MINISTRY FOR DEVELOPMENT OF INFORMATION TECHNOLOGIES AND COMMUNICATION OF THE REPUBLIC OF UZBEKISTAN

TASHKENT UNIVERSITY INFORMATION TECHNOLOGIES

Admit to protection
Head of department
Amirsaidov U.B.
____
2015 y.__________

FINAL QUALIFICATION WORK OF BACHELOR DEGREE

On the theme:

"DESIGNING DEVICE CONTROLLER SYSTEM THROUGH AVAILABLE OPEN SOURCE HARDWARE CORE"

Graduating student_______________ A. F. Ruziqulov
(signature)

Teacher______________ O.I. Urmonov
(signature)

Reviewer_______________ Z.Z. Rashitov
(signature)

Adviser on LSE______________ F. M. Qodirov
(signature)

Tashkent 2015

CONTENT

Introduction

Open source hardware has a long history and has gone through several cycles of growth. It can be argued that the first notion of open source hardware was in the early 1970's with groups such as the Homebrew Computing Club. In these early days of computer engineering a small group of engineers collaborated and openly shared designs of what would become one of the first personal computers. According to Apple cofounder Stephen Wozniak, “a lot of tech-type people would gather and trade integrated circuits back and forth. There was no official or formal organization and the main role of the club was to trade circuit designs amongst members. The theme of the club was "Give help to others” and membership to the group was mostly as a hobby.

Wozniak states that schematics and designs for Apple I products were "passed around freely" and help was provided in order to help other members build their own systems. Free revealing in this manner, provided the designers of the Apple computer with early feedback on their initial prototype by other lead users and at least part of the motivation for free revealing was the increased reputation gained amongst peers. Open source hardware designs in this case relied on the ability of users to tie different components together. The use of breadboards made it such that users did not need manufacturers in the process of developing products. Information was transferred from Innovating lead users to all users in the community through newsletters and Informal gatherings. From the early days of open source hardware at the "product" level, soon lead developers turned their attention into how to create the basic building blocks of the products. While the initial wave of Open Source Hardware was related to how to put the basic building blocks together the late 1970’s to mid 1980’s saw large development at the function level of the value chain. In the 1970’s Carve Meade and Lynn Conway developed a method by which users could begin to design large-scale integrated circuits. Several designs were grouped into a single production run in order to reduce development costs. Designs were transmitted using the Arpanet to manufacturing sites in order to be built. This process allowed universities access to low cost production of highly complex integrated circuits but users outside of this system could not easily develop or replicate the same products. Although there were several free development tools, the computing power needed to produce large designs was extremely expensive. In this phase of open source hardware users began requiring the services of manufacturers in order to develop their products.

The 1990's saw increased development of proprietary design tools and this increased the knowledge needed to design a particular platform making it difficult to switch and share designs at the user level. Although the design tools allowed for easy transfer of design output to manufacturers the increased segmentation in the tool market made it difficult to share designs between different users. Board design tools did not easily translate from one CAD manufacturer to another, for example design rules and databases for a board being designed with a tool such as "Mentor Graphics" were not easily translated to that begin designed under the "Cadence" environment. On the IC design side design input was begin driven by the IC manufacturers making it difficult to translate designs from one type of IC to other types of IC. This made it such that one ended up designing for a particular "targeted device" on a particular "tool flow", making it difficult to reuse the design or share it with others that were not operating on the same platform. This approach led to the creation of whole groups of digital designers segmented by manufacturer and/or design tool.

Today changes in tool design methodology and standardization of interfaces have made it easier for open source to thrive and for user innovation communities to develop. Open Source hardware is thriving in cases in which users require manufacturers and cases in which users require little to no manufacturing involvement. As will be shown later most of the growth appears to be in innovation communities in which manufacturer involvement is little and where users can easily share, modify and upgrade the hardware with low cost approaches.

1. CHALLENGES OF OPEN SOURCE HARDWARE (OSH) SYSTEMS' INTEGRATION

1.1 Open Source Hardware system benefits and convenience from the network engineer's perspective

Open Source hardware today exists at most stages of the value chain, from low-level design gates to functional cores that can perform tasks such as image analysis to platforms and complete system/product solutions. Figure 1 shows the value chain or hardware from Atoms to Solutions and an Open Source Hardware example that exists today at each stage.

Value Add Activities

	Example	**Open Source**
Product	**ECG system**	**Open Medic**
Sub-System	Signal Conditioner	Bug Labs
Function	Power Regulator	Open Cores
Gate	Amplifier	

Solutions

Figure 1.1 Value add activities

At the bottom of the value chain "atoms" such as transistor level gates exist. Production of gates at this level can be extremely capital intensive, and traditionally one does not build individual gates but at the most basic level a few low level functions such as small drivers, buffers, amplifiers and other basic building blocks. Current state of the art development of integrated circuits in a process can cost upwards, with older technology costing anywhere. It is clear that one can develop Open Source hardware at this level but activity is typically limited to sharing the physics and material implementation of the devices. Development of hardware at this level can be extremely costly in terms of fabrication because it relies heavily on economies of scale. Companies such as IBM tend to develop a process and open source the methodology so that other corporations can utilize their fabrication facilities. This method helps owners of semiconductor facilities continue to develop innovative technologies while at the same time reducing the cost of owning and maintaining the fabrication facilities [2].

By open sourcing the technology IBM draws developers towards its fabrication process and helps to maintain production levels high while at the same time generating enough revenue to continue to develop processes that are innovative. There is no "open source of the design" but mostly of the process. One step above the gate level, stand-alone circuit designs are available in the form of electronic schematics or cores. These designs typically relay on hardware only and require little to no external knowledge of system operation. Examples of designs at this level can range from simple controller such as universal serial bus (USB) drivers, Ethernet framers and more complicated functions such as CPU's and video chips. The cores by themselves provide valuable functions but need to be tied to other cores in order to perform a partial solution for the user. The integration of functions to form larger subsystems requires the user to understand both the core implementation and the interface characteristics. Developers of open source hardware typically generate custom cores through less expensive manufacturing paths such a reconfigurable logic or by wiring discrete atoms together.

Developers have been known to generate highly integrated cores but this approach usually involves a higher cost and greater interaction with manufacturers. An example of a subsystem is a Video Processing core tied to a display driver and an associated LCD screen in order to form a touch screen interface. This type of display/interface subsystem can be found in newer smart phones. This type of subsystem allows for the user hardware to display images and to receive input from a user but does not provide a full solution to a user need, it can however be used by a developer as a part of a system for a phone or a gaming device [5].

On the product side there are cases in which the entire product is truly open including schematics and software to re-engineer the entire product and other cases where only a small number of interfaces to the product are available. In some cases only portions of the hardware are "open" these cases, which typically refers to cases in which all documentation is provided to make the hardware function but no details as to how the hardware is built.

1.2 Evaluating Open Source Hardware innovation and its potential effects in the Value Chain

Von Hippel demonstrated that there was a continuing trend towards democratizing innovation. By democratizing innovation Von Hippel meant that users were increasingly capable of developing products and solutions by themselves. Von Hippel also explained the process by which this shift was occurring and how innovation by users complemented the innovation done by manufacturers. Open Source Hardware can help drive this innovation by functioning as a source of designs which users can quickly leverage to build upon and create new products. Von Hippel developed a series of attributes to see where and how innovation was being democratized. These attributes taken from his book Democratizing nnovation9 will be used to look through the value chain and see if the new sources of innovation are relying on open source hardware. The attributes will help to determine in a qualitative way if and how open source hardware is being utilized to build new and innovative products. In particular the following attributes of widespread innovation qualified by the use/or role of open source hardware will be utilized:

- Evidence of open source hardware developed or utilized by lead users
- Need for custom solutions at several levels of the value chain and whether those custom solutions are based on Open Source Hardware
- If the cost of implementation and use of open source hardware at that stage can reduce the cost of building hardware or if it is cheaper to purchase existing hardware outright
- User low cost innovation niches along with information stickiness at each level

- Evidence of free revealing and reuse of open source hardware not only the of the source itself but of the manufacturing and development process
- Existence and participation in innovation communities that develop open source hardware
- Availability of toolkits in that are assist development with open source hardware

Activity or evidence of innovation based on open source hardware present at any stage of the value chain can point towards a strong possibility that open source hardware can begin to displace or change established revenue models and organizations. In some cases manufacturers will have to retool to accommodate more of the activity being done by users as Von Hippel established. In other cases firms may see their business models in direct competition with open source hardware [1].

Von Hippel observed the following characteristics of lead users lead users are at the leading edge of an important market trend and are currently experiencing needs that will later be experienced by many users in that market. And secondly they "anticipate relatively high benefits from obtaining a solution to their needs and so many innovate". Clearly if lead users have adopted Open Source Hardware as a vehicle to develop products or subsystems it is a an indication that Open Source Hardware can play an important role in future innovations and be the basis for future products in the market, if lead users continue to develop products and hardware designs with little use of open source hardware or based on current closed systems then it is likely that the current players will see their positions unchallenged. Also the utilization of Open Source Hardware by Lead Users can be used as an indication that Open Source Hardware can provide a solution space not available through current existing models. Von Hippel also explained "high rates of user innovation suggest that users may want custom products. Von Hippel also explained that if individual users or firms want something different in a product type, it is said that heterogeneity of user need for that product type is high. Open source hardware will be analyze in this context to see if it can be useful in providing a better custom solutions in some particular markets than those provided by hardware commonly available.

For example can a user generate custom solution based on Open Source Hardware better than that available by an established company. Will users and developers be willing to spend resources and time working on a custom solution to obtain a custom solution at a particular point in the value chain and will they be willing to open source the design.

By having access to the hardware documentation schematics and diagrams users can gain insight into a particular subsystem or product and modify its characteristics to better suit their needs. Apache web server is an example of an open source software product whose specific characteristics and configuration allowed users to modify and develop web server security not available through other software systems, as described by Von Hippel. Many adopted Apache since they could modify it better to suit their needs. The result of this activity has pushed the open source based Apache server as the preferred vehicle for developing web server software. The question remains if the same can be said for an Open Source Hardware system.

Some trends such as the Open hardware Foundation were formed to meet the efforts of the Open Graphics Project whose aim is to provide amongst one of its charters the creation of a Open Source Graphics chip set which would indicate that there are enough needs unmet by current Graphic Chip sets that would make users want to generate their own graphics integrated circuit architecture and develop their own boards and circuits. This stands in stark contrast to other very active groups such as the General Purpose computation on Graphics Processing Units. This group was formed to develop software and applications based on Graphic processing units, which tend to achieve processing speed gains of orders of magnitude in computation versus available approaches. This group is formed of lead users that having experienced unmet needs in terms of the computational speed and processing provided by standard CPUs are looking for new innovative solutions through the use of graphic processing units. The software developers have remained tied to established graphic chip sets such as the NVIDIA CUDA. This is one example of where an established graphic chip sets provide a solution not seen by standard CPUs, however there is only a shift in where the users will obtain their hardware they will trade standard CPU's for Graphic Chip sets but they will not invest heavily on new platforms. In summary the users will shift to purchasing a different type of processing IC and innovate by utilizing it in a different way but do not feel the need to seek out an open source hardware platform.

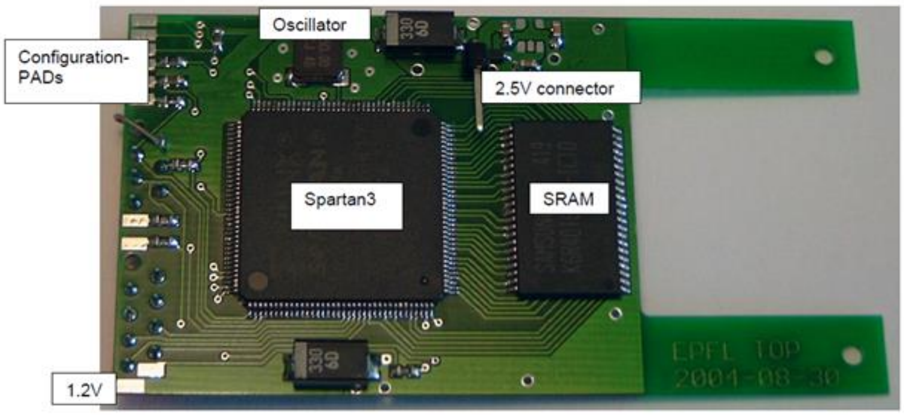

Figure 1.2 FPGA board top

Information Stickiness and users low cost innovation niches will be looked for as potential factors affecting the use of Open Source Hardware. Von Hippel explained that the term of information stickiness is a measure of how costly is it to transfer information from place to place16. The stickiness of information is defined as the incremental expenditure required in transferring a unit of information to a specific location in a form usable by a specific seeker. If the expenditure is low information stickiness is low, when the expenditure is high stickiness is high. Information stickiness can have a large impact on the use of Open Source Hardware. The cost required to transfer information in Open Source Hardware can vary greatly. In cases where schematics, integrated circuits and software is needed to recreate an Open Source Hardware solution the user must be able to access not only the design information but also the manufacturing requirements. There are cases where schematics are not enough to recreate a circuit board and a user must have access to board information such as which signals should be built with tighter constraints and which cases can have relaxed constraints. As an example a high speed Gigabit interface running at 1000 mb/second is more challenging in terms manufacturing than an IC interface running at 100 kb/sec. Higher digital speeds require greater control in the manufacturing process than those running at slower speeds. The designer must be able to effectively communicate to the manufacturer the constraints required for a particular design. Although some newer CAD tools support encoding this type of information there are no set rules and tradeoffs in terms of the time and cost of manufacturing and the design requirements are usually made. A designer would have to measure the final implementation and write up the final constraints in order to fully communicate how a design was achieved, the designer may not have this information available since it may rest with the manufacturer or may find that there is an extra cost involved in order to capture this information [4].

This can impact the utility of Open Source Hardware designs since they may not contain all of the information required to recreate a design. Von Hippel explained that low cost innovation niches as areas that are developed as a consequence of information stickiness and that information stickiness yields to information asymmetries that cannot be erased easily or cheaply. Since hardware relays on manufacturers, information stickiness can be high in certain areas of the value chain. If that is the case then it is likely that Open Source Hardware will only impact the design aspect of the product development cycle, the building aspect will remain with the manufacturer. However if the information stickiness is low in order to manufacture certain types of hardware then it is possible that Open Source Hardware can greatly affect the manufacturing points in the cycles of product development, in these cases the manufacturing will lose some of the revenue that it can generate by providing services associated with building of the hardware.

Evidence of free revealing and reuse of open source hardware in terms of not only the of the design itself but of the whole system integration and development process. Von Hippel characterized "free revealing" proprietary information as meaning that the innovator voluntarily gives up all existing and potential intellectual property rights to that information and that all interested parties are given access to it. Open Source Hardware is based on free revealing of the source code and schematics but it is bound by licensing. The type of licensing can have an impact on whether Open Source Hardware can challenge existing value chains.

Free revealing on the manufacturing process will also be used as characteristic of possible success of Open Source Hardware. If evidence is available that such information is available it can be used as measure of Open Source Hardware activity that involves not only copying the initial design but that open source hardware is also gaining ground on the manufacturing aspect. Re-use is a good measure of value being generated by Open Source initiatives. If there is evidence that other designers are re-using the information that has been revealed it is an indication that other users are benefitting from the information that has been made available. Von Hippel emphasized that valuable forms of re-use can range from those gaining general ideas of development paths to pursue or avoid to the adoption of specific designs [2].

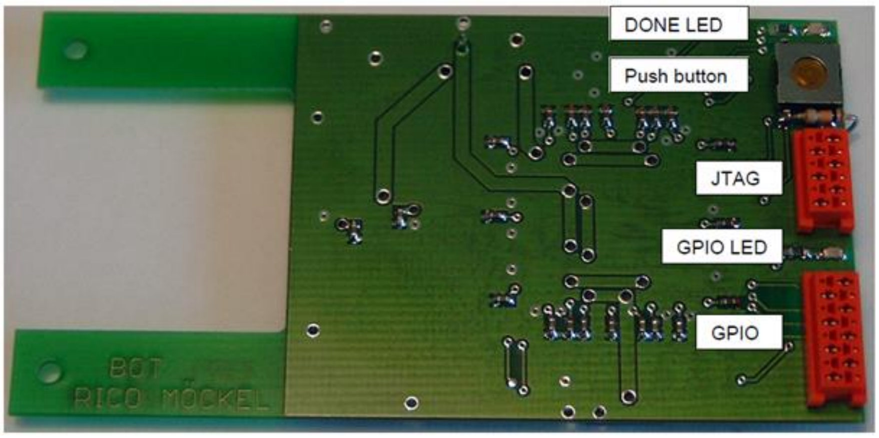

Figure 1.3 FPGA Board bottom

There may be certain cases in which Open Source Hardware architectures are available but designers do not utilize them. If there is little reuse it can be an indication that although the information is available there in no interest or that there is not enough information available to make use of the design in these cases it can mean that open source hardware designs will not compete with closed source designs.

Von Hippel defined innovation communities as nodes consisting of individuals or firms interconnected by information transfer links, which may involve face-to face, electronic, or other communication. Von Hippel also explained that innovation communities are often specialized around certain technologies and serve as collection points for information related to narrow categories of innovation. Another important value of innovation communities is that they can offer support in the form of tools and evaluation to developers by a large number of users. Innovation communities based around Open Source Hardware would be an indication that a large community of users who have embraced Open Source Hardware and are actively developing it. These communities are important because they can provide tool development and user-to-user assistance in the form of evaluation and hardware debug. This could be an indication of how widely diffused open source hardware is and how much momentum it has in certain areas of the value chain. It would also be an indication that traditional hardware development methods in which a user depends on established design houses or manufacturers for information on circuits could be changing. User to user assistance could mean that developers of certain types of hardware will see their positions challenged as users of the innovation community may increasingly rely on the knowledge of the of the innovation community rather than paying for design services from established companies. Another way that innovation communities can begin to challenge established companies is by directly developing hardware that competes with that of an established company.

For example if a company is proficient at developing certain types of designs interfaces such as high-speed USB interfaces and an Open Source Hardware innovation community begins to develop a competing implementation then the company could see some of its revenue disrupted Von Hippel explained took its as integrated sets of product design, prototyping and design testing tools [4].

Availability of high quality toolkits for open source hardware will lead higher utilization and adoption of open source hardware by designers. If designs are available for users to leverage but the users do not have a way of manufacturing, testing or building variants of the designs then access to a particular hardware design may not be useful at all. However if toolkits are available for users to develop and build and experiment with open source hardware then we could see the rapid adoption and development of open source hardware. Traditional design houses could see some of their work be replaced by open source hardware developers since most of their activity would now be shifted to supporting the manufacturing of the hardware but not the design of it.

Design houses traditionally provide assistance in design, layout, routing and material construction of PCB boards. If toolkits are available such that a user can design within certain layout and routing constraints then a design house may not be able to charge as much money for the services they provide. Another example would be the appearance of hardware platforms that allow the user to change or modify only certain portions. For example if a user wanted to test variant of a hardware implementation of an algorithm the user may need to supporting circuits and infrastructure in order to properly test his and her design. If a collection of libraries that exist but are not open source the developer may choose to utilize this path since access to the existing circuits and infrastructure will greatly reduce his development time. However if a large collection of supporting libraries of open source hardware is available for the user to build upon the designer may choose this development path.

1.3 Open Source Hardware deployment process and steps of integration

At present, there's clearly the whiff of prototype regarding these products--except for the Open-Flow compatible switches designed to work within a single vendor's SDN ecosystem.

Unfortunately, actual data center networks aren't homogeneous, and no one is ready to forklift upgrade to a single vendor's SDN architecture. That's the scenario the Open Networking Foundation, caretaker of the Open-Flow standard, seeks to obviate, but there's still plenty of work left in erasing the gray areas in defined standards and filling the gaps between compliant products and functioning, heterogeneous network.

The demo showed on-premise applications dynamically requesting new VMs and network resources from a cloud service using Open-Stack APIs and Open-Flow. Blue Planet proxies the request, does all the work necessary to spin up new VMs, virtual interfaces and other network services, and works with servers and various network devices to allocate the required capacity.

It sounds like the classic Open-Stack or Open-Flow controller example, but what set this demo apart was that it incorporated components from five other vendors: Accedian Networks' MetroNID carrier Ethernet endpoint, Arista switches, Canonical Ubuntu Open Stack platform, Overture Networks' carrier Ethernet aggregation switch and the NTT RYU open-source Open Flow controller. These five are joined by Boundary (cloud application management) and Embrane (virtual network appliances) to form what Cyan calls its Blue Orbit SDN ecosystem. Aside from the number of vendors involved, the fact that the demo included a carrier Ethernet WAN made it particularly interesting [1].

If you've never heard of Cyan, you're not alone. The company just went public this spring, and, according to its registration statement, generates "substantially all" of its revenue from a line of optical switches sold to carriers, primarily Wind stream, which accounted for 45% of the firm's sales last year.

The company has been building custom-developed software for its switches since day one, but decided last summer to refocus its software efforts on SDN. Cyan CTO Steve West says carriers, which he correctly notes don't just operate WANs and wireless networks but also large data centers, are likely to remain the company's biggest customer segment. The company also targets Web firms and large financial institutions for its SDN ecosystem.

Juniper also used Interop Tokyo as a venue to announce a couple significant software automation improvements that will undoubtedly be incorporated into its SDN strategy, first announced at its Partner Conference back in January.

The highlights include support for Puppet Labs' orchestration product, enabling Puppet to manage the firm's EX, MX and QFX switches, and OpenStack integration in Juniper’s EX, QFX and QFabric switches and Contrail SDN controller. The upshot is that Juniper can now expose both physical and virtual networks as OpenStack Quantum resources.

Another recent SDN interoperability development wasn't at Interop, but at the Open Day light Foundation's Hack Fest, where 65 developers from over 25 companies came out to work on specifications and write code.

While the event focused on low-level details like Open Flow support, threading models in the API and ideas for building virtual tenant networks, perhaps the most important accomplishment is the sense of community and cross pollination of ideas that happens simply by getting a diverse group of engineers and developers together. While it's far too early to declare victory, SDN proponents have reasons to be optimistic that the technology will be vendor- and hardware-agnostic, and will do for virtual network resources what Linux and Open Stack have done for computer servers [3].

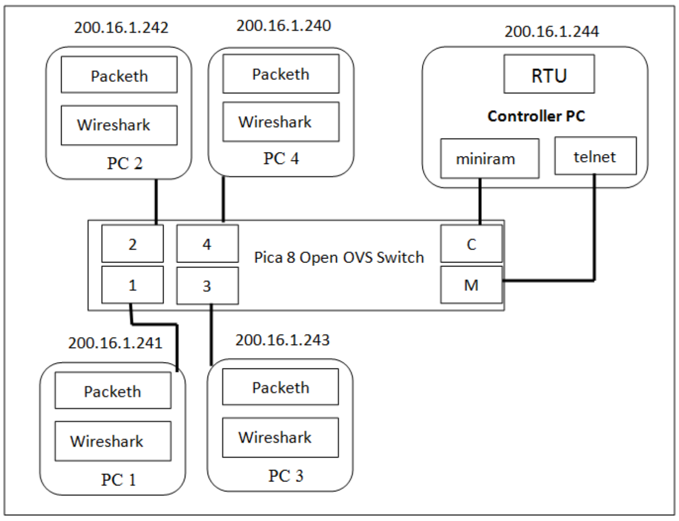

Figure 1.4 First Integrated Open OVS Switch and Open Flow Controller

In Matthew Palmer's blog "*v-Switch is the New Battleground. What Every Datacenter Operator Must Know ",* he made a point that "there is an Open-Source vSwitch, but no 'Open' vSwitch". Matt's concern is *"... What's different from these (open source) projects and OVS is the core contributions and governance for OVS is controlled by a single company: Nicira".*While we disagree OVS is controlled by a single company. Nicira, we recognize this could be many people's concern. No datacenter operator wants to be locked in.

At Pica, we are addressing this concern with an open platform that enables the open community to innovate on both the device AND the controller of Open Flow. Our goal is to enable the innovation and to build a complete platform that allows innovators to deploy applications quickly, just like Android.

This open platform includes the open OVS running on an open switch based on commercial silicones. At the controller side, we have integrated RYU, an open source OpenFlow controller, supported by NTT, Pica, and the open communities. With Pica8 supporting and integrating RYU and open OVS, these two SDN components will be synchronized and fully inter-operable to facilitate the application development.

Our goal is to move OpenFlow from labs to deployment. We want to involve the open source community to address and solve many deployment and real data center issues. The availability of a production ready hardware accelerated Open OVS switch along with a supported open source controller will open up the applications, just like Android enabling the innovations of mobile Apps.

Starting a SDN development and deployment is a daunting task. It is difficult to identify controller options, device capabilities and performance, and inter-operability between the controller and device. We want to make this easy. By providing RYU and OVS, both with open-source implementations, we want to enable users to innovate and customize their network [2].

We are publishing a trilogy to show users how to step-by-step set up the test environment and insert the first flow into their OVS switch. The three documents include:

- Open SDN Technical Reference – Pica8 OVS Power On and Configuration;
- Open SDN Technical Reference – Pica8 OVS Flow Configuration;
- Open SDN Technical Reference – RYU and Pica8 OVS Integration;

Besides the detailed step-by-step instruction, we also provide reference to several awesome open-source tools, such as packet generator and sniffer, to help users verify their applications. After going through these three technical guide, you should be able to bring up RYU and OVS, insert unidirectional flow, peer-to-many flows, and many-to-peer flows. The purpose of the documentation is to enable anyone with minimum Open Flow knowledge to configure the switch for the same scenarios in a couple of days. Once you have done the scenarios, you should be able to start exploring the switch and controller on your own.

SUMMARY

The presented chapter above is considering several aspect of Open Hardware Systems (OHS) emergence and there were made brief analysis in integral features of the earliest hardware compatibility. Needless to say, new concepts of inter-systems adaptation are urging network designer to create hard and soft based solution in order to overcome network complex diligence. Hardware compatibility quality can provide many opportunities in terms of upgrading overall network performance and the main exploited approach on fulfilling this could be only by open hardware systems.

2. OPEN SOURCE HARDWARE ELEMENTS IN CONROLLING NETWORK COMMUNICATION AND DEVICE INTERCONNECTION

2.1 Overview of network interfaces and interface designing aspects (PCI or NIC interface case)

This chapter provides a background to network interfaces. The traditional architectures of NICs and their operations are described in this chapter. Then the theoretical Gigabit Ethernet throughput and the throughputs of existing NICs are compared and discussed. Also, the implementation tradeoffs between programmable NICs and application specific NICs are compared. This fact needs to be considered that there is no research on designing an FPGA-based NIC with or without DRAM yet. Exiting programmable Network interfaces use software-based processors and most of them use SRAMs as a local memory. However, some previous related works on programmable network interfaces and some FPGA-based network interfaces are explored in this chapter.

Network interface cards allow the operating system to send and receive packets through the main memory to the network. The operating system stores and retrieves data from the main memory and communicates with the NIC over the local interconnect, usually a peripheral component interconnect bus (PCI). Most NICs have a PCI hardware interface to the host server, use a device driver to communicate with the operating system and use local receive and transmit storage buffers. NICs typically have a direct memory access (DMA) engine to transfer data between host memory and the network interface memory. In addition, NICs include a medium access control (MAC) unit to implement the link level protocol for the underlying network such as Ethernet, and use a signal processing hardware to implement the physical (PHY) layer defined in the network. The steps for sending packets from the main memory to the network are shown in Figure 2.1(A). To send packets, the host processor first instructs the NIC to transfer packets from the main memory through a programmed I/O. The NIC initiates DMA transfers to move packets from the main memory to the local memory. Packets need to be buffered in the Buffer-TX, waiting for the MAC to allow transmission. Once the packet transfer to local memory is complete, the NIC sends the packet out to the network through its MAC unit. Finally, the NIC informs the host operating system that the packet has been sent over the network.

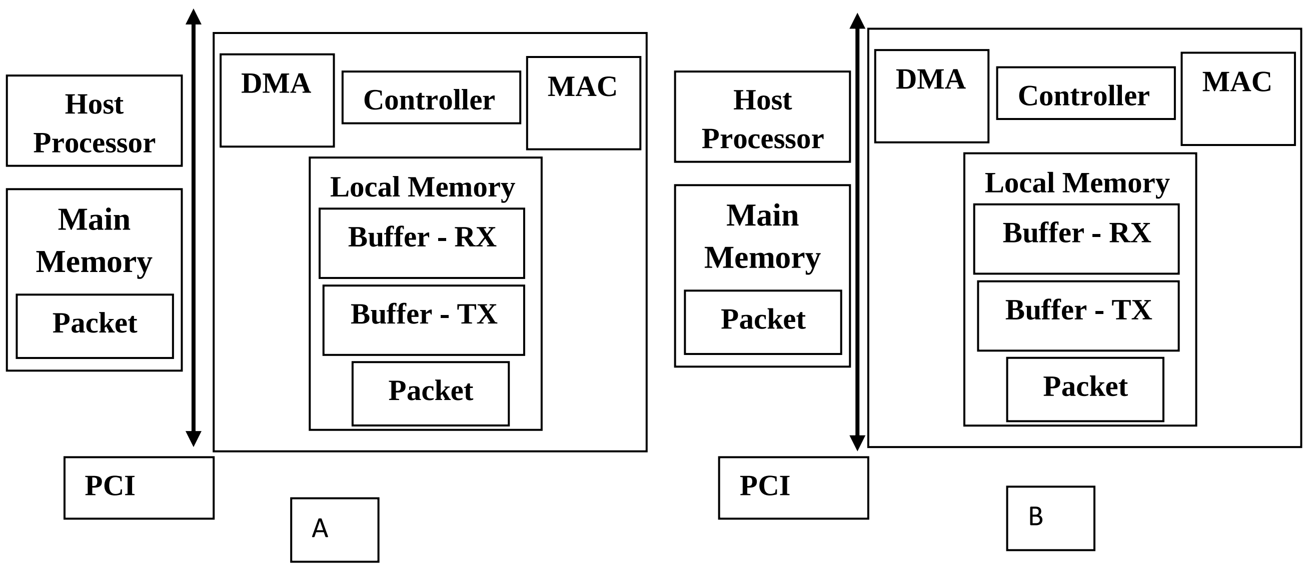

Figure 2.1 (A) Steps for sending packets to the network, (B) Steps for receiving packets from the network

The steps for receiving packets from the network are shown in Figure 2.1(B). A packet, which has arrived from the network, is received by the MAC unit and stored in the Buffer-RX. The NIC initiates DMA transfers to send the packet from local memory to the main memory. Finally, when the packet is stored in main memory the NIC notifies the host operating system about the new packet in the main memory. Mailboxes are locations, which are used to facilitate communication between host processors and network interfaces [6], [7].

Typically, these locations are written by one processor and cause an interrupt to the other processor or controller in the NIC. The value written may or may not have any significance. The NIC can map some of its local memory onto the PCI bus, which functions as a mailbox file. Therefore, these mailboxes are accessible by both host processor and the controller in the NIC. The serial Gigabit Ethernet transmit and receive interfaces can communicate with the controller in the NIC by using descriptors. For transmit packets, the descriptors contain the starting address and length of each packet. For receive packets the descriptors contain the starting address and length of the packet as well as indication of any unusual or error events detected during the reception of the packet.

In a programmable NIC, a programmable processor controls the steps shown in Figure 2.1 (A) and (B), while fixed state machines control these steps in ASIC-based NICs. A programmable NIC provides the flexibility to implement new services in order to improve and modify the functionality of the NIC. With a programmable network interface, it is easy to implement value-added network services like iSCSI or network interface data caching, which substantially can improve networking server performance. iSCSI provides SCSI-like access to remote storage using the traditional TCP/IP protocols as their transport layer. iSCSI has emerged as a popular protocol for network storage servers recently. By using inexpensive and ubiquitous TCP/IP network rather than expensive proprietary application-specific networks, iSCSI is gaining popularity.

Several iSCSI adapters aim to improve storage server performance by offloading the complete implementation of the TCP/IP protocols or a subset of the protocol processing on the adapters. In addition, previous research shows that network interface data caching is a novel network service on programmable network interfaces to exploit their storage capacity as well as computation power in order to alleviate local interconnect bottleneck. By storing frequently requested files in this cache, the server will not need to send those files across the interconnect for each request. This technique allows frequently requested content to be cached directly on the network interface. Experimental results indicate that using a 16MB DRAM or less on the network interface can effectively reduce local interconnect traffic which substantially improves server performance.

Such new services may be significantly more complex than the existing services and it is costly to implement and maintain them in non-programmable ASIC-based NICs with a fixed architecture. However, existing programmable network interfaces do not provide enough computational power and memory capacity to implement these services efficiently [7], [11].

Programmable processors provide flexibility by running some functions as software, but that flexibility comes at the expense of lower performance and higher power consumption than with ASICs. In a programmable processor, there are several steps for the execution of the instruction including Instruction Fetching, Decoding, Execution, Memory access and Write back to registers which slows down the processor performance.

This section explains some previous work on programmable NICs. Previous researches have focused on developing software-based NICs. Many of these platforms use standard microprocessors, like the Intel Pentium, AMD Athlon, or the Motorla/IBM PowerPC. Myrinet LANai is a very popular programmable NIC, which is based on a system area network, called Myrinet. The block diagram of the Myrinet-2000-Fiber/PCI interface is shown in Figure 2.2.

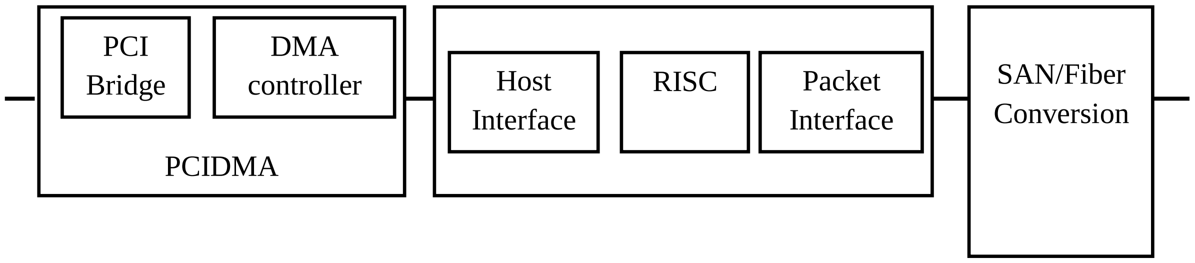

Figure 2.2 Block diagram of M3F-PCI64, universal, 64/32-bit, 66/33MHz, Myrinet-2000-Fiber/PCI interface

The interface processor includes LANai RISC operating at up to 133MHz for the PCI64B interfaces, or at up to 200MHz for the PCI64C interfaces. The on-board memory is a fast SRAM, ranging from 2MB up to 8MB depending on the revision of the board. The local memory operates from the same clock as the RISC, at up to 133 MHz for the PCI64B interfaces, or at up to 200MHz for the PCI64C interfaces. The PCI-bus interface, 66/33MHz, is capable of PCI data rates approaching the limits of the PCI bus (528 MB/s for 64-bit, 66MHz; 264 MB/s for 64-bit, 33MHz or 32-bit, 66MHz; 132 MB/s for 32-bit, 33MHz). However, the data rate to/from system memory will depend upon the host's memory and PCI-bus implementation.

The DMA controller allows multiple DMA transfers in PCI interface. A few projects used these programmable network interfaces ranging from implementing various optimizations for basic send and receive processing to implementing all or parts of network protocol processing on the network interface.

The Tigon is another representative programmable Ethernet controller that provides several mechanisms to support parallel and event-driven firmware. Figure 2.3 shows a block diagram of the Tigon controller [8].

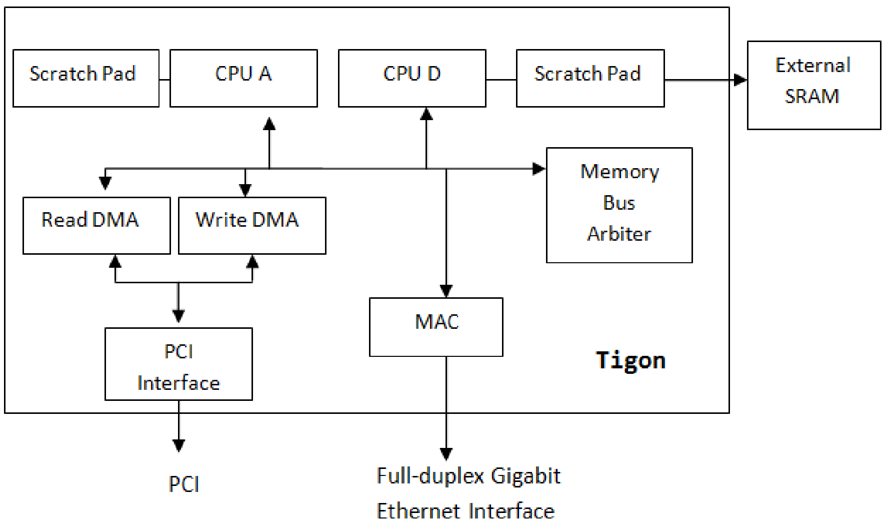

Figure 2.3 Block diagram of Tigon controller

The Tigon has two 88 MHz single issue, in-order embedded processors that are based on the MIPS R4000. The processors share access to external SRAM. Each processor also has a private on-chip scratch pad memory, which serves as a low-latency software-managed cache. Depending on the Tigon revision, the SRAM size varies from 2Kbyte to 16Kbyte for each processor. The Com 710024 Gigabit Ethernet network interface cards are based on the Tigon controller and have 1 MB of external SRAM. The Tigon programmable Ethernet controller supports a PCI host interface at 33MHz or 66MHz and a full-duplex Gigabit Ethernet interface.

Kim showed in that caching data directly on a programmable network interface reduces local interconnect traffic on networking servers by eliminating repeated transfers of frequently-requested content. Experimental results on Tigon-based NICs indicate that the prototype implementation of network interface data caching reduces PCI bus traffic by 35–58% on four web workloads with only 16MB cache, which was prototyped on an external DRAM memory. The Quadrics network uses programmable network interfaces called Elan to support specialized services for high performance clusters. Like Myrinet, the Quadrics network provides high performance messaging system using special-purpose network interfaces, switches, and links. However, the Quadrics network provides much greater transmission bandwidth of 400MBps as opposed to 250MBps of Myrinet.

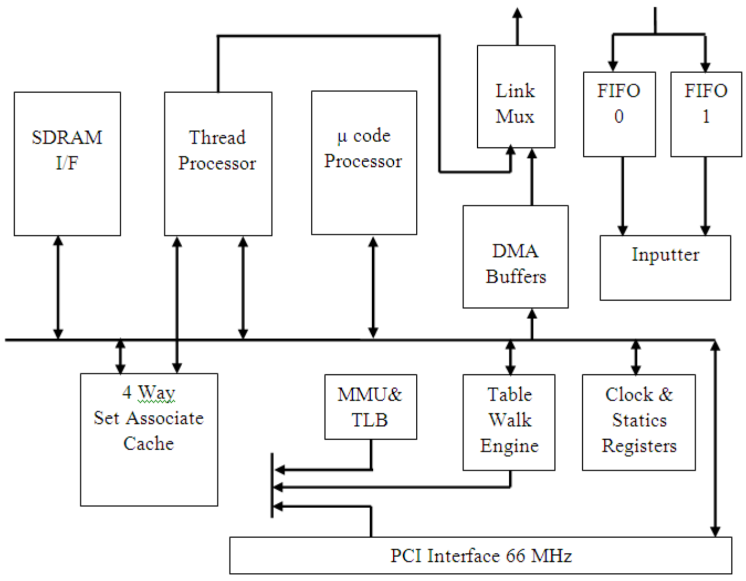

Figure 2.4 Block diagram of Elan functional units in Quadrics NIC

Figure 2.4 shows the functional blocks in Elan network interface. As shown in the figure, the Elan network interface includes two programmable processors (one as microcode processor and the other as a thread processor). The 32-bit microcode processor supports four separate threads of execution, where each thread can independently issue pipelined memory requests to the memory system. The thread processor is a 32-bit RISC processor used to aid the implementation of higher-level messaging libraries without explicit intervention from the main CPU [11].

The Elan network interface includes 8 KB cache, 64 MB on-board SDRAM, and a memory management unit. The memory management unit can translate virtual addresses into either local SDRAM physical addresses or 48-bit PCI physical addresses. While these software-based systems have outstanding flexibility, their packet processing functionality is still limited due to the sequential nature of executing instructions in software. Next section presents the throughput performance in existing NICs.

The Ethernet protocol supports frame sizes from 64 to 1518 bytes in length. In addition to payload data, each frame contains control information used to delimit, route and verify the frame. This extra information consumes a portion of the available bandwidth and reduces the throughput of the Ethernet. Figure 2.5 shows the Ethernet frame format. As shown in the figure, the preamble (64bits) and inter-frame gap (96bits) are appended for each frame, which prevent full utilization of 1Gbps for data. When the frame size is larger, more of the available bandwidth is used. As the frame size increases, the 20 bytes of inter-frame gap and preamble become less significant. An example is given here to show how the inter-frame gap and preamble affect on Ethernet throughput. For this example, assume the UDP packet size is 18 bytes (leading to minimum-sized 64-byte Ethernet frames after accounting for 20 bytes of IP headers, 8 bytes of UDP headers, 14 bytes of Ethernet headers, and 4 bytes of Ethernet CRC).

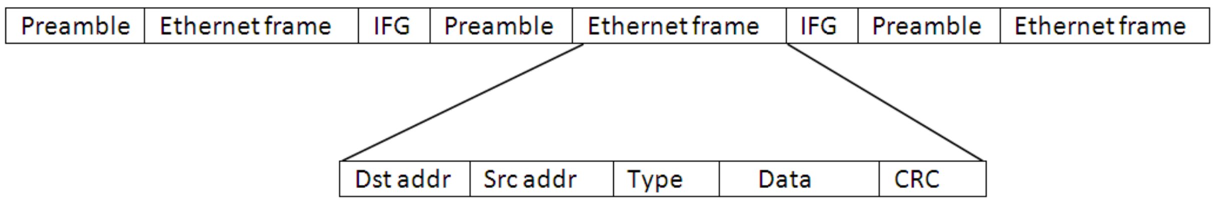

Figure 2.5 Ethernet Frame Format with Preamble and Inter-frame gap (IFG)

Knowing that there are 8 bytes of preamble and 12 bytes of inter-frame gap for each Ethernet frame, the maximum theoretical throughput is calculated as the following:

Ethernet frame size (including preamble and inter-frame gap) is 84 (64+8+12) bytes.

The number of frames is:

$$\frac{1000Mbps}{8bytses \times 8} = 14880.000 Frames/s$$

Formula 2.1

Therefore the maximum theoretical throughput for minimum-sized packet is:

1488.000x8x8=761Mbps

Lost bandwidth due to preamble is:

1488.000x8x8=95Mbps

Lost bandwidth due to Inter-frame gap is:

144.000x12x8=142.8 Mbps

The theoretical throughput for other packet sizes is calculated based on the above calculations. Figure 2.6 illustrates the theoretical bidirectional UDP throughput and the throughput achieved by existing programmable Tigon-based NIC, COM 710024, and the non-programmable IntelPRO/1000MT server Gigabit Ethernet adapter. Note that the Ethernet limit is now doubled since the network links are full-duplex. The X axis shows UDP datagram sizes varying from 18 bytes (leading to minimum-sized 64-byte Ethernet frames) to 1472 bytes (leading to maximum-sized 1518-byte Ethernet frames). The Y axis shows throughput in megabits per second of UDP datagram [12].

The Ethernet limit curve represents the theoretical maximum data throughput, which can be calculated for each datagram size based on the above example. The Tigon-PARALLEL curve shows the throughput achieved by the parallelized firmware in Com 710024 Gigabit Ethernet NIC, which uses existing Tigon programmable Ethernet controller. According Kim’s results the PARALLEL implementation, which uses both on-chip processors in Tigon, delivers the best performance. The Intel curve shows the throughput by the Intel PRO/1000 MT server Gigabit Ethernet adapter, which is a commonly used nonprogrammable NIC. The Intel PRO/1000MT server adapter is one representative of a non-programmable Gigabit Ethernet adapter.

The Intel NIC supports 64-bit or 32-bit PCI-X 1.0 or PCI 2.2 buses and a full-duplex Gigabit Ethernet interface. The controller processor is Intel 82545EM which integrates the Gigabit MAC design and the physical layer circuitry. The controller operates up to 133MHz and it has a 64Kbyte-integrated memory. The Intel NIC saturates Ethernet with 400-byte. With 1472-byte datagram, the Intel NIC achieves 1882 Mbps, close to the Ethernet limit of 1914 Mbps, whereas PARALLEL achieves 1553 Mbps. Note that in Figure 2.6, as the datagram size decreases in both NICs, throughput diverges from the Ethernet limit. This is a big issue in exiting NICs which can not send or receive small packets at gigabit Ethernet line rate. The decrease in throughput indicates that the controllers on both NICs can not handle packet processing when the frame rate increases. The other reason can be the limited amount of buffering memory in the NICs, which was discussed below.

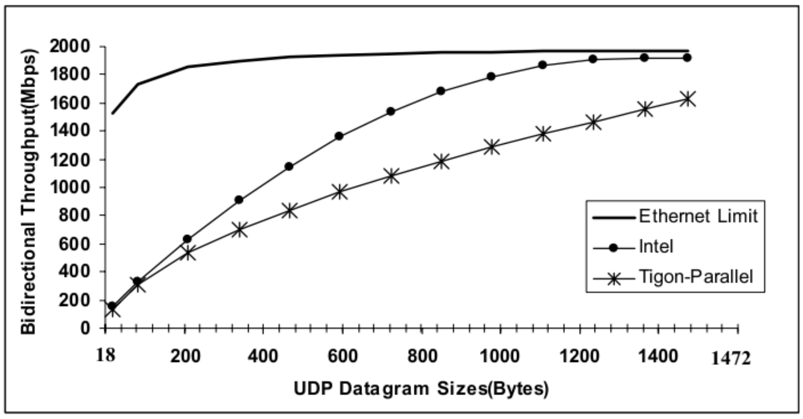

Figure 2.6 Theoretical bidirectional Gigabit Ethernet maximum throughput achieved in existing programmable Tigon-based NIC and non-programmable Intel PRO/1000MT NIC

In Intel, the throughput starts decreasing beginning at 1100-byte datagram, whereas in PARALLEL, throughput starts decreasing linearly beginning at 1400-byte datagram. The linear decrease indicates that the firmware handles a constant rate of packets regardless of the datagram size, and that the processors in the Tigon are saturated [16].

Field programmable gate arrays (FPGA) fill the gap between custom, high-speed and low power ASICs and flexible, lower-speed and higher-power microprocessors. An FPGA is a type of programmable device, that has evolved from earlier programmable devices such as the PROM (Programmable Read-only memory), the PLD (Programmable logic device), and the MPGA (Mask Programmable Gate Array). An FPGA is an IC (integrated circuit) consisting of an array of programmable cells. Each cell can be configured to program any logic function. The logic cells are connected using configurable, electrically erasable static-random-access memory (SRAM) cells that change the FPGA's interconnection structure. Just as a software program determines the functions executed by microprocessor, the configuration of FPGA determined its functionality.

Novel functions can be programmed into FPGA by downloading a new configuration, similar to the way a microprocessor can be reprogrammed by downloading a new software code. However, in contrast to microprocessor's functions, the FPGA runs in hardware. Therefore, there is no software's relatively instruction overhead, which results in higher speed and lower power dissipation for FPGAs than microprocessors.

FPGAs provide a new approach to Application Specific Integrated Circuit (ASIC) implementation that features both large scale integration and user programmability. Short turnaround time and low manufacturing cost have made FPGA technology popular for rapid system prototyping and low to medium-volume production. FPGA platforms are easily configurable, and with a growing number of hard cores for memory, clock management, and I/O, combined with flexible soft cores, FPGAs provide a highly flexible platform for developing programmable network interfaces. Although ASIC design is compact and less expensive when the product volume is large, it is not easy to configure at the stage of prototyping. FPGA provides hardware programmability and the flexibility to study several new designs in hardware architectures. It can easily achieve the concept of system-on-chip (SOC) with hardware configuration. When the design is mature, FPGA design can be easily converted to system on a chip for mass production. The increase in complexity of FPGAs in recent years has made it possible to implement more complex hardware systems that once required application-specific integrated. Current devices provide 200 to 300 thousand logic gate equivalents plus 100 to 200 thousand bits of static RAM [15].

Although FPGAs deliver higher performance than programmable processors, they still have a lot of setup overhead compared to ASICs, requiring as many as 20 transistors to accomplish what an ASIC does with one. This adds latency and increases the power consumption in the design. In addition, FPGAs are more sensitive to coding styles and design practices. In many cases, slight modification in coding practices can improve the system performance from 10% to 100%. Thus, for complex and high-speed designs, current FPGAs are still slow and power-hungry and they come with high transistor overhead. This makes FPGA designers face with additional challenges of architectures to meet difficult performance goals and different implementation strategies.

2.2 Board Design and layout implementation

Designing the FPGA-based PCI/Gigabit Ethernet NIC proposed in this thesis involves additional issues, which govern the decision for board design and affect on choosing the components of the NIC. The key issues are maximum power limit in PCI card, power consumption of the on-board components, maximum achievable bandwidth for each component, clock distribution and management in different clock domains, power distribution system in the FPGAs and high-speed signalling constraints in layout implementation.

The FPGA-based Gigabit Ethernet/PCI Network Interface Card (NIC) is designed to provide either a full or a half-duplex Gigabit Ethernet interface. The NIC connects the PCI-compliant server to a Gigabit Ethernet network and is implemented for use in systems that support either the 64 or 32 bit wide PCI Local

Bus operating at 66 or 33 MHz. The block diagram of the NIC is shown in Figure 2.7.

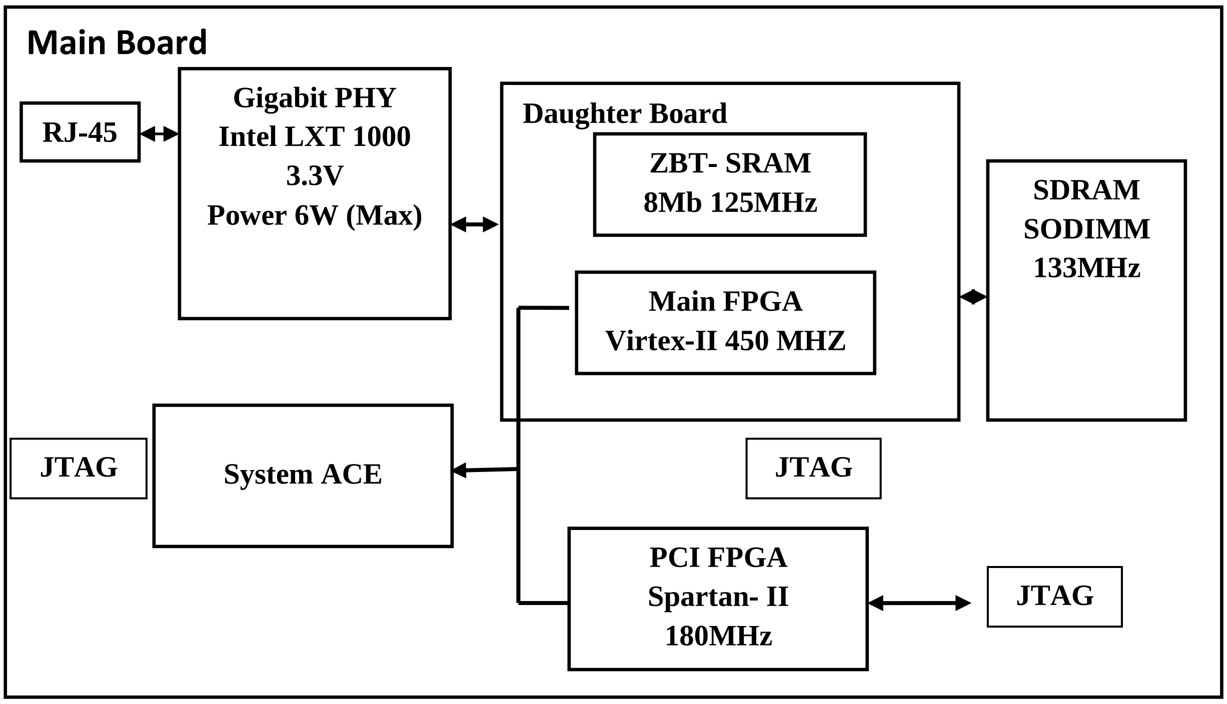

Figure 2.7 FPGA-based Gigabit Ethernet/PCI NIC block diagram

As shown in the figure, the NIC is designed on two separate custom boards, named main-board and daughter-board. The reason for having two separate boards is to provide more flexibility in the board layout implementation. The combination of the main-board and daughter-board functions as a Gigabit Ethernet/PCI network interface card. The NIC features two Xilinx FPGAs. The Virtex-II FPGA (main FPGA) is the focal point in the board design and is used to implement a programmable network processor, a Gigabit Ethernet MAC controller, memory controllers for ZBT SRAM and SDRAM SO-DIMM, a DMA controller in PCI interface and asynchronous FIFOs. The Virtex-II FPGA architecture has been optimized for high-speed designs with low power consumption. The Virtex-II FPGAs are capable of supporting high-density designs up to 10 million gates. The XC2V2000 part, which is used in this design, has 2 million gates and maximum available 896 IO pins. These devices have 16 global clock buffers and support 16 global clock domains, with a maximum internal clocking speed of 420 MHz [17].

This feature allows the implementation of multiple clock domains in the FPGA, which will be discussed below. Additional features include 8 DCMs (Digital Clock Manager) for generating desk ewe clocks, as well as fine-grained phase shifting for high resolution phase adjustments in increments of 1/256 of the clock period. DCI (Digitally Controlled Impedance) technology is a new feature in Virtex-II FPGA, which provides controlled impedance drivers and on-chip termination. The use of Virtex-II's DCI feature simplifies board layout design, eliminates the need for external resistors, and improves signal integrity in the NIC design.

The Spartan-II FPGA (PCI FPGA) is used to provide the interface between the PCI and the main FPGA. The functionality of the PCI bus interface is fixed and well defined. For this reason, the Xilinx PCI bus interface IP core is used for PCI interface. However, a user interface design is implemented which connects the PCI core interface to the rest of the system in Virtex-II FPGA. The NIC is designed as a universal card, which means that it can support both 5V and 3V signalling environments depending upon whether it is plugged into a 3V or 5V slot. The main reason for implementing the PCI interface in the dedicated Spartan-II FPGA is that the Xilinx PCI core supports only a few numbers of FPGAs, including Spartan-II. It does not support Virtex-II with the same package number used in the board design. In addition, Spartan-II devices are both 3.3Volt and 5Volt tolerant, while Virtex-II devices are only 3.3 volt tolerant. Using Spartan-II as the PCI interface eliminates the need for external bus switches in order to interface with Virtex-II, in a 5Volt signalling environment.

As shown in Figure 2.7, the NIC features two types of volatile memory. A pipelined ZBT (Zero Bus Turnaround) SRAM device, integrated in the daughter-board, is used for the low latency memory to allow the network processor to access the code. ZBT SRAM devices are synchronous SRAMs can be used without any dead cycles, even when transitioning from READ to WRITE. The ZBT SRAM with a speed grade of 133 MHz has access latency of 5 cycles and has a 32-bit data bus width interface with Virtex-II FPGA, which provides low latency access for the processor [11].

The SDRAM, 128 Mbytes SO-DIMM, is integrated as a large capacity, high bandwidth memory to store data and is used for adding future services like network interface data caching. The absolute maximum power consumption of the memory module is 4W. The SDRAM memory bus requires particular routing guidelines.

The trace lengths for all SDRAM data signals and address signals should be matched and as short as possible. The PHY chip, Intel LXT1000 Gigabit Ethernet transceiver, implements the physical layer defined in the Open System Interconnect reference module and is responsible for processing signals received from the MAC (which is implemented in the Virtex-II) to the medium, and for processing signals received from the medium for sending to the MAC. It requires a 3.3V, 1.5A power supply to all VCC balls for operation. A 25 MHz oscillator is required for the reference clock. The maximum power consumption of the PHY is 6Watts, which is significantly high and is considered in the calculation of the power drawn from the board. The PHY should be placed very close to the FPGA interface and special considerations must be given to the layout implementation of the PHY to meet the best performance.

The NIC design supports two different methods for configuring (programming) the Virtex-II and Spartan-II FPGAs. These methods include Boundary-scan and System ACE. Programming the Virtex-II via Boundary-scan is done by JTAG connector on daughter board. The JTAG connector on the main-board is for programming the Spartan-II via Boundary-scan. The iMPACT software in Xilinx ISE tools is used to program the FPGA devices by transferring bit files from the design environment to the NIC. Configuring the Virtex-II and Spartan-II FPGAs with the System ACE is another way of transferring bit files from the design environment to the board.

The system ACE family is a pre-engineered, high-density configuration solution in multiple FPGA systems. It allows managing multiple bit streams to be stored on the on-chip flash memory for immediate FPGA configuration. As shown in Figure 2.7, the main-board is designed to have system ACE MPM (multi-package module). The MPM consists of an on-chip AMD flash memory with 64 Mb densities, a non-volatile PROM and a Xilinx Virtex-E configuration controller. The Virtex-II and Spartan-II FPGAs can be configured through Select MAP mode, which is the fastest configuration mode with maximum configuration speed of 152Mb/s. As shown in Figure 2.7, three AMP Mictor connectors on the main board, compatible with Tektronix logic analyzers, provide high-speed signals that can be used for debugging the signals between SDRAM and Virtex-II or the PHY and Virtex-II FPGA. A 50-pin Logic Analyzer header connector provides access to additional pins in the Virtex-II FPGA for debugging purposes. A number of LEDs and pushbuttons are also on the board for testing purposes. The daughter-board is connected to the main board by eight dedicated connectors. The Virtex-II FPGA on the daughter-board has access to the PHY, Spartan-II FPGA, SDRAM, and the System ACE via these connectors [9].

To achieve one Gigabit per second throughput in the FPGA-based Gigabit Ethernet/PCI NIC, the major functional components in the NIC should provide minimum bandwidth of 1Gbps. These components are the on-board two FPGAs, PHY, SDRAM, SRAM and PCI interface. Table 2.1 shows the theoretical bandwidth provided by each component in the NIC. The data width and maximum operating frequency of each component are shown in the third and fourth column. The maximum theoretical bandwidth provided by each component can be calculated based on the data width and maximum operating frequency and is indicated in the last column.

Note that the bandwidth numbers in Table 2.1 are the maximum theoretical rates. However, in practice, it is not possible to achieve these rates for parts like SDRAM. In SDRAMs, due to their access latency, overlapped transfers are not possible. Depending on the DRAM access latency and the memory controller performance the maximum achievable bandwidth could be different. The maximum bandwidth is calculated.

Table 2.1

Theoretical bandwidth provided by each component in the NIC

	Data width (bits)	**Max operating Frequency (MHz)**	**Theoretical Bandwidth (Gbps)**
Gigabit Ethernet PHY	8x2	125	2.0 (full duplex)
SDRAM SODIMM	64	133	8.5
SRAM ZBT	32	133	4.2
PCI	64	66	4.2

The SDRAM SO-DIMM has the access latency of 10 cycles and has the minimum clock cycle time of 7.5 ns. The SDRAM SODIMM can support burst length of 1, 2, 4 and 8. Table 2.2 shows the actual maximum bandwidth for the SDRAM SO-DIMM with various burst length implementation. The SDRAM latency and the experimental results are discussed in below.

Table 2.2

Practical bandwidth in SDRAM SODIMM

Burst Length	**Transfer cycles**	**Number of bits per Transfer**	**Max Practical Bandwidth (Mbps)**
Burst-1	**10**	**64**	**853.3**
Burst-2	**12**	**64x2**	**1422.2**
Burst-4	**14**	**64x4**	**2438.0**

Burst-8	**18**	**64x8**	**3792.5**

SRAMs are faster than SDRAMs but there is still access time latency in read and write operations. According to the datasheet, the ZBT SRAM has access time latency and the minimum clock cycle time for this SRAM is 7.5 ns. The data width in the SRAM is 32-bits. This SRAM can only support burst lengths of 1 (single transfer) and four. Table 2.3 shows the maximum practical bandwidth for the ZBT-SRAM [18].

Table 2.3

Maximum practical bandwidth in ZBT-SRAM

Burst Length	**Transfer cycles**	**Number of bits per Transfer**	**Max Practical Bandwidth (Mbps)**
Burst-1	**5**	**32**	**853.3**
Burst-4	**9**	**32x2**	**1896.3**

Experimental results indicate the PCI bus can not deliver the theoretical maximum throughput due to overheads associated with data transfers. The Local I/O interconnect and main memory are two significant limiters in PCI transfers. Table 2.4 shows the actual maximum PCI throughput, which is almost 70% of the maximum theoretical bandwidth. These numbers are based on network interface data caching results in Kim's thesis.

Table 2.4

Maximum practical throughput in PCI traffic (Numbers are from)

Burst Length	**Max Practical Bandwidth (Mbps)**
PCI 64-bits/33 MHz	**~1250**
PCI 64-bits/66 MHz	**~1290**

As shown in the Table, the throughput in PCI 64-bits/ 66MHz is slightly better than the PCI 64-bits/33MHz.Theoretically, the PCI 64-bit/66 MHz PCI bus can provide twice bandwidth of the 64-bit/33MHz PCI bus. The use of a faster 64-bit/66 MHz PCI results in vastly increased overheads and little server improvements. In this case, the PCI bus may spend a large fraction of cycles waiting for the main memory, which results in the decrease of achieved throughput. The internal bus, which connects the MAC, PCI and memory interfaces, should provide at least one Gigabit per second bandwidth for each interface. The Virtex-II FPGA, which is used as the main FPGA for designing the Memory controller, MAC and DMA interfaces can internally operate up to 450MHz by using on-chip DCMs (Digital Clock Management). Depending on the internal bus architecture, different bandwidths are achieved. Table 2.5 shows the maximum bandwidth provided by different bus architectures in Virtex-II FPGA.

Table 2.5

Maximum bandwidth provided by different bus architectures in Virtex-II

Bus architecture	**Data width (bits)**	**Max Frequency (MHz)**	**Max Theoretical Bandwidth (Mbps)**
Custom-designed BUS	**64**	**200**	**12.8**
Power PC OPB Bus	**32**	**100**	**3.2**

The custom-designed bus was originally used for the internal system bus. Simulation results indicate that the custom-designed bus bandwidth can achieve up to 12.8 Gbps with adding pipeline stages in controller architecture. The PowerPC OPB (On-chip Peripheral Bus) bus is provided by Xilinx. As shown in Table 2.5, PowerPC OPB bus bandwidth is only 3.2 Gbps, which is not enough for each interface to achieve the maximum Gigabit throughput. The architecture design with OPB bus and the performance results are respectively discussed in below. Spartan-

II FPGA, which implements the PCI interface, should provide minimum 2Gbps throughput at the PCI bus interface for bidirectional DMA transfers [12], [9].

The total power drawn collectively from all four power-rails for the PCI card cannot exceed 25 watts. Due to this maximum PCI power limit, the maximum power consumption of the components on the main-board and daughter-board should be less than 25 watts. Otherwise, an external power supply is required to provide power to the board. The absolute power consumption of each component depends on many factors such as device architecture and manufacturer. Finding low power components, which provide the desired functionality, price and availability, is an additional issue in the board design. For example, the absolute power consumption for Micron SDRAM SODIMM varies significantly with the memory size.

The maximum power consumption of 128Mbytes SODIMM is 4Watts, whereas the maximum power consumption for 512 Mbytes SDRAM SODIMM is 16 Watts with the same architecture. Another noticeable fact is that the power consumption within an FPGA is design-dependent and can be difficult to calculate prior to implementing the design. Most other large, dense ICs (such as large microprocessors) are only designed to implement specific tasks in their hard silicon. Thus, their power supply demands are fixed and only fluctuate within a certain range.

FPGAs do not share this property. Since FPGAs can implement a practically infinite number of applications at undetermined frequencies and in multiple clock domains, it can be very complicated to predict what their transient current demands will be. To address these issues, a detailed power analysis for designs in both FPGAs and in other on-board components is required [4], [17].

2.3 Power and clock distribution management

The dynamic power consumptions of the Virtex-II and Spartan-II designs are calculated by power analysis tool XPower provided by Xilinx ISE tool. XPower calculates power based on switching activity of elements in the design. XPower determines the switching activity of elements by using design simulation results. The analysis was carried out following the synthesis, translation, mapping, netlist extraction, and the post-placement and routing phase. Extensive timing simulations were carried out in the ModelSim simulator to model true-device behaviour. All internal node transitions occurring during the course of the simulations were dumped into a ".vcd" (Value-Change-Dump) file format. The .vcd files were then analyzed by XPower.

The generated power report represents the total power consumed by the specific design. This report provides the estimated current and voltage drawn by each voltage supply in the FPGA. Table 2.5 shows the estimated power results from the XPower tool for each design part in Virtex-II FPGA. The slice utilization for each design is shown in second column. As shown in the table, the estimated power consumption is categorized to three voltage supplies in Virtex-II. The VCCINT 1.5V column displays the total power consumption from the core supply voltage (VCCINT), split to power and current drawn from each design.

Table 2.5

Power consumption in Virtex-II FPGA for each interface

Design Part	**Slices**	**Estimated power consumption**						**Total Power (mW)**
		V_{CCINT}		**V_{CCO}**		**V_{CCAUX}**		
		I (mA)	**P (mW)**	**I (mA)**	**P (mW)**	**I (mA)**	**P (mW)**	
Gigabit MAC CORE	1.437 (10%)	314	471	101	333.3	28	92.4	896.7
SDRAM Controller	259 (2%)	228	342	59	194.7	25	82.5	619.2
ZBT SRAM Controller	92 (1%)	140	210	37	122.1	15	49.5	381.6
FIFO	1003	400	600	91	300.3	27	89.1	989.4

(PCI+MAC)	(9%)							
Power PC Processor Design	1.950 (20%)	582	873	287	947.1	53	174.9	1995

The VCCAUX 3.3V column displays the power consumption from auxiliary circuits. The last row in the table indicates the estimated power consumption of PowerPC processor design. This includes the power consumption of processor cores and the processor buses. As shown in the table, the processor design consumes almost 2 watts, which is a considerable fraction of the total budget for the PCI board design. The estimated power consumption for each design is the sum of power drawn by each voltage supply and is shown in the last column. Adding the power consumption numbers in the last column results the total estimated power in Virtex-II FPGA, which is 4.8W.

Table 2.7 shows the estimated power results from the XPower tool for the PCI interface design with 66MHz and 64bits implementation in Spartan-II FPGA. As in previous table, second column shows the slice utilization for PCI interface design. Spartan-II voltage supplies include VCCINT2.5Volt, which is the core voltage, and VCCO3.3V, which is used for the 3.3V IOs. The power consumption of these voltages is shown in Table 2.7.

Table 2.7

Estimated power consumption in Spartan-II FPGA

Design Part	**Slices (% Usage)**	**Estimated Power Consumption**				**Estimated Power (mW)**
		V_{CCINT} (2.5)		**V_{CCO}(3.3)**		
		I (mA)	**P (mW)**	**I (mA)**	**P (mW)**	
PCI Core +User						

Interface (66 MHz/64bits)	632 (26%)	143	350	112	371	721

The estimated power consumption on Spartan-II FPGA is 721 mWatts. The estimated power consumption by Spartan-II and Virtex-II FPGA is used to calculate the total power of the board in the next section.

The absolute power consumption and current drawn by each component in the NIC can be found in related datasheets and are shown in Table 2.8. A fact to be noticed is that these numbers are the maximum power consumption numbers, which happen in the worst case usually at start up. Thus, in normal operation the power consumption and current drawn by the on-board components are less than the values in Table2.8.

Table 2.8

Absolute power and current drawn by main components

Part name	**Power consumption**	**Current**
Virtex-II FPGA	~4.8 W	1.008 A
Spartan-II FPGA	~0.8	0.552
Gigabit Ethernet PHY	6 W	1.8 A
System ACE	4.92 W	2.40 A
SDRAM-SODIMM	4 W	1.290 A
ZBT SRAM	1.65 W	0.5 A
Total Value	22.17 W	7.55 A

The estimated power consumption of the board is calculated by adding the numbers in table 2.8 and results in is 22.17 Watts. Total estimated current drawn by the components on main-board and daughter is 7.55A. The power results indicate that the total power consumption of the two boards does not exceed the PCI card power limit of 25W. Thus, the auxiliary power supply is not required.

However, an auxiliary power connector was added to the main board in case the total power consumption exceeds than the 25Watt PCI power limit [15], [16].

Various interfaces with different high-speed clock domains are implemented in the board. These clock domains are shown in Figure 2.7. The Gigabit Ethernet PHY requires 125 MHz as the reference clock. SDRAM and SRAM require a programmable clock ranging from 50 MHz to 133 MHz. The PCI interface, implemented on the Spartan-II FPGA, requires an input clock of 66MHz. Excessive use of multiple oscillators or clock multiplier ICs to generate the required clock for each interface is not recommended, since it adds more hardware and noise to the board. In addition, special consideration must be given to clock distribution to prevent the clock skew in the systems. Clock skew and clock delay can have a substantial impact on designs running at higher than 100 MHz. Thus to address these issues, a proper clocking management is required to generate a zero-delay and de-skewed clock for designs in the both FPGAs and the on-board component.

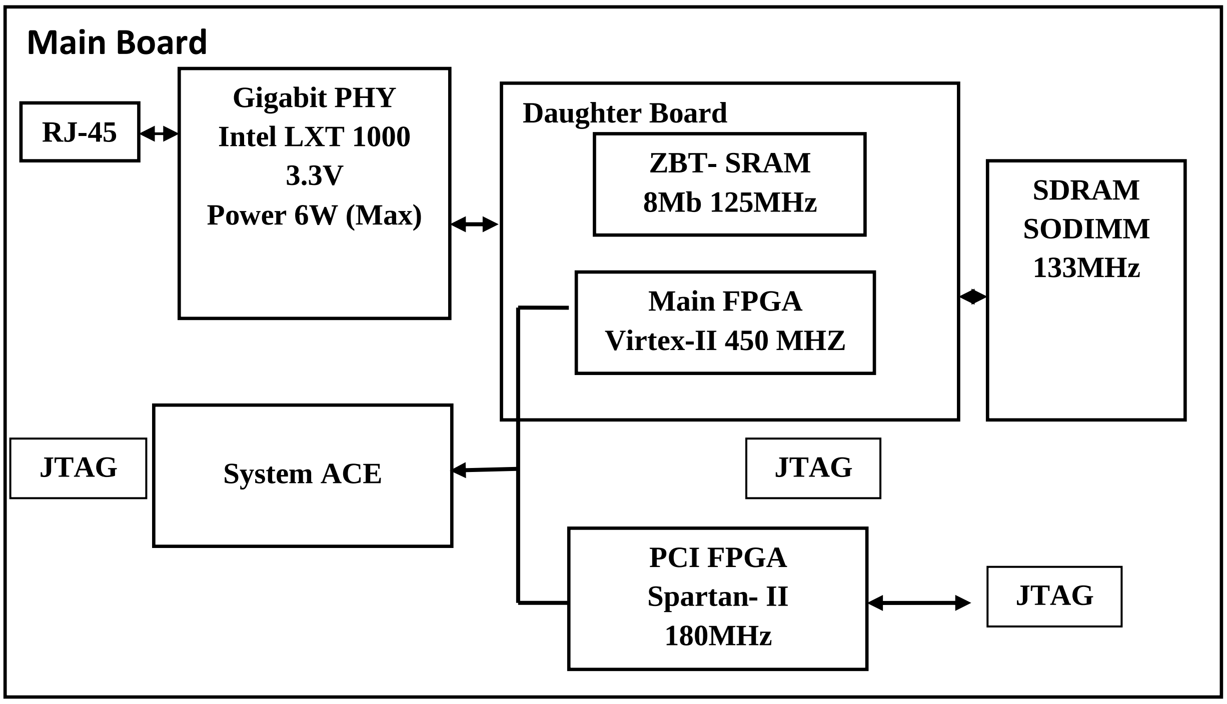

Figure 2.8 clock domains in FPGA-based Gigabit Ethernet/PCI NIC

As shown in Figure 2.8, a 50MHz external oscillator is used to provide the reference clock for the Virtex-II FPGA. All other clocks are generated using DCMs (Digitally Controlled Management) in Virtex-II. DCMs offer a wide range of powerful clock management features including clock de-skew, which eliminates the clock distribution delays. In addition, using the frequency synthesis feature, different frequency rates can be generated internally. This eliminates the need for external clock multiplier chips in the board design, and frees up the space in the board layout.

An example is given in Figure 2.9 to show the use of DCMs, which provide a de-skewed clock for the SDRAM. The input clock is 50MHz and the SDRAM clock needs to be 125MHz. As shown in the figure, DCM0 provides the CLKFX signal to the SDRAM, which is 125MHz. The frequency for CLKFX is generated by using the equation:

$$\text{FREQCLKFX} = (M/D) \times \text{FREQCLKIN} \qquad \text{Formula 2.2}$$

Setting the M equal to five and D equal to two results the 125MHz CLKFX for the SDRAM. As shown in the figure, DCM0 also receives the clock feedback from the SDRAM. DCM1 provides both CLK and CLKFX as internal clocks for the SDRAM controller in the FPGA. Therefore, a zero-delay clock for both SDRAM and SDRAM controller is generated in this way. To provide a zero delay clock for both SDRAM controller and the SDRAM, it is required to use two DCMs with the same clock input but separate clock feedbacks, which achieves zero-delay between the input clock, SDRAM controller and SDRAM clock. This design is used to generate a de-skewed clock for ZBT-SRAM and Gigabit PHY chips [14].

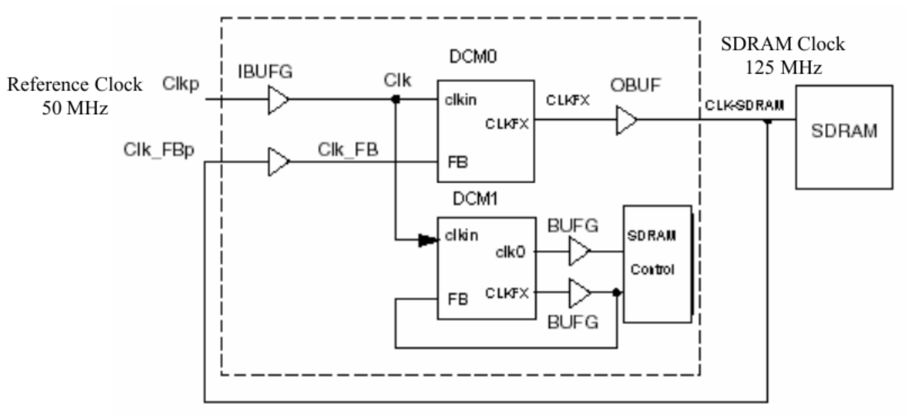

Figure 2.9 Using DCMs in providing the SDRAM clock

The PC board is no longer just a means to hold ICs in place. At today's high clock rates and fast signal transitions, the PC board performs a vital function in feeding stable supply voltages to the ICs and in maintaining signal integrity among the devices. Several issues should be considered in the board layout design. Design techniques, which were used at slower frequencies (33 MHz and lower) are no longer appropriate for these higher bus frequencies like 125MHz. More attention must be paid to board layout, bus loading, and termination to ensure that short clock cycle times can be met without noise, ringing, crosstalk or ground bounce.

In addition, it is required to design a proper power distribution system (PDS). Most other large, dense ICs (such as large microprocessors) come with very specific bypass capacitor requirements. The power supply demands for these devices are fixed and only fluctuate within a certain range. The reason is that these devices are only designed to implement specific tasks in their hard silicon. FPGAs do not share this property. FPGAs can implement a practically infinite number of applications at undetermined frequencies and in multiple clock domains. Therefore, it can be very complicated to predict what their transient current demands will be. Transient current demands in digital devices are the cause of ground bounce, and the bane of high-speed digital designs. In low-noise or high-power situations, the power supply decoupling network must be tailored very closely to these transient current needs, otherwise ground bounce and power supply noise will exceed the limits of the device [10], [11].

SUMMARY

This chapter provides a background to network interfaces. The traditional architectures of NICs and their operations are described in this chapter. Then the theoretical Gigabit Ethernet throughput and the throughputs of existing NICs are compared and discussed. Also, the implementation tradeoffs between programmable NICs and application specific NICs are compared. This fact needs to be considered that there is no research on designing an FPGA based NIC with or without DRAM yet. Exiting programmable Network interfaces use software based processors and most of them use SRAMs as a local memory. However, some previous related works on programmable network interfaces and some FPGA-based network interfaces are explored in this chapter.

As researchers vigorously develop advanced, processing schemes for programmable networks there exists a need for scalable network interface architecture capable of data processing at line speeds. Existing network interface architectures that provide sufficient flexibility employ software processing environments containing multiple Reduced Instruction Set Computer (RISC) cores and suffer from lower performance.

3. ARCHITECTURE DESIGN AND OPEN SOURCE HARDWARE (OSH) SYSTEM IMPLEMENTATION

3.1FPGA-based NIC Controlled System

A top-level block diagram of the FPGA-base NIC controller design with PowerPC bus interface is shown in Figure 3.2. The detailed block diagram of each interface is illustrated in following sections. The FGPA-based NIC controller is designed to provide a high flexibility in addition to providing the basic tasks of a network interface card. Using the flexible and configurable architecture design and additional interfaces in the NIC, new services can be implemented and can be quickly configured and tested in real hardware. As shown in the figure, different clock domains are illustrated by dashed lines in the controller design [10].

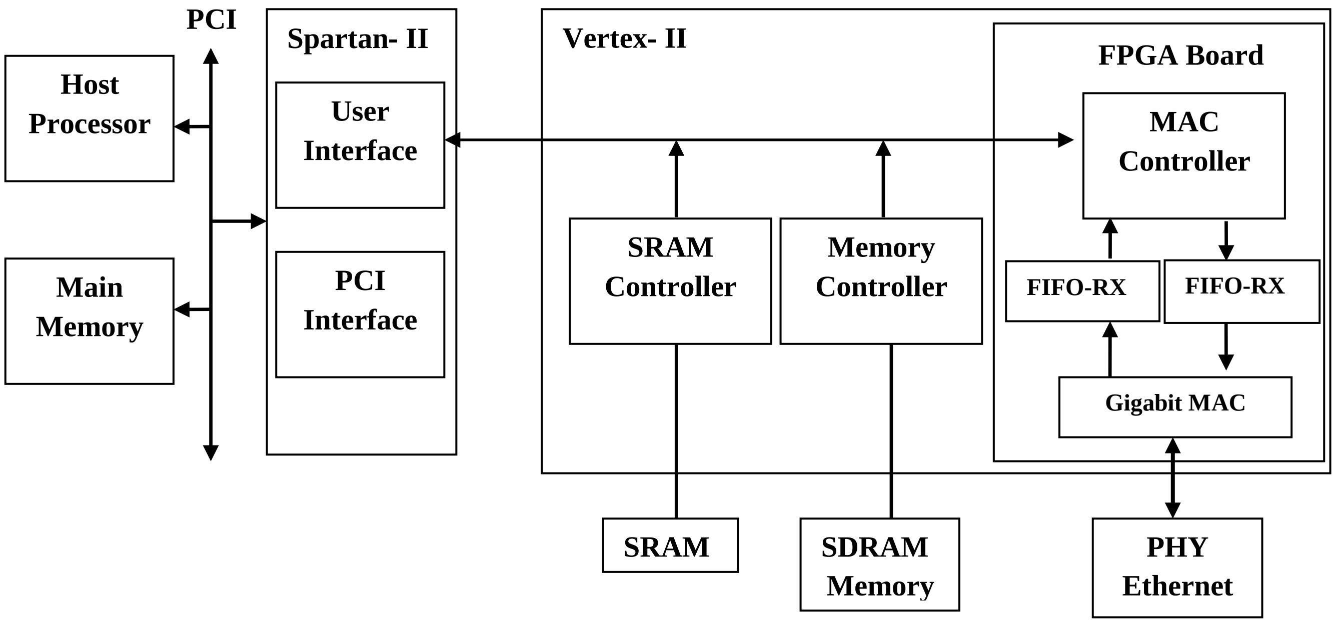

Figure 3.1 Ethernet/PCI NIC Controller Block diagram on the Avnet-board

The processor interface operates at 100MHz, which is the maximum operating frequency in PLB bus. This interface consists of embedded IBM PowerPC 405 processors in Virtex-II Pro FPGA. The software in PowerPC processor is either stored in on-chip Block RAM (BRAM) memories in Virtex-II

PRO or in the on-board SRAM memory. The OPB is used to provide the shared system bus for the processor interface, MAC controller, SRAM and SDRAM controllers and DMA interface. The Gigabit Ethernet interface operates at 125MHz, providing a 2Gbps bidirectional full duplex or 1Gbps half-duplex throughput. The main blocks in this interface include MAC controller, FIFO-RX and FIFO-TX, and a Gigabit MAC Core. Since the functionality of the Gigabit MAC is fixed, this controller uses Gigabit Ethernet MAC Core offered by Xilinx to interface with the on-board PHY chip. The MAC controller is a state machine, which manages packets transfer operation between FIFOs and OPB bus. FIFO-RX and FIFO-TX with asynchronous clocks are used to provide data synchronization and buffering between the MAC Controller operating at 100MHz and Gigabit MAC core operating at 125MHz.

The memory controller provides a 32-bit high-speed memory bus for the on-board SDRAM memory. Various implementations for the memory controller interface are investigated in this chapter. The first design considers single transfers to the SDRAM, whereas the second design improves the memory controller and OPB-slave interface to provide burst transfers to the SDRAM. The memory controller with this design is implemented for 100 or 125 MHz operation.

The 100MHz implementation is synchronous with OPB bus frequency and has a simpler architecture whereas 125MHz implementation requires DCMS and synchronizing interfaces to provide higher frequency than OPB bus frequency. The architecture for this design is more complex but provides faster read and write accesses to the SDRAM. The PCI interface operates at 66 or 33MHz with 64-bit data or 32-bit width data bus at the PCI connector. The main blocks for this interface in Virtex-II Pro FPGA are DMA controller, FIFO-RD and FIFO-WR. The DMA controller provides the DMA transfers from FIFOs to the OPB bus and operates at 100MHz.

FIFO-RD and FIFO-WR with asynchronous clocks are used to provide data synchronization and buffering between the PCI interface in Spartan-II and DMA controller in Virtex-II Pro. As mentioned earlier, since PCI protocol functionality is fixed and well-defined the Xilinx PCI core is used to provide the interface at PCI connector. The PCI core and the user interface are implemented in the dedicated Spartan-II FPGA. The user interface provides DMA transfers from the PCI interface to the FIFOs. Table 4.1 indicates how the local memory space is sub-divided for each interface on the OPB. The local memory map is implemented within a 32-bit address space and it defines address ranges, which are used for each interface in OPB including PLB-to-OPB bridge, OPB Arbiter, OPB-slave SDRAM, OPB-slave SRAM and UART(for debugging purposes) [15].

Table 3.1

Memory map for the NIC controller in PowerPC bus interface

Address Range	Access	Max Region Size
0x80000000-0xBFFFFFFF	PLB-to-OPB-Bridge	1 Gigabyte
0x10000000-0x1000001FF	OPB Arbiter	512 byte
0xFFFF8000-0xFFFFFFFF	PLB-BRAM-Controller	32 Kbyte
0xA0000000-0xA00000FF	OPB-UART	256 byte
0x88000000-0x880FFFFF	OPB-Slave SRAM	1 Mbyte
0x80000000-0x81FFFFFF	OPB-Slave SDRAM	32 Mbyte

3.2FPGA-based NIC Controller Basic Functionality

To send a packet from PCI interface to the network, the operating system in the main memory provides a buffer descriptor, which contains the starting memory address and the length of a packet along with additional commands. The device driver then writes the descriptor to a memory mapped register, located in the User Interface unit in Spartan-IIE FPGA. The PowerPC processor, which has access to these registers, initiates DMA transfers to move actual packets from the main memory to the FIFO-WR in the Virtex-II Pro FPGA.

Once the packets are received, they will be moved to the SDRAM using address and length information in buffer descriptor. The MAC controller sends the packets to the FIFO-TX and informs the Gigabit MAC Core to start transmitting packets to the PHY Ethernet chip. Once the packets are sent to the network, the NIC informs the device driver.

At Ethernet interface, when packets are received by the on-board PHY, the Gigabit MAC Core stores them in the FIFO-RX. The MAC controller reads the received packets from the FIFO-RX and stores them in the memory through the OPB Bus transaction. The memory controller provides interface between the local memory on the NIC and the OPB bus transactions. Once the packets are stored in the SDRAM memory, the DMA controller transfers the packets to the main memory through the PCI BUS [17].

The Gigabit Ethernet receive interface is responsible for accepting packets from the external network interface and storing them in the local memory (SDRAM) along with an associated receive descriptor. A top-level block diagram of the Gigabit Ethernet receiver interface is shown in Figure 3.3. The flow for receiving packets is the same as the flow described below. All data, which is received from the Ethernet interface goes through synchronizing FIFOs. As shown in the figure, all design parts in Gigabit Ethernet interface domain operate at 125 MHz whereas design parts in OPB bus interface operate at 100 MHz. The SDRAM controller operates with programmable clock of 100 and 125MHz generated by the Clock Generator unit.

To implement a 32 bit wide data at the OPB Bus interface the Receive-FIFO is designed on four modules of 8 bit wide asynchronous FIFOs. The output of the four FIFOs is concatenated to provide 32 bit data. The FIFOs are implemented using Virtex-II Pro Block Select RAM, which can be configured with different clocks at write port (125MHz) and read port (100MHz). Each FIFO is configured as 8-bit wide and a programmable depth size, ranging from 15 o 31,767 words.

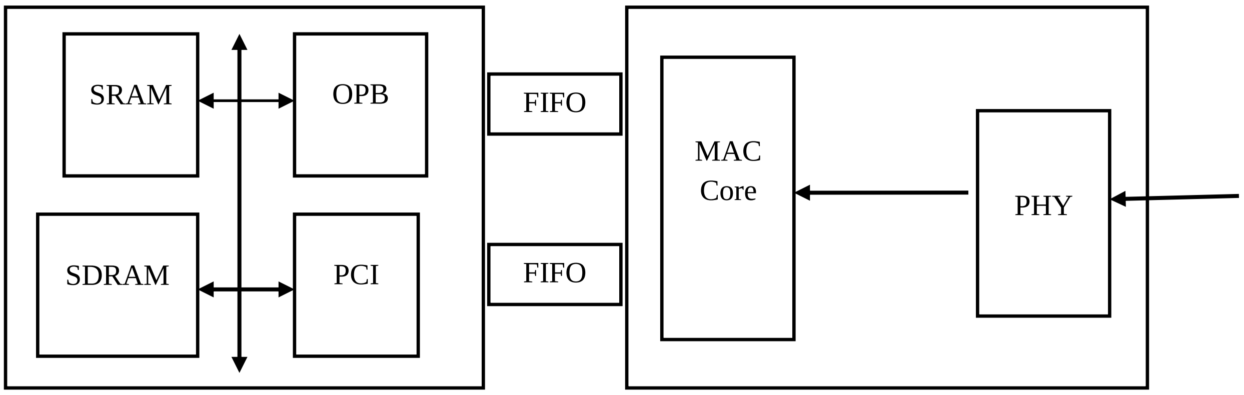

Figure 3.2 Ethernet receive interface block diagram

The NIC controller uses Receive Descriptors to keep track of packets. A frame descriptor is used by network processor to monitor the quality of the serial channel and perform any necessary error reporting to the host. The Ethernet Receive Descriptor is set up in order to provide the ending status information of the received frame and the frame length, which is required for the network processor. Figure 3.4 shows the Receive Descriptor format for each frame. As shown in the figure, the Receive Descriptor is four bytes in length and contains the information about the received frame including the overflow during frame reception, bad or good frame and the frame length. All four Bytes of the descriptor are written at the end of the received frame and are stored in the SDRAM [12].

Packet Status			Frame Length
Overflow	Frame Received	Bad/Good Frame	

Figure 3.3 Ethernet Receiver Descriptor format

The timing diagram for receiving packets is shown in Figure 3.5. The MAC core indicates that valid status of RX data by asserting RX data high. Therefore, as long as RX data is high, the data must be written into the FIFO. There is no feedback mechanism in place to request previous data; therefore, the FIFO must always be ready to write this data on each rising RX. A new frame is always started by writing to the top FIFO.

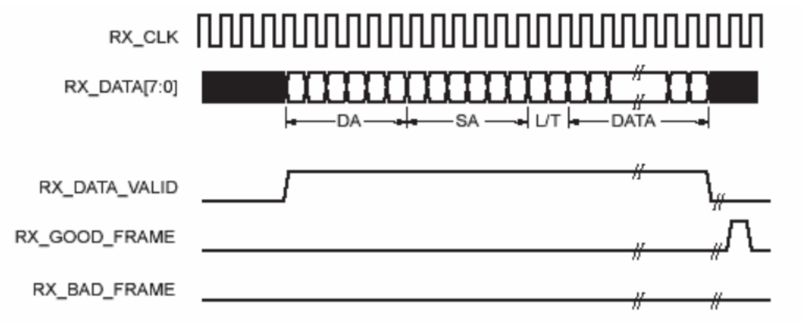

Figure 3.4 Timing diagram of receiving data in Gigabit Ethernet interface (Modified from).

The next 8-bit word of data from the MAC is written into the FIFO #2, then FIFO #3 and FIFO #4 and this repeats until the end of frame, which is indicated by RX_DATA_VALID going low. Write-Control state machine, shown in Figure 3.3, handles generating signal in this way. To align the last data in the four FIFOs, the Write-Control unit generates correct numbers of signal for the rest of FIFOs, which were not written after frame was finished. This ensures that the start of the next frame will be written to the FIFO#1 and the sequence can continue. The end of frame is indicated by the RX_GOOD_FRAME or RX_BAD_FRAME signals from the MAC core, which is used in Receive Descriptor.

The MAC Controller, shown in Figure 3.3, consists of ADDR generator, OPB-master interface and Descriptor generator units. The OPB-master interface translates the master device transaction into a corresponding OPB master transaction. In this section, two designs of OPB-master interface are explored for the MAC-controller. The first implementation of the MAC controller is performed by using OPB-master core provided by Xilinx, called OPB-IPIF master attachment.

IPIF stands for Intellectual Property Interface and the OPB IPIF master attachment is a standardized interface module for connecting custom designed master devices (here MAC controller) to the IBM On-Chip Peripheral Bus (OPB).

Figure 3.6 shows a top-level block diagram of Xilinx OPB-IPIF master attachment. The IP Core and service blocks shown in the IPIF figure use a set of signals and protocols called the IP Interconnect (or IPIC) [7].

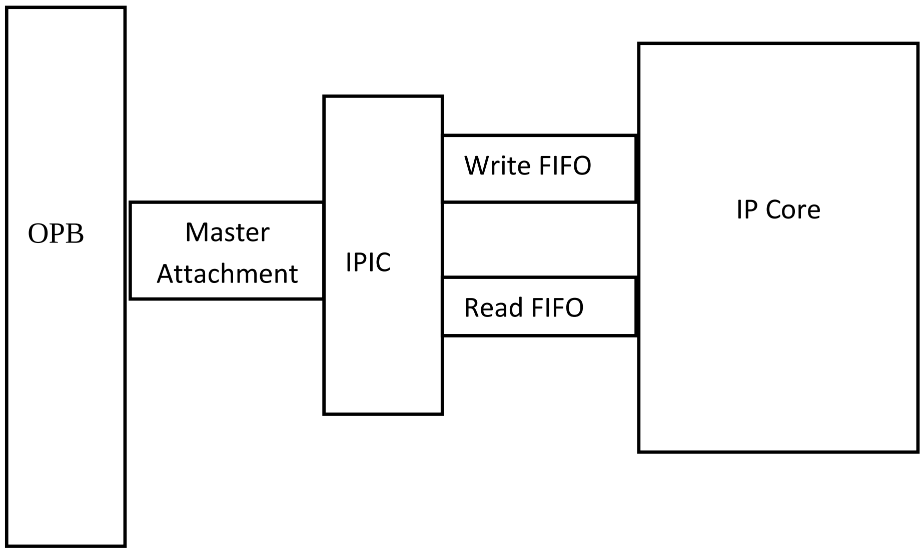

Figure 3.5 Xilinx OPB-IPIF master attachment top-level block diagram

The Master Attachment translates the IP Interconnect master transaction into a corresponding OPB master transaction. As OPB transfers complete, the Master Attachment generates the IPIC response signals. As shown in the figure, the Xilinx OPB-master is designed to be extremely flexible for different applications. However, as timing analysis results by ModelSim and Logic Analyze indicated, the first version of this core which was the only available OPB-IPIF master core by Xilinx was not designed for a high performance implementation. Timing analysis by ModelSim indicated that there is 7 cycles latency in Xilinx OPB-master core to assert the select signal for OPB transfers; this is discussed in detail below.

The HDL code of the Xilinx OPB-master core is not available to investigate the reason for this latency in detail. However, it seems that the RTL coding for the Xilinx OPB-master core was not efficient to enhance performance. Moreover, timing analysis results from Xilinx Timing Analyzer tool indicate that the critical paths in the Xilinx OPB-master core design cannot fit within 10ns requirement provide by OPB clock. Unfortunately, the HDL code of the Xilinx OPB-master core is not available, so it is not possible to improve its performance. The custom-designed OPB-master provides simpler architecture and decreases the slice counts by 75% compared to Xilinx OPB-master core. Moreover, it is designed by applying efficient RTL coding styles, which is described in next section to enhance speed through reducing logic levels and complexity in the state machines.

A noticeable difference in ASICs and FPGA is that ASIC architectures have the ability to tolerate a wide range of RTL coding styles while still allowing designers to meet their design goals. However, FPGA architectures are more sensitive to coding styles and design practices In many cases, slight modifications in coding practices can improve the system performance anywhere from 10% to 100%. A logic level in a FPGA is considered one Combinatorial Logic Block (CLB) delay. If the amount of logic that can fit into one CLB is exceeded, another level of logic delay is added which results in increased delay. For example, a module with 6 to 8 FPGA logic levels would operate at ~50MHz where as a module with 4 to 6 FPGA logic levels would operate at ~100MHz.

This section provides the coding techniques, which are applied in the NIC controller architecture design specifically in the OPB-master interface design. These techniques include duplicating register to decrease the famous, using one-hot state machines to decrease the logic complexity for each state, adding pipeline stages to break up long data paths in multiple clocks and using case statements instead of else-if statements to enhance the speed of multiplexing. As mentioned in last section the critical paths in the first version of Xilinx OPB-master core could not fit within 10 ns OPB clock. To work around these problems, an approximate amount of logic levels should be considered. The placement of the logic should be taken into consideration, too. Some of the techniques that are used to decrease the logic levels and enhance the performance of the design are discussed as the following [9].

Since FPGA architectures are plentiful with registers, duplicating the number of registers in a critical path, which contains a large number of loads, is very useful. This technique reduces the fan out of the critical path, which substantially improves the system performance. An example is given in Figure 3.7 and 3.8 to show how using register duplication can reduce the fan out. Figure 3.7 shows the memory address (Mem-ADDR-In) and the address enable signal (Tri-Addr-En) in MAC controller and SDRAM controller interface. As shown in the figure, the enable signal is generated from a single register so there is a 32 fan out for this register.

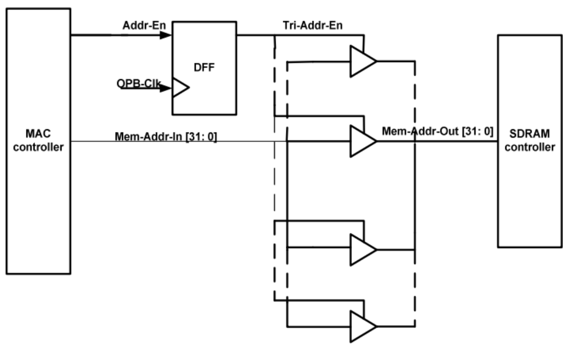

Figure 3.6 Memory address generation with 32 loads

By separating the address enable lines within two registers, the number of loads in each register is decreased by half, which results in a faster routing process. The block diagram of the improved architecture by duplicating the registers is shown in Figure 3.8.

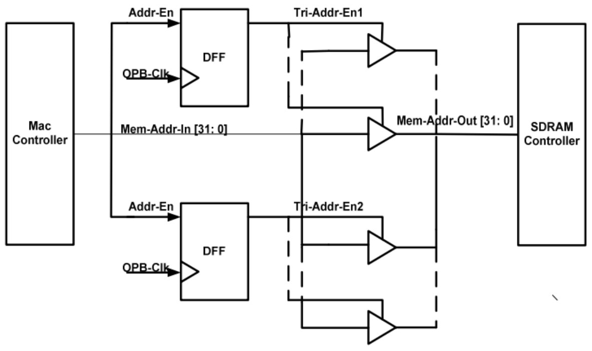

Figure 3.7 Register duplication to reduce fan out in memory address generation

The goal in designing fast FPGA designs is to fit the most logic into one Combinatorial Logic Block (CLB). In Virtex-II PRO FPGA, a 16:1 multiplexer can be implemented in one CLB, which is built by 4 slices. In general, If-Else statements are much slower than CASE statements. The reason is that each If keyword specifies a priority-encoded logic whereas the Case statement generally creates a parallel comparison. Thus, improper use of the Nested If statement can result in an increase in area and longer delays in a design. The following piece of VHDL code with IF statement is used to in MAC Controller to calculate the total bytes transfer for each frame:

```
if (fifo_read_status = “000”) then
byte-count <= count[0];
elsif (fifo_read_status= “001”) then
byte_count <= count[1];
elsif (fifo_read_status = “110”) then
byte_count <= count[2];
elsif (fifo_read_status= “111”) then
byte_count <= count[3];
end if;
```

The schematic level diagram of this code is shown in Figure 4.14.

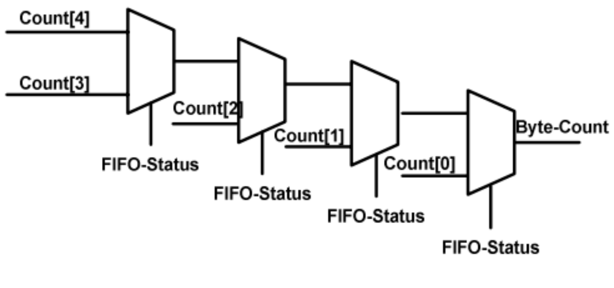

Figure 3.8 Inefficient multiplexed signals by using else-if statement

As shown in the figure, four individual 2 to 1 multiplexer is utilized to implement above piece of code. However, the same logic can be implemented using a Case statement:

```
Case (fifo_read_status ) is
When “000” =>
byte-count <= count[0];
When “001” =>
byte_count <= count[1];
When “110”=>
byte_count <= count[2];
When “111” =>
byte_count <= count[3];
When others =>
Null;
end case;
```

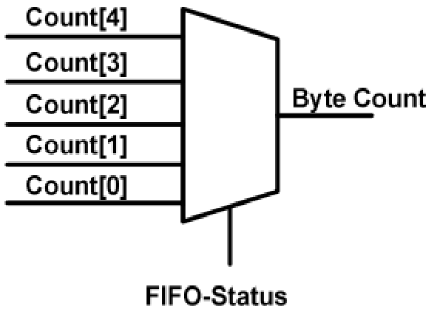

Figure 3.9 A 5 to 1 MUX implementation fit in one CLB

The schematic level of the above code is shown in Figure 4.15. As shown in the figure using the code can be implemented in one 5 to 1 Mux, which can be fit in one CLB. This yields a more compact design resulting in a faster implementation[14].

Table 3.2

Resource utilization and timing analysis results for the longest path in Xilinx OPB-master core and custom-designed OPB-master interface implementations.

Design	**Xillins OPB-master Core**	**Custom-Designed OPB-master**	**Improvement**
Slice count (% of chip slices)	278 (2%)	68(1%)	75.53 %
Longest Slack (ns)	-1.459	0.726	149.76%
Longest data path delay	10.89	9.36	14.04%
Longest Clock Skew(ns)	-0.569	0.086	115.11%
Logic Level	6	3	50%
Maximum Frequency (MHz)	91	106.83	17.39%

The first row in the table is the slack number, which identifies if the path meets the timing requirement. Slack is calculated from equation 4.1. Slack Requirement Time-(Data Path Delay- Clock Skew) (4.1)The requirement time is the maximum allowable delay. In this case, it is 10ns, which is the OPB clock period. This is set as period constraint in the UCF (User Constraints file) file. The Data Path Delay is the delay of the data path from the source to the destination. Clock Skew is the difference between the time a clock signal arrives at the source flip-flop in a path and the time it arrives at the destination flip-flop. If the slack is positive, then the path meets timing constraint by the slack amount. If the slack is negative, then the path fails the timing constraint by the slack amount. The levels of logic are the number of LUTs (Look-up Table) that carry logic between the source and destination. As shown in the table, the slack for the longest path in

Xilinx OPB-master core is -1.459ns, which indicates the timing violation of this path in the design. There are 6 logic levels for this path which adds excessive delay and causes the design runs slower. As shown in the table the longest delay path in custom-designed OPB-master interface is 9.26ns, which fits in the 10ns constraint requirement and improves the delay by 14%. Compared to Xilinx OPB-master core, the custom-designed OPB-master decreases the levels of logic from 6 to 3(50% improvement) and delivers up to 17.3% improvement in operating frequency [14], [15], [16].

SUMMARY

This chapter evaluates the FPGA-based NIC functionality in the Ethernet receiver interface. It investigates how improving the processing latency of each interface can improve the overall achieved throughput, transfer latency and bus utilization. Experimental results show that increasing FIFO size up to 8Kbytes does not affect on on-chip memory utilization. Therefore, the receive FIFO can provide enough buffering space for up to 10 incoming large frame with size of 1518 bytes which can improve the performance in receiving small sized packet.

The bus utilization measurements reveal that the receive interface with minimum sized frame and burst length 8 implementation consumes only 26% of the OPB bus. As a result, the receive interface implementation makes up to 74% of the OPB bus available for other OPB bus interfaces. Thus, with overlapped arbitration, the Ethernet transmit and bidirectional PCI-DMA interfaces can be implemented on the OPB to make a complete functional NIC transfer. Of course, any additional interface should be implemented based on the efficient architectures and RTL programming techniques, which were discussed in sections.

Thus, the FPGA-based Gigabit Ethernet NIC is a viable platform to achieve throughput competitive with ASIC-based NICs for real time network services. Such a research platform provides a valuable tool for systems researchers in networking to explore efficient system architectures and services to improve server performance.

OVERALL CONCLUSION

The continuing advances in the performance of network servers make it essential for network interface cards to provide services that are more sophisticated rather than simple data transferring. Modern network interfaces provide fixed functionality and are optimized for sending and receiving large packets. Previous research has shown that both increased functionality in the network interface and increased bandwidth on small packets can significantly improve the performance of today's network servers. One of the key challenges for networking systems researchers is to find effective ways to investigate novel architectures for these new services and evaluate their performance characteristics in a real network interface platform. The development of such services requires flexible and open systems that can easily be extended to enable new features.

This thesis presents the design and evaluation of a flexible and configurable Gigabit Ethernet/PCI network interface card using FPGAs. This system is designed as an open research platform, with a range of configuration options and possibilities for extension in both software and hardware dimensions. This FPGA-based NIC features two types of volatile memory. A pipelined ZBT SRAM device is used as a low latency memory to allow the network processor to access the code. The other memory is a 128 Mbytes SDRAM SO-DIMM, a large capacity and high bandwidth memory, which is used for data storage and adding future services like network interface data caching. This thesis first presents the design of the hardware platform for the FPGA-based Gigabit Ethernet/ PCI NIC. Then the thesis shows a high-performance architecture for a functional Gigabit Ethernet/PCI network interface controller in the FPGA.

The performance of Gigabit Ethernet receive interface is evaluated in the Avnet board. The experimental results indicate that adding pipelined stages and improving the RTL coding in the design, lead to the reduction of latency in bus interface operations, which results to 32% reduction in total data transfer latency between receive FIFO and SDRAM and 72.5% improvement in achieved throughput with single transfers in OPB Bus interface. The results presented in this thesis imply that, using burst transfers can alleviate the SDRAM access latency and results in throughput improvement and bus utilization reduction.

Compared to SDRAM single transfer implementation, implementation with burst length 4 and 8 can reduce the FIFO transfer latency to 84% and deliver up to 516.25 % more throughput in received maximum-sized 1518-byte Ethernet packet. In addition, the experimental results with burst length 4 and 8 indicate that the FPGA-based NIC can receive all packet sizes and store them in the SDRAM at Gigabit Ethernet line rate. This is a promising result since no existing network interface card use SDRAM due to SDRAM latency. Increasing the working frequency of SDRAM controller to 125MHz, 25% faster than the processor bus clock, allows faster access time in memory interface. Compared to 100MHz SDRAM controller, the 125MHz implementation reduces the SDRAM operation cycles to 20%. This results in reduc0tion of total transfer latency and increases the available bandwidth for other OPB bus interfaces.

The bus utilization measurements indicate that the receive interface with minimum sized Ethernet frame of 64-byte and burst length 8 implementation consumes only 26% of the OPB bus. As a result, the receive interface implementation makes up to 74% of the OPB bus available for other OPB bus interfaces. Thus, with overlapped arbitration, the Ethernet transmit and bidirectional PCI-DMA interfaces can be implemented on the OPB to make a complete functional NIC transfer. Of course, any additional interface should be implemented based on the efficient architectures and RTL programming techniques, which were discussed in below.

Thus, this thesis shows that the FPGA-based Gigabit Ethernet NIC is a viable platform to achieve throughput competitive with ASIC-based NICs for real time network services. Such a research platform provides a valuable tool for systems researchers in networking to explore efficient system architectures and services to improve server performance.