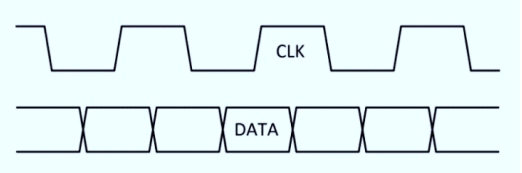


Figure 6.4 RGMII Interface with clock skew *[11]*

Required delay is added to the TX and RX clock signals in three stages in an FPGA based system as shown in *Figure 6.2* in FPGA or on the PCB traces (i.e. with clock traces longer than the data traces) or in the PHY. In an ideal RGMII interface, the delays at other two stages are disabled or not implemented and the delay is added at one stage in the TX and RX path.

The TX clock's delay and RX clock's delay may be independently dealt, on each path, the implementation may differ. The delay at each stage and different ways to enable or disable them are being discussed here.

Skew in FPGA

By default, the RGMII TX clock internal delay setting is enabled in the FPGA design of Vivado design. The RGMII TX clock is a delayed once if the default PHY configuration is being used then the RGMII TX clock is delayed twice i.e. in FPGA and in PHY that results in poor alignment in the sampled data. This is recognized at 1 Gbps, the internal delay at the PHY is 1.9 ns.

The RGMII TX clock and the RGMII TX data are in phase by changing it in the Vivado project. RGMII TX clock is delayed by 1.9 ns in the Marvell 88E1510 Ethernet PHYs design when an internal delay is selected. In FPGA, the RGMII TX clock and the RGMII TX data are output with the same phase, thus simplifying the design of RGMII interface.

```
set_property CLKOUT1_PHASE 0 [get_cells
design_1_i/axi_ethernet_0/U0/eth_mac/U0/tri_mode_ethernet_mac_supp
ort_clocking_i/mmcm_adv_inst]
```

Code snippet 1 Design constraints for adding skew

The RGMII TX clock delay is performed in the FPGA fabric by default for AXI Ethernet Subsystem. It uses an MMCM to produce a clock that is phase shifted by 90 degrees with respect to the "gtx_clk". Mixed Mode Clock Manager (MMCM) produces a clock that is phase shifted by 90 degrees with respect to the "gtx_clk". To remove the delay the simple way is to change the configuration of the MMCM to produce a phase shift of 0 rather than 90 using the above said command *Code snippet 1*.

If multiple AXI Ethernet Subsystem IPs are being used, only one of these IPs will have the shared logic (MMCM) and the phase shifted clock is input to the others. In this case only one MMCM has to be reconfigured whereas if the shared logic (MMCM) is included in more than one AXI Ethernet Subsystem IP, then the above command is applied for each of the MMCMs.

Skew on the PCB traces:

The clock and data traces of Ethernet FMC are length matched between the FMC connector and the PHYs, hence, the PCB traces add no delay.

Skew in the PHY:

There are two internal delays in the Marvell 88E1510 Ethernet PHY's design, which can be enabled to add skew to the incoming RGMII TX clock and the outgoing RGMII RX clock independently. The delay is always 1.9 ns despite of the link speed. These delays are enabled or disabled by writing to a particular register in the PHY and depends on whether it is Linux or a stand-alone application which will be discussed in 6.13.

We are using AXI Ethernet Subsystem IP cores in the current design which will be discussed in 6.3. The RGMII TX clock delay is performed in the FPGA fabric by default for AXI Ethernet Subsystem. Our actual design has four AXI 1G/2.5Gigabit Ethernet Subsystem IP cores for four Ethernet ports of Ethernet FMC module. An AXI4-Lite bus interface is provided in the subsystem for a simple connection to the processor core in order to access the registers. For moving transmit and receive Ethernet data to and from the subsystem 32-bit AXI4-Stream buses are provided. AXI Direct Memory Access (DMA) IP core, AXI4-Stream Data FIFO or any other custom logic are to be used with the above bus. In our design, four AXI DMA IP cores have been used with respect to four AXI 1G/2.5Gigabit Ethernet Subsystem IP cores as already discussed in previous sections.

6.3 AXI 1G/2.5G Ethernet Subsystem IP

An Ethernet PHY device is connected to the PHY side of the subsystem which performs the BASE-T standard at 1 Gb/s, 100 Mb/s and 10 Mb/s speeds. AXI 1G/2.5Gigabit Ethernet subsystem IP core has a benefit to use 1000BASE-X, MII, GMII, SGMII, RGMII interfaces to connect a MAC (media access control) to a PHY (physical-side interface) chip.

In our current design, RGMII, the Reduced Gigabit Media Independent Interface (RGMII) is being used which has a benefit of Double Data Rate (transfer data on both rising and falling edges of the clock signal) and provides support for Ethernet operation at 10 Mb/s, 100 Mb/s and 1 Gb/s speeds.

Ethernet MAC functionality related features are displayed under MAC features tab of Ethernet Subsystem as shown in the *Figure 6.5*. Two important parameters are Tx Checksum offload and Rx Checksum offload. In general, data integrity can be sustained by calculating and verifying checksum over the TCP and UDP frame data. The protocol stack software handles the checksum functionality that uses sufficiently more processor power for large frames at higher Ethernet data rates. An alternative idea is to offload some or whole transmit checksum generation and receive checksum verification in hardware. By using the TX Checksum offload and RX Checksum offload parameters, this can be achieved and it results in higher Ethernet performance by using more FPGA resources while freeing up processor use for other functions.

In our design, we have full checksum offload of Tx and Rx enabled on the hardware and a Tx memory size and Rx memory size of maximum value i.e. 32K is set as shown in *Figure 6.5*.

Figure 6.5 AXI 1 G/2.5 Gigabit Ethernet Subsystem in the current design

6.4 AXI Bus

A data bus called AXI is part of ARM AMBA, a family of microcontroller buses called an "Advanced eXtensible Interface".

Three types of AXI4 interfaces are defined as follows

- AXI4 for high performance memory-mapped requirements.
- AXI4-Lite for simple and low-throughput memory-mapped communication (to and from status and control registers).
- AXI4-Stream for streaming data at high speed.

6.5 Gigabit Ethernet subsystems and DMA engine

An in-built dual Gigabit Ethernet controller that supports 10/100/1000 Mb/s EMAC configurations is present in the Zynq- 7000 AP SoC. An additional soft AXI EMAC controllers can also be configured in Programmable Logic (PL) subsystem of the Zynq-7000 AP SoC if more than two Gigabit Ethernet Controllers is the requirement. An example block diagram of the Zynq-7000 AP SoC with GEMACs is shown in the following Figure 6.6.

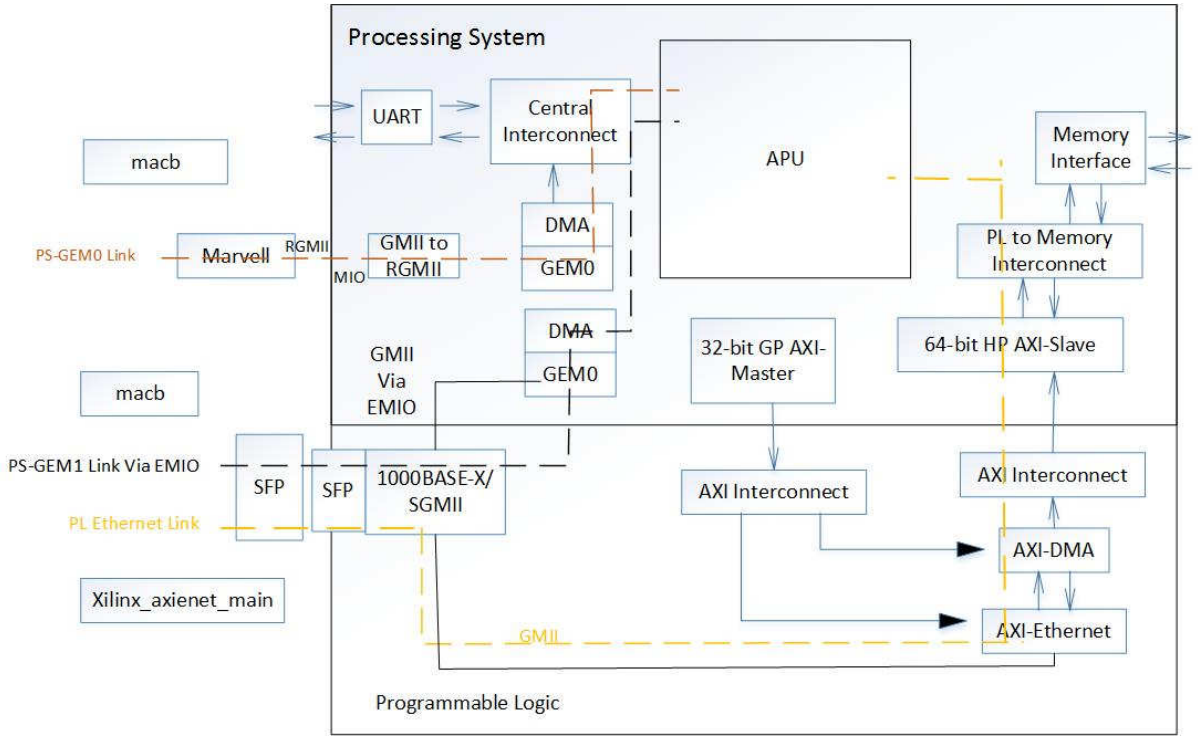


Figure 6.6 Gigabit Ethernet Design block diagram using Zynq-7000 AP SoC *[12]*

A brief description of various PS- and PL-based Ethernet implementations is given below. Among these

- PS-GEM0 is connected to the Marvell PHY through the reduced Gigabit media independent interface (RGMII) via MIO pins, which is the default setup for the ZC706 board.
- PS Ethernet (GEM1) that is connected to a 1000BASE-X physical interface (PHY) in PL through an EMIO interface.
- Ethernet implementation as soft logic in PL (MAC) using "AXI 1G/2.5Gigabit Ethernet subsystem" IP core and connected to the 1000BASE-X physical interface (PHY) and the GTH transceiver through RGMII (in our current design). Where 1000BASE-X PHY and the GTH transceiver are a part of the AXI Ethernet core.

On the board, 1000 BASE-X PHY is shared by PS-GEM1 and the PL Ethernet and only one can be used at a time. PS GEM0, PS GEM1 supports max frame size 1522 bytes. A jumbo frame size up to 16k can be supported for AXI EMAC in PL which is the main difference between PS and PL EMACs.

In the current project design, Ethernet implementation as soft logic in PL is chosen and four tri-mode (10/100/1000 Mb/s) Ethernet MACs are implemented using four AXI Ethernet subsystem IP cores as soft logics in PL and the AXI Ethernet subsystem IP core has been discussed in 6.3.

6.6 Application Data path on ZC706

In this section, as an example, Ethernet data movement in the Processing System of Zynq AP SoC is described before proceeding to describe Ethernet data movement in our actual design with four ports Ethernet implementation in Programmable Logic under the same section.

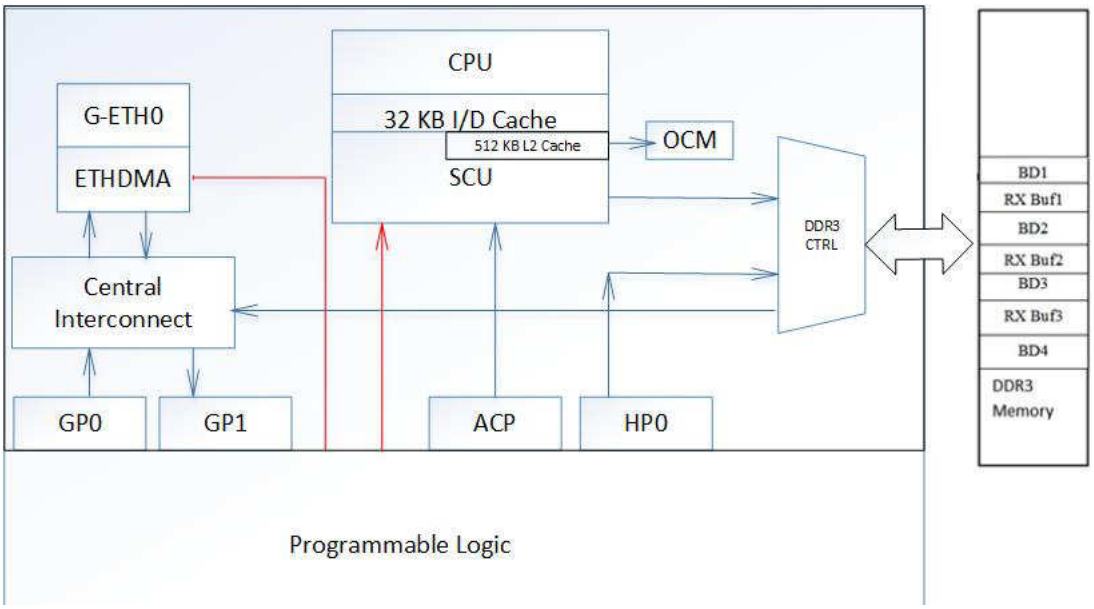


Figure 6.7 Ethernet Data movement in Zynq-7000 AP SoC [12]

On the Zynq Processing System, the Gigabit Ethernet MAC Controller has three main blocks.

1. MAC Controller
2. FIFO (Packet Buffer) and
3. Ethernet DMA Controller.

Receive Path

The Ethernet DMA controller is connected to the FIFO to give a scatter-gather capability for packet data storage in a Zynq processing system. Separate transmit and receive lists of buffer descriptors are used by the Ethernet DMA with every descriptor containing a buffer area in memory. The Ethernet DMA Controller writes the data received to pre-allocated buffer descriptor in system memory. Receive buffer queue has the list of these above said buffer descriptor entries. The Ethernet DMA has the Receive-buffer Queue Pointer register that points to this data structure on initialization. The Ethernet DMA uses the Receive-buffer Queue Pointer continuously and sequentially and copies the Ethernet packet received in the Ethernet FIFO to Memory address specified in the receive buffer queue.

DDR or OCM will contain these Rx Ring buffers and Tx Ring buffers, the speed at which the instructions execute for packet processing will also improve the overall performance. Thus, when an Ethernet Packet is received by the MAC, the address in the RX Buffer descriptor is used by the Ethernet DMA to push the packet buffered in the FIFO (Packet Buffer) on Ethernet interface to DDR3 memory, via the central interconnects.

Data Receive Path

ETH0 ➔ ETH0 DMA (32-bit) ➔ Central Interconnect ➔ DDR3 Memory Controller (64-bit AXI)

Transmit Path

The address in the TX Buffer descriptor is used by the Ethernet DMA in case of transmit to pull the data from DDR3 Memory, through the central interconnect and finally to the ETH0 Interface.

Data Transmit path

DDR3 Memory Controller (64-bit AXI) ➔ Central Interconnect ➔ ETH0 DMA (32-bit) ➔ ETH0

In the current project, the data receive path and data transmit path are as follows unlike the above paths because a four port Ethernet FMC module is used.

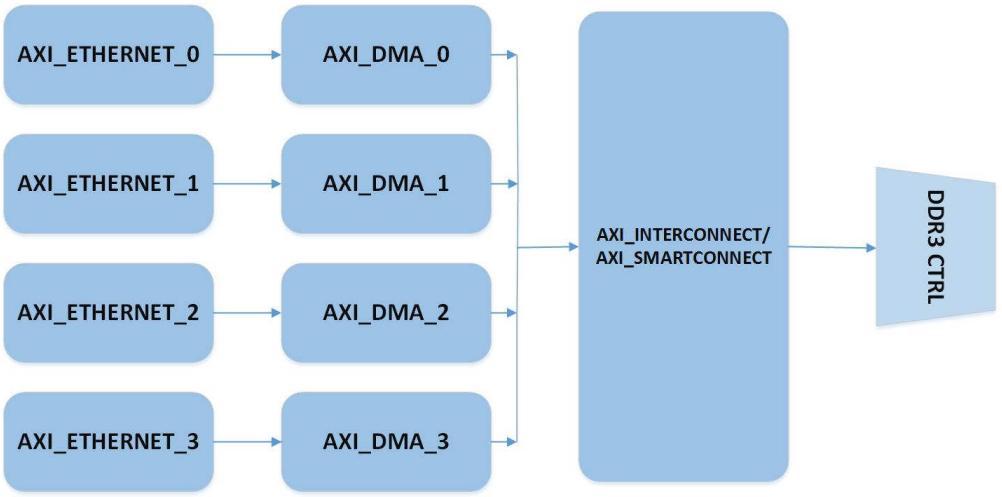


Figure 6.8 Ethernet Data movement in Zynq-7000 AP SoC with 4-port Ethernet FMC module

Data Receive Path (when Ethernet FMC module is used)

AXI _ETH[X] ➔ AXI_ETH[X]_DMA (32-bit) ➔ AXI Interconnect / AXI SmartConnect ➔ DDR3 Memory Controller (64-bit AXI).

where [X] = 0, 1, 2, 3

Data Transmit Path (when Ethernet FMC module is used)

DDR3 Memory Controller (64-bit AXI) ➔ AXI Interconnect / AXI SmartConnect ➔ AXI_ETH[X]_DMA (32-bit) ➔ AXI_ETH[X].

where [X] = 0, 1, 2, 3

6.7 DMA Engine

The DMA (Direct Memory Access) engine is an important part for maximizing performance in FPGA designs and helps to transfer data from one part of the system to another. As an example it is used to transfer data from one part of the memory to another. Also, the DMA engine is used to transfer data from any data producer like an ADC to a memory or from a memory to any data consumer like DAC. Previously, the processor had handled all data transfers between devices and memories. Due to increase in complexity and speed of the systems DMA was invented to free up the processor in dealing with data transfers from one place to another which removed a bottleneck in efficiency. The data throughput is typically very high for the processor to deal with so a DMA has an important role in high performance digital and FPGA systems. In the current design, the AXI Direct Memory Access (AXI DMA) IP provides high-bandwidth direct memory access between memory and AXI4-Stream-type target peripherals; in our case the target peripherals are AXI 1G/2.5G Ethernet Subsystem IP cores.

6.8 AXI DMA and scatter-gather mode

An overview of Scatter Gather, operation of Scatter Gather DMA and AXI DMA in scatter gather mode during the data acquisition and data transfers from PL to the DRAM in PS and vice versa are discussed in this section. The following IP core in the current design play an important role for data transactions to take place to and from the memory and AXI stream generator in our case AXI 1 G/2.5 G is the source for input and output stream of data in the implemented design.

6.9 Meaning of Scatter-Gather

If we have a data stream, which can be anything like a stream of Ethernet packets, a stream of packets over USB, a stream of packets over a PCI Express link or a custom interface connected to an A2D i.e. let data is entering the system and this data has to be processed and stored somewhere in the system. If the data has to be stored in the DRAM memory of our system, a set of blocks are allocated on the DRAM memory and the incoming data has to be arranged in a set of blocks as shown in the *Figure 6.9*. These blocks are not necessary to be one after another but they can be distributed across the physical memory. A stream of data enters the system and it scatters between different locations or physical addresses in the system.

The destination of all these packets may not be necessarily the DRAM but each of the packets may go to the different locations in the system. In the same manner, in the reverse direction suppose that we have different amounts of data and each amount is located at different physical addresses in the system memory and a stream of data is to be created out of these amounts of data at different locations. So, the DMA engine or the DRAM controller is responsible for reading and gathering all these amounts of data and putting them into a unified or unique stream of data. In our case these are Ethernet packets containing the data from the detector modules of the phenoPET project.

Figure 6.9 Example for scattering and gathering of data stream

6.10 Operation of Scatter-Gather DMA: Register mode

Previously the Scatter-Gather (SG) has been explained, here the operation will be explained. For each packet from the data stream entering the DMA engine CPU will define the transfer task which specifies where the packet should go. The DMA engine receives the transfer task and performs the transfer and as soon as it finishes the transfer, an interrupt is generated by the DMA engine to the CPU and the CPU defines and sends next transfer task to the DMA engine. In the reverse direction, same thing is true. There is some data stored in the physical address range of the system and a set of packets are to be produced and for each read and generation of the packets, the CPU defines from which address of the system the packet should be read and how much amount of data the DMA engine should read. As soon as the DMA engine finishes one transfer task the CPU defines a new transfer task. The CPU will receive an interrupt for

every transfer task and it should respond to the interrupt giving the next transfer task. The better way is explained in the next section.

Figure 6.10 Operation of DMA engine and buffer descriptors in register mode

6.11 Operation of DMA: SG Mode

The CPU defines all of the required transfer tasks at one point and it copies all of these transfer tasks into memory. When the DMA engine receives the data or when it wants to read the data from the physical addresses to produce the output stream, the DMA engine will first read the transfer tasks one by one and perform data transfer according to the definitions described in transfer tasks. The DMA engine interrupts the CPU only when all of the transfer tasks are finished. Thus, CPU instructs the DMA engine to begin the operation and provide address of the transfer tasks in the block memory and tells the DMA engine the ending address of final transfer task in the block memory. The CPU for the next set of data transactions will define a new set of transfer tasks. Each transfer task is called as a descriptor. Each descriptor is an indicator of an amount of data to be transferred from a source to the destination.

Figure 6.11 Operation of DMA engine and buffer descriptors in SG mode

In our current design, we are using AXI DMA in SG mode and an overview of the operation in SG mode for data transitions is as shown in the Figure 6.12. The heart of our system is the AXI DMA. This AXI DMA engine is used in SG mode (Scatter-Gather mode) in our design by connecting the SG mode interface pin to the HP0 register (High Performance port) of Processing System of Zynq. The AXI slave port of the PS i.e. HP0 is enabled. This AXI slave port HP0 is used to receive the data from the incoming AXI stream (S_AXIS_S2MM) and to put the data to the DDR3 (M_AXIS_S2MM). The same interface port is being used to do reverse operation i.e. to read the data from DDR3 (M_AXIS_MM2S) and put the data over the outgoing AXI stream (M_AXIS_MM2S). Through the GP0 port, AXI memory mapped master interface of the Zynq PS, the Stream generator or data source (in our case it is AXI 1G/2.5Gigabit Ethernet Subsystem IP core or 10 Gigabit Ethernet Subsystem) is handled. Here AXI DMA has two channels, in one channel it is writing data into DRAM memory and in another channel, it is reading the data from the DRAM memory. The descriptors (transfer tasks) are stored in the HP0 register of PS.

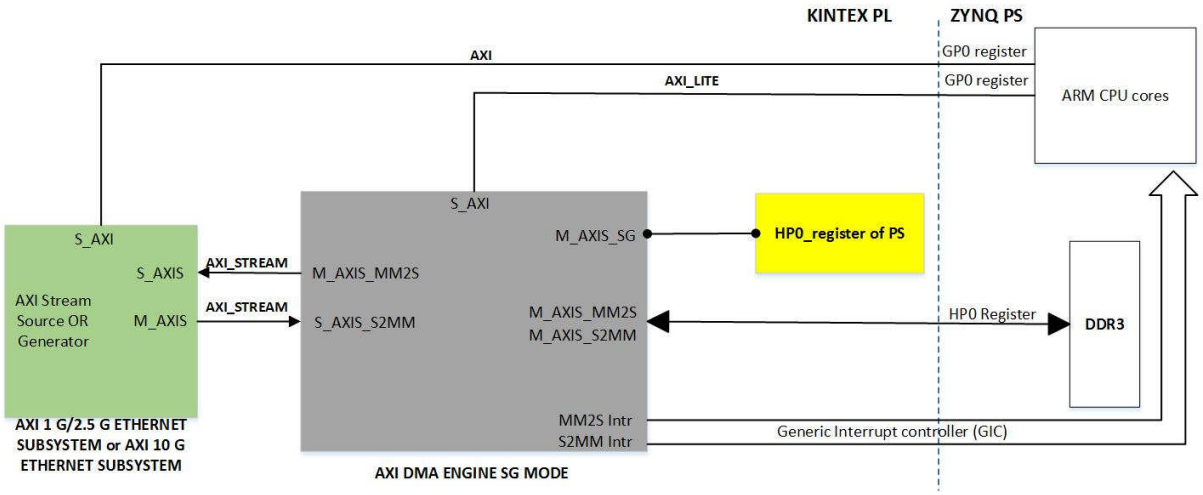


Figure 6.12 Operation of AXI DMA and its role in data transactions

During the design phase there are two important parameters to be changed in the customization of AXI DMA IP core and these parameters will affect the performance of data transfer from the PL to the PS for this component. As shown in the *Figure 6.13*, burst size is the number of data words we transfer in each read or write transaction. If we look at an AXI memory mapped connection between one AXI memory mapped master and one AXI memory mapped slave, at the time of initiating the transaction, the AXI memory mapped master can indicate the address of the transaction and the amount of data that the master wants to transfer. Indeed, the master doesn't need to tell address of every single word of data but master will tell only the start address to transfer this data and then the burst length helps in indicating how many words of data have to be transferred. Here in our design we have selected a Max Burst size of 16, it means that the master will transfer 16 words of data or 32 Bytes of data for every transaction.

Thus, the data on AXI interface can be transferred in bursts. In one transaction on AXI, N words can be transferred. Better throughput can be achieved for higher burst size. This parameter should be set to at least 16. We can set it up to 256 (maximum on AXI4). However the fact that PS AXI interfaces are AXI3 compliant limits the burst size to 16. Therefore, the AXI4 burst is split into several AXI3 bursts by the AXI interconnect. So we set the burst size parameter to 16 in our design.

The width of buffer length register is another parameter that affects the performance of the data transfer from PL to the PS. Indeed, the AXI DMA has a kind of internal buffer, here we have selected the width of buffer length as 16, and it means that after transfer of 2^{16} bytes i.e.,

65 Kbytes of data, CPU should configure the AXI DMA again, i.e. the CPU is allowed to ask DMA to perform transfers up to 2^{16} bytes.

Figure 6.13 Customization of AXI DMA IP core

6.12 Zynq PS

A dual-core ARM Cortex-A9 MP Core based processing system (PS) and Xilinx programmable logic (PL) in a single chip along with the on-chip memory, external memory interfaces, and an abundant set of IO peripherals are the main features in the Zynq-7000 family based Xilinx All Programmable SoC architecture. In the current design, the software interface around the Zynq-7000 is “Processing System 7” IP core. The Processing System (PS) and Programmable Logic (PL) are integrated in the SoC giving a flexible solution on a single platform. This core acts like a link between the processing system and programmable logic besides giving a feature to join other customized and embedded IP cores in the Vivado IP integrator.

6.13 Implementation of Linux on Zynq PS

We can run different types of software on the ARM host of Xilinx All Programmable SoC such as

1. Bare metal (standalone) application directly executed by ARM cores.
2. Linux OS running on the ARM cores and Linux kernel can handle both cores and can schedule different processes to different cores.
3. In addition to these, Free BSD can be run on the ARM host or Free BSD or Windows OS can also be run on the Xilinx Zynq device.

Since our hardware design with four Ethernet ports is ready, next step is to handle the four ports to receive the data from the detector modules in UDP packets by running UDP application to receive the packets and the chosen way is to boot Linux on Zynq and writing a software application that runs on processing system. PetaLinux tools from Xilinx offer *Embedded Linux* solution on Xilinx processing systems.

Why use Linux in our project:

If we are using Linux in our system, we will have these features already included

1. Network/File System support.
2. Inter process communication and memory management.

   This means two or more programs can exchange data between them and memory management meant for example, our program may ask some memory from the system to do something and leave this memory back to other programs.

3. Multithreading (it will use two cores in our Zynq).
4. Device drivers.

   Device drivers are special kind of software that make particular hardware available for our system.

5. Open source code available.

Compiling, porting the kernel and drivers can be challenging because Linux is complex, it takes time to boot, compared to booting the System on Chip for bare metal application.

6.14 Implementation of Linux on Zynq PS using PetaLinux

Hardware contains a lot of parameters and the important information that is used during software development. These parameters are mostly related to the initialization of the Zynq PS and with different address ranges, ARM host here can access different components that we have on the PL. Therefore, an export of hardware is needed in Vivado Design Suite before the software development. For example, when we want to use the DRAM controller, there are some set of parameters which are to be initialized before the DRAM controller is used effectively. When ARM host begins operating, the first thing it should do is, it should program all of these registers. And a reference to these registers can be found in a Xilinx Technical reference manual, the final part of the manual contains details of these registers.

Now we have Petalinunx Environment of the Xilinx in which we develop our Linux based application for our Zynq device. We have set of system configuration information which is reflecting different address values for different components inside our system and the type of the IP block present in our design. The set of information that is vital for the software development in fact specified in the hardware platform specification is exported from the Vivado environment to the Xilinx Software Development Kit is then being used for configuring the PetaLinux project.

A simple software application is created on the Linux OS running on the ARM host of the Zynq and cross compilation is done using the tool set provided by the Xilinx tool. Generated binary after cross compilation is executed on the ARM host.

After, the PL of the Zynq is programmed and after the bit stream is generated in Vivado as discussed in the 6.1, development of single Linux system at a time is supported by the PetaLinux project. Following are the main components in a PetaLinux project

1. First Stage Boot Loader
2. Device tree
3. U-boot
4. Linux kernel
5. Rootfs

The **First Stage Boot Loader** is responsible for programming the PL part and initializing the Zynq processor. And this is the first one that loads when we power up the Zynq. It will copy the Linux kernel into the processor's SDRAM and initialize the Zynq processor. The

Zynq boot process starts with running code inside the Boot ROM, boot medium is selected and FSBL is quickly loaded. The FSBL is created by Xilinx tools using information from our hardware project.

The **device tree or Flattened Device Tree (FDT)** is a data structure that contains byte coded format data, which is useful to the kernel when booting up. This amount of data is copied into the known address in the RAM before jumping into the kernel's entry point. Then it jumps the kernel's entry point. For example, in a PC, there are hardcoded initial registers and BIOS will supply the rest of the information. ARM processors do not have a BIOS, so Device tree is the opted solution. Thus, device tree is a way to convey the information about the specific hardware we have added or removed to the kernel so that a right driver is used by the kernel to handle the hardware. It is a file that describes all the devices and their drivers in the system. In simple words, Device Trees (DTB files) are used to describe the hardware architecture and address map to the Linux kernel.

Device tree configuration:

As discussed in 6.6, in handling the delays at the PHY side, it may depend on whether it is Linux or a stand-alone application. As such device tree is the best chosen way. Since we are doing a Linux application, to enable or disable the internal clock delays we specify a particular value for the "phy-mode" parameter in the device tree.

```
&axi_ethernet_0 {
    local-mac-address = [00 0a 35 00 01 22];
    phy-handle = <&phy0>;
    xlnx,has-mdio = <0x1>;
    phy-mode = "rgmii";
    mdio {  #address-cells = <1>;
            #size-cells = <0>;
            phy0: phy@0 {
            compatible = "marvell,88e1510";
            device_type = "ethernet-phy";
            reg = <0>;  };  };  };
```

Code snippet 2 Device tree configuration for phy-mode

For each of the Ethernet interfaces (axi_ethernet_0, axi_ethernet_1, axi_ethernet_2 and axi_ethernet_3) the device tree is scripted in the above said way (*Code snippet 2*).

To enable or disable the internal clock delays, we specify a particular value for the “phy-mode” parameter.

Both internal delays DISABLED:

```
phy-mode = "rgmii";
```

Both internal delays ENABLED:

```
phy-mode = "rgmii-id";
```

Only RX internal delay ENABLED:

```
phy-mode = "rgmii-rxid";
```

Only TX internal delay ENABLED:

```
phy-mode = "rgmii-txid";
```

U-Boot, is a piece of software commonly used to load Linux into the board.

Linux Kernel is the core of the Linux operating system with complete control over everything in the system. Kernel.org together with Xilinx additions (BSP and drivers) offer the Linux kernel for Xilinx Zynq. Linux kernel is the central part of the operating system that links the hardware with the applications.

Linux Kernel job

On a system, the job of a kernel is to manage all the following demands posed by different programs asking for resources at the same time:

1. Process management
2. Memory management
3. File systems
4. Device control
5. Network

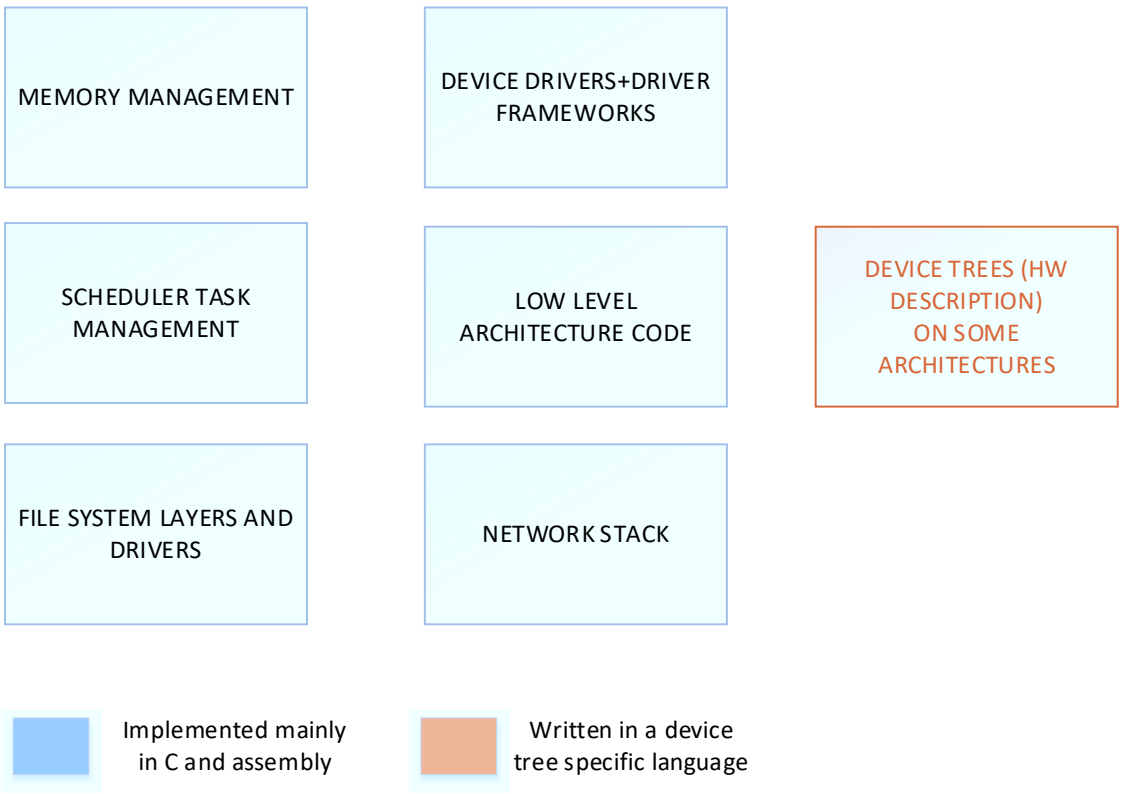


Figure 6.14 Linux Kernel Components

Rootfs

On Linux systems, several filesystems are mounted and create a global hierarchy of files and directories, a particular filesystem, the root file system is mounted as '/', this root file system contains all the libraries, applications and data of the system.

Therefore, building the root filesystem is one of the main tasks of integrating embedded Linux components into a device. User application written in C such as "multiUDP4final" application (see 11.1 under 0 Appendix) is added to our rootfs after cross compiling using arm-Linux-gnueabihf-gcc (cross compilers depend on the architecture of ARM in the Zynq PS, ZC706 AP SoC has ARM hard float architecture) .

For example, for compiling the user application (in our case it is multiUDP4final.c) for ARM hf architecture following command line in *Code snippet 3* will be used.

```
arm-Linux-gnueabihf-gcc multiUDPfinal.c -o multiUDPfinal -pthread
```

Code snippet 3 Command for cross compiling the user application on Linux

Once the binary called multiUDPfinal is generated, it is booted along the boot image or can be sent to the /usr/bin folder of rootfs over one of four Ethernet ports using scp commands since dropbear ssh is configured on the Zynq PS. The command to send the file using scp commands from a Linux terminal window is in the following way.

```
janani@janani-Latitude-E5550:~$ ssh root@192.168.4.14
The authenticity of host '192.168.4.14 (192.168.4.14)' can't be established.
RSA key fingerprint is SHA256:1KZf4q7y1UnYDXC1hHo/+UqKTc+hmsGQolG25XwL9Ws.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.4.14' (RSA) to the list of known hosts.
root@192.168.4.14's password:
root@plnx_arm:~#
```

Code snippet 4 Linux command for ssh login into Zynq PS

```
janani@janani-Latitude-E5550:~/projects/peta2017.1-zcu102/zc706/Clustering/files
$ sudo scp multiUDP4final1 root@192.168.4.14:/usr/bin
root@192.168.4.14's password:
multiUDP4final1                               100%   13KB  13.4KB/s   00:00
```

Code snippet 5 Linux command for SCP transfer to Zynq PS

The rootfs of the boot image for ZC706 contains following applications and libraries compiled and booted along the boot image or sent over the Ethernet ports using SCP commands. (See pictures in 11.3).

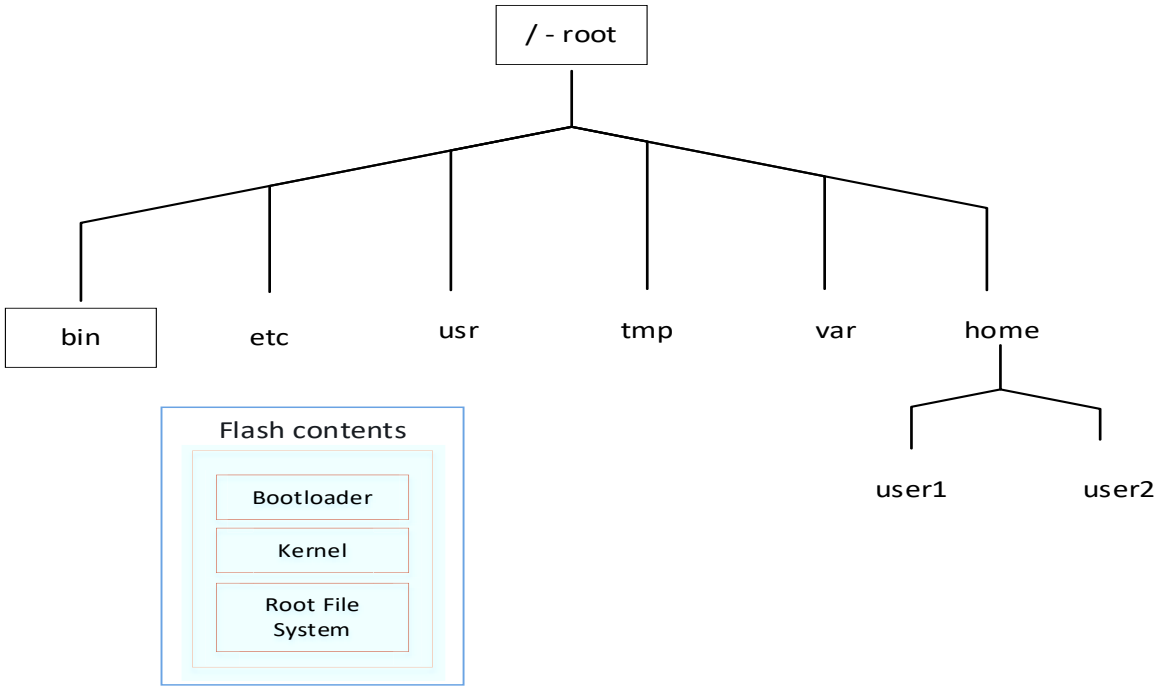


Figure 6.15 Flash or SD Card Contents and RootFS structure

FSBL is executed first followed by execution of U-Boot that loads device tree and Linux kernel into the memory and followed by mounting rootfs. Thus, after the hardware bit stream and software images have been built, the new PetaLinux platform with the Zynq kernel is booted using SD card. The *Figure 6.16* shows a high level block diagram of the design flow using Vivado and PetaLinux.

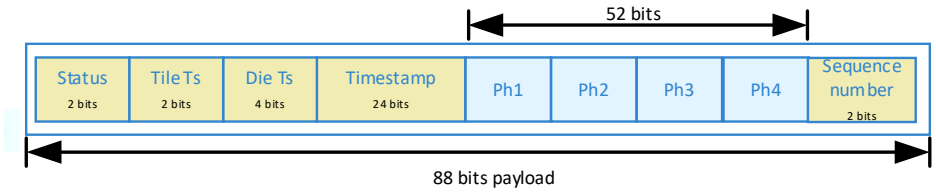
Figure 6.16 Xilinx tools design flow at Implementation level *[13]*

6.15 Implementation of clustering algorithm in PetaLinux

As discussed in the previous section, the rootfs contains user applications written in C programming such as "multiUDP4final" added to our rootfs after cross compiling using arm-Linux-gnueabihf-gcc. The following application is written in C for handling the four Gigabit Ethernet ports for acquiring data in UDP packets on four Ethernet ports where multi-threading concept is implemented for four Ethernet ports. And clustering algorithm is written in C program to process the data acquired from each Ethernet port. (See multiUDP4final program that has clustering algorithm in 11.1 in appendix)

The Clustering application contains the algorithm required to cluster the data, the algorithm will run in such a way that it extracts the data from each frame and clusters the data based on time stamp in a time window of 5 ns and keeps it into a buffer. Later, the data can be sent out on 10 Gigabit Ethernet port.

The data in each Ethernet frame contain data from 4 tiles starting with a command byte a6 and a header d2 followed by payload data at 27th byte. This data coming from the Tiles are sorted on Timestamps. The data is extracted from the payload in the required order from the frame and is in the following order.



This data is clustered based on timestamps in a window of 5 ns and the clustered data contains data in the following order:

- Ts of 1st hit in cluster
- Most intensive hit between sum of photon counts ($\sum ph_i$ where i = 1, 2, 3, 4)
- Tile_ts
- Die_ts

To send the clustered data out on 10 Gigabit Ethernet, implementation can be done using Linux driver for PL Ethernet on another evaluation board such as ZCU102 that has 10 Giga Ethernet support. This is not done in current thesis and can be seen in *Figure 6.17*.

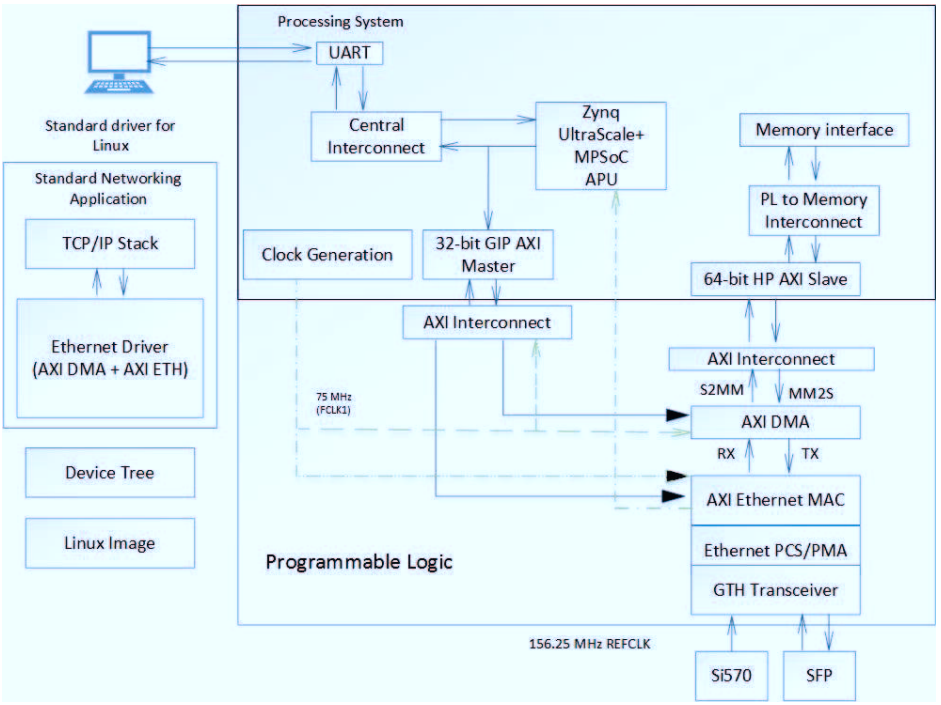


Figure 6.17 Block diagram for Linux driver for PL Ethernet for 10 Gigabit Ethernet *[14]*

7. System Verification

This study provides a measurement of Ethernet bandwidth utilization versus actual bandwidth from the stress tests performed on the datagrams. It provides information about the processor utilization during the tests and also when the user application for data acquisition on UDP packets and data processing is running.

Maximum no. of frames per second $= \frac{1000}{Ethernet\ frame\ size\ in\ Bytes} \left(\frac{Mega\ Bits}{Second}\right)$

$= \frac{125}{Ethernet\ frame\ size\ in\ Bytes} \left(\frac{Mega\ Bytes}{Second}\right)$

$= \frac{125}{1538\ Bytes} \left(\frac{Mega\ Bytes}{Second}\right)$

$= 81275\ \left(\frac{frames}{Second}\right)$

Thus, a maximum of 81275 frames per second on 1000 Mbps or 1 Gbps occurs for an Ethernet size of 1538 Bytes in the Ethernet frame.

Transaction time for each frame $= \frac{1}{81275} \left(\frac{seconds}{frame}\right) = 12.3\ \left(\frac{\mu\ seconds}{frame}\right)$

Whereas for jumbo Ethernet frame of size 9000 Bytes, the transaction time would be 64 µs.

If a frame transaction of 12.3 µs occurs in an Ethernet network interface, it means that the software on the Processing System of Zynq has to finish handling and processing of a frame in 12.3 µs and then available to handle the next arriving frame. This time bound execution will ensure that software is working in relation with the Ethernet hardware. If the software is unable to handle and process the frame in this time then it is unable to reach the much needed equilibrium with respect to Ethernet hardware to endure a line rate of 1 Gbps. A backpressure (during RX) or a starvation (during TX) are created on the hardware. The Ethernet hardware will overrun and drop the frames from the Ethernet wire because of this backpressure or the hardware will under-run and the wire is underutilized due to the starvation.

7.1 Solutions for better performance from Ethernet design

Few solutions for better performance of Ethernet have been discussed in 6.11 and 6.3 at design level. Also, the use of jumbo frames in high data intensive applications will increase the throughput. The performance is improved by larger frame size by decreasing the no. of fragments for a given size of data.

7.2 Solutions for better performance from user space

When Linux kernel/XAPP1082 image is booted on Zynq 7000 AP SoC, following commands can be applied

1. Configuring the MTU (Maximum Transmission Unit) to jumbo size on both the server and client.

The size of the largest data unit in network layer that can be communicated in a single transaction is called Maximum Transmission Unit (MTU).

In other words, MTU relates to the size of the final product from the transport layer after adding headers and checksum to the payload or data i.e., MTU relates to the data packet in the network layer before adding source and destination IP addresses in order to convert it into complete Ethernet frame in the datalink layer. So, if MTU is more, more throughput or data in a data segment in the transport layer and less number of data packets in the network layer and so less number of heads and tails appear in the Ethernet frame which implies that more throughput will fit in it. If MTU is less, then less throughput in the transport layer and more number of data packets in the network layer and so more heads and tails appear in the Ethernet frame. Thus, a greater efficiency comes with larger MTU because more user data fits into each network packet. A larger MTU resulting in fewer packets are processed for the same amount of data.

```
On server side (on Zynq)
ifconfig eth0 192.168.1.11 netmask 255.255.255.0 up
ifconfig eth0 down
ifconfig eth0 mtu 9000
ifconfig eth0 up

On client side (on the host PC)
ifconfig enp1s0f0 192.168.1.20 netmask 255.255.255.0 up
ifconfig enp1s0f0 down
ifconfig enp1s0f0 mtu 9000
ifconfig enp1s0f0 up
```

Code snippet 6 Rising MTU to 9000 for network interfaces

2. The task set-2 feature is being used in the current project to share the load between two cores of ARM Cortex A9s when the application is being launched.
3. Configuring Window size for better performance.
4. For better performance, configuring Window size is one of the options. The client's receive window on the receiver side and it is the server's send window i.e. the no. of bytes, the client would like to receive from the server at one time. Similarly, the server can tell the client no. of bytes of data it would like to take from the client at one time and is a server's receive window and client's send window. Window size may drop down to zero dynamically if the receiver is unable to handle the data as fast as sender is sending the data. Better performance can be achieved with the larger size of the window.
5. –W option is used to specify the window size while using iPerf3 benchmarking.

7.3 Test Results

These measurements are obtained against iPerf3 client program running on Linux host PC with Ubuntu 16.04 OS and iPerf3 server program (an application in the rootfs or bin folder of the image) running on the Zynq PS with petaLinux image.

7.4 Performance tests of Ethernet ports

In the below figures, throughput achieved for actual bandwidths in Mbps and data loss occurred in percentages are shown. These are obtained during the stress tests performed on the Gigabit Ethernet ports.

Figure 7.1 Performance of Port 0

Figure 7.2 Performance of Port 1

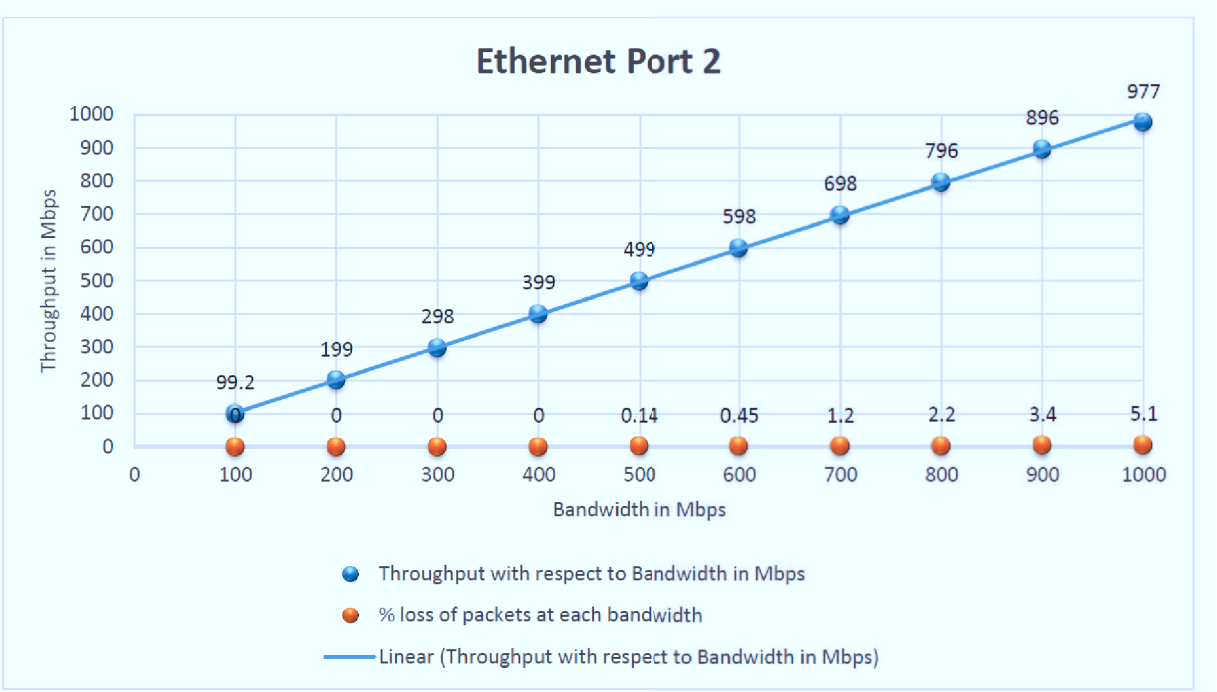


Figure 7.3 Performance of Port 2

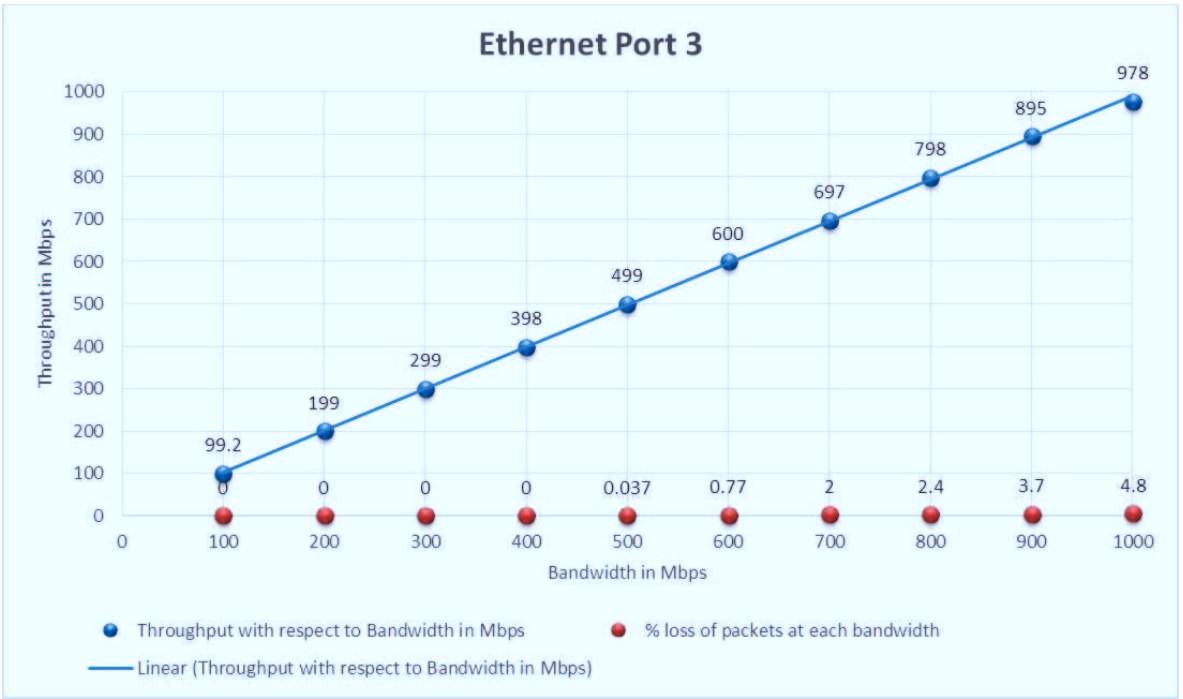


Figure 7.4 Performance of Port 3

7.5 CPU core utilization

Figure 7.5 presents the percentage utilization of ARM core on Zynq PS during performance test on each Ethernet port at 1 Gbps bandwidth whose results at different bandwidths are already presented above.

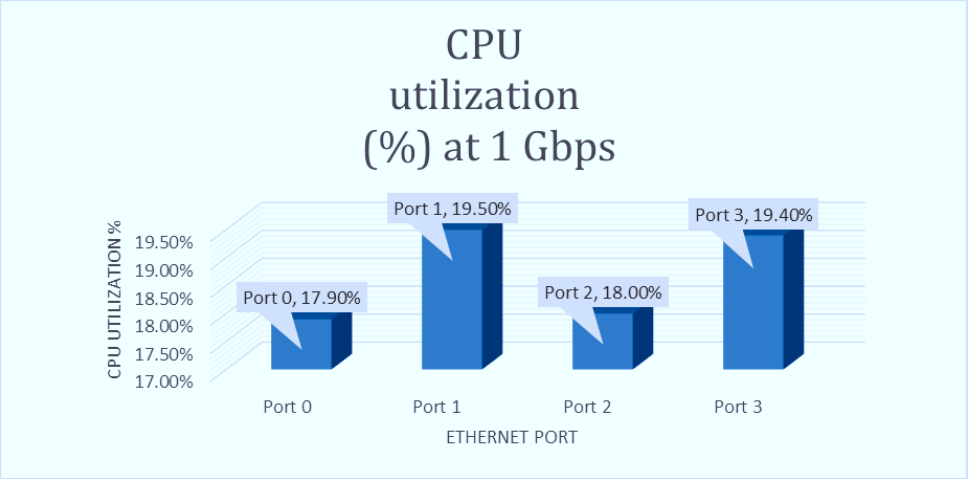


Figure 7.5 CPU utilization for stress test on each Gigabit port at 1 Gbps

Figure 7.6 and *Figure 7.7* present the utilization of both ARM cores in percentages during the parallel tests performed over four Ethernet ports together at different bandwidths from 100 Mbps to 1 Gbps. It also presents the respective CPU core being used by the Ethernet ports on Zynq PS.

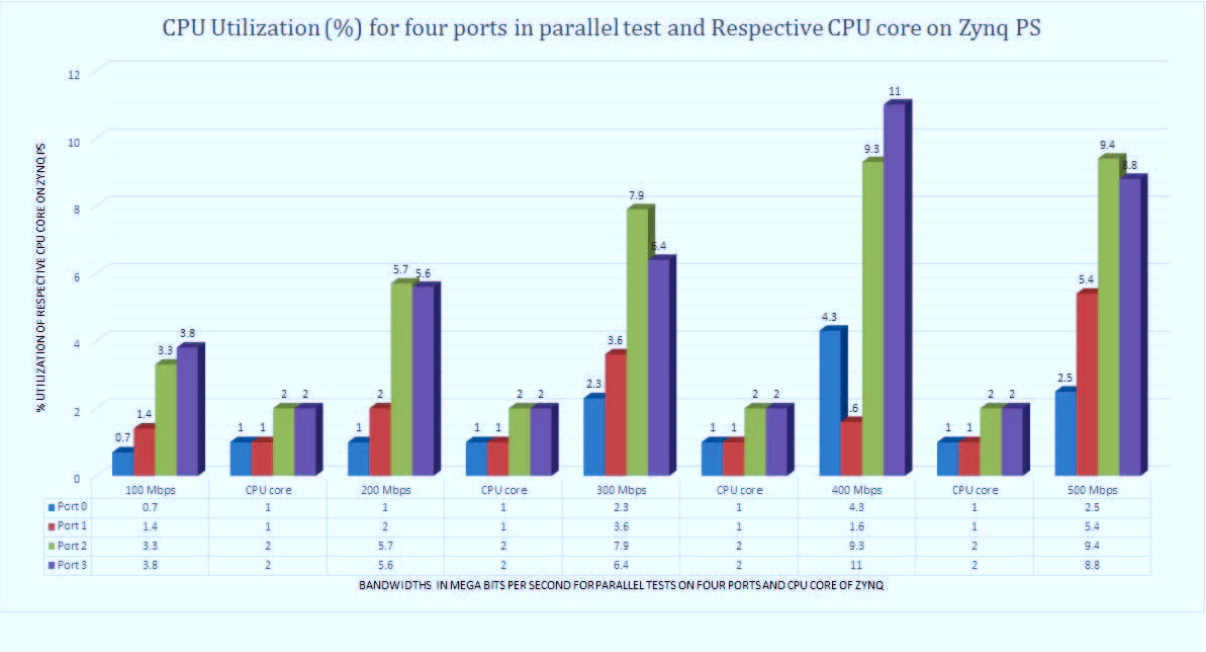


Figure 7.6 CPU Utilization (%) in parallel test and respective CPU core

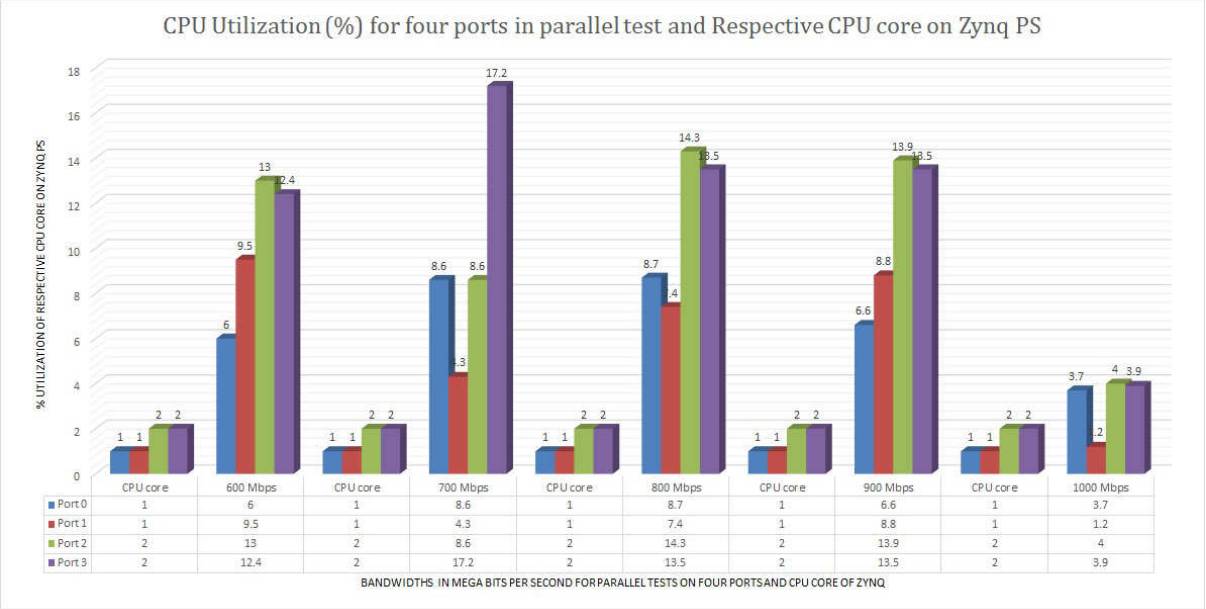


Figure 7.7 CPU Utilization (%) in parallel test and respective CPU core

Figure 7.8 presents the CPU utilization in percentage and respective CPU core (see % CPU and CPU) when *multiUDP4final* application is running on Zynq PS.

```
Mem: 36576K used, 994140K free, 156K shrd, 3820K buff, 9700K cached
CPU:  0.0% usr  0.2% sys  0.0% nic 99.5% idle  0.0% io  0.0% irq  0.0% sirq
Load average: 0.02 0.02 0.00 3/99 1206
  PID  PPID USER     STAT   VSZ %VSZ CPU %CPU COMMAND
 1194   984 root     S    35628  3.4   0  0.1 multiUDP4final
 1191  1120 root     R     3256  0.3   1  0.1 top
 1096     2 root     SW       0  0.0   1  0.1 [kworker/1:0]
  163     2 root     SW       0  0.0   0  0.1 [kworker/u4:2]
  977     1 root     S    13528  1.3   1  0.0 /usr/sbin/tcf-agent -d -L- -l0
  984   982 root     S     3744  0.3   1  0.0 -sh
 1120  1119 root     S     3744  0.3   1  0.0 -sh
```

Figure 7.8 CPU Utilization (%) for *multiUDP4final* application

Figure 7.9 presents the CPU utilization in percentage and respective CPU core (see % CPU and CPU) when *multiUDP4final* application along with clustering algorithm is running on Zynq PS.

```
Mem: 40444K used, 990272K free, 156K shrd, 3888K buff, 9576K cached
CPU:  0.2% usr  0.3% sys  0.0% nic 99.1% idle  0.0% io  0.0% irq  0.1% sirq
Load average: 0.00 0.00 0.00 2/98 1076
  PID  PPID USER     STAT   VSZ %VSZ CPU %CPU COMMAND
 1069   984 root     S    64308  6.2   1  0.5 multiUDP4final1
 1076  1047 root     R     3256  0.3   0  0.1 top
 1046   951 root     S     2832  0.2   0  0.1 /usr/sbin/dropbear -r /etc/dropbear/dropbear_rsa_host_key -p 22 -B
 1030     2 root     SW       0  0.0   0  0.1 [kworker/0:2]
    7     2 root     SW       0  0.0   1  0.1 [rcu_preempt]
```

Figure 7.9 CPU Utilization (%) for *multiUDP4final* application with cluster algorithm

8. Conclusion

The Ethernet port setup for data acquisition of phenoPET detector data is successfully implemented using the four Gigabit Ethernet ports on the PL of Zynq-7000 XC7Z045-2FFG900C AP SoC using Vivado Design Suite. Data acquisition and data processing of PhenoPET data has been implemented successfully on PS of Zynq using Xilinx PetaLinux tool chain. After data acquisition for phenoPET data is performed on four Ethernet ports over UDP application written in C, clustering of the data from each port is performed over Zynq PS and buffered into system memory.

In the current project we achieved almost full Ethernet bandwidths which can be seen under section 7.3 under System Verification chapter during the stress tests on the multiple Gigabit Ethernet ports. Percentage utilization of CPU cores of ARM is estimated during the stress test on each Ethernet port and during the parallel tests on the four Ethernet ports together. Percentage utilization of CPU cores of ARM during the user application task for UDP and clustering running on the Zynq Processing System is also done. These results for percentage utilization of CPU cores of ARM are shown in Figure *7.8* and Figure *7.9* under 7.5 section of *Test Results* chapter.

The four Ethernet ports those are validated for UDP with iPerf3 are successfully verified using the same at different bandwidths in an interval of time. The clustering algorithm which has been validated for the already available PhenoPET data using MATLAB is successfully verified after the implementation of the algorithm over the Zynq PS.

The UDP user application handling data acquisition on four Gigabit Ethernet ports written in the programming language C has been tested for the acquisition of phenoPET data and the clustering algorithm is also tested by running the algorithm on the Processing System of Zynq for data processing.

9. Outlook

The results of CPU percentage utilizations while testing the four Ethernet ports (presented in 7.5 under Test Results chapter) did not reach our expectation during the stress tests on each of four Ethernet ports. This seems to be due to the processor being a bottleneck.. The same implementation might give better performance for percentage utilization of CPU on ZCU102 ultrascale+ MPSoC since it has quad core architecture. If we want very low utilization of ARM cores in the stress tests, we need complete parallelization of our hardware design, including the processors on FPGA or more than two ARM cores to handle the multi Gigabit ports.

Due to time constraints, the implementation of 10 Gigabit Ethernet for sending the clustered data out is an upcoming challenge and can be implemented in future approaches in this project. The implementation of Ethernet interfaces to send the clustered data out at higher bandwidths adds fulfilment to the current project resulting in a complete setup for data acquisition and data processing for phenoPET project. The concept behind this thesis for data acquisition and processing at higher data rates can be extended for other projects also.

10. References

[1] Jülich Forschungszentrum, “Jülich Forschungszentrum Press Release: German Plant Phenotyping Network (DPPN) Launched,” 2013. [Online] [Accessed 12/06/2017].

[2] P. Jahnke, et al., “Design and initial performance of PlanTIS: a high-resolution positron emission tomograph for plants,” Vols. Physics in Medicine & Biology-55, January 13th 2010.

[3] Jülich Forschungszentrum, www.fz-juelich.de/ibg/ibg-2/EN/methods_jppc/PET_PlanTIS_phenoPET/_node.html, [Online] [Accessed 12/06/2017].

[4] P. Wüstner, et al., “*The Use of USB 3.0 for Fast Data Transfer in a PET Detector*” in *19th IEEE-NPSS Real Time Conference*, Nara, 2014.

[5] M. Streun, “*phenoPET The Jülich Plant-PET Development*”, 63rd Crystal Clear Collaboration Meeting, Prague, Czech Republic, April 16th 2015.

[6] M. Streun, "*phenoPET Time Skew Calibration",* Forschungszentrum Jülich, Jülich, Germany, October 17th 2017.

[7] *ZedBoard, zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf,* [Online] [Accessed 12/06/2017].

[8] Xilinx, zedboard.org/product/zedboard, [Online] [Accessed 12/06/2017].

[9] Xilinx, "*ZC706 Evaluation Board for the Zynq-7000 XC7Z045 All Programmable SoC*", User Guide*,* [Online] March 29th, 2016.

[10] Opsero Electronic Design Inc., ethernetfmc.com/wp-content/uploads/2014/10/quad-gige-w-zedboard-zynq-small-1.jpg, [Online] [Accessed 12/06/2017].

[11] Opsero Electronic Design Inc., ethernetfmc.com/rgmii-interface-timing-considerations, [Online] [Accessed 12/06/2017].

[12] Anil Kumar, et al., "*PS and PL Ethernet Performance and Jumbo Frame Support with PL Ethernet in the Zynq-7000 AP SoC*", Application Note, 2015.

[13] Xilinx, wiki.xilinx.com/PetaLinux+Getting+Started. [Online] [Accessed 12/06/2017].

[14] Bhargav Shah, et al., "*PS and PL-Based 1G/10G Ethernet Solution*", Application Note, 2017.

[15] M. Streun, et al., “*PhenoPET: A dedicated PET scanner for plant research based on digital SiPMs (DPCs)*", Seattle, WA, USA, 2014.

Jül-4406 • Februar 2018
ISSN 0944-2952

Mitglied der Helmholtz-Gemeinschaft