

Design and Performance Analysis of a High-Speed Network Processor

By

Md. Jahidul Islam

Roll No: 0803507

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Electrical and Electronic Engineering



Khulna University of Engineering & Technology
Khulna 9203, Bangladesh

August 2011

Declaration

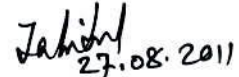
This is to certify that the thesis work entitled "Design and Performance Analysis of a High-Speed Network Processor" has been carried out by Md. Jahidul Islam in the Department of Electrical and Electronic Engineering, Khulna University of Engineering & Technology, Khulna, Bangladesh. The above thesis work or any part of this work has not been submitted anywhere for the award of any degree or diploma.



27.08.11

Signature of Supervisor

(Prof. Dr. Md. Abdur Rafiq)



27.08.2011



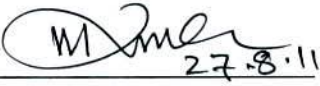


Signature of Candidate

(Md. Jahidul Islam)

Approval

This is to certify that the thesis work submitted by Md. Jahidul Islam entitled "Design and Performance Analysis of a High-Speed Network Processor" has been approved by the board of examiners for the partial fulfillment of the requirements for the degree of Master of Science in Engineering in the Department of Electrical and Electronic Engineering, Khulna University of Engineering & Technology, Khulna, Bangladesh in August 2011.

BOARD OF EXAMINERS

1. 
27.08.11
_____ Chairman
(Supervisor)
Dr. Md. Abdur Rafiq
Professor
Department of Electrical and Electronic Engineering
Khulna University of Engineering & Technology
2. 
27.08.11
_____ Member
Head
Department of Electrical and Electronic Engineering
Khulna University of Engineering & Technology
3. 
27.8.11
_____ Member
Dr. Md. Rafiqul Islam
Professor
Department of Electrical and Electronic Engineering
Khulna University of Engineering & Technology
4. 
27.08.11
_____ Member
Dr. Md. Shahjahan
Associate Professor
Department of Electrical and Electronic Engineering
Khulna University of Engineering & Technology
5. 
27/8/2011
_____ Member
(External)
Dr. M. M. A. Hashem
Professor
Department of Computer Science and Engineering
Khulna University of Engineering & Technology

Acknowledgement

First of all, I would like to grateful to almighty ALLAH.

As a master's student I not only furthered my technical skills but also got an opportunity to enrich my professional life. I have realized the importance of patience, intellectual humility and optimism during this course of time.

I would like to sincerely thank my supervisor Professor Dr. Md. Abdur Rafiq for providing me the opportunity to work on such an interesting topic of Network Processor. He has been very encouraging and patient with me, besides giving me a lot of independence. His regular guidance, inspiration and the time providing played the vital role in completion of this thesis work. I would also like to thank him for all the long hours he spent reviewing my thesis and timely feedback, even while on vacation.

I would grateful to the researchers in the field of network processor, packet processor, networking systems, data communication, high-speed communication system and FPGA/ASIC design for their research work and recent publications.

It is mentionable that, my work would not have been possible without the necessary CAD software. So, I would like to thank EZchip microcode development tools and NepSim Network processor simulator. I specially would like to thank Xilinx Foundation for the ISE Design software tools and Actel Corporation for their Libero IDE FPGA Design tools, without which I could not have completed my work. I would also like to thank Mentor Graphics for their ModelSim waveform simulator.

Finally, I would like to thank my family for all their love and support all along this effort. Especially thanks to my parents for always having faith in my abilities and giving inspiration for successfully completion of the research work. I would also like to thank to my friends for giving mental support and encouragement that has helped me cope with difficult times.

Abstract

Network processor is the key architecture of the recent communication technology. Most of the high performance network equipments especially routers, switches and traffic management systems are designed with network processor to process their network packets. This research work introduces a design architecture for high-speed network packet processor and also analyzes its performance. There are two proposals in this thesis, first is the proposed hardware architecture of a high-speed network processor system and second is the proposed modified architecture of packet processing unit. A hierarchical 4 level layered processing architecture is developed for efficiently process the packets. To capture all traffic from a high-speed I/O interface without any loss a load-balancer with an efficient load distribution algorithm is implemented. High throughput pipelined memory architecture is also developed to minimize the rate of memory access time. The processor units have four basic operational tasks – parse, search, resolve and modify. To design the processing unit, the thesis provides some modification of the Task Optimized Processing core (TOPcore) technology and proposed a modified processing core architecture. This is a super-pipelined parallel architecture. The performance of the proposed network processor is evaluated for some real applications and compared with reference NPs. Results shows that the proposed architecture is efficient and provides better performance. Finally, the design is modeled and simulated in RT level using VHDL and then synthesized to schematic. The synthesis is done for both Xilinx Spartan3 and Actel ProASIC3 FPGA. Design requires very little FPGA logic while efficiently processing packet and implementation of green technology provides saving of power consumption at ideal condition.

To my parents

Contents

	PAGE
Title Page	i
Declaration	ii
Approval	iii
Acknowledgement	iv
Abstract	v
Contents	vii
List of Tables	ix
List of Figures	x
CHAPTER I Introduction	
1.1 Background and Previous work	1
1.2 Motivation and Objectives	2
1.3 Scope of the thesis	3
1.4 Approaches	4
1.5 Challenges	4
1.6 Organization of the thesis	4
CHAPTER II Network Processor Review	
2.1 Overview	6
2.2 Evaluation	8
2.2.1 First-Generation Systems	8
2.2.2 Second-Generation Systems	9
2.2.3 Third-Generation Systems	10
2.3 Applications	10
2.4 Generic Architecture	12
CHAPTER III Data Networking and Packet Processing	
3.1 Data Networking Models	15
3.1.1 ISO/OSI Model	15
3.1.2 TCP/IP Model	16
3.1.3 Data Networking	16
3.2 Packet Processing	18
3.2.1 Packet Processing Flow	19
3.2.2 Ingress and Egress	20
3.2.3 Framing	21
3.2.4 Parsing and Classification	21
3.2.5 IP Lookup and Forwarding	22
3.2.6 Modification	23
3.2.7 Compression and Encryption	23
3.2.8 Queuing and Traffic Management	24
3.3 IP Addressing, Routing and Forwarding	24
3.4 Search Engines	27
3.4.1 Simple Switching and Forwarding	31
3.4.2 IP Address Lookup	31

CHAPTER IV	Architecture and Methodology	
4.1	Network Processing Trends	32
4.1.1	Network Processing Functions	33
4.2	Architectures of Network Processor	34
4.2.1	Basic Architectural Approaches	35
4.2.2	Parallel Pipelines of Homogeneous Processors	36
4.2.3	Pipeline of Parallel Heterogeneous Processors	37
4.2.4	Superpipeline and Superscalar Architecture	37
4.3	Design Methodology	38
4.3.1	Layered Processing Architecture	38
4.3.2	Pipelined Memory Architecture	39
4.3.3	High Throughput Pipelined Memory	39
4.3.4	Load Balancing for High-Speed Links	41
4.3.5	Load Distribution Algorithm	42
4.4	Proposed Architecture	44
4.4.1	High-Speed Ethernet MAC	45
4.4.2	Load Balancer	45
4.4.3	Processor Unit	46
4.4.4	Increasing the Throughput of Common Operations	47
4.5	Overall Power Model	47
CHAPTER V	Modeling and Simulation	
5.1	EZdesign Microcode Development Tools	50
5.1	Network Processor Simulator: NepSim	52
5.3	VHDL Modeling, Synthesis and Schematic Diagram	53
5.3.1	Working on Xilinx ISE Design Suite	53
5.3.2	Working on Actel Libero IDE Design Flow	60
5.4	Simulation Waveform	68
5.5	Synthesized Power Report	70
CHAPTER VI	Performance Evaluation	
6.1	General NP Performance	71
6.2	Architecture Analysis and Comparison	71
6.3	Evaluation Methodology	73
6.3.1	Real Packet Processing Operation	74
6.4	Summary	75
CHAPTER VII	Conclusion and Future Work	
7.1	Conclusion	76
7.2	Future Work	76
	References	77

List of Tables

Table		Page
3.1	IP Address Example	25
3.2	Example of Routing Table	26
3.3	IPv4 and IPv6 Non-routable Addresses	27
4.1	Cache capacitance for 0.35- μm technology	49
6.1	Comparison of packet processing tasks	74

List of Figures

Figure		Page
2.1	Software-Based Architecture	9
2.2	Generic Network Processor	13
3.1	ISO/OSI seven layers model	15
3.2	TCP/IP model	16
3.3	Data (payload) encapsulation	17
3.4	Network Model	17
3.5	Router layers	18
3.6	Internet Model	18
3.7	A general framework of packet processing	20
3.8	Ingress and Egress Processing Configuration	20
3.9	Simplified CAM scheme	29
3.10	TCAM priority mechanism	29
3.11	CAM/RAM system for IP address lookup	30
4.1	Evolving the Network Design System Philosophy	32
4.2	Graphical comparison between network processor-based systems and their hardwired counterparts	33
4.3	Embedded processor architecture in which a single processor handles all packets	35
4.4	Parallel architecture in which the incoming packet flow is divided among multiple processors	35
4.5	Pipeline Architecture in Which Each Incoming Packet Flows Through Multiple Stages of a Pipeline	36
4.6	An example architecture that uses parallel pipelines of homogeneous processors (Cisco)	36
4.7	An example architecture that uses a pipeline of parallel stages with heterogeneous processors (EZchip)	37
4.8	EZchip TOPcore superpipeline and superscalar processor architecture	37
4.9	Layered processing architecture	38
4.10	Pipelined wide-word memory architecture	40
4.11	Load-balancer system	41
4.12	Diagram of Load-balancer	41
4.13	Large flow accommodation through reassignments	43
4.14	Block diagram of proposed Network Processor	45
4.15	High-speed Ethernet MAC Interface	45
4.16	Load balancer diagram	46
4.17	Proposed modified architecture of processor unit	46
5.1	EZchip microcode development environment	50
5.2	EZchip NP simulation status	51
5.3	Generating Frame structure	51
5.4	NePSim software structure	52
5.5	I/O pin layout of the top module	53
5.6	Total schematic diagram of the top module	54
5.7	Schematic circuit of control module	55
5.8	Schematic circuit of scheduler module	56

5.9	Schematic circuit of memory module	57
5.10	Actel Libero IDE Design Flow	60
5.11	RTL Synthesis block diagram	61
5.12	Full synthesized schematic circuit	62
5.13	synthesized diagram of memory and control unit	63
5.14	Schematic circuit of control unit	63
5.15	Schematic circuit of memory unit	63
5.16	Schematic of memory register	64
5.17	Data path diagram of control operation	64
5.18	Critical data path	65
5.19	Simulation waveforms of packet receive interface	68
5.20	Simulation waveforms of packet transmit interface	68
5.21	Simulation waveform of routing application	69
5.22	Simulation waveform of switching application	69
5.23	Simulation waveform of Load scheduler	69
6.1	Block diagram of proposed Network Processor architecture	72
6.2	Proposed modified architecture of processor unit	72
6.3	Performance comparison graph	75

CHAPTER I

Introduction

The bandwidth requirement of communication network is increasing with the increases of various network applications. This rapidly increasing consumption of bandwidth significantly shapes the development of the transport networks and the communication terminals. Running high-speed networking functionalities on today's communication systems requires that a lot of functions are to be implemented in a hardwired manner. The ultimate throughput of communication systems depend on its network processing equipments. In recent years, fiber optic link technologies throughputs have expanded at greater rates and it increases the burden on the routing technology. With greater bandwidth and greater network processing requirements to handle traffic, it has become apparent that previous methods of traffic processing are unable to meet the current market requirements and high performance. This has brought about the development of one of the most promising technologies in this area, namely Network Processor (NP).

With the introducing of recent faster communication medium the total communication performance is not increased well with communication lines. The reason is the lagging of faster networking devices. In recent technology most often engineers are trying to use fiber-optic communication medium to speed up the network communication, but the conventional networking devices are not simultaneously speedy with the optical medium. As a result the total performance is depending on the processing element of the network devices. So, for getting high performance network operation it is needed to speed up the network processors, the main packet processing element of the networking device. Researchers are still trying to increase the speed of the network processor and for that they introduce many more methods and techniques regarding this.

1.1 Background and Previous work

During past 20 years, engineering of network systems has changed a lot. Their architectures can be divided into three main generations: [1]

- The first generation goes back to 1980's when standard processors were used for network applications, e.g., like a minicomputer for routing.
- The second generation runs by mid of 1990's, speed and complexity of systems had gone up, so designers added special hardware blocks to relieve the load from CPU.
- The third generation systems employed very specialized hardware in ASIC (application specific integrated circuit) and even attempted to use several of them for higher performance systems. Because the protocols were consolidating around Ethernet and IP at those days, not much flexibility was needed and fully hardware fixed solutions were satisfactory.

As can be noted in the transition above, with each new generation the programmability of network devices were traded for higher speed through adding more hardwired components. Towards the end of 1990's, the Internet boom period, convergence of voice and data networks started to become more imminent. As a result, new and wider range of

protocols and services, e.g., multimedia services, needed to be developed by the industry. The pace of introduction of new services and their further upgrades had become faster shortening the product cycle and requiring faster time-to-market. Furthermore, more complex services were expected to become the norm, for example moving the routers beyond just store and forward machines and increasing the required processing power several order of magnitudes of the exiting level.

The best solution seemed to be bringing back programmability which was the hallmark of the first generation network but not at the cost of performance which was the hallmark of second generation network. The result was a new hardware known as network processor (NP) today. Network processors need to deliver the speed of silicon which is combined with the intelligence and flexibility of programmable microprocessors. The key to network processor success is an architecture that enables implementation of high-level applications in high-speed networking environments [3].

Owing to increasing network data rates and more sophisticated networking protocols, networking solutions continue to demand more powerful processing capabilities [7]. The processing of packets is the main job of the network systems [8]. Current packet processing device approaches designed for performance at multiple gigabit-per-second data rates and used network processors that are used in routers or switches. True Network Processors are usually large SoC (System-on-Chip) with multiple processing devices. These processing devices are usually very simple, specially designed generic processors [9]. True Network Processors can also be implemented using other techniques such as massive multithreading and usually include custom hardware for packet processing [10].

From the late 1999 the researches on network processing hardware for high speed communication is significantly growing up and still continue. Architecture for fast packet processing [14], slow response time, higher throughput, multithreading [9] on network processor, parallel processing [15], adaptive processing [11] approach, FPGAs [13] and ASIC implementations for high speed packet processing etc. is the major area of that research.

In April 2002, researchers at the University of Southern California's Information Sciences Institute demonstrated a first-generation giga-bit rate packet processing system implemented in several FPGAs [12]. The researchers sought to address the performance bottleneck imposed by the Central Processing Unit (CPU) in single-processor gigabit networking systems. Packet processing functions including basic routing, encryption, and framing are performed in the reconfigurable components. Three separate Xilinx Virtex XCV1000 FPGAs (X0, X1 and X2) are used to process data traffic [12].

In 2002, Gordon Brebner of the University of Endinburgh described a gigabit IP router [13] that fits on a single Xilinx Virtex-II Pro XC2VP7 FPGA device. This router takes advantage of the configurability of the FPGA to maintain system performance as traffic migrates from IPv4 to IPv6.

1.2 Motivation and Objectives

Network systems, such as routers and switches, started as conventional central processing units. They have a CPU, RAM and ROM to store the operating system and interfaces to connect to the network. At the beginning, their performance was sufficient. With the rapid growth of the Internet and applications, they became a bottleneck. They could not reach

the required speed for packet throughput. To solve this bottleneck, ASIC was introduced. ASIC is an integrated circuit designed to perform the networking functions at wire speed. The networking functions are designed into silicon hardware permanently.

Bandwidth is important and critical to network applications. Because emerging Internet applications increase the network traffic, it is pushing the limit of the capacity of communication lines and semiconductor technologies. Therefore, network equipment providers are searching for better technologies and methods to handle, support and manage the traffic.

Currently, researchers are exploring several different approaches to high speed networking. Most of the work has concentrated in the areas of improving switching architectures [1]. Therefore, network equipment providers are searching for better technologies and methods to handle, support and manage the traffic [2]. Network processors present a solution, which can help maximize bandwidth utilization and traffic flow [3, 4]. And now it is the main component in the network systems to meet the new bandwidth, speed, and performance requirements.

From the study of this related works, it is found that network devices that use network processor are not speedy well with the communication medium. Moreover, researchers are still trying to increase the speed of the network processor. So, research on this area is the important issue to increase the network performance and meet the current networking demands.

Again due to higher bandwidth requirements and to support the real-time applications with high-speed communication medium (like: optical fiber), the main issue for improving the routing systems are to design such a network processor which considers the following:

- Increase processing speed and reduce mean system wait time
- Faster time scheduling and memory queuing architecture
- Use adaptive methodology for support various packet processing
- Parallel packet processing from multi queuing system
- Multi protocol supporting and multi-threading capability
- Parallel processing with multi-core processor unit
- Programmable and scalable processing architecture
- Synchronous processing capability with different communication medium

The objective of this thesis work is to design such a network processor architecture that speeds up the packet processing power.

1.3 Scope of the thesis

Network processing is an active field both academically and commercially. Network researchers are always try to speed up the processing power of a network processor to reach its incoming traffic speeds. Different methods and techniques are introduced by the different academic researchers. On the other side, commercial network processors are viewed as the next step towards high-speed data traffic processing and are currently enjoying a third generation systems. At the same time, academic research continues with new methods of providing network services in reconfigurable hardware.

1.4 Approaches

In the first step of this thesis work is to study on various network processors with their design architecture and functionality. Then study on the methods and algorithms that the existing processors used to achieve their specific purpose. From the study and analysis, this research finds some techniques and methodology for designing efficient network processor hardware. The Task Optimized Processing core (TOPcore) technology is chosen for designing the core processing unit. To processing in a parallel way pipelined architecture is developed with pipelined memory interface. An efficient load balancer is used for high performance parallel processing. The whole processes are followed by a hierarchical layered processing strategy.

Finally, the design will be modeled and simulated in RTL level using VHDL and then synthesized to a schematic. The design is also simulated in ModelSim Simulator and generates simulation waveforms.

1.5 Challenges

The performance aspects of network processor design continue to be challenging. Additionally, there are now other concerns of growing importance that focus on software and new applications for network processors. The design and application spaces are very large. Programmability remains a substantial challenge due to the different types of NP architectures, the evolution of networking standards and applications, and the unavailability of a unified and widely accepted set of software development and performance prediction tools.

On the other hand, with introduction of optical fibers in transport networks, the serial time-division multiplexing (TDM) synchronous optical network/synchronous digital hierarchy (SONET/SDH) transmission speed grew exponentially and reached 40 Gb/s rate by 2000 [2]. Though the speed increase was not expected to go beyond 100 Gb/s because of the limits of transceivers, this had already put pressure on the network device designers. To make the situation more challenging, deployment of WDM transmission technology brought radical changes increasing transmission capacity of fiber links to 1.6 Tb/s and beyond [2].

1.6 Organization of the thesis

This section reviews the topics covered in sequence and remainder organization of the whole thesis work.

- Chapter 2 describes the overview of the Network Processor (NP) systems. The evaluation and generation, application and generic NP architecture are described there.
- In chapter 3, data processing models and details packet processing functions are described.
- The chapter 4 is for the architectural definition of NP and describes the methodologies that are used in this thesis. The proposed architectural design and the proposed modified design of processor unit are given in detail there.

- The architectural microcode simulation, VHDL modeling and synthesized results are given in detail in chapter 5.
- The performance evaluation of the thesis is described in chapter 6.
- Chapter 7 gives the conclusion and future work of the thesis.

CHAPTER II

Network Processor Review

2.1 Overview

Communication over the Internet is built on packet switching. The processing of packets is the main job of the network systems such as switches and routers [2]. These network systems examine each packet, and then decide what to do with them. Typically, this decision depends on the headers of the packets. They can be forwarded to interfaces of the system or returned to the sender as an error message. The functions and the services that the network system provides depend on the architecture of the network system processor.

A network processor, unlike the conventional computer processor unit (CPU), combines hardware functional units with software, and is designed and highly optimized to perform network functions [2-4]. For high bandwidth and performance, parallelism and pipelining are used in the design of Network Processors.

A Simple definition of a network processor is that it is a programmable device which is specially designed to process packet data at wire speed [7]. However, this definition doesn't really clarify what the Network Processor has to process. This is summarized in more detail below.

1. **Pattern Matching:** Here the network processor compares packet fields with specific patterns, to classify the type of packet, for example to decide whether a packet is an IPv4 or an IPv6 packet.
2. **Lookup:** Here the network processor takes a packet field and performs a lookup in a table to return the relevant table entry. This table may be either in internal memory or external memory. For small fields the table can be one-to-one and only require a single lookup. However, for larger fields, a tree search may be required to find the correct table entry, in this case multiple lookups may be required. A typical lookup may be to perform a lookup on the destination address to identify the IP address of the next hop.
3. **Data Manipulation:** This is where the packet is modified in some way. This could be decrementing the Time-To-Live (TTL) field in an IP packet, recalculating the CRC check, performing packet segmentation and reassembly, or encryption/decryption of a packet.
4. **Queue Management:** This is where the scheduling and storage of the packets is handled to provide traffic shaping and quality of service priority queuing.

Network processors vary in their level of:

- programmability,
- processing power,
- extent and ease of configuration,

- flexibility,
- adaptability and
- inclusion of specialized hardware structures [20]

Programmability is the ability of loading a program (set of instructions to perform a certain task) or a set of registers (setting the initial state of the processor), such as in Switch core NP. While the level of programmability may enable the support for a wider variety of applications, it also increases the level of complexity of product development (creating a program requires more work than setting a set of configuration registers).

Processing power is a measure of the data processing capacity, usually measured in millions of instruction per second (MIPS), millions of floating-point operations per second (MFLOPS), millions of bits per second (Mbps, megabits per sec.), millions of bytes per sec. (MBps, megabytes per sec.), millions of clock cycles per sec. (MHz megahertz), millions of packets per second (MPps). The units used differ between manufacturers, but most use a measurement of the throughput of data on the communication medium. These are measured either in 'Mbps' or in 'MPps'. 'Mbps' gives a good indication of the communication mediums that can be connected to the system, but does not give any indication of the amount processing that can be done on the packets. 'MPps' is a better indication of the amount of processing, but is highly dependent on the protocols and traffic patterns on the network. It requires information on the protocols, types of processing, and types of traffic. In the network processor market, the 'MPps' is by far the most favored unit of processing power. There is no perfect type of processing power measurement. The processing power of the Network Processor depends not only on the number and the speed of the core processor(s), and the amount of threads on each core, but also on their efficiency and specialized hardware structures. For example, both the MMC NP7120 and the Agere (Lucent) FPP/RSP support a line bandwidth of 2 x OC-48 (5 Gbps), but the Agere Network Processor has 1700 MIPS versus 440 MIPS for MMC's network processor. The Network Processor from Agere has 4 CPU cores versus 2 for the MMC and the Agere also has hardware support for a total of 64 threads versus 8 per CPU [21]. Multiprocessor and multithreaded systems increase the power of the Network Processor, but that requires the breaking of the algorithms into multiple parts. While networking naturally lends itself to parallelism, it can still be difficult to split the algorithms into the 64 threads that some of these Network Processors have and to balance the load over the 16 processors.

Extent and Ease of Reconfiguration is related to the programmability, as it is partly measured by the ease of re-programmability or reconfiguration, whether it can be done in the field, on the fly or if it requires removal from the system. It is also related to whether the network processor has any FPGA-like structures to allow the designer to load custom hardware structures into the design, as in the Chameleon CS2000 Family, and to the type of programmability (on the fly, etc...). This feature can increase overall throughput of the system by allowing for time-division-multiplexing, but will increase the latency of the Network Processor.

Flexibility is the ease with which the network processor can be configured or adapted to a given task, protocol or service. Flexibility is related to which functions can be implemented in software, which functions can be implemented in custom hardware and whether the interconnect between chip components can be reconfigured.

Adaptability is the ability of a chip to support different types of protocols, services, communication links and bus standards (PCI, AMBA, etc...). It is the ease with which it can adapt to the environment in which it is placed.

Inclusion of specialized hardware structures provides on-chip specialized packet processing structures that would normally be present in an ASIC design and usually delivering considerable performance enhancement for network specific tasks. Structures such as polynomial-based hashing hardware, SRAM read write queues, address lookup tables, parallel processing power and even integrated SRAM and SDRAM controllers are among the cores that can be included in a Network Processor. An example of such hardware structures integration is the IBM Power Network Processor. Such inclusion of specialized hardware structures provides performance enhancements and/or greater system integration.

2.2 Evaluation

Over the last 20 years, network systems especially router architectures, have evolved through three generations, each marked by improvements in packet processing mechanisms.

2.2.1 First-Generation Systems

Up to the mid 1990's, router architecture was similar to the conventional PC systems. Figure 2.1 illustrates a CPU that performs networking functions, controlled by the router's operating system. Like conventional PCs, the router's operating system resides in the system's volatile memory in RAM and controls all the system's functions and services.

In such a system, all tasks are controlled and performed by software, and routers built in this system are called software-based routers. Because routing is software-based, adding new functions and services to the router can be done by simply changing or adding new instructions to the software [2-4].

It is good for the vendors because it does not take much time to change or upgrade the router's software. They could quickly develop new or special purpose products within a short time. The Cisco 2500 Router is an example of a software-based router (Figure 2.1). Cisco 2500 uses its Central Processor Unit to execute and conduct its routing instructions stored in nonvolatile RAM.

As Networking technology and applications changed, the drawback of this system became apparent. Software-based architectures had a limited ability to scale to higher bandwidth demands and new routing services [4]. For example, the majority of software based routers can only support wire speed throughput for less than 155 Mbps [4]. For perform complex networking functions, like filtering, policy based routing, and examining traffic statistics, the throughput of software-based routers is reduced. This creates a bottleneck in the network [4].

Networking technology was constantly developing, but software-based architecture could not keep up with the bandwidth demand and started to suffer in performance. In addition, maintaining this architecture became very expensive.

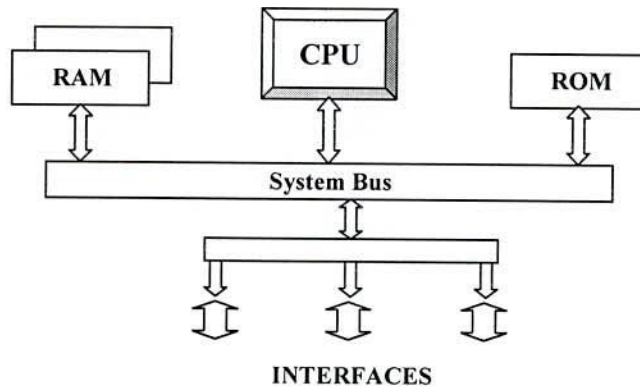


Figure 2.1: Software-Based Architecture.

2.2.2 Second-Generation Systems

After the mid 1990s, companies started to find new solutions to support high bandwidth and fast processing networking systems. Vendors used Application Specific Integrated Circuits (ASIC) and combined them with embedded Reduced Instruction Set Computer (RISC) processors yielding greater speed and performance. Companies that built high-speed network systems started to hire VLSI design engineers to design ASICs for their systems and products [2].

ASIC-based forwarding and switching have resulted in a new generation of very high-speed routers and switches. ASIC is an integrated circuit manufactured with embedded instructions to perform specific functions. The functions are programmed in silicon hardware permanently. So, for ASIC, since there is no memory instruction fetch cycle, it is significantly faster than software-based systems. It works at wire speed. That is, in the software-based architecture, the CPU must make memory accesses to execute instructions, and memory accesses take too much time compared with the execution time of ASICs.

With ASICs, manufacturers improved the performance by creating special chips that could do packet forwarding directly in the hardware. These chips make decisions about packet forwarding and, when packets need special treatment, they are forwarded to the RISC processor for special treatment. With very high packet forwarding speeds (approximately tens of millions of packets/second), routers became very inexpensive. They became common in academic and industrial networks [2, 4]. Now, one can buy sophisticated routers for \$100 or less.

ASIC technology became very popular because it can process packets at wire speed. But, after several years, its drawbacks started to be understood. ASICs are created by designing and fabricating networking functions into silicon permanently. In the meantime, Internet applications are becoming more complex. Thus, they need still more functionality. Some of those applications, such as Firewall Capability (stateful firewalls), Virtual Private Networks (VPN), and Quality of Service (QoS) implementation demand new processing capability from the network hardware [6].

To add a new function to an ASIC, we have to design and produce it from the beginning. This procedure takes from several months to two years. ASIC can be designed for several

different functions, but since those functions are embedded into silicon, adding new functions, or designing new ASICs are very expensive and time consuming [6].

To summarize, ASICs have numerous disadvantages: They are costly, require much time to market, and exhibit difficulties in simulation, design, and modification [2].

2.2.3 Third-Generation Systems

Network systems vendors can no longer afford to wait as long as two years designing and developing ASICs for an application. The network requirements could change during the development of a special purpose ASIC, and a lot of effort and money could be wasted [7]. The solution is Network Processors. Network Processors were introduced in the market in the late 1990s. Network Processors combine two approaches, hardware structure (about as fast as ASIC) and software that makes the system flexible.

Network processors are not for a special application. A vendor can produce different systems with different network functions with only one type of network processor. Today, designers can build a layer-3 unicast router; tomorrow designers can build a stateful firewall. Applications that are overwhelming for ASICs, because of the complex functionality, are implemented with network processors, such as Virtual Private Networks, firewalls, and Quality of Service mechanisms. These functions require more scalability, flexibility, and programmability.

2.3 Applications

The application of network processors is another area where abundant growth will take place. These processors will be targeted at "edge applications" and "backplane applications". Edge applications are where LAN and WAN protocol and packet translation frequently are required. Backplane applications are where a high concentration of source ports must each be forwarded to the appropriate destination port. Uses areas are Remote Access Servers, Web Switches, Internet Core Routers, Enterprise Routers and SOHO Intrusion Detection systems etc.

Remote Access Servers

These are servers that function as a large bank of modem to allow remote users to dial-in to access an enterprise network or the Internet. Recently, the access to the Internet through Internet Service Providers (ISP) has been a major focus of RAS. ISP's have customers wanting to do all kinds of different things. A RAS needs to be programmable to meet the needs but also faster and at a smaller cost per port. Current architectures have simplified line cards and a central switching point to route traffic. The line cards are inexpensive but this approach does not scale well. Traditional CPUs are designed based on locality of reference; they make the assumption that data recently referenced is likely to be referenced again in the near future. A cache helps make these systems run fast by keeping recently referenced information close to the processor. Thus the traditional CPU design bogs down in the face of constant streams of new data. A network processor takes a different approach. High-speed data movers collect data from network devices and move it directly to memory for queuing. While data is being moved to memory, it is selectively copied to special high speed engines that are able to parse the packet data, and make on the fly decisions about how to forward this data. However, as mentioned above in the [Table Lookup algorithms] section, network processors can benefit for specially designed memory cache to aid in the routing table computation to make decisions on how

to forward a given packet. Multiple high-speed engines allow the network processor to handle the many simultaneous threads of traffic flowing in a high-speed network. Sophisticated memory interfaces insure that the data can flow in and out of the processor's memory without creating a bottleneck.

Web Switch

Web Switches' are a new breed of network switch devices that help businesses and other content providers serve the needs of their clients. These switches delve deep into the network packets to determine not just what destination was intended, but also what application is being run, and what kind of transaction is being requested within the application. This information can then be used to make intelligent decisions about how to forward this traffic. The Web Switch needs to handle several different types of tasks. Packet handling from many connections must be processed at a fast rate. State information about each client connection is maintained so those subsequent packets for the session are forwarded to the appropriate server. Other network load and management processing is also done. This requires a device that is very flexible. Using network processors to build such a device because of their inherent nature of being programmable offers such flexibility.

Internet Core Routers

These routers serve as the backbone to the Internet. ATM technology dominates these routers now and is very fast and efficient for [Layer 2] switching. However, [Layer 3] and [4] forwarding is needed; this requires that the network packet must be investigated to make a more intelligent forwarding decision. A switch that can handle existing requirement, known new requirement and unknown future requirements is needed. A router built upon a network processor has the flexibility and performance necessary to succeed at such a task.

Enterprise Routers

In similar regards to the requirements of the Internet Core Routers, enterprises need [Layer 3] and [4] switching but also want a substantial reduction year to year in the cost per port of network devices. A system based on a network processor because of its programmability and low parts count can attempt to satisfy the needs of enterprises.

SOHO Intrusion Detection systems

Small Office Home Office (SOHO) network device could benefit from using a network processor in a device that not only connect the client to the Internet but also does intrusion detection. Most SOHO's can not afford high priced network equipment or technical staff to protect against intrusions. A system that was programmable and relatively inexpensive with a long life span would be ideal in this situation. Because intrusion techniques are constantly changing, a network processor and its inherent programmability point to an ideal match of problem and solution.

Intelligent Processing

Because the architecture incorporates flexibility by being programmable other areas of future direction will involve intelligent processing. This will take the form of system based on a network processor to be able to provide customers with features such as Quality of Service (QoS), IP billing, security and monitoring, and IPv6 to name a few.

Quality of Service (QoS)

This feature is to ensure that a defined level of service is provided for network traffic. Most prominently this is in terms of a constant data rate; so important in audio and video data transmissions. To accomplish this intelligent processing must be done in the network devices. Currently this would have to be defined in silicon (ASIC's) and therefore, not easily changed. With network processor doing the intelligent processing, the programming can be changed to meet specific needs.

IP Billing

Currently, all packets travel at "bulk rate" because routers cannot tell them apart. Routers that can analyze headers more thoroughly without compromising wire speeds will allow network operators to sort packets into digital equivalents of priority mail, first class, second class, etc... For example, in return for higher fees, the encrypted traffic on a corporation's virtual private network (VPN) could get a higher priority on an Internet backbone than the idle gossip in an AOL chat room. Such processing must be done without compromising wire speeds - and hence, the network processors.

Security and Monitoring

Another application of intelligent processing is to investigate packets in order to monitor their behavior and to detect security violations. This requires a delving into the packets to examine the contained information. By processing this information intelligent decisions related to security and monitoring can be made. This is made possible with the programmability and wire-speed performance of network processor based systems.

IPv6

The next revision of the Internet Packet (IP) protocol is called IPv6. This revision will incorporate many new features into the protocol; the most notable is the address range of IP addresses. The source and destination addresses will be 64 bits instead of the current 32 bits. Network devices that are not programmable will have to be replaced to take full advantage of the features in IPv6. With systems built upon a network processor the devices do not have to be replaced, only the software that controls them.

2.4 Generic Architecture

Network processor architectures make CPU architectures look staid and boring [31]. Figure 2.2 is a block diagram of a generic network processor. It does not represent a specific network processor, but includes traits common to most. These traits are:

- Multiple RISC cores
- Dedicated hardware for common networking operations
- High-speed memory interface(s)
- High-speed I/O interfaces
- Interface to general purpose CPU

Compared to general purpose processors, network processors do not need very sophisticated arithmetic and caching capabilities. Instead, they usually contain multiple execution threads running in parallel. Also specialized hardware units provide added functionality depending on the mainstream applications the network processor is targeted for. Packet processing is different from other applications intended for regular CPU in several ways [23]:

- It is I/O centric rather than process-centric.
- It has to be handled in real-time fashion.
- It consists of many simple tasks instead of few big tasks.
- It keeps states per flow instead of per application.
- It requires atomic access to shared data as it is more geared towards parallel processing than the usual CPU applications.

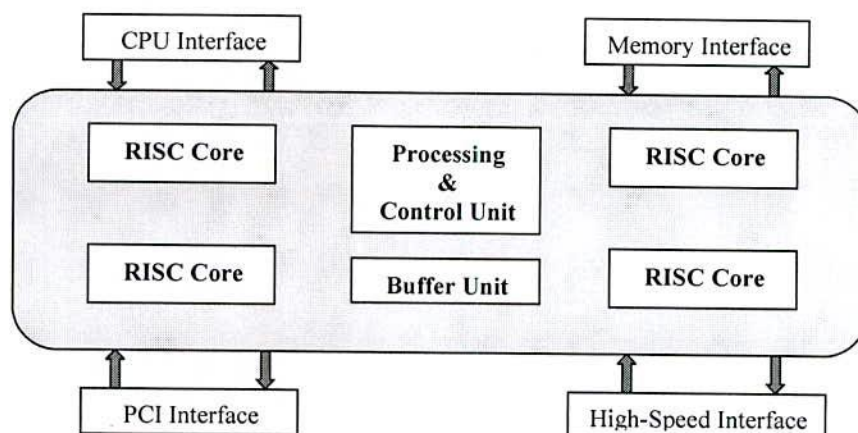


Figure 2.2: Generic Network Processor

Dedicated Hardware

Network processors, particularly RISC based ones; provide specialized hardware or integrated co-processor to perform common networking tasks. Typical hardware functionality includes Lookup Engines, Queue management, CRC calculation and Encryption Functions. In some Instance this additional hardware functionality may be targeted specifically for a particular application and would be of no use in a different application.

Network Interface

A key feature of Network processors is their network interfaces. This is where the data packets enter and exit the network processor. These interfaces connect to the Framers or MAC device of the line interface. (Although some network processors have integrated framers and MACs, in which case the network processor connects directly to the PHY). Some of the early network processor manufacturers developed their own network interface, which means designing new framers with this interface or providing a bridge chip. Now, however most network processors implement standard interface to the Framers and MACs, such as UTOPIA level 2 and 3, SPI-3 and for 10Gbps traffic SPI4.2.

Switch Fabric Interface

Many of the high-end network processors are targeted towards routing applications and provide an interface to a switch fabric. In general the switch fabric interfaces comply with a standard switch interface such as CSIX or may be SPI4.2.

Control Processing

The main function of the network processor is to process packet data at wire-speed. Certain packets known as control and management packets do not need to be processed at

wire-speed. These include exception packets and table updates. These packets in general have more complex processing requirements than the data packets, which are processed by the PPEs, would slow down the packet throughput. These packets are therefore separated out of the main data path and passed onto a separate processor known as the control processor. This processor may also be responsible other functions such as statistics gathering. A general-purpose processor normally implements the functionality of the control processing. Many network processors contain an integrate core specifically for control processing. Alternatively the control processing functionality will need to be provided externally via a host interface, such as a PCI bus, to a general-purpose processor.

Memory

Network processors require memory to store program code, lookup tables, packet data and queue information. Where feasible this memory is provided internal to the network processor. However, inevitable external memory will be required for many applications.

Most network processors will have at least two external memory interfaces, one to SRAM (Static Random Access Memory) and one to DRAM (Dynamic Random Access Memory). Accessing the external memory can cause a bottleneck; so many manufacturers are looking to the technologies to provide faster memory accesses, such as using DDR (Double Data Rate) or QDR (Quadruple Data Rate) SRAM, DDR SDRAM (Synchronous DRAM), FCRAM (Fast Cycle RAM) or RDRAM (Rambus DRAM). Packet data is typically requires large buffer storage, so in general it is stored in DRAM. Queuing information will usually be stored in SRAM.

The location of lookup tables depends on the size of the lookup table. large tables will need to be stored in DRAM, whereas smaller tables can be stored in SRAM or internally. Some network processors provide interface to CAMs (Content Addressable Memory), which allow very fast lookups. However, CAMs are still fairly small and so will limit the number of entries in the lookup table.

CHAPTER III

Data Networking and Packet Processing

3.1 Data Networking Models

Network modeling can be done in any one of the following two ways: either by modeling the data and the communications' protocols between the communicators, or by modeling the physical components of the network and their interconnections. Eventually, the two models converge into one representation of network modeling.

As data communications and telecommunications programming, interfaces, and equipment grew more sophisticated, the International Standard Organization (ISO) suggested a structured, layered architecture of networking called Open System Interconnect (ISO/OSI). At about the same time, the U.S. Department of Defense (DoD) offered another layered model that concentrated on data-network modeling (often called the Internet model, or more commonly, the TCP/IP model). These two models provide fundamental concepts in communications, and most systems and definitions use their language.

3.1.1 ISO/OSI Model

According to the ISO/OSI model, which is also called the seven-layer model, any two peered layers interact logically, carrying the relevant data and parameters, and executing the functionality of that layer. These layers actually interface with the layers above or below them (i.e., they hand them the data and parameters). The seven layers are shown in Figure 3.1.

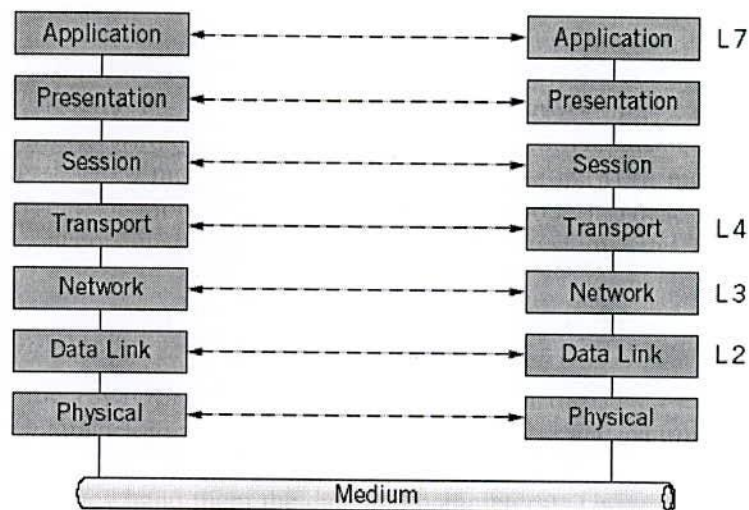


Figure 3.1: ISO/OSI seven layers model

The physical layer handles bits and the physical transmission of bits across the communication channel through some sort of medium (whether it be a kind of wire, fiber, radio-waves or light). The second layer (referred to as L2) is the datalink layer, which takes care of framing bytes or a block of bytes, and handles the integrity and error

recovery of the data transmitted at this level between two nodes connected by a physical channel. The third layer (referred to as L3) is the network layer, which is responsible for carrying blocks of data between two or more nodes across a network that is composed of multiple nodes and data-links; L3 responsibilities include the required addressing, routing, and so on. The transport layer is the lowest application (or host) layer that carries data between applications (or hosts), independently and regardless of the networks used. It is responsible for the end-to-end data integrity and reliability, and it works through either connection or connectionless transport mechanisms. The session layer controls the session (e.g., determining whether the relationship between the nodes is peered or master/slave; establishing, maintaining, and terminating a session). The presentation layer determines such things as the format, encryption, compression, structure, and encoding of the application data. The application layer determines the way the application uses the communication facilities, that is, e-mail, file transfer, and so on.

The upper four layers (the transport, session, presentation, and application) are considered the host layers, while the lower three (the physical, data-link, and network) are the network layers. The network layers are considered the most important in network processing; nevertheless, many networking decisions are made based on the upper four layers, such as priority, routing, addressing, and so on.

3.1.2 TCP/IP Model

The equivalent data-networking model of the DoD (often called the Internet model, or more commonly, the TCP/IP model), is simpler, and contains fewer layers. (It originally had only four layers, without the physical layer; see Figure 3.2)

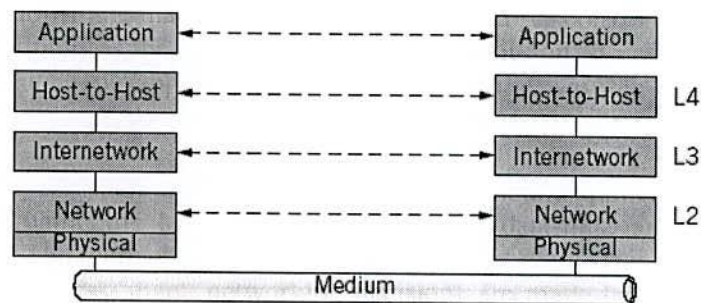


Figure 3.2: TCP/IP model

The ISO/OSI model layers are not mapped exactly onto the TCP/IP model layers; however, roughly speaking, the TCP/IP model shrinks all host layers into the host-to-host (transport) layer (L4), and adds a new, internetworking layer that is composed mainly of the ISO/OSI network layer (L3). TCP/IP's network layer (L2) is composed mainly of the functionalities of ISO/OSI's data link layer and some of its network layer. Recently, this model has been amended by a "half" (or a "shim") layer as new technologies have been introduced, so this model can better fit systems more precisely.

3.1.3 Data Networking

The way the applications and other entities are multiplexed in one host and demultiplexed in the target hosts is by using headers in each layer, and data encapsulation. In ISO/OSI and TCP/IP models, each layer's entity interfaces with the entities in the layer underneath

it by adding a layer-header describing the parameters required for the entity in the layer underneath to function properly. This header also lets the peered entity in the same layer in the other host to have the required information about how to process the data (according to the protocol). These headers are added as the data travels downwards through the layers, and they are removed as the data travels upwards. Figure 3.3 depicts data encapsulation in the TCP/IP model, and names the data units (Protocol Data Units, PDUs) that result; that is, datagrams in the application layer, packets in the IP layer, and frames in the physical layer.

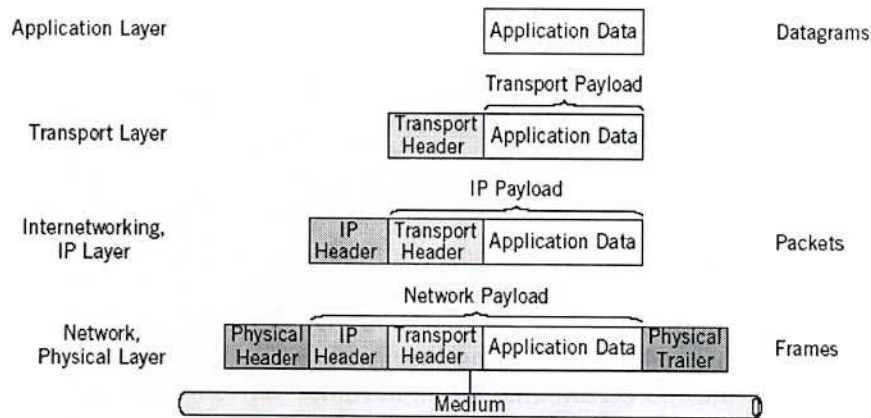
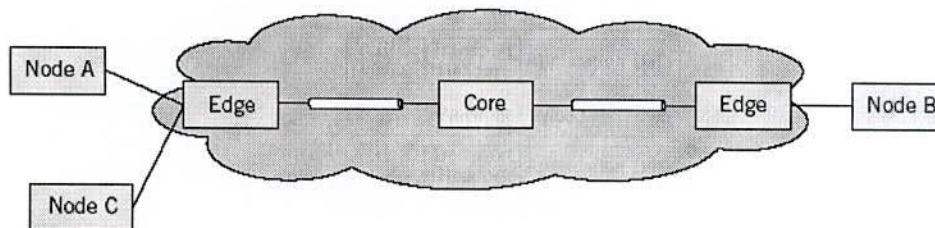


Figure 3.3: Data (payload) encapsulation

The other network modeling emphasizes the physical components of the network, and is composed basically of nodes and links. Some of the nodes are hosts or end systems (clients and servers, or peer communicators), and some are network edge devices or network core devices, as shown in Figure 3.4. Edge and core devices are gateways, routers, switches, bridges, hubs, and repeaters.



Node: Host PC, Server
 Edge/Core: gateways, routers, switches, bridges

Figure 3.4: Network Model

Network devices work up to the third layer (with the exception of gateways that work on all layers, and connect very different types of networks and applications). Generally speaking, repeaters and hubs work only on the first, physical layer, bridges, and switches work on the second layer, and routers work on the third layer. Thus, for example, routers interconnect networks and links by analyzing and forwarding packets based on the

headers of the third layer, as shown in Figure 3.5, where node A and router A are connected by one link (the left medium), and node B and router B are connected by another link (the right medium). The routers are connected by a third link (the middle medium).

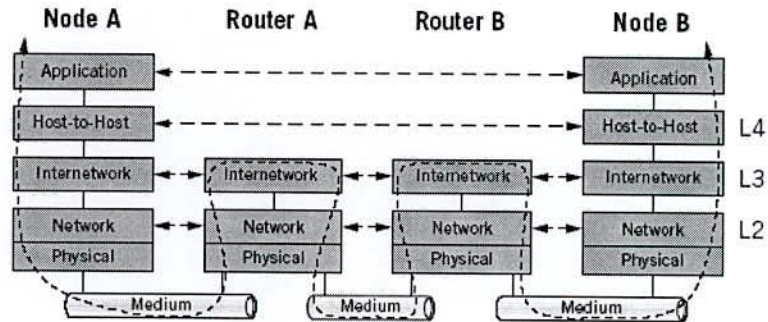


Figure 3.5: Router layers

Links can be replaced, or generalized, by networks, such that, for example, the routers are connected by networks in between them, as shown in the Internet model in Figure 3.6.

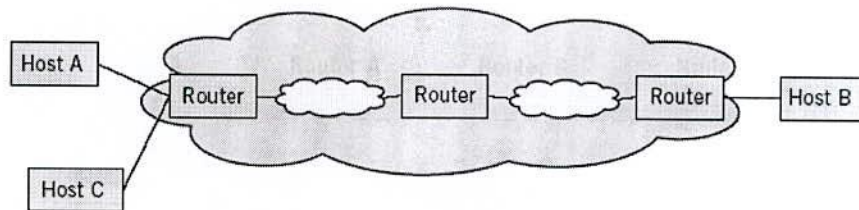


Figure 3.6: Internet Model

3.2 Packet Processing

Network processors are sometimes distinguished from packet processors by being programmable, whereas packet processors are just configurable. The programming paradigms, models, styles, and languages that are used for NPs are also very different from those used for applications running on general purpose processors. No interrupts are used in NPs, and the main principle is to use an event-driven control program, in which packets dictate how the software runs. Network processors perform several key functions, including:

- Parsing the incoming frames in order to understand what they are, and where to find the relevant information that is required for processing.
- Retrieving the relevant information from the frames, which may be complicated as encapsulation, variable fields length, and various levels of protocols may be involved.
- Deep packet analysis when required, such as understanding HTTP names, identification of XML protocols, and so on. This may be required for priority assignments for various kinds of traffic flows.
- Searching for related information in repositories; for example, routing tables, access lists, and so on.

- Classifying the frames according to the various forwarding and processing schemes that the frames should undergo.
- Modifying the frame's contents or headers, possibly adding tags, changing addresses, or altering the contents.
- Forwarding, which may potentially be coupled with various traffic management tasks such as metering, policing, shaping, queuing, scheduling, and statistics.

Packet processing tasks (or functions) include:

- Framing.
- Parsing and classification.
- Search, lookup, and forwarding.
- Modification.
- Compression and encryption.
- Queuing and traffic management (measurement, policing and shaping).

Packet processing can follow one of two paths:

- Data path (fast path).
- Control path (slow path).

Packet processing can be discussed according to direction:

- Ingress (entering the equipment or the network processor, from the network).
- Egress (exiting the equipment or the network processor, to the network).
- Combinations of Ingress and Egress.

3.2.1 Packet Processing Flow

Processing functions are separate tasks, each following the other. The process starts with the packet entering the network processor and immediately goes through framing, whose function is to make sure that the packet arrived correctly. (In the other direction, framing is the last task, and is targeted to ensure valid packet output.) The second phase is to parse and classify the packet, which simply means that the network processor must understand what the packet is, what type it is, and then must classify it according to the application requirements. Usually for this classification function, searching is required. The last function that the network processor carries is the required modification of the packet, which includes dropping the packet if required, multiplying it, or altering its content as required. Finally, transmitting the packet usually involves an extra function of queuing, prioritization, and traffic management of the packet to make sure that the receiver can receive the transmitted packet at traffic patterns that it expects. Queuing and traffic management sometimes happens inside the network processors and sometimes happens outside the network processors. Optionally, compression and encryption tasks are utilities that packets sometimes undergo and usually they are done outside of the network processor, although there are some network processors that contain an embedded security functional unit.

The main processing functions are classification of the packet (at real time or at wire speed), and searching for various values (e.g., next hop address) that correspond with some fields in the packets (e.g., IP address). These two functions have received extensive treatment in the industry, to the extent of special purpose search engine coprocessors, and the development of parsing and classification languages.

A general framework of the three primary aspects of packet processing is depicted in Figure 3.7. The packets enter from left, in the ingress direction, and take either the slow path (through some kind of upper level processing, for example, updating routing tables of the network processor), or the fast path (going through the network processor functions of searching, modification, etc.). The packets are then forwarded either to a switch fabric or to the network (line interface) again, in the egress direction.

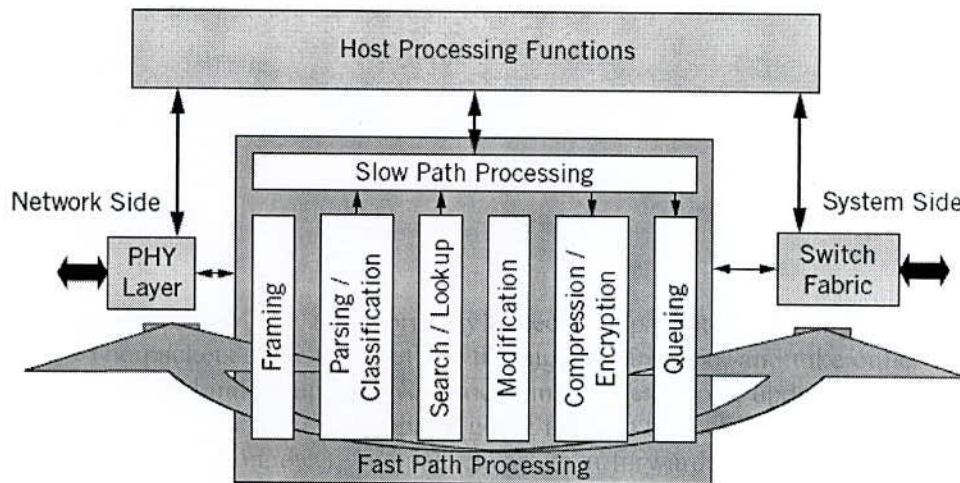


Figure 3.7: A general framework of packet processing

3.2.2 Ingress and Egress

The packet processing flow is divided into three stages: ingress, switching and egress. The ingress stage is responsible of receiving packets arriving into the system. The switching stage is responsible for the transfer of packets from packet processing units on the ingress to packet processing units on the egress. The egress stage is responsible of transmitting packets exiting the system, See Figure 3.8.

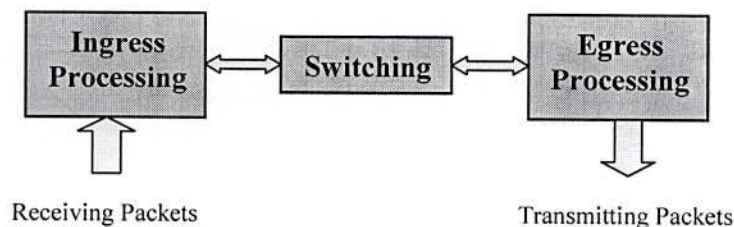


Figure 3.8: Ingress and Egress Processing Configuration

In the Ingress configuration, the processing performed in the ingress queue includes such operations as checksum calculation, TTL updating, destinations address swapping and address lookup. After all the required processing operations are performed, packets are transferred to the Packet Buffering Unit, through the switch matrix. The Packet Buffering Unit places the packet on the output queues.

In the Egress Processing Configuration, the only information extracted from the packet on the input queue is the destination address. The information is used to determine the destination port and the packet is transferred to a Protocol Processor on the output queue through the switch matrix. Any needed processing is then performed on the assigned Buffer Processor on the output queue.

3.2.3 Framing

Received or transmitted frames should receive “framing” treatment, in order to assure that the correct and full packets or datagrams can be extracted from these frames. This means that incoming frames should undergo correctness tests (to make sure that the entire frames’ bits are received without error), correcting attempts if required and integrity checks (to make sure that all packets’ content arrived). Outgoing packets should be fragmented or segmented as required and “framed” correctly, that is, adequate headers should be attached or altered, proper terminators (trailers) should be appended or modified, and error detection and correction information should be added.

3.2.4 Parsing and Classification

After a complete valid packet is received and verified, the next step of packet processing is to look at the incoming packets in order to classify them for various treatments, as the processing requirements dictate. This step involves two combined subtasks: parsing and classification. Packet processing has two basic architectures, or design philosophies, that are crucial for parsing and classification implementation; these architectures are store-and-forward and cut-through.

In store-and-forward architecture, packets are first received in their entirety, stored temporarily, examined, analyzed, processed, and then, after a decision is made regarding them, transmitted, and the memory is cleared. Store-and-forward allows a complete, even complex treatment of the packet, before the packet is injected back into the network. This comes, however, at the expense of having to deal with buffering issues and delaying the packet, that is, adding end-to-end delay to the packet flowing from the ingress of the processing unit to its egress, due to the increased latency incurred by the buffering time that is required before processing starts.

The shortcomings of the store-and-forward are remedied by the cut-through architecture, where processing begins as packets flow into the processing units, and continues as bits continue to come in. The packet is transmitted after the required analysis, and the decision and processing is done “on-the-fly.” This can be accomplished by examining specific bit patterns in various fields of the incoming packet, that is, parsing and classifying, as well as other decision and processing tasks that must be carried out in real-time. Although cut-through saves buffering and latencies, it allows only simple analysis, decision and processing tasks, and might even cause network overhead (e.g., transmitting a packet that eventually turned out to be invalid, bad CRC, or, because a premature decision was made based on the first bits of the packet, the output-channels were loaded in vain, forcing packet retransmits).

Examples of these two architectures are, on the one hand, in low-end routers that use store-and-forward architecture (no “wire-speed” constraints), and on the other, VLAN or MPLS switches that use cut-through architecture, based on a tag in the beginning of the

frame that allows forwarding decisions to be made instantly, while the packet is still inflowing.

Parsing

Each incoming packet must go through some sort of parsing to examine and understand what it is as well as its requirements, and then it must be classified, or handled according to its type and its required processing. Parsing therefore is the first analysis and action done on the packet content. Parsing is basically identifying the relevant fields in the incoming packets, according to their place and type, and picking the field's values for continuing the parsing process, or using these values for classification. Therefore, parsing and classification are tied together and sometimes are not separated. A simple parsing example in an IPv4 packet would be to detect its destination IP address.

Classification

Classification means categorizing packets into "flows," in which they are processed in a similar way by the network entities. These flows are defined by rules that the packets obey, and the collections of these rules are called classifiers. The rule database contains many entries, each of which is composed of a pair of a specific rule description and its appropriate action. These specific rules are matched with the incoming packets, and the best match determines the appropriate action to be taken on the incoming packet. Very often, the action is to mark the incoming packet with a notation, so that a subsequent process will take the appropriate action, based on this notation.

A specific case of packet classification is packet forwarding, which deals with searching and packet lookups. In packet forwarding the rules are represented by the packet destination address fields, and the action is simply to forward the packet to its appropriate destination. In traffic management literature, classification is sometimes considered part of the traffic management process, when it classifies packets solely for traffic management purposes.

3.2.5 IP Lookup and Forwarding

The importance of rapid IP lookup cannot be exaggerated in the context of packet processing. The most common IP lookups are done for forwarding. IP packet forwarding is executed in any router or switch, where forwarding decisions have to be made in order to find the address of the next hop router and the egress port to be used to send the packet through. In most cases, IP forwarding is based on IP addresses. IP address lookups are obviously based on the IP address, that is 32-bit or 128-bit keys (IPv4 or IPv6, respectively), whereas other lookups can have much wider keys that can reach hundreds of bits, composed from multiple fields in the IP (or layer 2) packet.

Almost every packet processing activity starts with an IP-lookup, and therefore, in high-speed networks, the speed of IP-lookups is critical. Huge efforts are made to accelerate this specific task in any network processor or, as a matter of fact, in any networking device. In a 10 Gbps network, for example, the required time to forward a packet of minimal size is <50 ns; therefore, this is the maximum allowed time for looking for a match (or a longest match) of a specific IP address in a table of hundreds of thousands of entries (as well as doing some other things in this time frame).

3.2.6 Modification

In all applications other than packet forwarding, modification of the packets is eventually the purpose of packet processing. Sometimes, some packet modification is required even for forwarding applications. For example, changing the IP header may be required in the time-to-live (TTL) field, as well in the IP addresses, hence recalculation of the checksum header is also required, and so on.

Packet modification can include all of these operations:

- Modification—Changing the contents of a packet (usually changes in its header, or changes both in the header and in the payload, as a result of some processing, e.g., compression or encryption described in the next section)
- Deletion—Some of the packet contents or headers are deleted (e.g. de-encapsulation of packets)
- Adding—Additional information is added to the packet (e.g., encapsulation, authentication information)
- Canceling the entire packet—Simply removing the packet from the system (e.g., exceeded traffic, wrong addressing)
- Duplicating the packet—Copying the entire packet (e.g., multicasting, port-copy operations)

Since checksum is often changed in packet processing, some network processors contain an internal checksum functional unit to assist with maintaining the right checksums.

In addition, traffic analysis, some management tasks and other applications can be performed in the modification phase, although most of these operations are expected to be performed in the control path. Statistics collection, however, must be done in the fast path, at the modification phase, to accommodate the packet rate.

3.2.7 Compression and Encryption

Compression and encryption processing are optional phases in packet processing that are more typical to access network processors than to high-end network processors. The reason is that packets do not undergo any compression or encryption in the main trunks of the core and the metro networks, whereas these processes might happen at the edges of the network, at the access points. Usually, compression and encryption is executed by specific coprocessor, or in the mid-range and access network processor that contains an encryption functional unit.

Compression is used mainly in places where bandwidth is critical (mainly wireless applications), and most of the compression efforts focus on TCP/IP header (many protocols run very small packets, while most of these packets are just predictable headers).

Encryption is used for privacy, data integrity, and authentication to confirm the communicating parties' identities. There are many standards for security in the IP world; the main standard is Internet Protocol Security (IPsec) framework, operating at layer 3, and SSL/TSL operating above this layer. IPsec is used mainly in Virtual Private Networks

(VPN), whereas SSL/TSL is used typically in client server applications (mail, web browsing, telnet, and so on).

The security protocols primarily use encryption for confidentiality, where encryption simply transforms unsecured information (“plaintext”) to coded information (“ciphertext”), using some key and a transformation algorithm. The other components of the security suit are to make both sides know what encryption algorithm to use, and what key to use, and some specific protocols are doing just that.

3.2.8 Queuing and Traffic Management

Finally, the processed packet ends its quick journey in the network processor, and is about to leave. The last small but very complicated task is to decide how to get rid of this packet, or, more precisely, how to pass it on to the receiving party (equipment or communication link) on its way to its final destination.

As previous stages in the packet processor (classifiers, modifiers, and so on) determined the output path (port), priority, and some handling parameters, the traffic management process forward this packet to an appropriate queue, and schedule it for transmission according to the conditions of the lines, the receivers, and the parameters that this packet has.

In many cases, incoming packets also have to go through traffic management (metering, queuing, prioritization, and so on, which are determined by incoming port or communication lines or other parameters) in order to be processed in the network processor according to predefined scheduling scheme.

3.3 IP Addressing, Routing and Forwarding

IPv4 addressing is based on a 32-bit address field. The 32 bits of the IPv4 address are usually represented by the decimal value of each of the bytes in this 32-bit address, separate by a dot, that is, 131.44.2.1 means: 10000011001011000000001000000001, or 832C0201 (hex). The IP address is further broken down to network address (or net ID) part, sub-network (or subnet) part, and host address (or host ID, which is the physical machine connected to the network); routing and forwarding takes place among the networks and the sub-networks.

The principal of routing and forwarding is quite simple: A routing table maintains all the addresses of the next hop and interfaces (ports) that should be used to forward an incoming packet. The chosen interface and next-hop address are functions of the incoming IP packet (its destination network address, as it appears in the IP destination address field of the header of that packet).

The network address part in the IP header went through some changes over the last decades of Internet use. Up until 1993, in what was later called classful (or class-based) networks, the IPv4 address was categorized in four address classes (A to D), with one reserved class (E). Originally, the network address part of the IP address was defined by the IP class. Class A address always starts with a “0” bit in its MSB, that is, addresses 0.0.0.0 to 127.255.255.255 and the network ID is defined by the remaining 7 bits of the first byte of the IP address (i.e., a total of 128 networks). Class B addresses start with a “10” bit pattern in the MSB bits, and the remaining 14 bits of the first two bytes are the

network ID, that is, class B contains addresses 128.0.0.0 to 191.255.255.255, and about 16 thousands networks. Class C address starts with a “110” bit pattern, and the remaining 21 bits of the first three bytes of the address define the network ID. Class C addresses thus are in the range of 192.0.0.0 to 223.255.255.255, organized in roughly 2 millions networks. Class D is used for multicast addresses, and their IP addresses always start with an “1110” bit pattern, followed by the remaining 28 bits for the multicast address. This class therefore is in the range of 224.0.0.0 to 239.255.255.255. The last class, E, is reserved, and its IP addresses start with an “11110” bit pattern. Some IP addresses (networks) are non-routable, and are reserved for specific use (as outlined in Table 3.2).

The rapid use and growth of the Internet and the consumption of IP addresses resulted in a severe shortage of IP networks and addresses with unique IP network addresses. Two million networks, most of them containing about 250 hosts, were simply not enough. To cope with this unbearable situation, many solutions were suggested, starting from many intermediate solutions to extending the IP address to 128 bits (IPv6). These intermediate solutions turned out to be so efficient that IPv6 was no longer required so desperately. However, the use of these various mechanisms to expand the original IP address space makes things a bit more complex. These mechanisms may include Border Gateway Protocol (BGP), Classless Inter Domain Routing (CIDR), and Network Address Translation (NAT). The most influential mechanism in solving the shortage of IP networks is NAT, but it is not impacting addressing mechanism, routing or forwarding. The most influential change in addressing, routing and forwarding is due to CIDR, which is described in the following.

Classless Inter-Domain Routing added hierarchy to the network addresses by defining subnets with a prefix length noted /n (where n defines the number of initial bits in the IP address that should be considered as the network ID and sub-network), or using a network mask. This hierarchy also allows aggregation (also called summarization or supernetting) or defines a range of networks by a simple notation and thereby shrinks the routing tables, and reduces routing advertises throughout the Internet. It also created the potential of overlapping address ranges, or more importantly, inclusions of address ranges within broader address ranges, a situation we call exceptions. This makes it possible to specify a broad range of addresses in a routing table in one entry, and to define networks that are exceptions within this range by additional entries in the routing table (that are providing specific routing rules).

An example will clarify this concept: IP address 192.168.16.0/20 (or 192.168.16/20 for short, or 192.168.16.0 with a network mask 255.255.240.0) means all IP addresses from 192.168.16.0 to 192.168.31.255 (the first 20 bits of the address are masked, which leave the last 12 bits unspecified):

Table 3.1: IP Address Example

	IP Address	Network Mask
Regular notation	192.168.16.0/20	255.255.240.0
Hexadecimal presentation	C0 A8 1X XX	FF FF F0 00

Obviously, 192.168.16/20 also contains 192.168.24/21, 192.168.16/21 (which are two sub-networks having together the same span of 192.168.16/20), and 192.168.27/24. Actually, any IP address with network mask bigger than 255.255.240.0, or prefix length

bigger than 20, of the same initial IP address C0 A8 1x xx, will be part of this network range. This means that in a routing table, a specification of a route (next-hop and interface) can be defined to a small range of IP addresses within the range of another group, having another specification route (next-hop and interface).

Table 3.2: Example of Routing Table

Row	IP Address	Next hop	Interface
1	192.168.16/20	10.10.1.1	2
2	192.168.27/24	10.16.54.2	3
3	192.168.0/17	10.16.1.6	1
4	0.0.0.0/0	10.1.1.0	7

Assume, for example, a small routing table as shown in Table 3.2. IP destination address 192.168.156.5, for example, matches only row 4, by definition, the default routing. IP destination address 192.168.1.3 matches only row 3 (and row 4, by definition). IP destination address 192.168.18.1 matches rows 1 and 3 (and row 4), and IP destination address 192.168.27.56 matches all rows. This overlapping, or network exception, raises the question of how to route packets of IP destination 192.168.27.56.

Here comes the longest prefix match (LPM), or actually, the best matched prefix search. The longer the match is (with regards to the prefix length), the better the routing that will be chosen for this packet. In other words, a router needs to search more than just the right (matched) prefix; the router has to find the most specific match, which is the longest matching prefix in our case. The algorithm used to search for the best match is a bitwise comparison of the addresses, or more specifically:

Algorithm:

Assume the routing table like the one given in Table 3.2 (always, the last row [entry] should be the default route, that is entry 0.0.0.0/0), and an IP destination address, for which a next hop and interface is searched. Make sure the routing table is sorted by the network mask (the prefix length), from the most specific (the longest prefix) to the least specific network (the shortest prefix).

For each row (entry) in the routing table do

```

{
    if (IP destination address AND entry's network mask == entry's IP address)
    {
        Use the entry's next-hop and interface
        Exit
    }
}

```

Please note that the above algorithm will always stop at the last row of 0.0.0.0/0. One of the biggest issues in maintaining such routing tables is the need to keep these tables sorted by the prefix lengths. In a large routing table, adding an entry of /31 for example (close to the top of the table), might require a shift for large portions of the table.

IPv6 addressing is based on an address field of 128 bits long, allowing $3.4 \cdot 10^{38}$ IP addresses (compared to the $4.3 \cdot 10^9$ IP addresses of IPv4). An IPv6 address is presented by

8 groups of 4 hexadecimal digits (16 bits) each, the total 128 bits represents in a notation like 0123:4567:89AB:CDEF:0123:4567:89AB:CDEF. The first 64 bits usually refer to the network (or subnet) address, whereas the last 64 bits are the host ID (usually drawn from the MAC address of the machine). IPv6 addressing uses the CIDR concept, that is, a range of addresses can be written by an address with a prefix length (/n, where n is the length of the network prefix). IPv4 addresses are recognized by an IPv6 addressing mechanism, for backwards compatibility, by attaching the IPv4 address to the ::/96 IPv6 prefix (all zeros), for example, ::192.168.0.1. Table 3.3 summarizes all unique and special addresses that usually are not routable, in IPv4 and IPv6.

Table 3.3: IPv4 and IPv6 Non-routable Addresses

Address	CIDR	Purpose
0.0.0.0 – 0.255.255.255	0.0.0.0/8	Zero address
10.0.0.0 -10.255.255.255	10.0.0.0/8	Private IP address
127.0.0.0 – 127.255.255.255	127.0.0.0/8	Localhost/Lookback address
169.254.0.0 – 169.254.255.255	169.254.0.0/16	Zeroconf/APIPA
172.16.0.0 – 172.31.255.255	172.16.0.0/12	Private IP address
192.0.2.0 – 192.0.2.255	192.0.2.0/24	Documentation and Example
192.88.99.0 – 192.88.99.255	192.88.99.0/24	IPv6 to IPv4 relay Anycast
192.168.0.0 – 192.168.255.255	192.168.0.0/16	Private IP address
198.18.0.0 – 198.19.255.255	198.18.0.0/15	Network Device Benchmark
224.0.0.0 – 239.255.255.255	224.0.0.0/4	Multicast
240.0.0.0 – 255.255.255.255	240.0.0.0/4	Reserved
0000:0000:0000:0000:0000:0000:0000:0000	::/128	Any address
0000:0000:0000:0000:0000:0000:0000:0001	::1/128	Localhost/Lookback address
	FC00::/7	Unique local unicast address
	FE80::/10	Local address (like zeroconf)
	FF00:/8	Multicast address

3.4 Search Engines

In order to cope with demanding search operations—the many search operations per packet in complex applications carried out in high speed communication trunks that can end up in billions of operations per second, and the necessity to search at wire speed. Search engines are software processes, hardware circuitry, or a combination of the two, sometimes designed as a functional unit on the network processor, and sometimes packed in a designated chip, search processor or coprocessor. The search engine functionality is simple: it returns a value (a result) when presented with a value (a key).

The simplest search engine is plain memory, where it returns a value when presented with the address of that value. As a matter of fact, memory returns a value contained in an address when presented with that address. The opposite operation happens with a special type of memory, called associative memory, or Content Addressable Memory (CAM), which, when presented with a value, returns a value that is the address of the presented value.

More complex search engines, as mentioned above, are special hardware assists. These search engines are internally based either on CAM memories, or on search algorithms materialized by hardware circuitry that use regular memories inside these search engines, or are attached to them.

These external search engines (sometimes called network search engines [NSE], or search accelerators) can support, as of 2008, billions of searches per second, with search keys of hundreds of bits wide among millions of entries. These search engines can interface with the network processor like a regular memory, and work transparently with the network processor, or by using the standard LA-1 interface.

There are many NSEs that look like hardware search engines to the network processor. These can be categorized into algorithmic NSE and those that indeed have hardware-based schemes that carry the searches within a few clock cycles. Some of these hardware-based schemes might also be used inside the network processor. Content Addressable Memory (CAM) is by far the most common way to execute searches by hardware.

Before discussing CAMs, however, another word about search engine metrics: the hardware search engines (CAM included) have the same performance metrics that we used before (that is size of the data store and the number of searches per second that can be achieved). Nevertheless, hardware-based search engines should also be examined by their power consumption and the additional chip count required for their implementation. In CAM's case, for instance, power consumption is a real challenge.

CAM is a hardware search component that enables searching in a single clock cycle. It is a memory composed of conventional semiconductor memory (usually like the static random access memory). CAM simultaneously compares a key (the searched data) against all of the table's entries stored in the CAM, and is thus capable of returning the address of the entry that matches the key in one single clock cycle. Although CAMs are fast, efficient, and flexible in terms of lookup functionality, they are also big energy consumers due to the large amount of parallel active circuitry. They are also big physically due to the large silicon area required for the memory cells and logic; they are small in terms of capacity compared to other types of memories; and they are very expensive.

A simplified scheme of a CAM having m words, each consists of n bits, is depicted in Figure 3.9. The CAM core cells are arranged into the horizontal words, where each bit is a CAM core cell that contains storage (the bit value) and the comparison circuitry. Typical implementations of CAMs have 36 to 144 bits per word, and as many as from 128K rows (words) to 512K rows in recent CAMs. A key (a searched word) is broadcasted vertically on the CAM core cells, through the search lines. Each stored word has a horizontal match line that indicates whether the stored word is identical to the searched word (activated match line) or not (a mismatch status). This indication is the result of a logic "and" between all comparisons of the searched word bits with the CAM core cells of a row (stored word); any mismatch in any of the core cell of a row causes the match line of that row to be pulled down to indicate mismatch. All match lines are sensed and amplified separately, and fed into an encoder that produces the location (address) of the matched key (the searched word) in the table.

There are two kinds of CAMs, that is regular (or binary) and ternary. Binary CAMs (BCAMs) store and compare binary bits, that is zero or one. Ternary CAMs (TCAMs) support an additional don't care bit, which causes the match line to remain unaffected by the do not care CAM core cell, regardless of the searched bit. This ability of TCAM can be used for masked searches, and is perfectly useful for IP lookup applications, or complex string lookups, as described in the following.

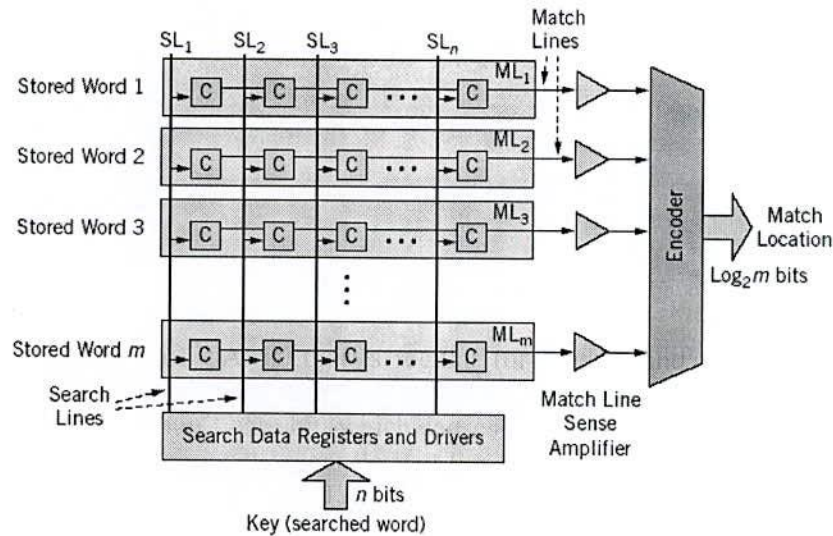


Figure 3.9: Simplified CAM scheme

It is possible in a BCAM to find more than one match between the search word and the stored words. In such a case, instead of using a simple encoder, as depicted in Figure 3.9, a priority encoder is used, which selects the highest priority location, usually defined by its address in the table (a lower position in the table grants higher priority). TCAM, on the other hand, works a bit differently. First of all, masking of bits usually creates more hits, as the chance of hits increases when fewer bits are compared. Second, multiple priority mechanisms can be used, emphasizing not only location, as in the BCAM case, but also the number of consecutive matched bits in each of the hits or the total number of matched bits along the entire word width. Figure 3.10 provides an example for clarification.

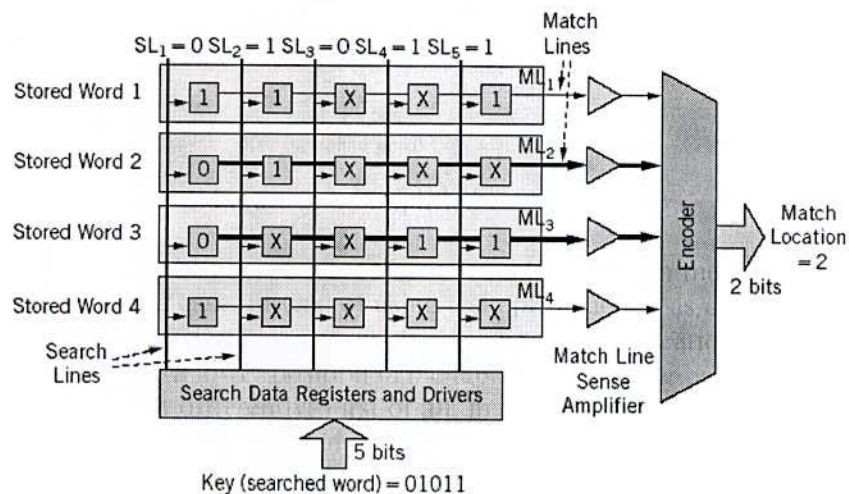


Figure 5.19: TCAM priority mechanism

The search operation begins with putting all the match lines temporarily in the match state (high). Then the search line drivers broadcast the key 01011 on the search lines to all CAM core cells, which then compare their stored bit with the bit on their corresponding search lines. Cells with a mismatch put the match line in the mismatch state (pulling it down). The match line sense amplifier gets the word match result, which is if the match line was pulled down by any bit, it has at least one mismatch. In Figure 3.10, match lines 2 and 3 remain activated (high), indicating a match, while the other match lines were pulled down (both because of bit 0), indicating a mismatch. The encoder receives two matches, and produces the search address location of the matching data. In the case that the encoder chooses the first match line, it will generate the match address 2. In the case that it chooses the longest prefix, it will also generate 2. But if it chooses based on the number of hits, then it will generate the address 3. In the two later cases, the encoder requires more circuitry to enable counting the hits and the places of the hits.

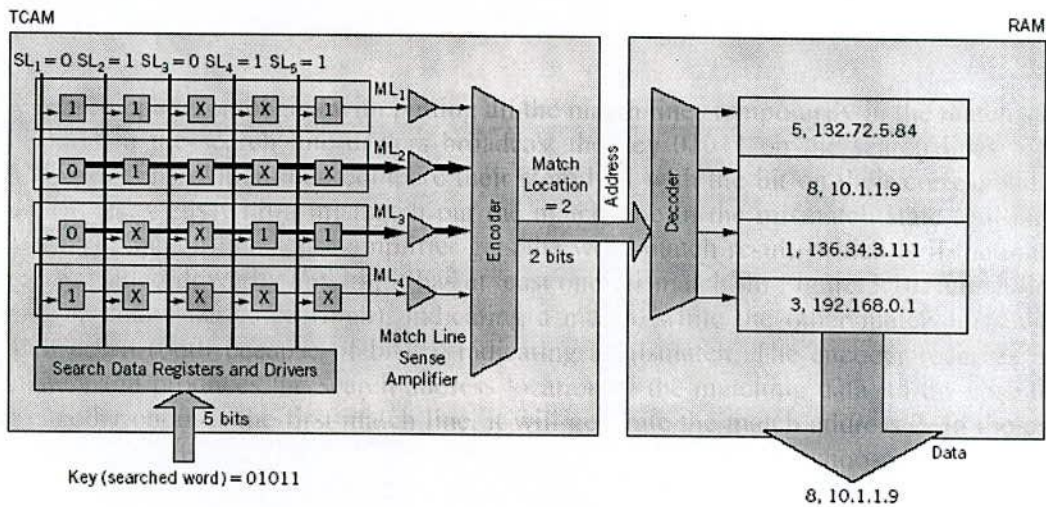


Figure 3.11: CAM/RAM system for IP address lookup

In most implementations, the output of the CAM is used to drive a regular RAM for receiving a value attached to the key that was searched in the TCAM. A common example, shown in Figure 3.11 is IP address lookup, using an IP address as an input key to the TCAM, while the resulted TCAM address location of that IP address is used to look in a RAM for a forwarding port and next hop IP address, which match that IP address. The output data from the RAM (forwarding port and next hop address), addressed by the match location of the TCAM, is associated with the TCAM key (IP address in example). The combination is very powerful, enabling a single cycle searching, and two cycles updating of this dictionary lookup system.

Using TCAMs for IP lookup is done by storing all routing prefixes in decreasing order of their prefix length, padded with “don’t care” bits in the rightmost side of the stored words. For example, a routing prefix 132.72.0.0/16 will be stored in the TCAM after routing prefixes that are longer than 16 are stored, and it will store the first 16 bits of this routing prefix and another 16 “don’t care” bits to their right.

3.4.1 Simple Switching and Forwarding

When a simple decision is required for switching, forwarding, or queuing a packet, based on few numeric keys of a small range, then a small table residing in memory (RAM) is sufficient to make such decisions. This table can be direct addressable if the number of entries is about the same as the key-range, or hash table if the number of entries is small compared to the range of the keys. An example for using a direct addressable table for look-up is VLAN (Virtual Local Area Network) forwarding, for example, assigning an output port based on the VLAN. An example for using a hash-table for look-up is MAC addresses based-decisions, for example, assigning MAC addresses to specific VLANs, bridging devices operating in layer 2, and so on.

3.4.2 IP Address Lookup

This is the most common usage of searching in network processing. Generally speaking, there are several reasons for IP address searches. Searching in the routing table in the classful IP network environment was straightforward: In order to receive the next hop and the output interface, an exact match search of the fixed length network ID (according to the address class) should be performed. These tables were not very dynamic, that is they were not changed very often. Hash tables were very efficient in performing this task. The impact of CIDR on IP address lookup, however, was enormous; from fixed sized, hash table lookups, search took place in variable length prefixes of aggregated entries. The focus shifted to tree, tries, and LC-tries data structures, algorithms and their variants that best matched the IP address lookup in terms of data distribution and design, simultaneously with using TCAMs and other hardware search engines (coprocessors).

CHAPTER IV

Architecture and Methodology

4.1 Network Processing Trends

There are several different architectural solutions for packet processing. Each having some advantages and disadvantages. General-purpose micro-processors are used for their flexibility to adapt to protocol changes in the field and short time to complete the software development, but do not have enough performance to process data at wire rates. On the other hand, hardwired (custom) chips are designed to process packets at wire rates, but are not flexible. They are difficult and expensive to modify, to add features, fix bugs or adapt to rapidly changing network processors. Network processors are proposed as a solution to both these problems. They are processors optimized to perform packet processing at wire rates and are programmable and therefore flexible. There is a trade-off between performance and flexibility in these three solutions (General-purpose micro-processors, Network processors, dedicated ASIC). General-purpose micro-processors are very flexible, but don't have enough performance, network processors are less flexible but have much better performance and finally, dedicated ASICs are not flexible but can perform at wire rates.

In semiconductor technology, the philosophy of network system design has changed [21] (Figure 4.1). In 1995, networks employed traditional routing/switching devices, such as a general purpose CPU, a packet processing engine, and a forwarding engine. By 2000, hybridized architectures were common. Such systems consisted of a general purpose CPU and a custom ASIC or an off-the shelf application-specific standard product (ASSP), or a combination of these devices. After 2001, technology-driven systems consist of a dedicated control CPU and a full-fledged application-specific network processor.

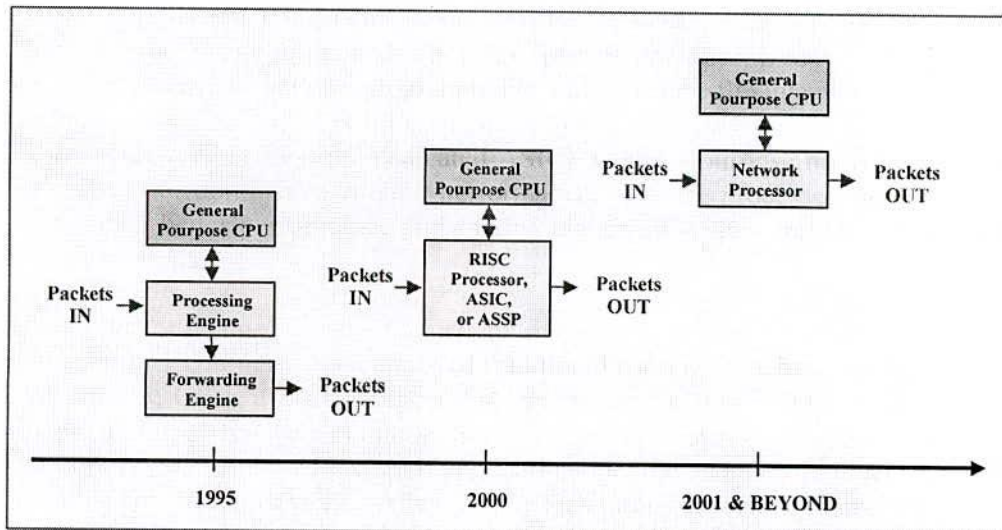


Figure 4.1: Evolving the Network Design System Philosophy

This migration of system architecture has also shifted Network Processing functionality from hardwired solutions to programmable solutions (Figure 4.2). Note the change from

fixed to programmable media, as well as the change from ASIC/ASSP technology to a single-chip (Network Processor) solution.

The target of every network processor is to perform packet processing with the flexibility of a microprocessor, but with the performance of a dedicated ASIC. There are two issues that concern network processors' performance. The first one is packet (data) processing and alternative solutions that improve data processing performance [16]. Network processing requires transfer of packets and additional information to and from large memories with large access latency. Therefore, memory latency and the alternative ways [16] to "hide" it, is the second issue.

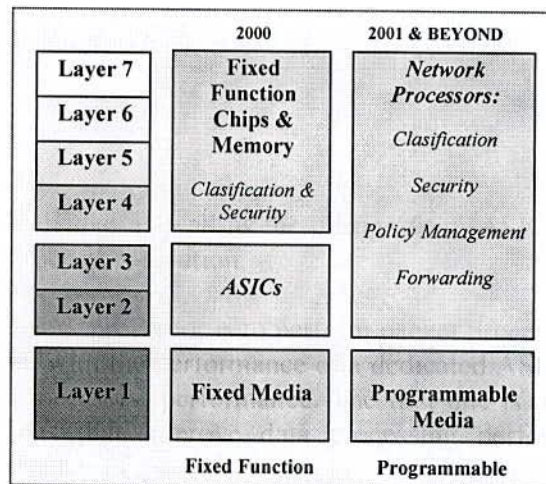


Figure 4.2: Graphical comparison between network processor-based systems and their hardwired counterparts

4.1.1 Network Processing Functions

Not all network devices have the same processing requirements. However, a lot of similarities exist. Roughly a network processor describes the packet processing duties of a router and a web switch. Routers are the workhorses of the Internet. A router accepts packets from one of several network interfaces, and either drops them or sends them out through one or more of its other interfaces. Packets may traverse a dozen or more routers as they make their way across the Internet. Here is a simplified version of the IP routing algorithm:

- Remove the link layer header
- Find the destination IP address in the IP header
- Do a table lookup to determine the IP address of the next hop
- Determine link layer address of the next hop
- Add link layer header to packet
- Queue packet for sending
- Send or drop packet (if link is congested)

Web switches, by contrast, are a new type of network device. They address the problem of trying to increase the responsiveness of a popular web site by using more than one web server. A web switch can direct incoming HTTP requests to different servers based on a variety of networking parameters, including the URL itself. For instance, all secure HTTP

requests could be forwarded to a special web server with cryptographic hardware to accelerate those requests. Here is a simplified web switch algorithm:

- Accept incoming TCP connection (three-way handshake)
- Buffer incoming TCP data stream (TCP/IP protocol)
- Parse the stream to find the URL being requested
- Do a table lookup to determine where to forward the request
- Open TCP connection with web server (three-way handshake)
- Send buffered request (TCP/IP protocol)

Note that, for a given bandwidth, the web switch processing requirements are much higher, and require much more state than the router processing requirements. The difference arises because a router processes packets, but a web switch processes connections.

4.2 Architectures of Network Processor

Network processors can be categorized according to their use and the way they have evolved. These categories can significantly impact the architecture of the network processors. The three main categories of network processors are:

- Entry-level network processors, or access network processors, which process streams of up to 1 to 2 Gbps packets, and are sometimes used for enterprise equipment. Applications for such access network processors include telephony systems (e.g., voice gateways and soft switches), xDSL access, cable modems, wireless access networks such as cellular and WiMAX, other narrowband and broadband wireless networks, and optical networks. A few examples of such network processors are EZchip's NPA, Wintegra's WinPath, Agere, PMC sierra, and Intel's IXP2300.
- Mid-level network processors (2–5 Gbps) contain two subgroups of network processors (NPs): legacy NPs and multiservice NPs, which usually are used for service cards of communication equipment, data center equipment and Layer 7 applications (security, storage, compression, etc.). In the legacy subgroup, one can include the classical, multipurpose NPs like AMCC, Intel's IXP 1200 and 2400/2800 NPs, C-port, Agere, Vitesse, and IBM's NP. Examples of multiservice and application (Layer 7) network processors are Cavium, RMI, Broadcom, Silverback, and Chelsio.
- High-end network processors (10–100 Gbps) are used mainly for core networking and metro networking, usually on the line cards of the equipment. Examples of such network processors are EZchip's NPs, Xelerated, Sandburst (who was bought by Broadcom), and Bay Microsystems.

Each of these categories was either designed for specific applications and thus has specific architecture patterns that support these applications or was designed using almost general-purpose processing elements in their cores.

The architecture of network processors can be described in many ways, and the three ingredients common to computers (processing unit, memory, and I/O). An extended

general framework for classifying network processors includes the following five dimensions.

- Parallel processing approach
 - processing element level
 - instruction-set level
 - bit level
- Hardware assistance
 - coprocessors
 - functional units
- Memory architectures
- Network processor interconnection mechanisms (i.e., on-chip communications)
- Peripherals

4.2.1 Basic Architectural Approaches

Hardware engineers use three basic techniques to achieve high-speed processing: a single processor with a fast clock rate, parallel processors, and hardware pipelining. Figure 4.3 illustrates packet flow through a single processor, which is known as an embedded processor architecture or a run-to-completion model. In the figure, three functions must be performed on each packet.

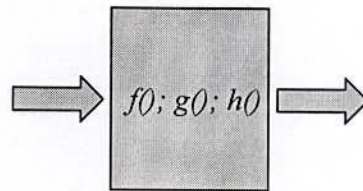


Figure 4.3: Embedded processor architecture in which a single processor handles all packets

Figure 4.4 illustrates packet flow through an architecture that uses a parallel approach. A coordination mechanism on the ingress side chooses which packets are sent to which processor. Coordination hardware can use a simplistic round-robin approach in which a processor receives every Nth packet, or a sophisticated approach in which a processor receives a packet whenever the processor becomes idle.

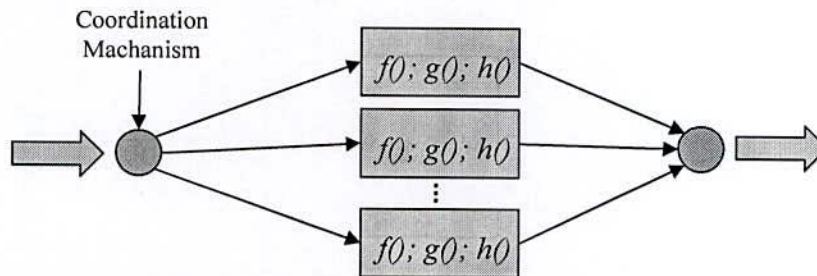


Figure 4.4: Parallel architecture in which the incoming packet flow is divided among multiple processors

Figure 4.5 illustrates packet flow through a pipeline architecture. Each packet flows through the entire pipeline, and a given stage of the pipeline performs part of the required processing.

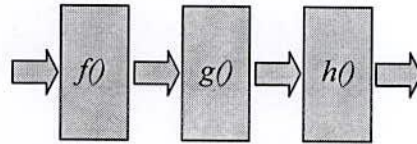


Figure 4.5: Pipeline Architecture in Which Each Incoming Packet Flows Through Multiple Stages of a Pipeline

Pipelining and parallelism can be combined to produce hybrid designs. For example, it is possible to have a pipeline in which each individual stage is implemented by parallel processors or a parallel architecture in which each parallel unit is implemented with a pipeline.

4.2.2 Parallel Pipelines of Homogeneous Processors

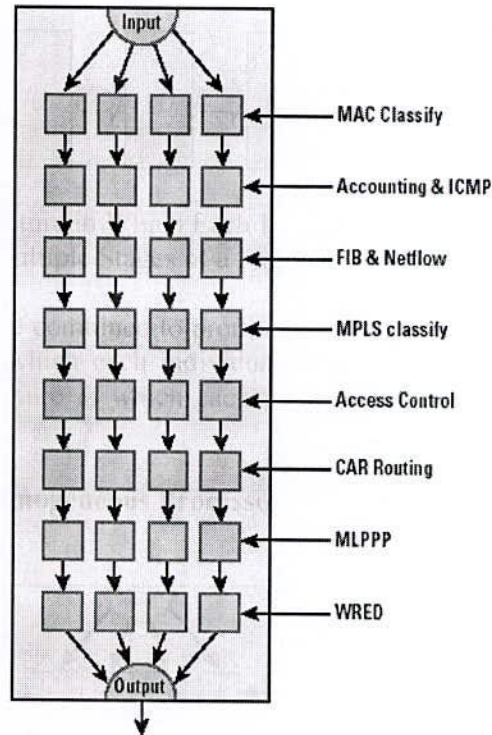


Figure 4.6: An example architecture that uses parallel pipelines of homogeneous processors (Cisco)

One of the more interesting designs employs parallel pipelines of homogeneous processors. Figure 4.6 illustrates the architecture of the Cisco chip. When a packet enters, the hardware selects one of the pipelines, and the packet travels through the entire pipeline.

4.2.3 Pipeline of Parallel Heterogeneous Processors

EZchip Corporation sells a network processor that combines pipelining and parallelism by using a four-stage pipeline in which each stage is implemented by parallel processors. However, instead of using the same processor type at each stage, the EZchip architecture employs heterogeneous processors, with the processor type at each stage optimized for a certain task (for example, the processor that runs forwarding code is optimized for table lookup). Figure 4.7 illustrates the architecture.

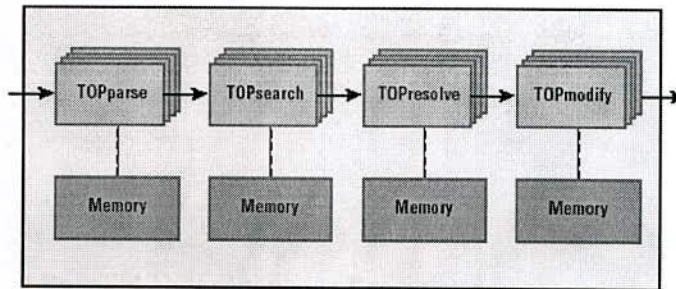


Figure 4.7: An example architecture that uses a pipeline of parallel stages with heterogeneous processors (EZchip)

4.2.4 Superpipeline and Superscalar Architecture

A single processor core containing multiple execution units is referred to as a Superscalar processor. Such a device is capable of issuing one instruction to each execution unit per clock cycle, greatly improving the performance of the core. The core is pipelined so each instruction executes over several clock cycles. Several instructions overlap; each being at a different stage of execution each clock cycle. This creates complications due to dependencies in the instruction flow. The packet processing tasks are pipelined, passing packets from TOPparse to TOPsearch to TOPresolve to TOPmodify [15]. The superpipelining and superscalarity of the core architecture provide the network-specific processor with massive processing power.

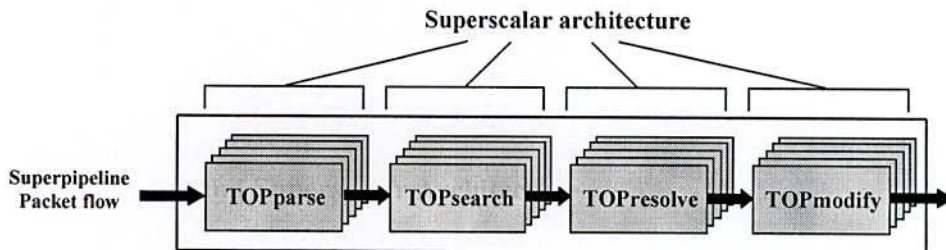


Figure 4.8: EZchip TOPcore superpipeline and superscalar processor architecture

Benefits of TOPcore technology:

The task of providing 7-layer packet processing at wire speeds for next-generation equipment presents huge challenges. EZchip's Task Optimized Processing Core (TOPcore) technology has the following advantages [15]:

- Ten-fold faster performance through modification of instruction sets and data paths.
- Intelligent 7-layer packet processing at Gigabit and Terabit wire speed.
- Scalable and expandable architecture.
- Flexible and programmable processing.

EZchip Technologies is designing a line of network-specific processors based on its TOPcore technology. The EZchip network-specific processor is a single chip IC that combines an array of numerous TOP processors optimized for 7-layer switching at 10 Gigabit speeds [15]. EZchip's network-specific processors can be applied to high-speed switches and routers at both the backbone and the edge.

4.3 Design Methodology

The design philosophy that the thesis implement to design a High-speed network processing system are pointed bellow.

- Maintain a 4 level Layered architecture
- Used pipelined Memory architecture
- Add a Load balancer with an efficient load distribution algorithm
- Design a super-pipeline and parallel packet Processing unit

4.3.1 Layered Processing Architecture

From the analysis of various incoming traffic to network processor, it is found that all packets are not follow the same processing strategy. A basic processing step is enough for all packets and some traffic required more additional processing and few traffic needs more extra processing. So, all packets do not need to pass through all processing steps. This thesis implements the concept and developed a hierarchical 4 level layered processing architecture, shown in Figure 4.9.

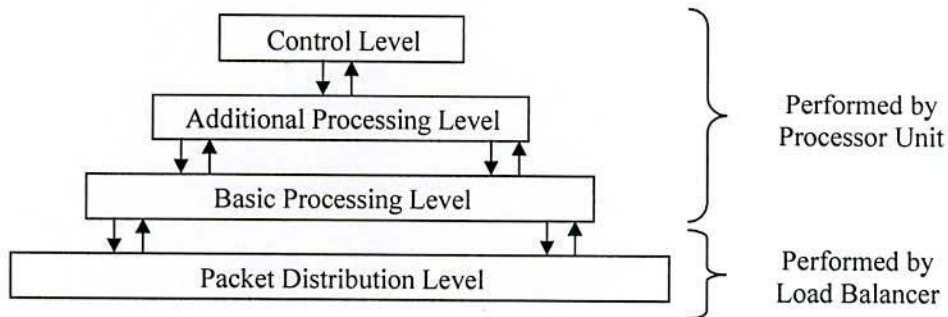


Figure 4.9: Layered processing architecture

In this layered architecture the level of hierarchy is packet distribution level, basic processing level, additional processing level and control level. Packet distribution is performed on a dedicated level and therefore does not reduce or limit packet processing. Packet distribution is therefore limited only by the Packet Distribution Level. The Load Balancer unit is work in the packet distribution level with using load distribution algorithm. Packet processing is performed on the Basic Processing Level and the Additional Processing Level. Processing Control Unit controls the processing sub unit to perform these levels. As packets flow through the Packet Distribution Level, packets are

queued for processing in Packet Buffers. The destination of the packet on the Packet Distribution Level is determined by the lookup performed on the Basic Processing Level if available or the Additional Processing Level alternatively. The number of clock cycles available for packet processing is determined by the number of clock cycles each packet can be held for processing in the Packet Buffer without reducing the packet throughput.

4.3.2 Pipelined Memory Architecture

Internally, the processor contains several CPUs. The packet stream is divided among CPUs, with each processor working on multiple separate packets (threads) at the same time. When the processing for one packet stalls because of a memory reference, a CPU resumes processing for the next thread. The main drawback of this design is that it is built around traditional memory, which is optimized for latency rather than throughput. To achieve high memory bandwidth, multiple processors and thread contexts are used to generate multiple pending accesses. The problem is that each processor does all of the forwarding for an assigned packet, and each independent processor needs independent access to a large amount of shared state (e.g., the global forwarding database). The only known way around this problem is to either replicate the state for each processor or to share the state in some fashion. Replication is prohibitively expensive for the large problem sizes that routers need to deal with memory. And another problem is the lack of memory bandwidth.

Memory pipelining is a solution to overcome these problems. This Scheme avoid these problems as follows: First, make each memory access wide to increase the number of bits retrieved per access. Second, observe that a memory subsystem consists of memory banks and an interconnect that ferries data read from a bank to the output. Traditional memory maintains the invariant that two readers do not access the same block by ensuring that there is only one reader in the entire subsystem. However, this is unnecessarily restrictive. In summary, this memory subsystem uses:

Wide Words: The performance of many network algorithms can be increased significantly by the use of wide data words. Instead of having multiple independent readers of memory, one reader at a time through wide word access can provide aggressive memory bandwidth.

Internally Pipelined Memory: By internally pipelining the memory to allow the memory system to work on multiple packets at once can greatly increase the throughput.

4.3.3 High Throughput Pipelined Memory

While multithreading and pipelining are very standard ideas in the network processor context, one of main contributions is showing how to get high worst case memory throughput by internally pipelining the memory. In the traditional memory layouts that prior network processors are designed around, only one or two (if multi-ported) memory lookups can be performed at the same time in the worst case. These designs use large memory tiles, trying to minimize the interconnect delay for latency, and rely on common case techniques such as locality and independent accesses to provide sufficient bandwidth. As such, the worst case throughput of such systems is limited to one or two transactions being processed by the memory at the same time, when, in the worst case, all the lookups fall into the same bank.

In contrast, the new memory design trades off latency for throughput. Here propose pipelining the memory design using smaller memory tiles to process many transactions at the same time (see Figure 4.10). This will result in a higher latency per transaction, but the number of transactions per unit time will be significantly higher than in a traditional memory design. To achieve this, design is start by breaking the memory into smaller tiles, each of which is capable of reading out a wide word each memory cycle [32]. The cycle times for these smaller banks are two to three times faster than monolithic memory banks, even assuming optimal internal partitioning. The design will have a larger interconnect delay, and hence a larger overall latency; however by pipelining the interconnect, it is possible to achieve better overall throughput because the delay of each stage is smaller than the overall latency of a traditional (non-pipelined) memory design. Therefore, the memory design can have N lookups in flight, where N is the number of pipeline stages (twice the height of the tree to go up and back down) of the memory layout.

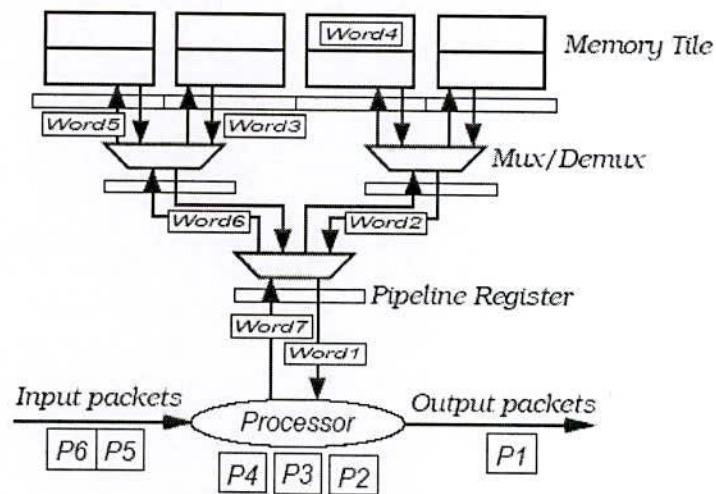


Figure 4.10: Pipelined wide-word memory architecture

A high level diagram of the design is shown in Figure 4.10. It shows how the pipelined tree-based memory can have many accesses flowing through it at the same time. At any given time, there is only one wide word access going to a memory bank, and all the memory banks are at the leaves of the tree. At each level in the tree (for the single ported version) there are at most two words in flight, one going up to the memory arrays and one coming back down.

In the Figure 4.10 the memory has been divided into many small banks. At the bottom of each small bank is a pipeline latch that grabs the data. Data read from a bank then needs to travel from the tile (which could be quite far away) back to the processor. This is done over a deeply pipelined tree interconnect. When an access is made, a lookup is performed in parallel at the leaves of the tree, and each cycle the correct data item marches its way down through the pipelined tree to the root where it is read out by the processor. Since the tree is balanced and full, and there is only one memory access actively accessing the banks at a time, so it is guaranteed that during the entire journey over the interconnect there will be no conflicts. This ensures that design will always be getting one wide word of data each cycle. The tree interconnect is not a trivial piece of real estate, because it needs to mux, demux, latch, and move large words of memory over a long distance [32].

4.3.4 Load Balancing for High-Speed Links

In Figure 4.11, two parts of the load balancer are distinguished: A receive (Rx) part, which distributes incoming packets from the high-speed link across the NPs, and a transmit (Tx) part, which re-joins the processed packets onto a single link to the switch.

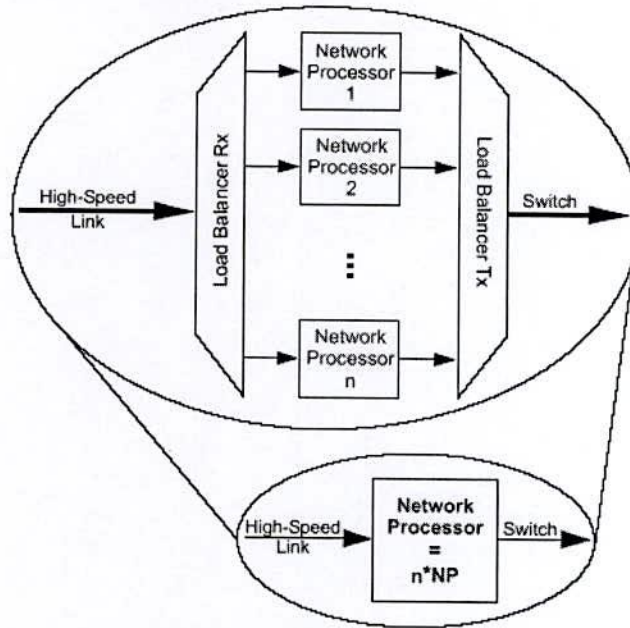


Figure 4.11: Load-balancer system

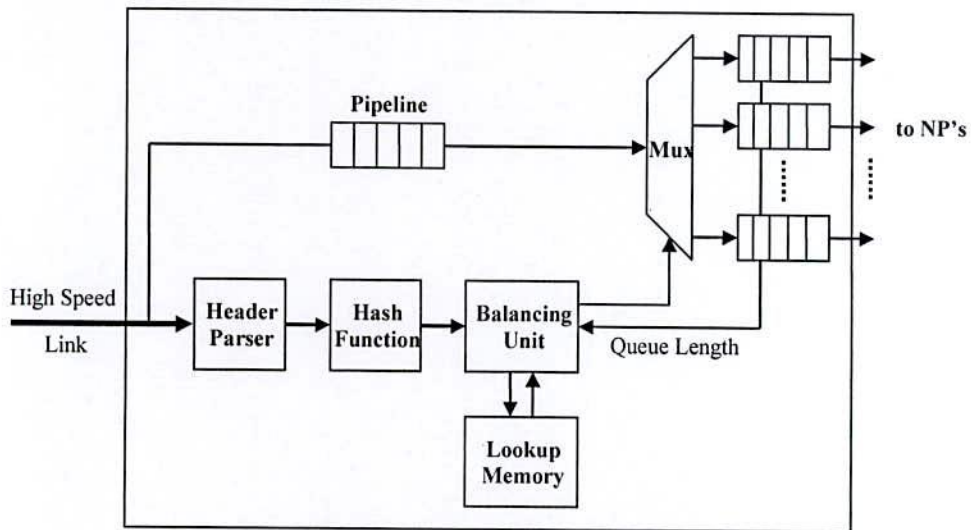


Figure 4.12: Diagram of Load-balancer

Figure 4.12 shows the internal structure of the receive part. Upon entering the load balancer, a packet is first inspected by a header parser. Here, the fields in the packet header that uniquely identify a flow are extracted. In the case of TCP or UDP over IP for instance, this is the five-tuple consisting of source and destination addresses, both layer-4

ports, and transport protocol number. These fields are then compressed by a hash function to an index of fixed length. Even with an ideal hash function, there will be hash collisions as soon as the active flows outnumber the available hash indices. The index serves as an address into the lookup memory of the balancing unit, where the number of one of the output queues is stored. Compared with direct hashing to a queue number, this two stage approach allows dynamic revisions of the flow-queue assignments.

The balancing unit decides to which NP a packet will be sent based on the information stored in the lookup memory and on the length of the individual queues towards the NPs. To be implementable in the high-speed domain with today's memory access latencies, the balancing algorithm has to be designed such that it does not require more than one read and update of a table entry per packet. For a packet size of 40 bytes (TCP acknowledgments) on a 10-Gb/s link that leaves a memory access time per packet of less than

$$t_{pkt} = \frac{40 \text{ Bytes}}{10 \text{ Gb/s}} = 30 \text{ ns}$$

and at 40 Gb/s this is further reduced to 7.5 ns.

The balancing unit controls the inverse multiplexer, through which packets are enqueued. During the decision making process, the corresponding packet is delayed in a pipeline until the queuing decision is available. The queues are implemented in a shared buffer to allow a certain tolerance of temporary queue imbalances [19].

The transmit part of the load balancer towards the switch in Figure 4.11 reunites the packet streams from the NPs. Implementation is straightforward because the NPs do not deliver more traffic than the switch link can carry and hence no congestion can occur. The NP traffic only has to be multiplexed. This can be realized with queues for speed adaptation, and a simple packet-scheduling algorithm such as deficit round robin.

4.3.5 Load Distribution Algorithm

If for a longer period of time no data arrives from a particular flow, it can be assumed that this flow has terminated. If that is true for all flows in a flow bundle, the association between the flow bundle and a queue can be changed safely without negative impact on the traffic. No packet reordering can occur after the last packet, and flow statistics in the NPs expire at the end of the flow.

```

for each packet {
    flow ← lookup(packet.id)
    if (currentTime - flow.timeStamp > timeOut) {
        flow.targetQ ← minQ()
    }
    flow.timeStamp ← currentTime
    packet.targetQ ← flow.targetQ
}

```

This fact is exploited by storing a time stamp for each flow bundle ID in the lookup memory, which is updated every time a packet arrives with that ID. When the next packet with the same ID comes in, the stored time stamp is compared with the actual system

time. If the difference is greater than a configured time-out value, the association with an NP has expired and is replaced by the NP with the currently shortest queue (minQ). In this way, newly initiated flows are assigned to the least-loaded NP, which smoothes out long-term imbalances in the load distribution.

Although the flow time-out mechanism helps avoid overload situations for the NPs, it does not prevent them completely. Its effectiveness depends on the frequency of flow expirations, i.e. the fewer flows time out, the less smoothing takes place. Also, burstiness of traffic and sustained data-rate changes of flows over their lifetime can lead to short-term load imbalances between queues, which cannot be handled well with flow time-outs only.

The proposed solution is a variation of sender-initiated adaptive load sharing [5] with a global task queue. In our case not only single threads are migrated but entire flow bundles are redirected away from overloaded queues, because all packets of a flow share their complete context. A queue is considered to be overloaded if it occupies more than a configurable share of the shared buffer. This condition is tested for every incoming packet. If the output queue to which a packet should be sent exceeds the buffer share threshold, then the NP currently associated with that flow bundle in the lookup memory is substituted with the minQ. The packet and all following packets of that flow bundle are sent to the new queue. This is called flow reassignment.

```

for each packet {
    flow ← lookup(packet.id)
    if (flow.targetQ.size > qThreshold) {
        flow.targetQ ← minQ()
    }
    packet.targetQ ← flow.targetQ
}

```

Flow reassignments also help solve the problem of accommodating a flow that exceeds the free capacity of any individual NP, although it is less than the combined free capacity of all NPs (Figure 4.13). This can be a consequence of an even load distribution. Now, the large flow is assigned to one of the queues. Because the old flows in that queue combined with the new one exceed the capacity of one NP, the queue starts filling up until it reaches the buffer share threshold. Then, the reassignment mechanism redirects flows from that queue until the overload is relieved. In this way, all flows can be served.

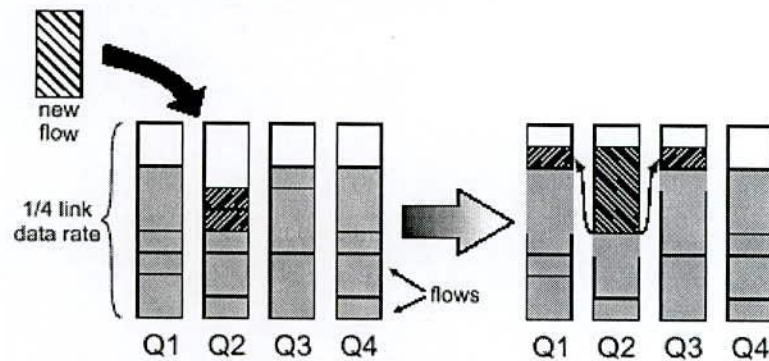


Figure 4.13: Large flow accommodation through reassignments

Packets from excessive flow bundles could simply be dropped. But because the throughput of all NPs together should match the data rate of the link and thus be able to handle all packets, this seems to be an unnecessary degradation of service. An alternative is to send packets from an excessive flow bundle to the currently shortest queue. As the minQ will change over time, this leads to the packets being distributed across several NPs. We call this packet spraying.

```

for each packet {
    flow ← lookup(packet.id)
    packet.targetQ ← flow.targetQ
    if (currentTime - flow.resetTime > tMeasure)
    {
        if (flow.count > countThreshold) {
            flow.count ← flow.count * reduction
        }
        else {
            flow.count ← 0
        }
    }
    flow.resetTime ← currentTime
}
if (flow.count > countThreshold) {
    packet.targetQ ← minQ()
}
flow.count ← flow.count + packet.size
}

```

Obviously, some packets might be reordered, but for receivers that are able to resequence packets this service is preferable to simply dropping packets. Moreover, statistics in the NPs for sprayed flows will not be consistent. If policing is an issue here, then the NPs should enforce policies earlier, well before a flow approaches the maximum throughput of an NP [19].

Packet spraying, on the other hand, provides a means to degrade service in a controlled manner for otherwise problematic flows. And most importantly, only these problematic flows will be affected, while all others are isolated and experience normal forwarding. Nevertheless, even problematic packets are forwarded.

4.4 Proposed Architecture

The total architectural design of the proposed Network processor is shown in Figure 4.14. In this design packets arrived at the high-speed media access control (MAC) interfaces and then goes to incoming memory queue. The flow of packets then scheduled for processing to the processor units through a load balancer. Load balancer actually distributes packet flow to all processor in a balanced load using load distribution algorithm. Here multi-processor approach is used and each processor has their individual pipeline fashion for processing a packet. Besides this, incoming packet flow goes to processor unit through another pipeline fashion for parallel processing. Every processor is directly connected to the pipelined memory interface for accessing the memory. External memory interface is also connected to all processor for especial purpose when needed.

After processing each processor transfer the packet to outgoing memory queue. Then the packet flow goes to its destination through high-speed MAC interfaces.

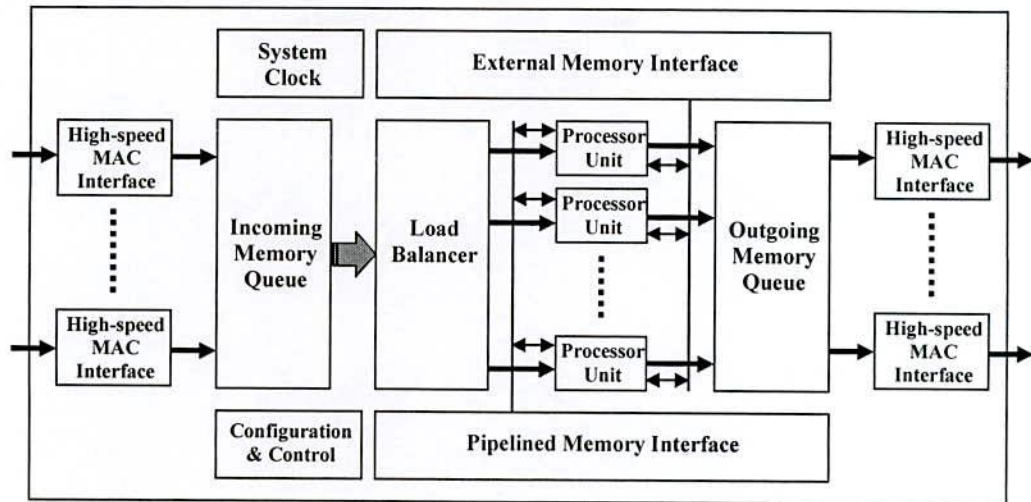


Figure 4.14: Block diagram of proposed Network Processor

4.4.1 High-Speed Ethernet MAC

The High-speed Ethernet MAC uses a single intellectual property (IP) core operating at 10, 100, or 1,000 Mbps. The 10/100/1000 Ethernet MAC core operates in full duplex mode, and supports termination and generation for transparent (switching) and full Ethernet frame (NIC or line card) applications.

Figure 4.15 shows the diagram of the Ethernet MAC interface. When an application data is come in the port it goes to a first in first out (FIFO) data structure then a control circuit controls the receiving and transmitting data separately through individual RX/TX controller. A configuration circuit is used for configure the MAC to operate as 10/100/1000 mbps rate.

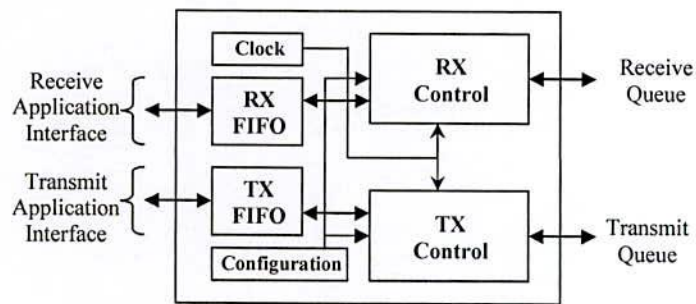


Figure 4.15: High-speed Ethernet MAC Interface

4.4.2 Load Balancer

Load Balancer scheme is already discuss earlier in section 4.3.3. The balancing unit decides to which NP a packet will be sent based on the information stored in the incoming

memory queue and on the length of the individual queues towards the NPs through a distribution algorithm described in section 4.3.4. The diagram of the Load Balancer is shown in Figure 4.16.

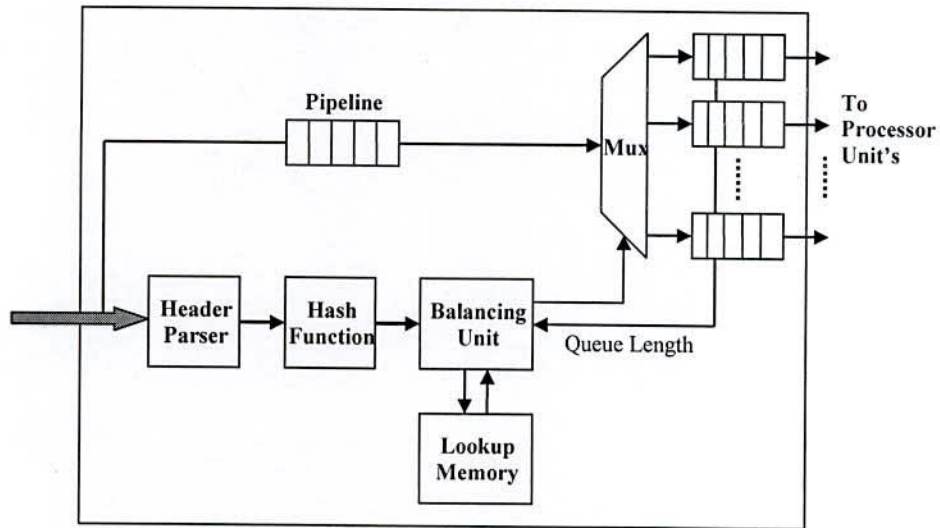


Figure 4.16: Load balancer diagram

4.4.3 Processor Unit

Actually all packet processing involves four basic tasks – parse, search, resolve and modify. Network specific processors optimize each of these tasks to boost the processing performance. Figure 4.17 shows the design. In the design EZchip's TOPcore technology is used for getting high-speed (at giga rate) packet processing with some modification. Modification is made for further speedup of processing capacity. In EZchip's technology each processor have their own memory module but here in this design a single pipelined memory is used which is directly connect to all individual processor unit. The four task-part of this processor unit are process packets through in a pipelined fashion. Parallel modules are connected with a synchronized manner in a superscalar architecture. The processing control unit controls the processing operation with generates some control signals to individual task part for the processing.

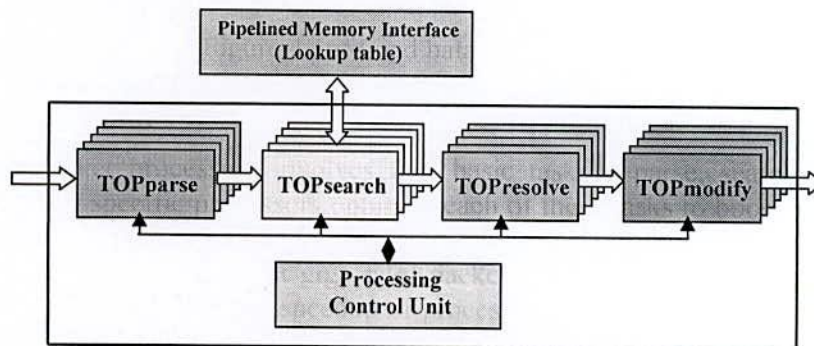


Figure 4.17: Proposed modified architecture of processor unit

TOPparse identifies and extracts the various headers and fields within the packet. It handles all seven layers including fields with dynamic offsets and length. TOPsearch performs the various table look-ups required for Layer 2 switching, Layer 3 routing. This TOPsearch uses a variety of search algorithms, optimized for various searched objects and properties. These algorithms feature innovative enhancements to hash tables, trees and CAMs. Multiple searches using different search methods can be applied simultaneously to yield wire speed throughput. TOPresolve assigns the packet to its appropriate output port and queue. It forwards the packet to multiple ports for multicast applications. TOPresolve also gathers traffic accounting information on a per flow basis, for network usage analysis and billing. TOPmodify modifies packet contents in accordance with the results of the previous stages. It modifies relevant fields, e.g. VLAN assignments, Network Address Translation (NAT), QoS priority setting and more.

4.4.4 Increasing the Throughput of Common Operations

There are several ways of increasing the throughput of common tasks or operations: using application specific components, pipelining and parallel processing. These enhancements can be applied on various structures to enhance performance. Application-specific components can be introduced to perform operations such as caching, table lookups and queuing. Pipelining can be introduced in almost any application specific component, local interconnect and processor core to enhance its efficiency. Parallelism can be applied to structures and operations such as buses, instruction processing and caches among many others.

In order to enhance the performance of common tasks, A Highly Adaptive methodology [3] is designed in a hierarchical structure. A three level hierarchy for packet processing has been devised. All packets are processed at Level 1, while only a small number of packets require Level 2 processing and only rarely does a packet require Level 3 processing. In the processing hierarchy, these Level processing is performed by processor unit and controlled by processing control unit shown in Figure 4.17.

4.5 Overall Power Model

The principal components considered in the power calculations are:

- Processor ALUs
- Processor clock
- Processor instruction and data caches (level 1, on-chip)
- Off-chip memory and I/O bus

ALU Power Model:

ALU power depends on the voltage, V_{dd} ; processor clock frequency, f ; the ALU utilization, a_{ALU} ; and capacitance, C_{ALU} :

$$P_{ALU} = C_{ALU} \cdot V_{dd}^2 \cdot a_{ALU} \cdot f \quad (4.1)$$

Using Wattch, the capacitance for .35- μ m technology (the process specification of an Alpha 21264 [7] simulated by Wattch) can be obtained as 310 pF. V_{dd} for this case is 2.5 volts.

The value for a_{ALU} (that corresponds to the ALU utilization, p_{ALU}) used by Wattch is 1. As discussed later, this value is used to verify the analytic power model by comparing model results with the results obtained from Wattch. However, by using a value of 1, the Wattch simulator assumes that the ALU is busy on every cycle. This is not true during stalls due to cache misses. Thus, the value used in the optimization studies (as contrasted with the power model verification work) is obtained from Equation ($a_{ALU} = p_p$) and reflects the effects of cache misses on component utilization.

Clock Power Model:

In a similar fashion, clock power consumption can be obtained:

$$p_{clk} = C_{clk} \cdot V_{dd}^2 \cdot a_{clk} \cdot f \quad (4.2)$$

Since the clock is changing state in every cycle, $a_{clk} = 1$. From Wattch, we obtain $C_{clk} = 3.33$ nF. With differing cache configurations, the clock power consumption can vary by up to $\pm 8\%$; however, the model does not consider this effect.

Caches Power Model:

The expression for cache power consumption is:

$$p_{ci} = C_{ci} \cdot V_{dd}^2 \cdot a_{ci} \cdot f \quad (4.3)$$

The dynamic power consumption of caches is due to memory accesses. For the instruction cache, the i-cache is accessed for each instruction. Additionally, the i-cache is accessed after each pipeline stall due to i-cache misses or branch miss-prediction. Adding in the effects of cache usage occurring after a miss, one obtains:

$$a_{ci} = p_p \cdot (1 + mi_{ci,a}) \quad (4.4)$$

Where $mi_{ci,a}$ is the instruction cache miss probability associated with application a and instruction cache size ci .

The data cache is accessed for each read/write (load-store) instruction and for each d-cache miss, thus:

$$a_{cd} = p_p \cdot ((f_{load_a} + f_{store_a}) \cdot (1 + md_{cd,a})) \quad (4.5)$$

The cache capacitance, C_{ci} and C_{cd} , is shown in Table 4.1. These numbers are given by the Cacti tool [12] for .35- μ m technology. The cache line size is 32 bytes and associativity level is 2. For instruction caches, one read/write port and one read port are assumed. For data caches, two read/write ports are assumed.

Memory and I/O Bus: The same approach taken in Wattch is used to calculate the power consumption of the memory and I/O busses. The memory channel is characterized by its width, $width_{mchl}$; physical length on the chip, $length_{mchl}$; clock frequency, f_{mchl} ; and utilization, $a_{mchl} = \rho_{mchl}$.

Table 4.1: Cache capacitance for 0.35- μm technology.

Cache size (KB)	i-cache capacitance (nF)	d-cache capacitance (nF)
1	0.369	0.378
2	0.397	0.406
4	0.440	0.450
8	0.541	0.570
16	0.708	0.739
32	0.957	1.030
64	1.368	1.412

The capacitance, C_{mchl} , is based on the width and length parameters and is given by:

$$C_{mchl} = 2 \cdot C_{.35\mu\text{m}} \cdot \text{width}_{mchl} \cdot \text{length}_{mchl} \quad (4.6)$$

The factor of 2 is due to the coupling capacitance between wires. The length of the memory channel is taken to be $\text{length}_{mchl} = 5\text{mm}$, which is the expected distance to a processor from the edge of a chip. Here also explored a larger channel length, of 20mm. This, however, only affects the overall results by about 1%. The width is set to 32 bits. The capacitance parameter associated with using .35- μm technology is obtained from scaling the capacitance associated with Wattch's "result bus," yielding $C_{.35\mu\text{m}} = 0.275\text{fF}/\mu\text{m}$.

CHAPTER V

Modeling and Simulation

This chapter describes the modeling of the design, simulation, analysis and implementation. The processor unit is designed and verified in EZchip microcode development environment (a demo system for Network Processors: Architecture, Programming, and Implementation). The Network Processor is also simulate in NepSim simulator (a network processor simulator in Linux environment based on Intel IXE NP architecture) for process execution, power and other parameter observation. Then the high-speed MAC and the NP design is modeled in VHDL using Xilinx ISE Design suit and *Microsemi* Libero IDE design tools. All simulation is performed in ModelSim and synthesized by Synplify Pro synthesis tools.

5.1 EZdesign Microcode Development Tools

EZdesign is part of a comprehensive set of design and testing tools for developers, enabling rapid delivery of new designs based on the NP network processor family to production. The EZdesign development kit allows designers to create, verify, and implement NP applications to meet specific functionality and performance targets. Designers can quickly create new code for the NP network processor using the simulation and debugging tools. An additional EZdriver toolset facilitates the development of host software for NP-based systems. Together EZdesign and EZdriver help designers create a comprehensive NP-based system while reducing overall system development time.

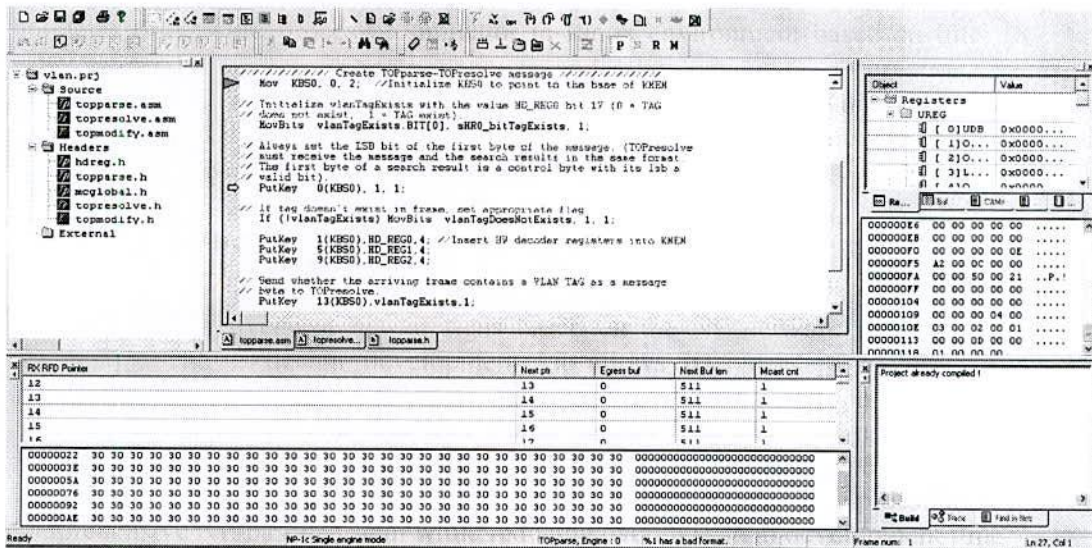


Figure 5.1: EZchip microcode development environment

The processor unit of the proposed design is simulate in the EZchip microcode development tools. These tools support C++ programming language for developing all designs. Figure 5.1 shows the EZchip microcode development environment. All codes are compiled and build by EZDesing compiler. Then run the simulator script and connect with host PC for debugging. Then debug and analyze the code in run time environment

with respect to corresponding registers and memory values. Then generate frames and structures for simulation. Figure 5.2 shows the EZchip NP simulation status and Figure 5.3 shows the Frame structure. The source code of this design is given in Appendix A.

```

C:\EZchip\EZdesign_Demo\bin\EZsim.exe

EZchip Network Processor Software Simulator
Release version 3.60a build 3, built Dec 1 2005 14:27:18

Chip version: NP-1c step 2
EZsim priority class 0x4000 (was 0x40), thread priority 0 (was -2)
=== Initialization ===
Opening frame file...
Connecting debug client to '127.0.0.1' port 2233... succeeded
Connecting host client to '127.0.0.1' port 4509... succeeded
Connecting interrupt client to '127.0.0.1' port 4510... succeeded

=== Execution ===
External memory 0 MB
PARSE RESOLVE MODIFV SRCH_I1 EBDMA GENERAL PARSE MODIFY STAT
RFD ETFD RFD XSRMSG UOQ GENERAL PARSE ECFD IBDMA IFDMA EBDMA
EFDMA ICFD UOQ ECFD ETFD HIFD SRCHMSG STAIMSG EXTARB 30000
47000
  
```

Figure 5.2: EZchip NP simulation status

The screenshot displays the EZchip NP simulation interface. The top window shows the 'Frame parsing...Frame #1' process, detailing the protocol stack from Layer 2 (Ethernet With Tag) to Layer 5-7 (HTTP). The middle window shows the 'EzCoid' details for the frame, including DA, SA, TAG, and PRIQ. The bottom window shows the 'Structure Generator' tool, which is used to generate the frame structure. The 'Filter on Input frame' section shows the selected layers: Layer 2 (Ethernet With Tag), Layer 3 (IP), Layer 4 (UDP), and Layer 5-7 (HTTP). The 'Structure Generator' window shows the resulting structure parameters, such as CHANNEL_ID, SEARCH, STRUCT_NUMBER, and STRUCT_TYPE. The bottom-most window shows the 'Memory Partition Creator' tool, which provides details about the memory partitioning, including the number of structures, structure number, and memory sizes for SigPage and ResPage.

Figure 5.3: Generating Frame structure

5.2 Network Processor Simulator: NePSim

NePSim is the first open source integrated infrastructure for analyzing and optimizing NP design and power dissipation at architecture-level. NePSim contains a cycle-accurate simulator for a typical NP architecture, an automatic verification framework for testing and validation, and a power estimation model for measuring the power consumption of the simulated NP. NePSim achieves satisfactory accuracy in both performance and power modeling. NePSim2 is an updated version that simulates more advanced network processor architecture. Figure 5.4 shows the NePSim hardware model's software architecture. NePSim's inputs are network packet streams.

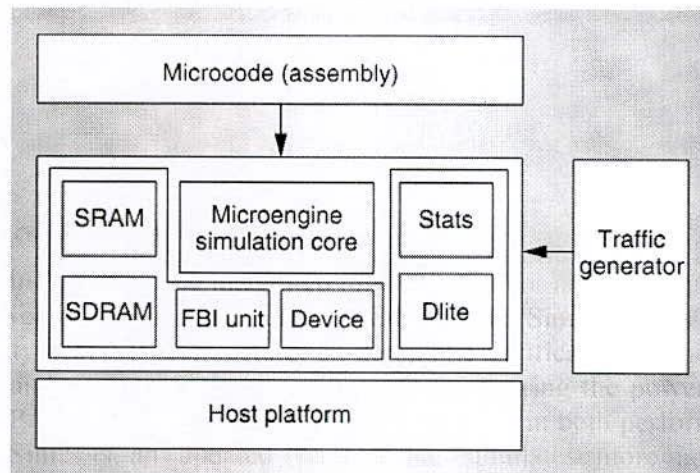


Figure 5.4: NePSim software structure

The NePSim body (the microengine simulation core) is the module that simulates the following five stages of the Micro-Engine (ME) pipeline:

1. instruction lookup;
2. initial instruction decoding and formation of the source register address;
3. reading of operands from the source registers;
4. ALU operations, shift or compare operations, and generation of condition codes;
5. writing of result to destination register.

An ME's threads share the pipeline and functional units such as the ALU. Thread execution is not preemptive, which means a thread cannot gain control of the pipeline unless the running thread yields control.

At the end of simulation, NePSim2 produces a large set of statistics data. Some important output data is given in the following.

```
Simulation state - 1: ** simulation statistics **
sim_cycle           101 # total number of simulated cycles
sim_time            0.1683 # simulated absolute time in microseconds
sim_num_insn        83 # total number of instructions committed
sram_reads_words    134755872 # total number of words read from SRAM
sim_elapsed_time    1 # total simulation time in seconds
sim_inst_rate       83.0000 # simulation speed (in insts/sec)
proc_speed          600.0000 # processor speed in MHz
```

```
throughput_factor      4800.0000 # * 8 bit * proc_speed
<..more lines follow..>
```

```
Simulation state - 2: ** simulation statistics **
sim_cycle              1001 # total number of simulated cycles
sim_time               1.6683 # simulated absolute time in microseconds
sim_num_insn           843 # total number of instructions committed
sram_reads_words       134755872 # total number of words read from SRAM
sim_elapsed_time       1 # total simulation time in seconds
sim_inst_rate          843.0000 # simulation speed (in insts/sec)
proc_speed             600.0000 # processor speed in MHz
throughput_factor      4800.0000 # * 8 bit * proc_speed
<..more lines follow..>
```

5.3 VHDL Modeling, Synthesis and Schematic Diagram

The design is modeled and simulated in hardware description language VHDL and synthesized for Xilinx Spartan3 and generate schematic diagram for the designed module. Then Actel Libero IDE is used to follow the structural design flow for Actel ProASIC3. Synthesis was done in both Xilinx XST synthesized and Synplicity Synplify Pro. Finally, all simulations are done in Modelsim Simulart v6.6d and generates waveform.

5.3.1 Working on Xilinx ISE Design Suite

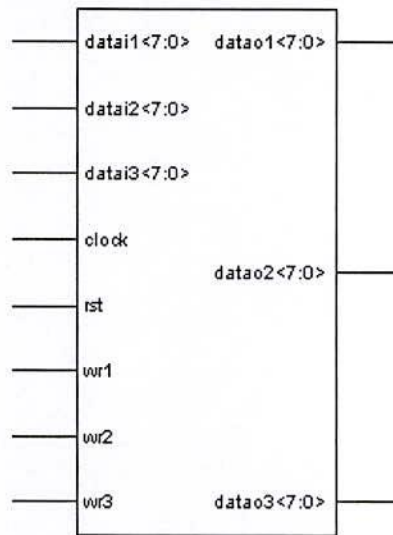


Figure 5.5: I/O pin layout of the top module

In the design phase of VHDL hardware modeling each block of the architecture is designed in different individual models such as FIFO module, control module, memory module, register module, processing module etc. Modules are connected through a hierarchical module interface. The top level module is the main module of the design. Figure 5.5 shows the I/O pin layout of the top module. After compilation with VHDL Xilinx compiler the design is synthesized and generates Register Transfer Logic (RTL) schematic. The total schematic diagram of the top module is shown in Figure 5.6. The Schematic diagram of control module, scheduler module and memory module are shown in Figure 5.7, Figure 5.8 and Figure 5.9 respectively.

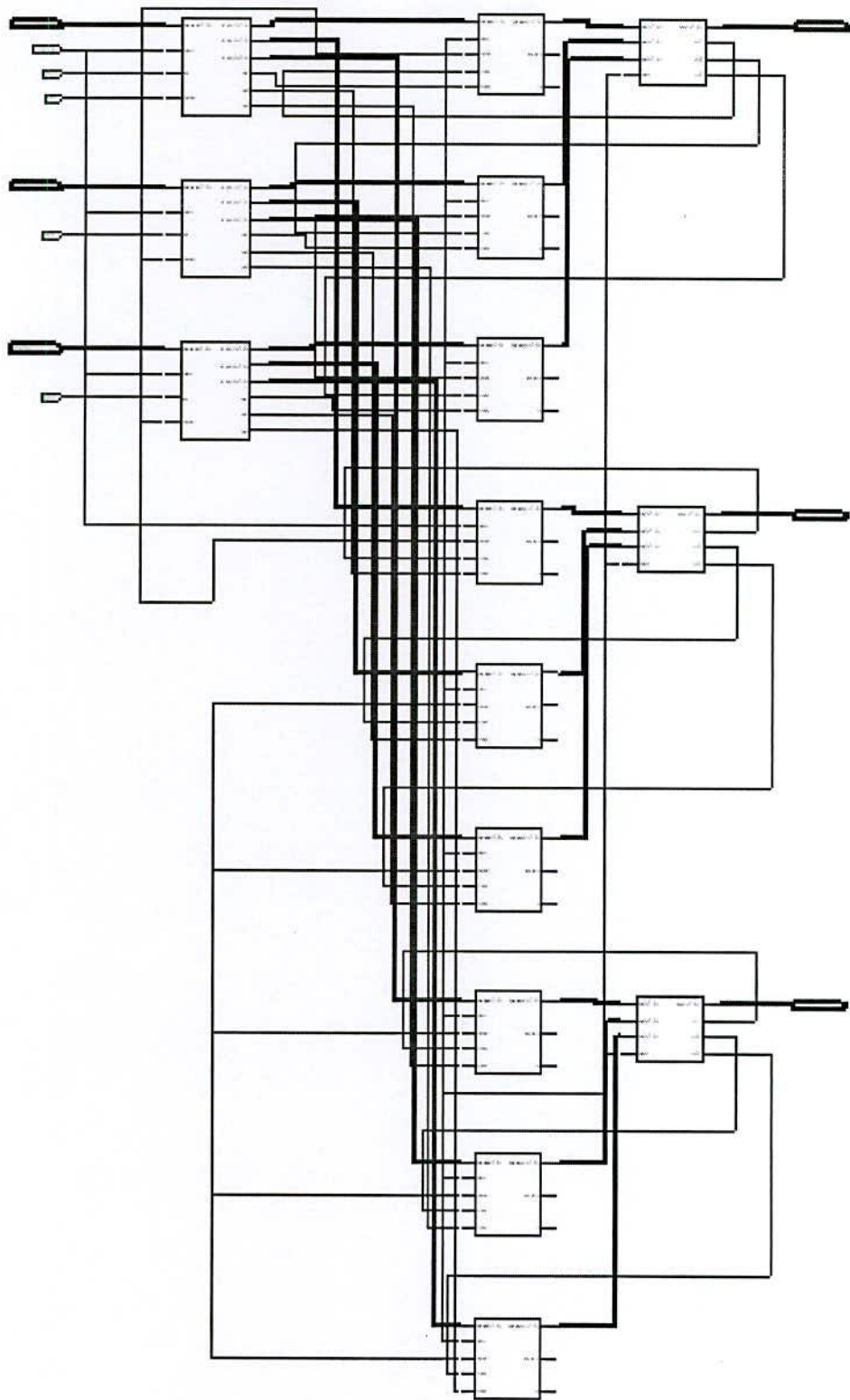


Figure 5.6: Total schematic diagram of the top module

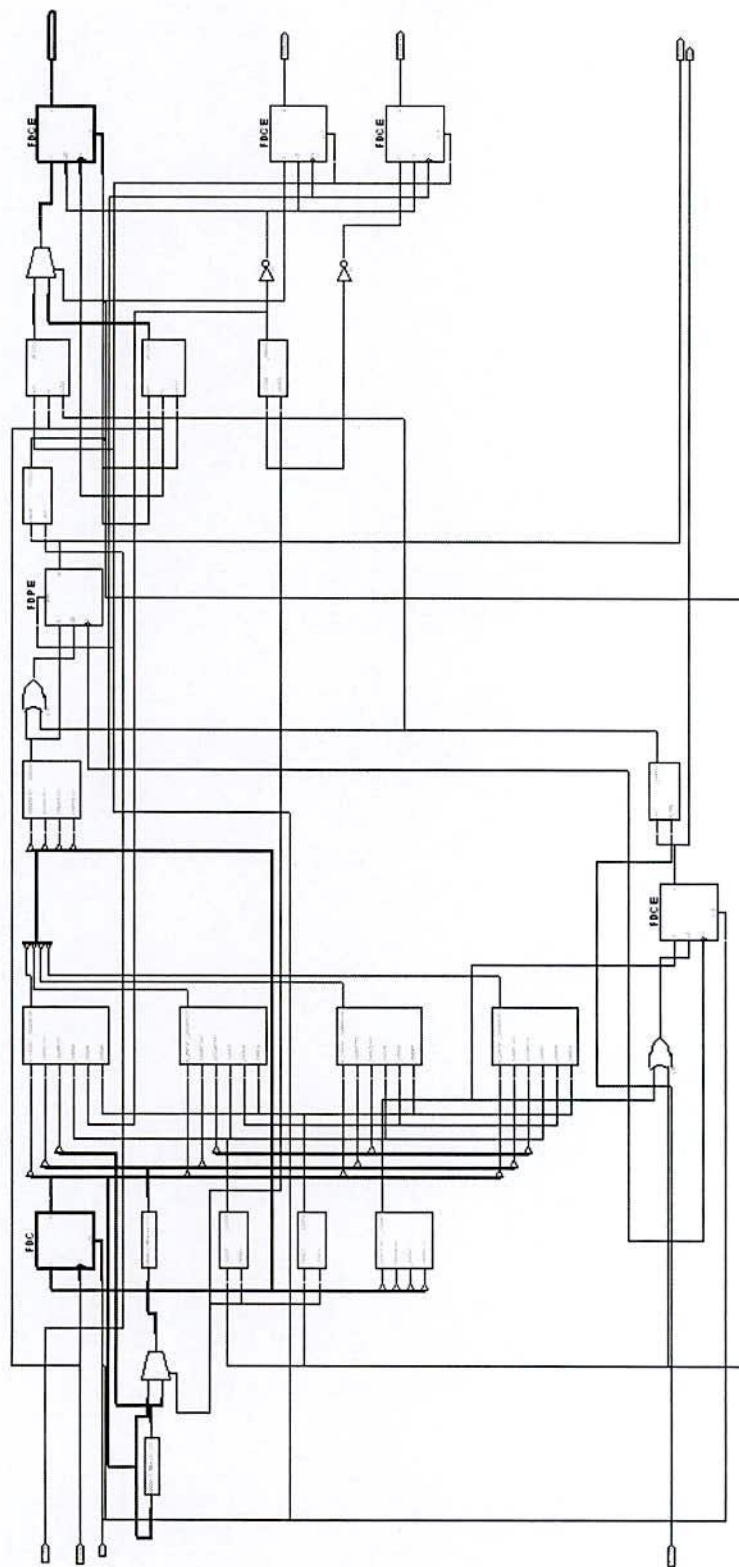


Figure 5.7: Schematic circuit of control module

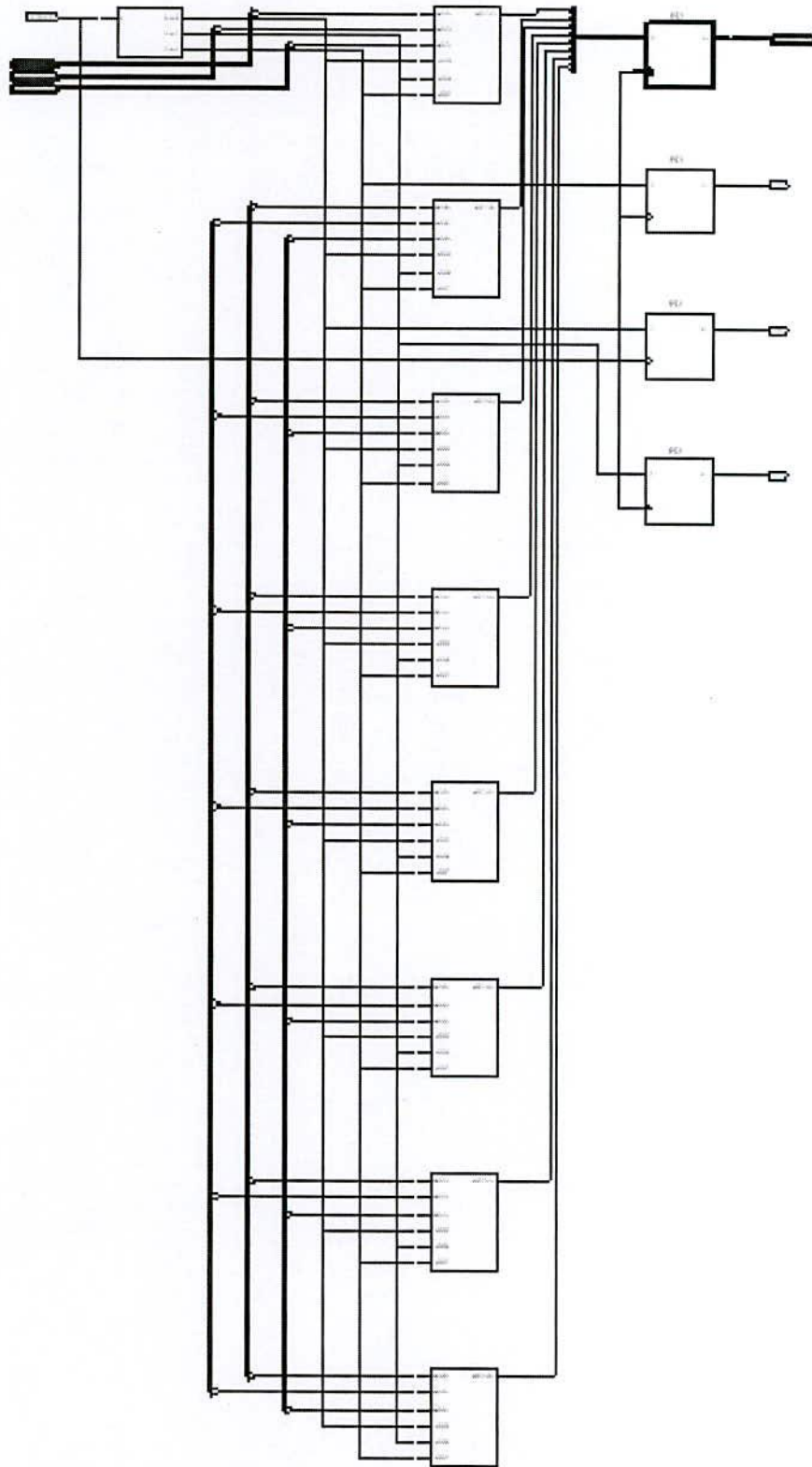


Figure 5.8: Schematic circuit of scheduler module

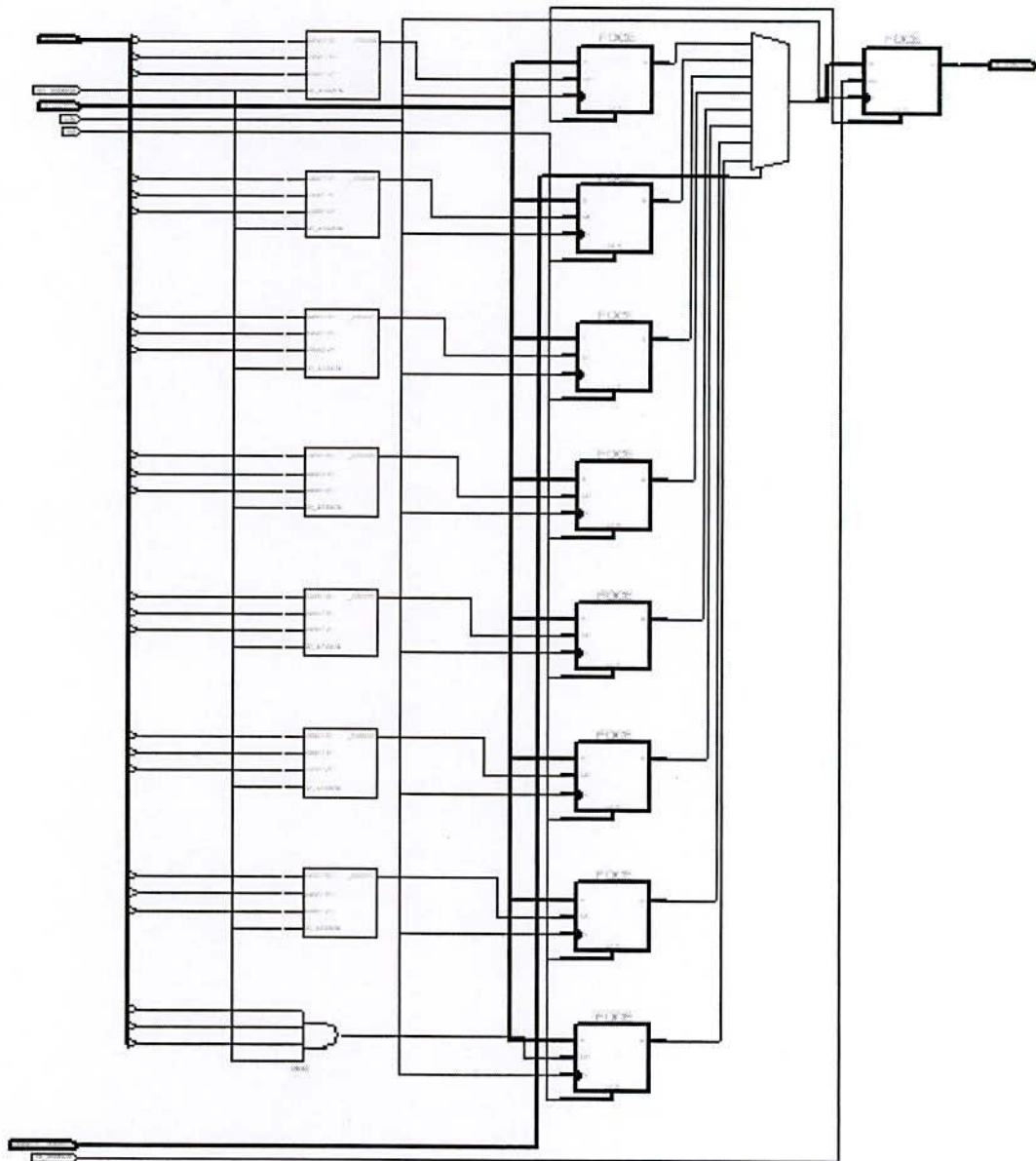


Figure 5.9: Schematic circuit of memory module

The synthesis analysis report generated from Xilinx XST synthesizer for spartan3 is given below:

```

=====
*                               Final Synthesis Report                               *
=====
Final Results:
Output Format                   : NGC
Optimization Goal               : Speed

Design Statistics:
# IOs                           : 53
  
```

Macro Statistics :

```
# Registers : 168
#   1-bit register : 45
#   3-bit register : 9
#   4-bit register : 27
#   8-bit register : 87
# Multiplexers : 27
#   2-to-1 multiplexer : 18
#   8-bit 8-to-1 multiplexer : 9
```

Cell Usage :

```
# BELS : 997
#   BUF : 1
#   GND : 1
#   LUT1 : 18
#   LUT2 : 36
#   LUT2_D : 9
#   LUT2_L : 18
#   LUT3 : 340
#   LUT3_L : 18
#   LUT4 : 220
#   LUT4_D : 32
#   LUT4_L : 106
#   MUXF5 : 132
#   MUXF6 : 66
# FlipFlops/Latches : 790
#   FD : 34
#   FDC : 36
#   FDCE : 711
#   FDPE : 9
# Clock Buffers : 1
#   BUFGP : 1
# IO Buffers : 52
#   IBUF : 28
#   OBUF : 24
```

=====
Device utilization summary:

Selected Device : 3s50pq208-5

Number of Slices:	627	out of	768	81%
Number of Slice Flip Flops:	790	out of	1536	51%
Number of 4 input LUTs:	797	out of	1536	51%
Number of bonded IOBs:	52	out of	124	41%
Number of GCLKs:	1	out of	8	12%

=====
Timing Summary:

Speed Grade: -5

Minimum period: 4.369ns (Maximum Frequency: 228.885MHz)
Minimum input arrival time before clock: 6.171ns
Maximum output required time after clock: 5.106ns
Maximum combinational path delay: No path found

Timing Detail:

All values displayed in nanoseconds (ns)

Timing constraint: Default period analysis for Clock 'clock'
Delay: 4.369ns (Levels of Logic = 3)
Source: reg_demux_3_data_1 (FF)
Destination: fifo_33_control_unit_full_signal (FF)
Source Clock: clock rising
Destination Clock: clock rising

Data Path: reg_demux_3_data_1 to fifo_33_control_unit_full_signal

Cell:in->out	fanout	Gate Delay	Net Delay
FDCE:C->Q	3	0.626	0.577
LUT4_D:I1->O	12	0.479	0.865
LUT4_L:I2->LO	1	0.479	0.100
LUT4:I1->O	1	0.479	0.240
FDCE:CE		0.524	

Total		4.369ns (2.587ns logic, 1.782ns route) (59.2% logic, 40.8% route)	

Timing constraint: Default OFFSET IN BEFORE for Clock 'clock'
Offset: 6.171ns (Levels of Logic = 4)
Source: wr3 (PAD)
Destination: fifo_33_control_unit_full_signal (FF)
Destination Clock: clock rising

Data Path: wr3 to fifo_33_control_unit_full_signal

Cell:in->out	fanout	Gate Delay	Net Delay
IBUF:I->O	37	1.679	1.326
LUT4_D:I2->O	12	0.479	0.865
LUT4_L:I2->LO	1	0.479	0.100
LUT4:I1->O	1	0.479	0.240
FDCE:CE		0.524	

Total		6.171ns (3.640ns logic, 2.531ns route) (59.0% logic, 41.0% route)	

Timing constraint: Default OFFSET OUT AFTER for Clock 'clock'
Offset: 5.106ns (Levels of Logic = 1)
Source: scheduler1_dout_7 (FF)
Destination: dataa1<7> (PAD)
Source Clock: clock rising

Data Path: scheduler1_dout_7 to dataa1<7>

Cell:in->out	fanout	Gate Delay	Net Delay
FD:C->Q	1	0.626	0.240
OBUF:I->O		4.240	
Total		5.106ns (4.866ns logic, 0.240ns route) (95.3% logic, 4.7% route)	

CPU: 12.13 / 17.55 s | Elapsed: 12.00 / 17.00 s

Total memory usage is 79084 kilobytes

5.3.2 Working on Actel Libero IDE Design Flow

Actel Libero IDE Design package is a famous design tool for FPGA and ASIC designer. HDL Editor, Synplify Pro Synthesis tool, ModelSim for HDL compiles & wave form simulation etc are built-in in Libero IDE Design. The Design Flow is shown in Figure 5.10. This is a step-wise design flow that guide designer to follow in a sequence manner. First add source code then pre-synthesize and perform pre-simulation then post-synthesis and post-simulation and finally implementation. After synthesis the VHDL by Synplify a performance report and synthesized schematic are generate. Summary of performance analysis is given at the end of this section and the synthesized schematics are shown in following Figures.

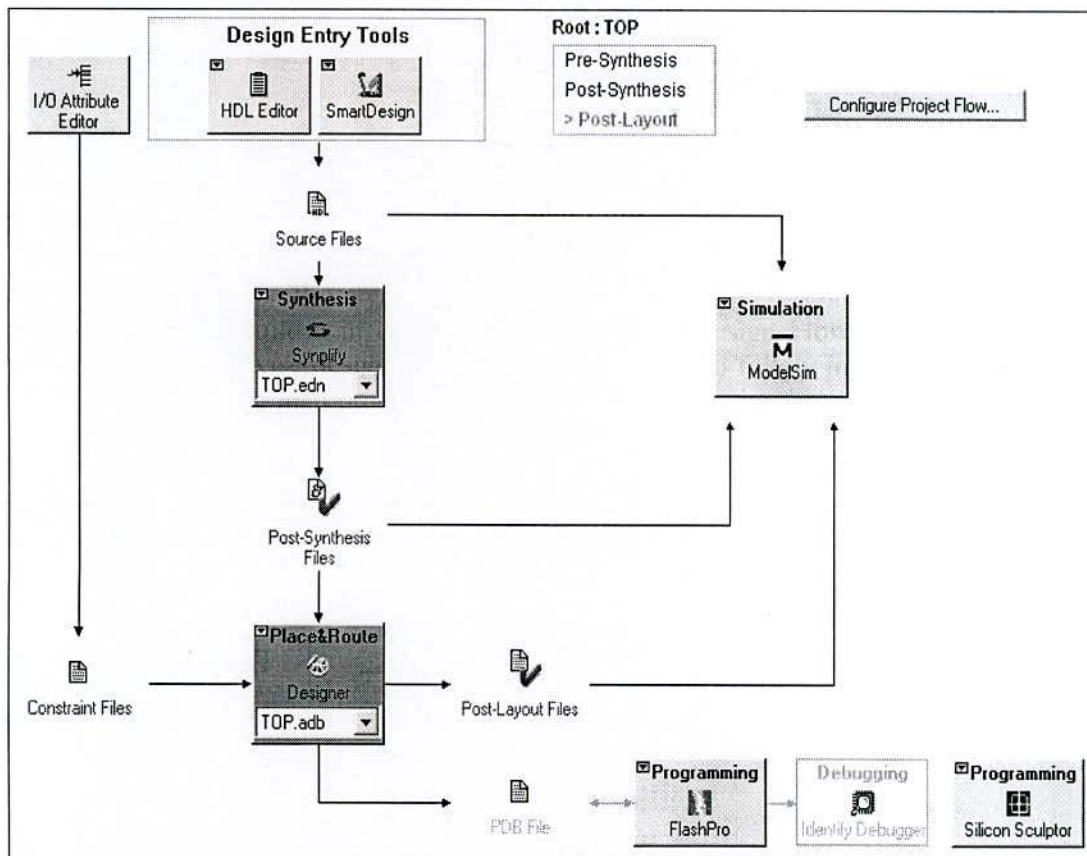


Figure 5.10: Actel Libero IDE Design Flow

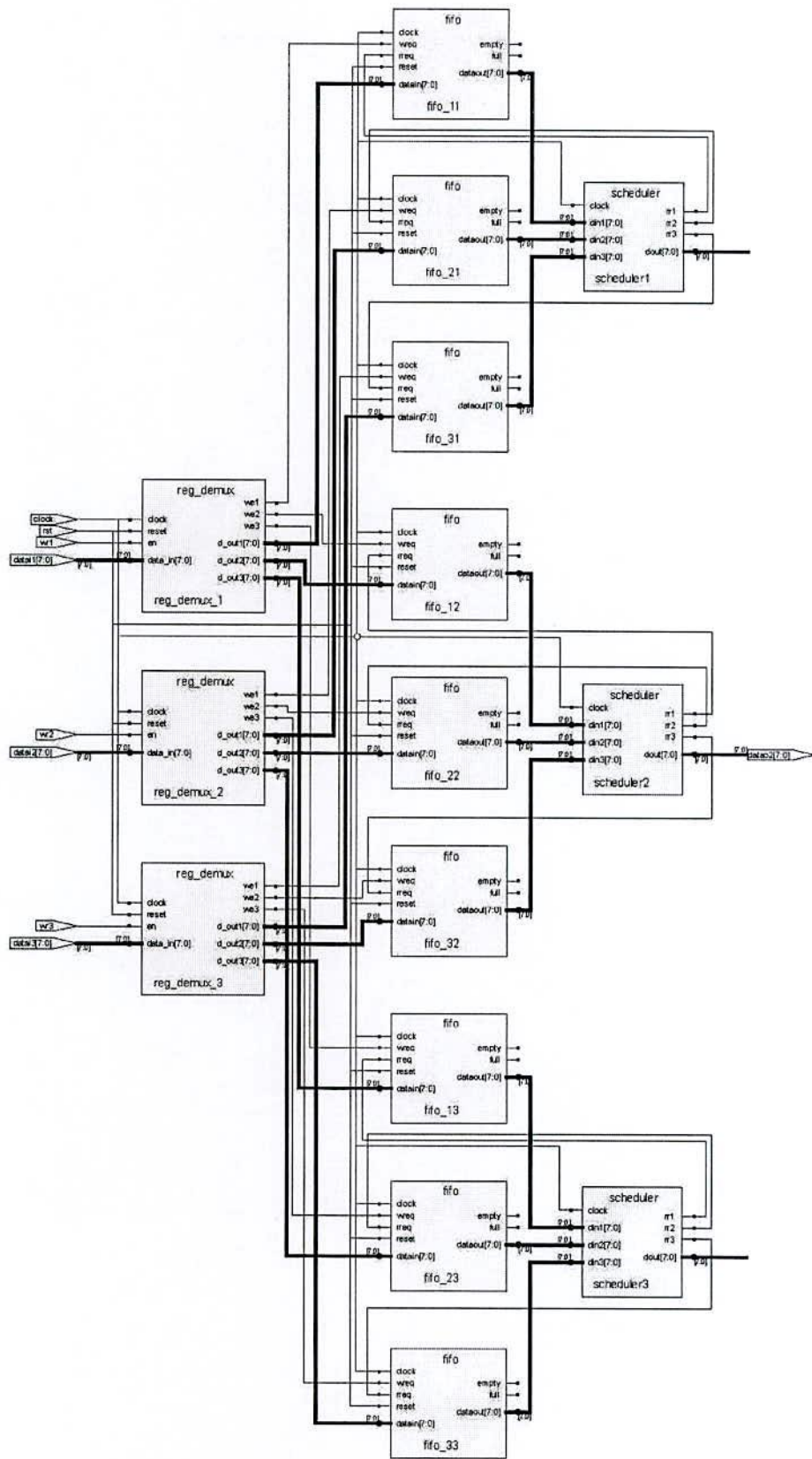


Figure 5.11: RTL Synthesis block diagram

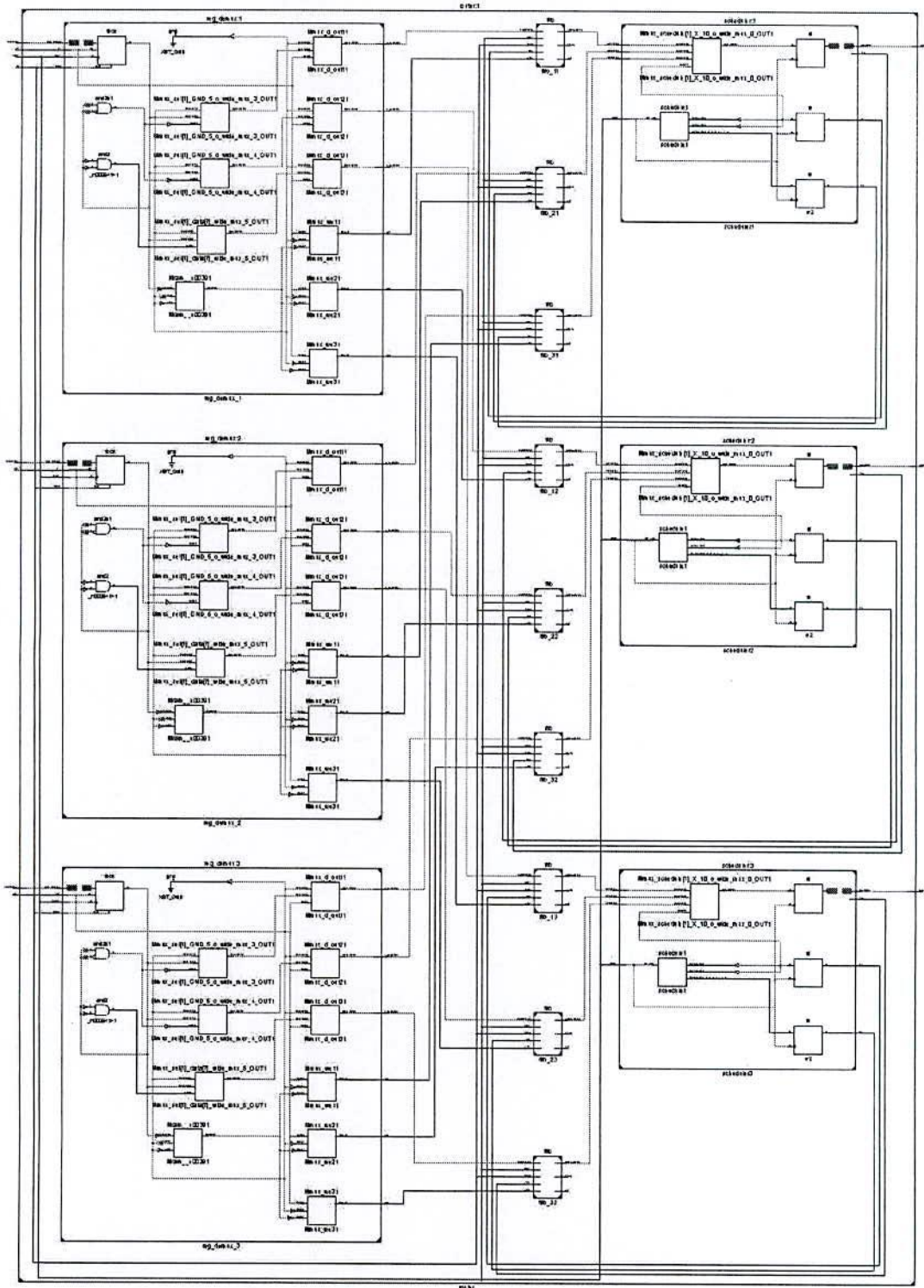


Figure 5.12: Full synthesized schematic circuit

The total RTL synthesis block diagram is shown in Figure 5.11 and the full synthesized schematic is shown in Figure 5.12. The Synthesized diagram of control unit and memory unit is shown in Figure 5.13 and schematic circuits of these units are shown in Figure 5.14 and Figure 5.15 respectively.

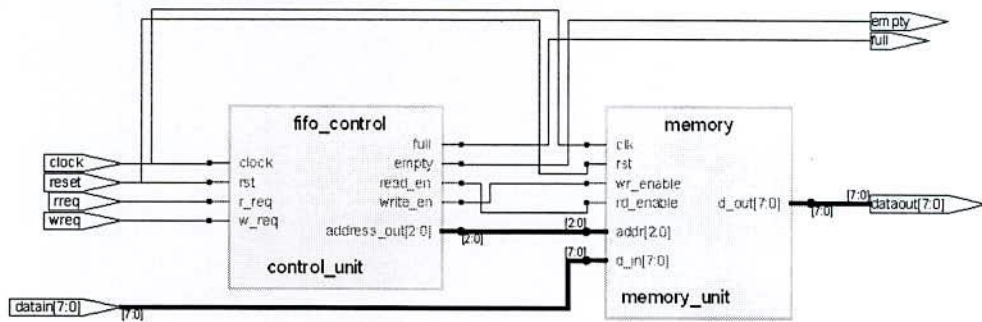


Figure 5.13: synthesized diagram of memory and control unit

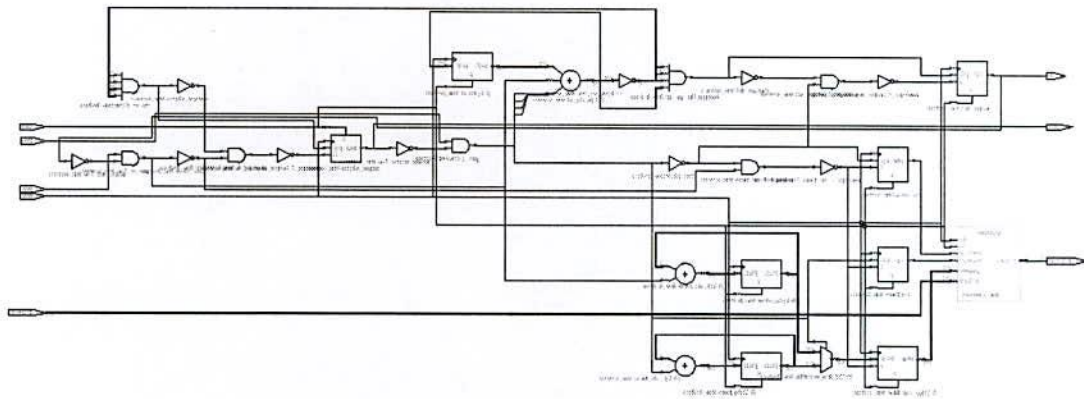


Figure 5.14: Schematic circuit of control unit

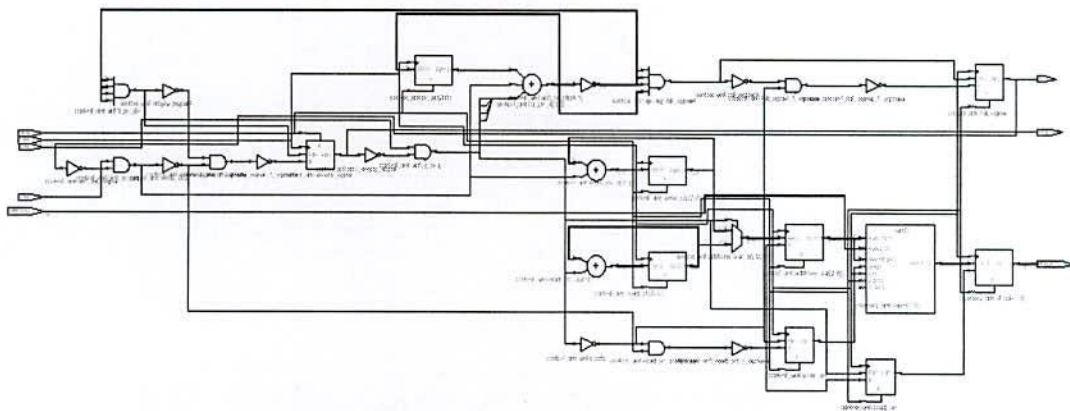


Figure 5.15: Schematic circuit of memory unit

In the control unit module all decision and processing operation is done in single unit. It generates all control signals and distributes incoming packets data to processing and this is also implement in this control unit. This control unit is interface with memory unit for search lookup data from memory and store routing/switching information in a table form.

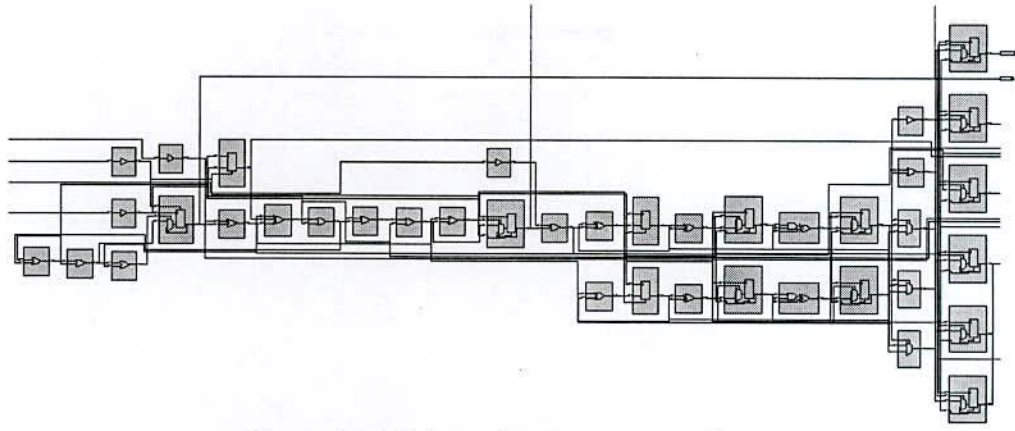


Figure 5.16: Schematic of memory register

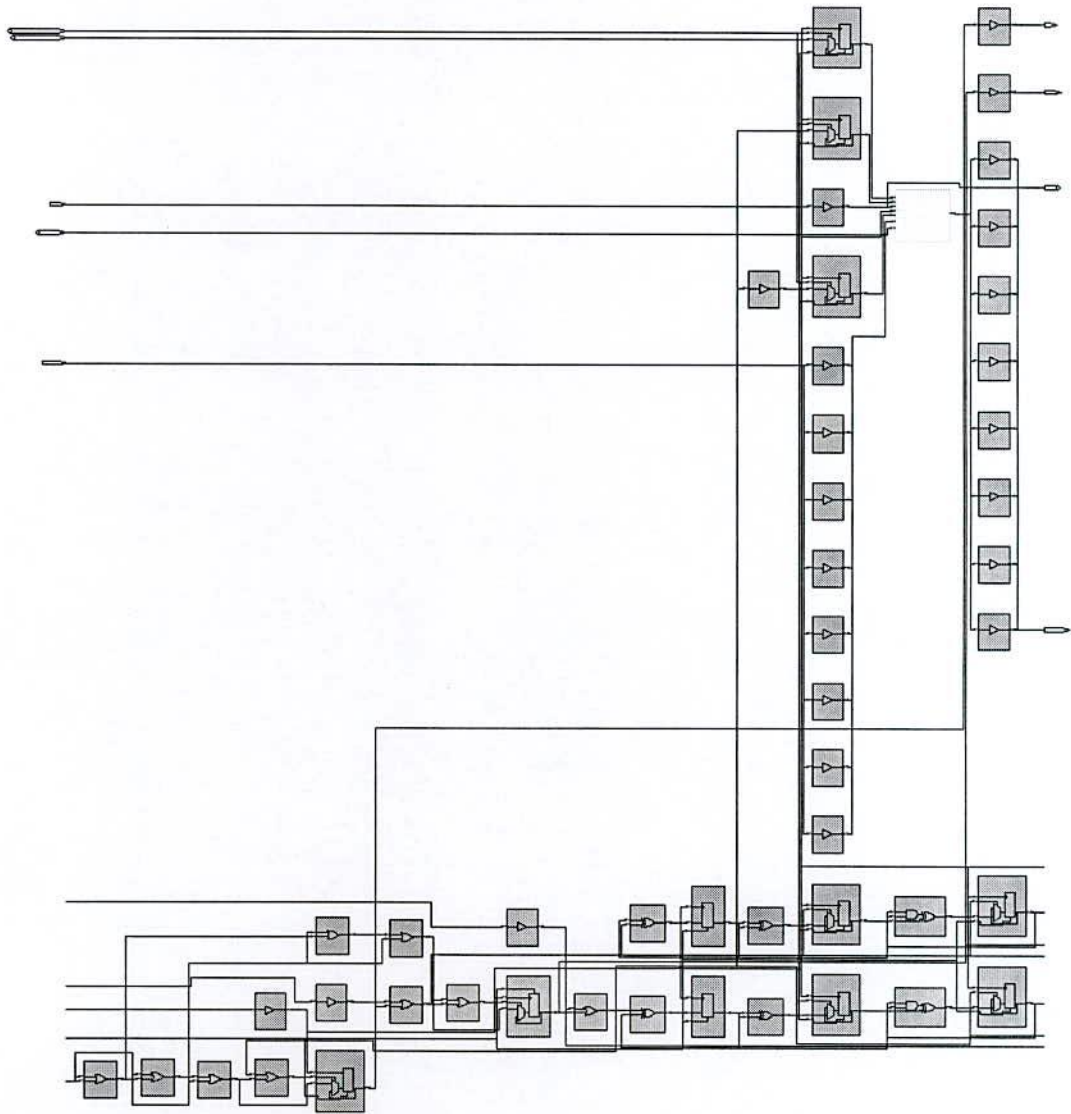


Figure 5.17: Data path diagram of control operation

Data Path: Network Processors are characterized by their data and instruction paths. Among the most important parts of any Network Processor is the data path. Since Network Processors are designed to process large amounts of data, the data paths are often independent of the instruction paths to avoid creating unnecessary bottlenecks. The data paths are also necessarily fast and wide, often requiring the use of switch matrices and point-to-point links. Figure 5.17 shows the data path of control operation and Figure 5.18 shows the critical-data path of the design.

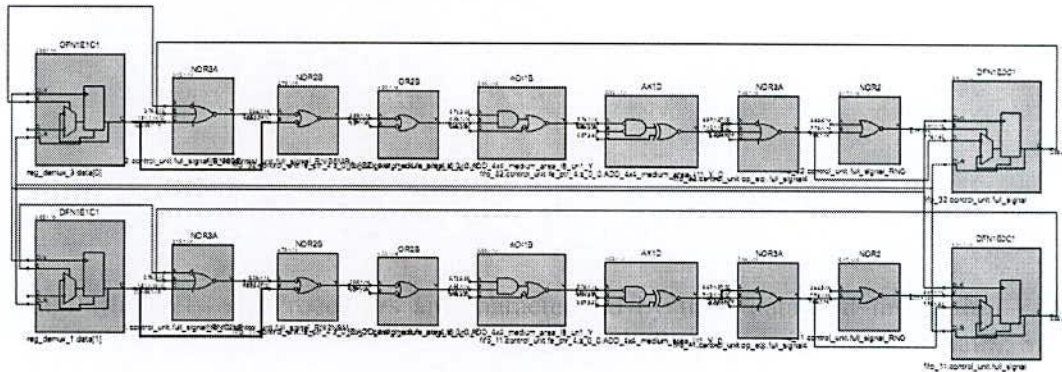


Figure 5.18: Critical data path

The performance analysis report from Synplicity Synplify Pro. is given below:

```

=====
*                               Performance Summary                               *
=====

@P: Total Area : 2199.0
@P: CPU Time : 0h:00m:05s

@P: Worst Slack : 1.139
@P: router|clock - Estimated Frequency : 112.9 MHz
@P: router|clock - Requested Frequency : 100.0 MHz
@P: router|clock - Estimated Period : 8.861
@P: router|clock - Requested Period : 10.000
@P: router|clock - Slack : 1.139

@P: Worst Slack(min analysis) : NA
@P: router|clock - Estimated Frequency(min analysis) : 112.9 MHz
@P: router|clock - Requested Frequency(min analysis) : 100.0 MHz
@P: router|clock - Estimated Period(min analysis) : 8.861
@P: router|clock - Requested Period(min analysis) : 10.000
@P: router|clock - Slack(min analysis) : 1.139

```

Starting Points with Worst Slack

Instance	Starting Reference Clock	Type	Pin	Net	Arrival Time	Slack
reg_demux_3.data[0]	router clock	DFN1E1C1	Q	data[0]	0.550	1.139
reg_demux_1.data[1]	router clock	DFN1E1C1	Q	data[1]	0.550	1.139

reg_demux_1.data[0]	router clock	DFN1E1C1	Q	data[0]	0.550	1.269
reg_demux_2.data[1]	router clock	DFN1E1C1	Q	data[1]	0.550	1.269
reg_demux_3.data[1]	router clock	DFN1E1C1	Q	data[1]	0.550	1.269

Starting Points with Worst Slack

Instance	Starting Reference Clock	Type	Pin	Net	Arrival Time	Slack
control_unit.empty_signal	fifo clock	DFN1E0P1	Q	empty_c	0.627	1.250
control_unit.full_signal	fifo clock	DFN1E0C1	Q	full_c	0.494	1.717
control_unit.address_out[1]	fifo clock	DFN1E0C1	Q	addr_s[1]	0.494	1.788
control_unit.fe_ptr[0]	fifo clock	DFN1C1	Q	fe_ptr[0]	0.627	1.802
control_unit.address_out_0[0]	fifo clock	DFN1E0C1	Q	addr_s_0[0]	0.494	1.811

Ending Points with Worst Slack

Instance	Starting Reference Clock	Type	Pin	Net	Required Time	Slack
control_unit.full_signal	fifo clock	DFN1E0C1	E	full_signal_1_sqmuxa	9.482	1.250
control_unit.empty_signal	fifo clock	DFN1E0P1	E	empty_signal_1_sqmuxa	9.482	1.378
memory_unit.d_out[0]	fifo clock	DFN1E1C1	D	mem[0]	9.542	1.788
memory_unit.d_out[1]	fifo clock	DFN1E1C1	D	mem[1]	9.542	1.788
memory_unit.d_out[2]	fifo clock	DFN1E1C1	D	mem[2]	9.542	1.788

Ending Points with Worst Slack

Instance	Starting Reference Clock	Type	Pin	Net	Required Time	Slack
control_unit.full_signal	router clock	DFN1E0C1	E	full_signal_1_sqmuxa	9.546	1.139
control_unit.full_signal	router clock	DFN1E0C1	E	full_signal_1_sqmuxa	9.546	1.139
control_unit.empty_signal	router clock	DFN1E0P1	E	empty_signal_1_sqmuxa	9.546	1.252
control_unit.empty_signal	router clock	DFN1E0P1	E	empty_signal_1_sqmuxa	9.546	1.252
control_unit.full_signal	router clock	DFN1E0C1	E	full_signal_1_sqmuxa	9.546	1.269

Worst Path Information

Path information for path number 1:

Requested Period:	10.000
- Setup time:	0.454
+ Clock delay at ending point:	0.000 (ideal)
= Required time:	9.546
- Propagation time:	8.407
- Clock delay at starting point:	0.000 (ideal)
= Slack (critical):	1.139

Number of logic level(s):

7

Starting point:

reg_demux_3.data[0] / Q

Ending point: control_unit.full_signal / E

The start point is clocked by router|clock [rising] on pin CLK
The end point is clocked by router|clock [rising] on pin CLK

Core Cell usage:

cell	count	area	count*area
AO1B	9	1.0	9.0
AOI1B	89	1.0	89.0
AX1C	18	1.0	18.0
AX1D	5	1.0	5.0
AX1E	18	1.0	18.0
GND	34	0.0	0.0
INV	9	1.0	9.0
MX2	533	1.0	533.0
MX2C	4	1.0	4.0
NOR2	80	1.0	80.0
NOR2A	58	1.0	58.0
NOR2B	130	1.0	130.0
NOR3	11	1.0	11.0
NOR3A	13	1.0	13.0
NOR3B	9	1.0	9.0
NOR3C	21	1.0	21.0
OA1B	4	1.0	4.0
OAI1	9	1.0	9.0
OR2	91	1.0	91.0
OR2A	23	1.0	23.0
OR2B	39	1.0	39.0
OR3A	8	1.0	8.0
OR3B	19	1.0	19.0
OR3C	25	1.0	25.0
VCC	34	0.0	0.0
XNOR2	8	1.0	8.0
XOR2	77	1.0	77.0
DFN1	37	1.0	37.0
DFN1C1	126	1.0	126.0
DFN1E0C1	63	1.0	63.0
DFN1E0P1	9	1.0	9.0
DFN1E1	528	1.0	528.0
DFN1E1C1	126	1.0	126.0

TOTAL	2267		2199.0

IO Cell usage:

cell	count
CLKBUF	2
INBUF	27
OUTBUF	24

TOTAL	53

Core Cells : 2199 of 24576 (9%)
IO Cells : 53

```

Mapper successful!
Process took 0h:00m:05s realtime, 0h:00m:05s cputime
# Mon Aug 15 11:59:17 2011

```

5.4 Simulation Waveform

All Simulation is done in Mentor Graphics ModelSim Simulator v6.6d. The simulation waveforms of 10G Ethernet MAC interface are generated from a test vector. The waveform of packet receive interface is shown in Figure 5.19 and the waveform of packet transmit interface is shown in Figure 5.20.

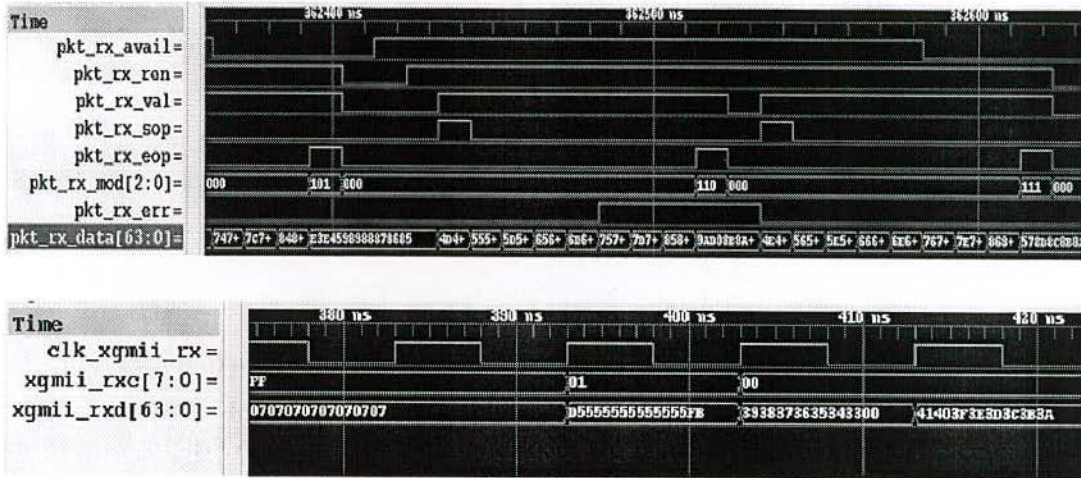


Figure 5.19: Simulation waveforms of packet receive interface

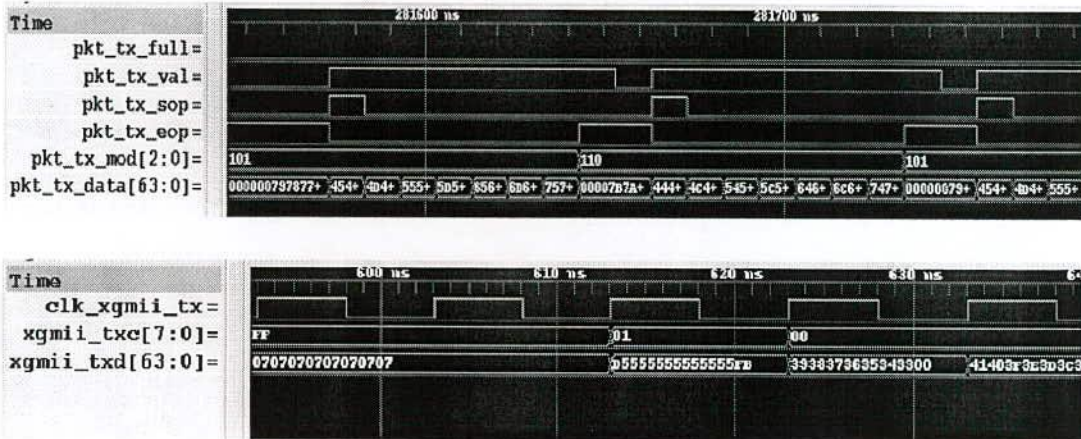


Figure 5.20: Simulation waveforms of packet transmit interface

The core design model is simulated for both routing and switching operation, i.e. two stimulus of test-bench one for router and another for switch. The results of simulation waveforms of routing and switching application are shown in Figure 5.21 and Figure 5.22 respectively. Switching operation is faster than router, because in switch there are less processing requirements. The load scheduler is performing the same task both in switch and router. The simulation waveform of load scheduler is shown in Figure 5.23.

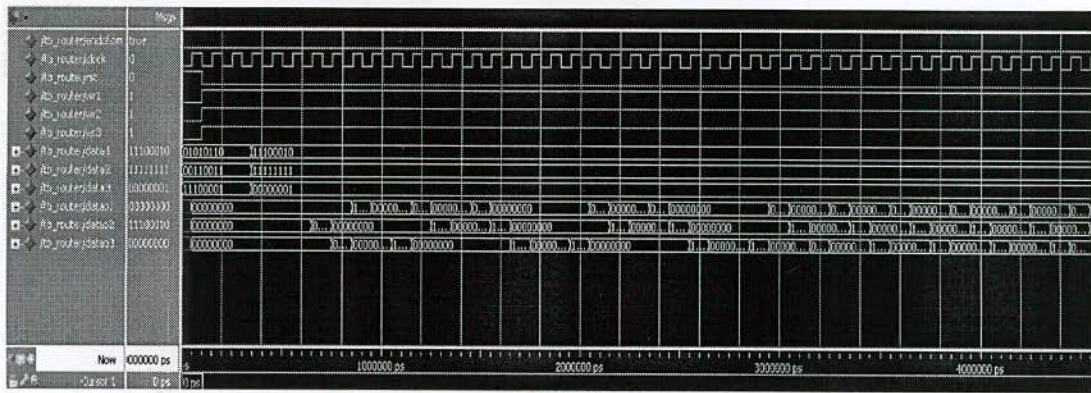


Figure 5.21: Simulation waveform of routing application

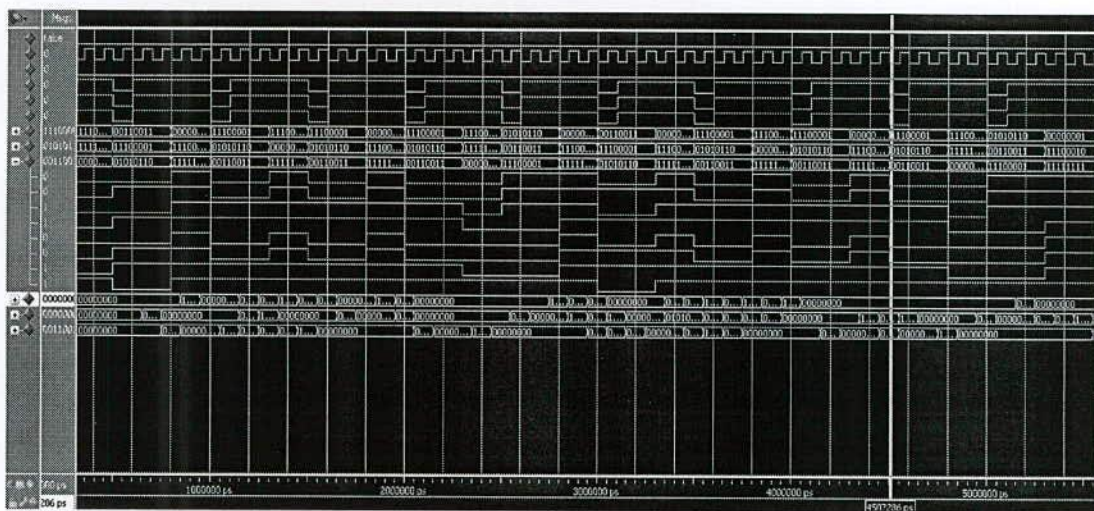


Figure 5.22: Simulation waveform of switching application

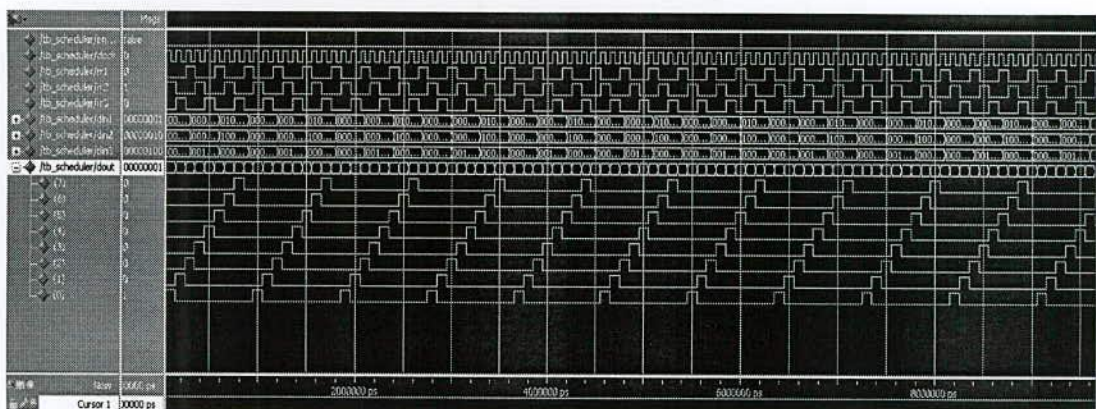


Figure 5.23: Simulation waveform of Load scheduler

5.5 Synthesized Power Report

```
-----
|                               Xilinx XPower Analyzer                               |
-----
```

On-Chip Power Summary

```
-----
|                               On-Chip Power Summary                               |
-----
| On-Chip | Power (mW) | Used | Available | Utilization (%) |
-----
| Clocks   | 0.49       | 1    | ---       | ---             |
| Logic    | 0.00       | 808  | 2400      | 34              |
| Signals  | 0.00       | 1014 | ---       | ---             |
| IOs      | 0.00       | 53   | 102       | 52              |
| Quiescent | 13.69      |      |           |                 |
| Total    | 14.18      |      |           |                 |
-----
```

Power Supply Summary

```
-----
|                               Power Supply Summary                               |
-----
|                               | Total | Dynamic | Quiescent |
-----
| Supply Power (mW)          | 14.18 | 0.49    | 13.69     |
-----
```

```
-----
|                               Power Supply Currents                               |
-----
| Supply | Supply | Total | Dynamic | Quiescent |
| Source | Voltage | Current (mA) | Current (mA) | Current (mA) |
-----
| Vccint | 1.200 | 4.48  | 0.41    | 4.07       |
| Vccaux | 2.500 | 2.52  | 0.00    | 2.52       |
| Vcco25 | 2.500 | 1.00  | 0.00    | 1.00       |
-----
```

CHAPTER VI

Performance Evaluation

This chapter covers the general performance analysis of the designed system and compares its architecture to other well known Network Processor architectures. The evaluation of the system performance for Network Processors is mainly packet Forwarding. The analysis shows that the proposed architecture can provide adequate performance for real world applications and can even exceed the performance of common architectures.

6.1 General NP Performance

Performance of network processors is determined by the average amount of processing cycles per packet available for network space applications. Network space applications are the ensemble of all packet processing operations performed on a per packet basis. To evaluate the performance of a network processor, it is necessary to take into account only the cycles spent on packet processing operations, discounting any clock cycles spend on overhead operations, such as packet forwarding, lookup table maintenance and system control.

6.2 Architecture Analysis and Comparison

Studies of network processor performance have analyzed and compared with several architectures. In recent growing technology the most two network processor architectures are the Intel IXE architecture (analyzed in NepSim) and EZchip TOPcore architecture (analyzed in EZdesign). The thesis revised their technology and provides some new techniques to increase the total processing performance and reduce complexity. The following section describes the architectural performance improvement and comparison with existing.

(i) Processing strategy and Layered Architecture:

Generally networking system performance degraded when packet delivery reached the system maximum and became the bottleneck. To avoid this limitation, a four level layered architecture was developed and increase the buffer memory such that no packet losses are occurred. In addition external memory interface is designed for very high speed packet delivery systems. From the analysis of various incoming traffic to network processor, it is found that all packets are not follow the same processing strategy. A basic processing step is enough for all packets and some traffic required more additional processing and few traffic needs more extra processing. So, all packets do not need to pass through all processing steps. This thesis implements the concept and developed a hierarchical processing level architecture. In the processing hierarchy, these Level processing is performed by processor unit and controlled by processing control unit. Figure 6.1 shows the proposed Network Processor architecture and Figure 6.2 shows the Proposed modified architecture of processor unit. These two figures are already described in chapter 4.

In this layered architecture the level of hierarchy is packet distribution level, basic processing level, additional processing level and control level. Packet distribution is

performed on a dedicated level and therefore does not reduce or limit packet processing. Packet distribution is therefore limited only by the Packet Distribution Level. The Load Balancer unit is work in the packet distribution level with using load distribution algorithm. Packet processing is performed on the Basic Processing Level and the Additional Processing Level. Processing Control Unit controls the processing sub unit to perform these levels. As packets flow through the Packet Distribution Level, packets are queued for processing in Packet Buffers. The destination of the packet on the Packet Distribution Level is determined by the lookup performed on the Basic Processing Level if available or the Additional Processing Level alternatively. The number of clock cycles available for packet processing is determined by the number of clock cycles each packet can be held for processing in the Packet Buffer without reducing the packet throughput.

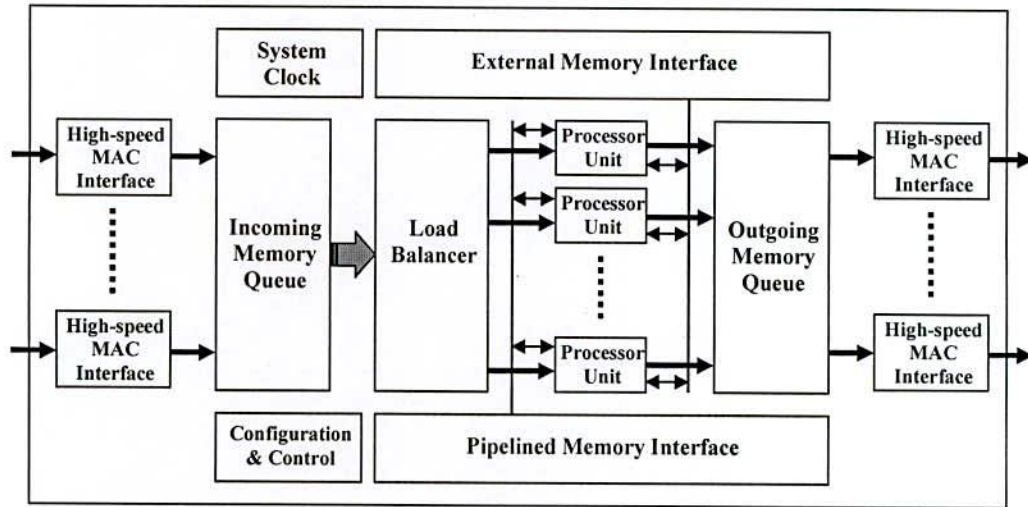


Figure 6.1: Block diagram of proposed Network Processor architecture

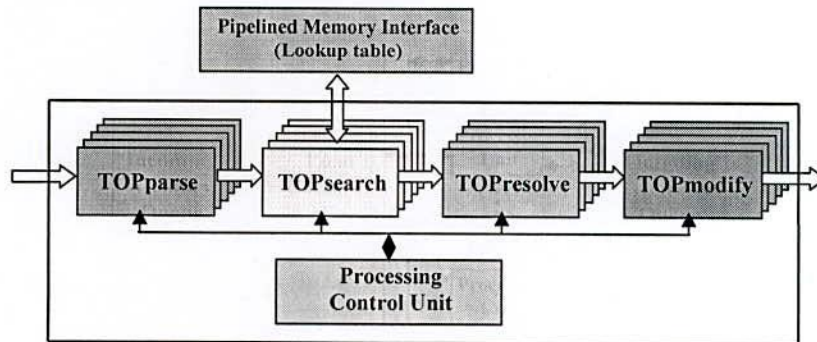


Figure 6.2: Proposed modified architecture of processor unit

(ii) Super-pipeline Parallel Architecture:

In the design of network processor this thesis combines pipelining and parallelism by using a four-stage pipeline in which each stage is implemented by parallel processors. Again in the processor core there are four basic processing units namely parse, search, resolve and modify. These units are also placed in parallel. Each unit process data through

in a parallel pipeline and the whole processor unit takes data over a pipeline, so it follows super-pipeline architecture. As a result packet processing is much faster than existing processor. Figure 6.1 illustrates the architecture.

(iii) Pipelined Memory Architecture:

Every network processor maintains a lookup database stored in memory and the processing performance directly depends on the access of memory or the speed of search engine. To achieve high memory bandwidth, multiple processors and thread contexts are used to generate multiple pending accesses. The problem is that each processor does all of the forwarding for an assigned packet, and each independent processor needs independent access to a large amount of shared state (e.g., the global forwarding database). The existing known way around this problem is to replicate the state for each processor or to share the state in some fashion. The thesis implements memory pipelining to overcome these problems and ensure high bandwidth for memory access. For a packet size of 40 bytes (TCP acknowledgments) on a 10-Gb/s link that leaves a memory access time per packet of less than

$$t_{pkt} = \frac{40 \text{ Bytes}}{10 \text{ Gb/s}} = 30 \text{ ns}$$

and at 40 Gb/s this is further reduced to 7.5 ns.

(iv) Green technology for saving power consumption:

Green technology is a field of new, innovative ways to make changes in daily life. Basically, green technology is that in which the technology is environmentally friendly and is created and used in a way that conserves natural resources and the environment. In electronics area this green technology conserves the unnecessary power consumption at the time of inactive periods. The thesis implements this green concept to save power.

In networking communication the incoming of packet flow are not occur all time. Some times it ideal and some times it is in heavy packet load. Whatever it is, the network device are run in 24 hours and consume power for all interface. But it is possible to change a interface to sleep mode when there are no packet flow. The interfaces that are in sleep mode saves some power. This thesis work implements this and saves power consumption when the I/O interfaces and the processor are in ideal.

(v) Implement as FPGA soft-core:

In today's design world soft processor core are available for used in FPGA. As this thesis modeled and design the proposed system in VHDL and synthesized for FPGA (Xilinx Spartan3 and Actel ProASIC3), this model program can be used for any other FPGAs vender and if necessary implement with modification.

6.3 Evaluation Methodology

Performance evaluation was performed by generating the VHDL model described previously and mapping them to a Xilinx Spartan3 FPGA, using Xilinx ISE Design Suit v13.1 and to Actel ProASIC3 FPGA, using Actel Libero IED v9.1. The design was synthesized using both Xilinx Synthesis Technology (XST) and Synplicity Sinplify Pro Actel Edition. The design was simulated using Model Technology ModelSim v6.6d. Performance was evaluated in three stages.

In the first stage, performance of the Packet Distribution Level was evaluated using a VHDL testbench which simulated packet distribution by sending data streams to the input ports of the IO-Buffers and monitoring packet distribution. Packet data was generated using Ezchip microcode development tools. In the simulation, design was able to reach the packet throughput required for handling wire speed on all MAC interfaces. The MAC interface was not included in the simulation to reduce the simulation time and to simplify the interpretation of the results.

In the second stage, performance of the Processor unit was evaluated by giving 8 bit data to processor unit and observes the time to leave this data to output. This was also done by a VHDL testbench.

In the third stage, the overall performance was evaluated by experiment with some real packet processing examples for IPv4 Forwarding execution, given in next section.

6.3.1 Real Packet Processing Operation

The following examples compare the typical number of clock cycles (where each instruction takes a clock cycle) required for the identical packet-processing task using proposed modified processor core, Ezchip TOPcore technology [15] and a general RISC processor [15]. The comparisons are shown in Table 6.1 below.

Table 6.1: Comparison of packet processing tasks

Task	Packet Processing Task	Clock Cycles		
		Proposed Modified Processing Core	EZchip TOPcore NP	General RISC NP
1	Parsing a IPv4 packet	53	60	400
2	Searching lookup tables for switching	8	6	200
3	Resolving a routing decision	10	8	80

The task 1 represents to parsing a typical IPv4 packet for determining its destination. The task 2 represents the searching of a 32 bit IPv4 IP address in a typical IPv4 packet from the lookup database. Finally the task 3 represents the resolving a routing decision.

From the table it is clear that EZchip TOPcore technology takes very less clock cycles compared to general RISC NP core. And the proposed modified architecture improves the EZchip TOPcore technology. Figure 6.3 shows a graph of this comparison tasks.

Parsing and searching is common operation mostly for all types of packets and for routing decision it takes extra resolve operation. In general, to process a packet the no. of clock cycles required for proposed modified core is $(53 + 8) = 61cc$ whereas for the EZchip TOPcore technology it takes $(60 + 6) = 66cc$. For the routing resolve operation the required clock cycles is $(53 + 8 + 10) = 71cc$ in proposed modified core and $(60 + 6 + 8)$

= 74cc in EZchip TOPcore technology. Therefore, the proposed modified architecture gives better performance.

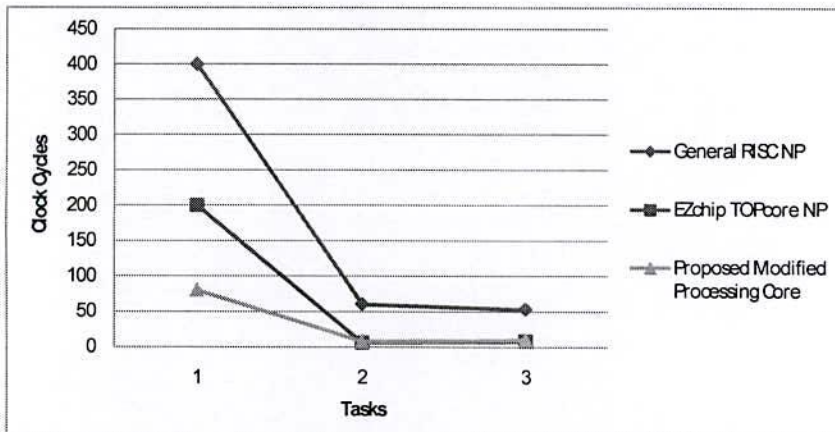


Figure 6.3: Performance comparison graph

6.4 Summary

Results show that the proposed designed architecture is efficient and provides better performance compared to common architectures. It also shows that for some specific tasks of interest to Network Processor system designers, the FPGA adaptability can provide the flexibility to process packet at wire speed. The performance summary of the system is -

- This processing technology provides more processing power per clock than other network processor designs.
- Super-pipeline and parallel architecture of the processor array boosts the performance.
- Pipelined memory architecture speeds the memory access time.
- Load balancer with an efficient load distribution algorithm provide higher processing throughput.
- Green technology provides saving of power consumption at ideal condition.

CHAPTER VII

Conclusion and Future Work

7.1 Conclusion

This thesis developed a high speed network processing system for efficient FPGA implementation. For this purpose, a 4 level layered processing architecture was developed which was scalable and allowed efficient packet processing. The layered architecture allowed for packet processing was independent of the packet distribution and for exceptions was processed outside the standard processing flow. The thesis also developed high throughput pipelined memory architecture to reduce the rate of memory access time. This design architecture was enabled by the development of a fast and effective load distribution algorithm for efficiently balanced the distributed load. For the design of high-speed packet processing core, EZchip's TOPcore technology was used with some modification. The modification was made for further speedup of processing capacity. The processor unit was designed and verified in EZchip microcode development environment. It was also simulated in NepSim simulator for process execution, power and other parameter observation. Then the design was modeled and simulated in RT level using VHDL and was synthesized to schematic. The synthesis was done for both Xilinx Spartan3 and Actel ProASIC3 FPGA.

The performance of the proposed network processor was evaluated for some real applications and compared with reference architectures. Results show that the proposed designed architecture is efficient and provides better performance. Design requires very little FPGA logic while efficiently processing packet. Implementation of super-pipelined parallel processing increases the performance and the FPGA adaptability can provide the flexibility to process packet at line speed. And implementation of green technology provide saving of power consumption at an ideal condition.

7.2 Future Work

The recent concentration of the researchers of this field are not only point to processing speed and performances, they are always try to make the network processor reconfigurable and supports all types of network protocols in a single processing unit. In this research work the thesis boosts the processing performance and prevents packet losses at the time of traffic overflow by providing load balancer and increasing buffer memory size. The techniques that used for load distribution in a load balancer in this thesis may be replaced by another way to further increase the performance. In this work, pipelined memory architecture is used for increase the rate of memory access time. For future work other memory access method may be reconsidered for enhancing the bandwidth of memory access. The core processing unit in this research work is based on Task Optimized Processing (TOP) core technology, other methods may also be considered to design the processing core faster and efficient.

REFERENCES

- [1] Andrew Heppel, "An Introduction to Network processors" Roke Manor research, January 16, 2003.
- [2] W. Slater, "Internet History and Growth", Internet Society, Chicago Chapter of the Internet Society, September 2002.
- [3] Yanai Danan, "HANP: A Highly Adaptive Network Processor" Thesis of Master of Engineering, McGill University, Montreal Canada, September 2002.
- [4] Xiaoning Nie, Lajos Gaz, Frank Engel and Gerhard Fettweis "A New Network Processor Architecture for High-Speed Communications"
- [5] D. Husak "Network Processors: A Definition and Comparison", White Paper. C-Port Corp., May 3, 2000.
- [6] N. Cravotta, "Network Processors: The sky's the limit" EDN, pp. 108-118, Nov. 24, 1999.
- [7] "Building Next Generation Network Processors", Agere, Inc. White paper. September 10, 1999.
- [8] "The Challenge for Next Generation Network Processors" White paper. Agere Systems Proprietary, April 2001.
- [9] Scott J. Harper, "A Secure Adaptive Network Processor" Doctor of Philosophy In Electrical Engineering, the faculty of Virginia Polytechnic Institute and State University, April 30, 2003.
- [10] Timur Diler, "Network Processors And Utilizing Their Features In A Multicast Design" Thesis, Master of Science In Computer Science and Electrical Engineering, Naval Postgraduate School, Monterey, California, March 2004.
- [11] Douglas Comer, John Steele and Raj Yavatkar, "A White Paper on High-Speed Network Architecture" Computer Science Department, Purdue University, West Lafayette, India, November 1988.
- [12] Jon S. Quarterman, Smoot Carl-Mitchell, "The Internet Connection - System Connectivity and Configuration" Addison-Wesley Publications, April-1994.
- [13] Tilman Wolf and Jonathan S. Turner, "Design Issues for High-Performance Active Routers" IEEE Journal On Selected Areas In Communications, Vol. 19, No. 3, pp-404-409, March 2001.
- [14] Lothar Thiele, Samarjit Chakraborty, Matthias Gries and Simon Kunzli, "Design Space Exploration of Network Processor Architectures"
- [15] "Network Processor Designs for Next-Generation Networking Equipment" White paper. EZchip Technologies, December 1999.
- [16] Dan Munteanu and Carey Williamson, "An FPGA-based Network Processor for IP Packet Compression"
- [17] Barry Katz, Douglas Burns, Stephen Coe, Robert Moles and Daniel Nilsson, "High-Speed Design Challenges for a 1.4GHz Network Processor" High-Performance Systems Design Conference, DesignCon 2003.

- [18] Patrick Crowley, Mark A. Franklin, Haldun Hadimioglu and Peter Z. Onufryk, "Network Processor Design" Issues and Practices, Volume 3, ELSEVIER 2005.
- [19] Gero Dittmann and Andreas Herkersdorf, "Network Processor Load Balancing for High-Speed Links"
- [20] Ada Gavrilovska, Karsten Schwan, Ola Nordstrom, Hailemeleket Seifu, "Network Processors as Building Blocks in Overlay Networks"
- [21] Madhu Sudanan Seshadri, John Bent, Tevfik Kosar, "Network processors: Guiding design through analysis"
- [22] "Next Generation Network Processor Technologies" Network Processor Division Intel Corporation, October 2001.
- [23] Hassan Shojania, "A brief study of network processors and their next generation trend" April 11, 2006.
- [24] A. N. M. Ehtesham Rafiq, M. Watheq El-Kharashi, Esam Khan, and Fayez Gebali, "A Study on Design Approaches for Network Processor Units" pp.-1000-1003, 2003 IEEE
- [25] Bob Wheeler and Jag Bolaria, "A Guide to Network Processors", Tenth Edition, Published by Linley Group, March 2009.
- [26] Maged Attia and Ingrid Verbauwhede, "Programmable Gigabit Ethernet Packet Processor Design Methodology" ECCTD'01 - European Conference on Circuit Theory and Design, August 28-31, 2001, Espoo, Finland
- [27] Gabriel Marchesan Almeida, Sameer Varyani, Gilles Sassatelli, R'emi Busseuil, "Self-adaptability in Multi-processor Embedded Systems"
- [28] Jurgen Foag, Nuria Pazos, Thomas Wild and Winthir Brunnbauer, "Self-Adaptive Parallel Processing Architecture For High-speed Networking" Proceedings of the 16th Annual International Symposium on High Performance Computing Systems and Applications (HPCS'02)
- [29] Robert Ennals, Richard Sharp, and Alan Mycroft, "Task Partitioning for Multi-Core Network Processors"
- [30] Patrick Crowley & Jean-Loup Baer, "A Modeling Framework for Network Processor Systems"
- [31] Mark Kohler, "Network Processor Overview"
- [32] Timothy Sherwood, George Varghese and Brad Calder, "A Pipelined Memory Architecture for High Throughput Network Processors" In Proceedings of the 30th International Symposium on Computer Architecture (ISCA), June 2003.
- [33] Matthias Gries, "Algorithm-Architecture Trade-offs in Network Processor Design" Thesis, Doctor of Technical Sciences, Swiss Federal Institute of Technology, Zurich, Germany, May 21, 2001.
- [34] Henrique Cota de Freitas, Carlos Augusto P. S. Martins, "Didactic Architectures and Simulator for Network Processor Learning"
- [35] Gokhan Memik, Seda Ogrenci Memik, William H. Mangione-Smith, "Design and Analysis of a Layer Seven Network Processor Accelerator Using Reconfigurable Logic"

- [36] Andrew Seawright, Renu Mehra, Arjuna Ekanayake, Barry Pangrle, "RTL C-Based Methodology for Designing and Verifying a Multi-Threaded Processor"
- [37] Taskin Kocak, "A Hybrid Network Processor with Support for High-Speed Execution of CPN and Upper Layer IP Protocols"
- [38] "Low Power Network Processor Design Using Clock Gating"
- [39] Tilman Wolf, and Jonathan Turner. "Design Issues for High Performance Active Routers," IEEE Journal on Selected Areas in Communications, vol. 19, issue: 3, pp. 404 - 409, March 2001.
- [40] Pak K. Chan and Martine D.F. Schlag. "New Parallelization and Convergence Results for NC: A Negotiation-Based FPGA Router," ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA 2000. Proceedings. Feb.2001. Page(s) 165 - 174.
- [41] A. S. Vaidya, Anand Sivasubramaniam and C. R. Das, "LAPSES: A Recipe for High Performance Adaptive Router Design," IEEE Symposium on High-Performance Computer Architecture, Jan. 1999. HPCA-5. Proceedings. Page(s) 236 - 243
- [42] F. Le Faucheur. "IETF Multiprotocol Label Switching (MPLS) Architecture", Feb 98. IEEE International Conference on ATM, ICATM 1998. Page(s): 6 -15
- [43] Christopher Metz, "IP ROUTERS: New Tool for Gigabit Networking," IEEE Internet Computing, vol. 2, issue: 6, pp. 14-18, Nov.-Dec. 1998.
- [44] L.S. Chen, and T.E. Stem, "Throughput Analysis, Optimal Buffer Allocation, and Traffic Imbalance Study of Generic Nonblocking Packet Switch," IEEE Journal on Selected Areas in Communications, vol. 9, issue: 3, pp. 439 - 449, April 1991.
- [45] P. Crowley, M. E. Fluczynski, J.-L. Baer, and B. N. Bershad. "Characterizing Processor Architectures for Programmable Network Interfaces," ACM International Conference on Supercomputing, ICS 2000. Proceedings. May 2000. Page(s) 54 - 65.
- [46] J. W. Lockwood, I. S. Turner and D. E. Taylor. "Field Programmable Port Extender (FPX) for Distributed Routing and Queuing," ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA 2000. Proceedings. Page(s) 137 - 144
- [47] Faraydon Karim, Anh Nguyen, Sujit Dey, and Ramesh Rao. "On-Chip Communication Architecture for OC-768 Network Processors," IEEE Design Automation Conference, DAC 2001. Proceedings. June 2001. Page(s) 678 - 683.