2004-03

# Network processors and utilizing their features in a multicast design

Diler, Timur

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

**MONTEREY, CALIFORNIA**

# THESIS

**NETWORK PROCESSORS AND UTILIZING THEIR FEATURES IN A MULTICAST DESIGN**

by

Timur DILER

March 2004

| | |
|---|---|
| Thesis Advisor: | Su WEN |
| Thesis Co-Advisor: | Jon BUTLER |

| | | | |
|---|---|---|---|
| **REPORT DOCUMENTATION PAGE** | | | *Form Approved OMB No. 0704-0188* |
| Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503. | | | |
| **1. AGENCY USE ONLY** ***(Leave blank)*** | **2. REPORT DATE** March 2004 | **3. REPORT TYPE AND DATES COVERED** Master's Thesis | |
| **4. TITLE AND SUBTITLE**: Network Processors and Utilizing their Features in a Multicast Design | | | **5. FUNDING NUMBERS** |
| **6. AUTHOR(S)** Timur DILER | | | |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Naval Postgraduate School Monterey, CA 93943-5000 | | | **8. PERFORMING ORGANIZATION REPORT NUMBER** |
| **9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)** N/A | | | **10. SPONSORING/MONITORING AGENCY REPORT NUMBER** |
| **11. SUPPLEMENTARY NOTES** The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. | | | |
| **12a. DISTRIBUTION / AVAILABILITY STATEMENT** **Approved for public release; distribution is unlimited** | | | **12b. DISTRIBUTION CODE** |
| **13. ABSTRACT (maximum 200 words)** In order to address the requirements of the rapidly growing Internet, network processors have emerged as the solution to the customization and performance needs of networking systems. An important component in a network is the router, which receives incoming packets and directs them to specific routes elsewhere in the system. Network processors and the associated software control the routers and switches and allow software designers to deploy new systems such as multicasting forwarder and firewalls quickly.This thesis introduces network processors and their features, focusing on the Intel IXP1200 network processor. A multicast design for the IXP1200 using microACE is proposed.This thesis presents an approach to building a multicasting forwarder using the IXP1200 network processor layer-3 forwarder microACE that carries out unicast routing. The design is based on the Intel Internet exchange architecture and its active computing element (ACE). The layer-3 unicast forwarder microACE is used as a basic starting point for the design. Some software modules, called micoblocks, are modified to create a multicast forwarder that is flexible and efficient. | | | |
| **14. SUBJECT TERMS** IXP1200, multicasting, ACE, microace, network processors, Intel IXA, microengine | | | **15. NUMBER OF PAGES** 74 |
| | | | **16. PRICE CODE** |
| **17. SECURITY CLASSIFICATION OF REPORT** Unclassified | **18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified | **19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified | **20. LIMITATION OF ABSTRACT** UL |

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

# NETWORK PROCESSORS AND UTILIZING THEIR FEATURES IN A MULTICAST DESIGN

Timur DILER
Lieutenant Junior Grade, Turkish NAVY
B.S., Turkish Naval Academy, 1998

Submitted in partial fulfillment of the
requirements for the degrees of

**MASTER OF SCIENCE IN COMPUTER SCIENCE**
**and**
**MASTER OF SCIENCE IN ELECTRICAL ENGINEERING**

from the

**NAVAL POSTGRADUATE SCHOOL**
**March 2004**

Author: Timur Diler

Approved by: Su Wen
Thesis Advisor

Jon Butler
Thesis Co-Advisor

John P. Powers
Chairman, Department of Electrical and Computer Engineering

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

# ABSTRACT

In order to address the requirements of the rapidly growing Internet, network processors have emerged as the solution to the customization and performance needs of networking systems. An important component in a network is the router, which receives incoming packets and directs them to specific routers elsewhere in the system. Network processors and the associated software control the routers and switches and allow software designers to quickly deploy new systems such as multicasting forwarders and firewalls.

This thesis introduces network processors and their features, focusing on the Intel IXP1200 network processor. A multicast design for the IXP1200 using microACE is proposed.

This thesis presents an approach to building a multicasting forwarder using the IXP1200 network processor layer -3 forwarder microACE that carries out unicast routing. The design is based on the Intel Internet exchange architecture and its active computing element (ACE). The layer -3 unicast forwarder microACE is used as a basic starting point for the design. Software modules, called micoblocks, are developed to create a multicast forwarder that is flexible and efficient.

THIS PAGE INTENTIONALLY LEFT BLANK

# TABLE OF CONTENTS

I. INTRODUCTION . . . . . . . . . . 1
- A. BACKGROUND . . . . . . . . . . 1
- B. THESIS PROBLEM STATEMENT . . . . . . . . . . 2
- C. THESIS OVERVIEW . . . . . . . . . . 2

II. NETWORK PROCESSORS . . . . . . . . . . 5
- A. INTRODUCTION . . . . . . . . . . 5
- B. THE EVALUATION OF THE NETWORK PROCESSORS . . . . . . . 5
  - 1. First-Generation Systems . . . . . . . . . . 6
  - 2. Second-Generation Systems . . . . . . . . . . 7
  - 3. Third-Generation Systems . . . . . . . . . . 8
- C. FEATURES OF NETWORK PROCESSORS . . . . . . . . . . 9
  - 1. Flexibility with Programmability . . . . . . . . . . 9
  - 2. Scalability . . . . . . . . . . 10
  - 3. Pipelined Processing . . . . . . . . . . 11
  - 4. Parallel Processing . . . . . . . . . . 12
  - 5. Memory Management . . . . . . . . . . 13
- D. INTEL EXCHANGE ARCHITECTURE . . . . . . . . . . 14
  - 1. Data Plane . . . . . . . . . . 15
  - 2. Control Plane . . . . . . . . . . 15
  - 3. Management Plane . . . . . . . . . . 15
- E. SUMMARY . . . . . . . . . . 15

III. INTEL IXP1200 ARHITECTURE . . . . . . . . . . 17
- A. INTRODUCTION . . . . . . . . . . 17
- B. OVERVIEW OF IXP1200 . . . . . . . . . . 17
- C. IXP1200 COMPONENTS . . . . . . . . . . 18
  - 1. StrongARM Core . . . . . . . . . . 18
  - 2. MicroEngines . . . . . . . . . . 19
  - 3. SRAM and Internal SRAM Interface Unit . . . . . . . . . . 24
  - 4. SDRAM and Internal SDRAM Interface Unit . . . . . . . . . . 25
  - 5. PCI Unit . . . . . . . . . . 25
- D. SUMMARY . . . . . . . . . . 25

IV. IXP1200 PROGRAMMING AND MICROACE STRUCTURE . . . . . . . . . 27
- A. INTRODUCTION . . . . . . . . . . 27
- B. IXP1200 PROGRAMMING . . . . . . . . . . 27
  - 1. Software Development Kit . . . . . . . . . . 28
  - 2. Instruction Set . . . . . . . . . . 29
  - 3. MicroEngine Assembly Syntax . . . . . . . . . . 31

4. Simple Packet Data Flow in IXP1200 . . . . . . . . . . . . . . . . . 33
C. ACE PROGRAMMING MODEL . . . . . . . . . . . . . . . . . . . . . . . . 34
1. IXA Application Programming Interface . . . . . . . . . . . . . . . . 34
2. Active Computing Element . . . . . . . . . . . . . . . . . . . . . . . . 35
3. MicroACE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
4. An Example of MicroACE Processing . . . . . . . . . . . . . . . . . 37
5. The Dispatch Loop . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
D. SUMMARY . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39

V. A MULTICASTING FORWARDER DESIGN USING MICROACE . . . . . 41
A. INTRODUCTION . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
B. MULTICASTING FROM UNICAST FORWARDING . . . . . . . . . . 41
C. MULTICASTING FORWARDER MICROACE . . . . . . . . . . . . . . 43
D. SUMMARY . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49

VI. CONCLUSIONS AND RECOMMENDATIONS . . . . . . . . . . . . . . . . . . . 51

LIST OF REFERENCES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53

INITIAL DISTRIBUTION LIST . . . . . . . . . . . . . . . . . . . . . . . . . . . 55

# LIST OF FIGURES

Figure 1. Software-Based Architecture. ....................6
Figure 2. Scalability Achievement. ....................11
Figure 3. Parallelism in the IXP1200 NPU. ....................13
Figure 4. IXP1200 Block Diagram (From Ref. 9.). ....................18
Figure 5. StrongARM Block Diagram (After Ref. 9.). ....................19
Figure 6. MicroEngine Internal Structure (From Ref. 9.). ....................20
Figure 7. Thread Context Swapping (From Ref. 9.). ....................22
Figure 8. GPR Addressing (From Ref. 9.). ....................23
Figure 9. Transfer Reg. Addressing (From Ref. 9.). ....................24
Figure 10. IXP1200 Macro Example (From Ref. 18.). ....................33
Figure 11. Simple Data Flow Hardware (From Ref. 9.). ....................33
Figure 12. Packet Flow (From Ref. 16.). ....................35
Figure 13. MicroACE App. Using Pseudo-code (From Ref. 17.). ....................37
Figure 14. IP Forwarding MicroACE (After Ref. 2,16.). ....................38
Figure 15. Dispatch Loop (From Ref. 2,17.). ....................39
Figure 16. Basic Multicasting System. ....................42
Figure 17. Multicasting MicroACE (After Ref.2,16.). ....................45
Figure 18. Multicasting MicroACE Algorithm. ....................46
Figure 19. Initialization Routine. ....................48

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF TABLES

Table 1. Internet Growth Trends (From Ref. 1.)..............................................................1
Table 2. MicroEngine Execution Pipeline (From Ref. 9.). ............................................12
Table 3. IXP1200 SDK Items (From Ref. 2.). ...............................................................29
Table 4. MicroEngine Basic Inst. Set (From Ref. 15.). .................................................30
Table 5. Manual Register Assignment Directives (After Ref. 15.)................................31
Table 6. Register Addressing (After Ref. 2,15.). ...........................................................32
Table 7. Data Flow Steps (After Ref. 9.). ......................................................................34
Table 8. Multicast Identifier Examples..........................................................................44

THIS PAGE INTENTIONALLY LEFT BLANK

# ACKNOWLEDGMENTS

I would like to thank to my thesis advisors Prof. Su Wen and Prof. Jon Butler for their suggestions and advice during the development of this thesis.

I also would like to thank my Computer Science professors and Electrical and Computer Engineering professors, Prof. Jon Butler, Prof. Hersch Loomis and Prof. Douglas Fouts who made me enthusiastic about computer engineering again.

Special thanks to my wife and my daughter, for the sacrifices they made in support of my completing this thesis in their time.

This thesis is dedicated to my wife, Munevver Diler and my daughter Yagmur Diler.

THIS PAGE INTENTIONALLY LEFT BLANK

# EXECUTIVE SUMMARY

The main goal of this thesis was to design a multicasting forwarder using Intel Internet exchange architecture and the IXP1200 network processor. The combat system computers on warships are connected to each other with a network to respond any attack immediately. The sensors and the weapons of the ships communicate with each other by using this network. The communication between systems must be lightweight to increase the respond time. This thesis can be used in these systems to eliminate network overheads. The sensors can send the information packets to control consoles by using unicast based multicasting to eliminate network overhead.

Originally, conventional CPU-based systems were used to build networking systems (routers and switches). They had sufficient computing capacity to handle the network requirements. Around 1990, the Internet begun to become a global network, and CPU-based systems could no longer handle the Internet applications requirements because of demand for wide bandwidth and high packet processing rates. In the middle of 1990, the application specific integrated circuit (ASIC) was introduced to address new requirements. ASIC is an integrated circuit with networking functions built-in permanently. ASICs are fast and have a high packet processing rate. However, since functions are permanently built into ASIC, they cannot be easily modified. As the number and variety of Internet applications grew, new networking functions were needed. It is very expensive and time consuming to design and produce new ASICs. This is one of the main reasons network processors were introduced. Network processors are processors that are designed for network processing and have special features to handle the high packet rate, new networking functions, and new services.

Network processors are fully programmable processors. This makes them flexible enough to address new application needs in a short time. They use parallelism and pipelining to increase the throughput and support high packet processing rates.

Intel designed and produced Internet Exchange Architecture (IXA) and its IXPxxxx family network processors. IXP1200 is one of them. Internet Exchange Architecture introduced the programming model Active Computing Element (ACE) to modularize network processor programming. The IXP1200 has six hardware multithreading RISC MicroEngines, and one StrongARM core RISC processor. Microengines and StrongARM work in parallel using a 5-stage pipelining execution queue.

Intel provides the IXP systems with a hardware testbed and platforms to help designers evaluate their applications. The Software Development Kit (SDK) is one of these platforms. SDK includes sample applications and library codes to help the designer.

The ACE software programming structure divides the tasks performed by an IXP-based system and allows each task to be handled by a module. MicroACE has two components, microblocks which run in MicroEngines and a core component, which runs in the StrongARM core processor. Microblocks handle the common packets and provide a fast data path. If they encounter a packet requiring special handling, they pass it to the core component.

The multicast forwarder, simply a router that also performs multicast forwarding, forwards each packet to one or more receivers. This thesis investigates a unicast-based design that provides multicast service using the existing unicast forwarder. It is an effective and flexible methodology because the system does not require additional routing protocols or another routing algorithm. End hosts in the network are responsible for multicast state maintenance.

THIS PAGE INTENTIONALLY LEFT BLANK

# I. INTRODUCTION

## A. BACKGROUND

The Internet has improved the quality of our lives. It has made it more convenient to do our jobs, to communicate with each other, to shop, to conduct research and a host of other tasks. Today, the Internet is everywhere. Even the telephone system is changing from the old dedicated analog transmission system to voice-IP technology over the Internet. Military applications use computer network systems to communicate in the battlefield. On warships, data from the sensors can be used as input to weapon systems. This combination can be is achieved using a computer network. A sensor must send its information to the control consoles of the combat information center to decide about the threats. Our unicast-based multicast design can be used in the warship combat systems as multicasting forwarder. It provides effective and reliable way to deliver information to the consoles.

The Internet grew rapidly especially after 1990, and it is still growing. Table 1 shows the growth of the Internet as measured by the total number of hosts.

| Year: | Number of Hosts on Internet: |
|---|---|
| 1977 | 111 |
| 1981 | 213 |
| 1983 | 562 |
| 1984 | 1,000 |
| 1986 | 5,000 |
| 1989 | 100,000 |
| 1992 | 1,000,000 |
| 2001 | 150 – 175 million |
| 2002 | Over 200 million |
| By 2010 | About 80% of the planet |

Table 1. Internet Growth Trends (From Ref. 1.).

Since the number of hosts on the Internet is growing, applications and services also are being developed. These growing applications and services require high bandwidth to work reliably.

Networking technology has an important role in supporting this growth. Protocol systems, transmission media, processing systems like routers, switches, and bridges are the main components of computer networking.

To handle the huge demand on the Internet, networks must be fast, reliable, and flexible. As a result, processors used in networking hardware have evolved from general CPU to specialized packet handling type Network Processor Units that are faster and optimized for moving data.

## B. THESIS PROBLEM STATEMENT

The main goal of this thesis was to propose a design of multicasting forwarding service that uses the existing IXP1200 ACE programming design module. To support the multicast design, this thesis explores the network processor and its features, and investigates the Intel IXP1200 network processor and its ACE programming model.

## C. THESIS OVERVIEW

Chapter II examines network processors. First, it introduces the evaluation of the network processors. Second, it gives the features of network processors that must address the high bandwidth requirement. Third, it presents the Intel Internet Exchange Architecture technology.

Chapter III describes the hardware and the architectural concepts of the Intel network processor IXP1200. It explains the external and internal blocks of the IXP1200 and introduces the concepts of hardware multithreading, memory management, and interconnection between blocks.

Chapter IV explores the software component of the IXP1200. It explains the programming structure and models of IXP1200, including the software development kit, IXA application-programming interface, and the advanced programming model, the active computing element.

Chapter V provides our design for a multicasting forwarder. Our methodology is done by modifying the layer-3 unicast forwarder microACE. The microACE of the unicast forwarder includes three main microblocks. We modify the ingress and egress microblocks to duplicate specific packets effectively converting from a unicast mode to a multicast mode.

The last chapter discusses the conclusions and recommendations for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

# II. NETWORK PROCESSORS

## A. INTRODUCTION

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.

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.

Network processors present a solution, which can help maximize bandwidth utilization and traffic flow [3,4]. Network processors are becoming the main component in the network systems to meet the new bandwidth, speed, and performance requirements.

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 [2,3,4]. For high bandwidth and performance, parallelism and pipelining are used in the design of Network Processors.

This chapter examines the features of Network Processors, focusing on the system processor IXP1200 in the INTEL Network Processor Architecture.

## B. THE EVALUATION OF THE NETWORK PROCESSORS

Over the last 15 years, network systems especially router architectures, have evolved through three generations, each marked by improvements in packet processing mechanisms.

### 1. First-Generation Systems

Up to the mid 1990's, router architecture was similar to the conventional PC systems. Figure 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.

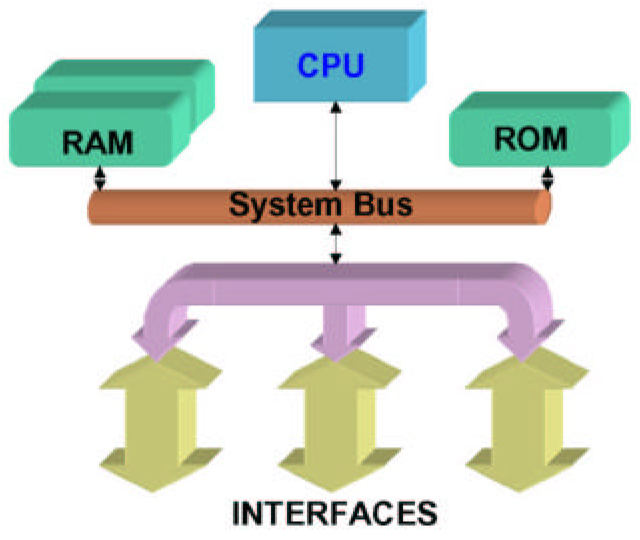

Figure 1. Software-Based Architecture.

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 [2,3,4].

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 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]. When you want to make them 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.

### 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, you 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].

### 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. These features are implemented with parallelism and pipelining and are discussed in detail in the next section.

## C. FEATURES OF NETWORK PROCESSORS

Network Processors brought new concepts and new technologies to networking systems. In this section, the INTEL IXP1200 Network Processor will illustrate the features of network processors in general. The IXP1200 has StrongARM, a core RISC processor, 6 pipelined Multithreaded (4 threads for each Microengine) RISC-type Microengines for packet processing, SDRAM to store packet data, SRAM to store packet headers and variables, and system buses to establish communications between the units.

### 1. Flexibility with Programmability

Internet technology and applications are changing every day. New network systems must be adapted to new protocols, functions and services at low cost. Unlike ASICs, Network Processors are not limited to a particular layer stack or a protocol.

Flexibility of network processors arises from programmability [2]. With frequently changing network requirements and standards, programmability is an important characteristic of network processors. Rather than designing and producing a new chip like ASIC, one can design and create systems for new protocols or applications by only developing new algorithms and implementing them in programs.

Programmability allows designers to reuse the components and programs for different versions of protocols. System software tools designed for network processors shorten the development time for a system. Tools allow extensive testing capabilities with intelligent debugging features [4] with real-world conditions.

The flexibility of Network Processors means one can develop any network system with a network processor for any network protocol or services by just programming the network processor. Network processors yield low cost, reduce the development cycle, and allow programs to be reused.

The flexibility of network processors also means new chips may allow system designers perform tasks that people never imagined before [6]. Network Processors let designers handle complex functions and edge systems, like quality of service implementation and firewall mechanisms.

Also vendors are not stuck with only switches and routers. With the Network Processor, they can design and manufacture new systems.

### 2. Scalability

Scalability is an important issue for Computer Engineering. Processors and chips must handle the growing load demand by new applications and services. VLSI silicon systems have working limits, such as a maximum clock rate and latency. These conditions allow processors and chips to handle a limited amount of load for network systems packets.

With an increasing number of packets to be processed, systems must be scalable enough to handle that amount of packet load. It must have the ability to scale high data and packet rates. Responding each time by making faster processors for new systems is difficult and costly.

Instead of using faster processors for scalability, designers used parallelism, pipelining, and memory management to achieve scalability for network processors. These features are keys to scale high packet rates. Figure 2 shows simple internal structure of the IXP1200 to illustrate parallelism, pipelining, and memory management.

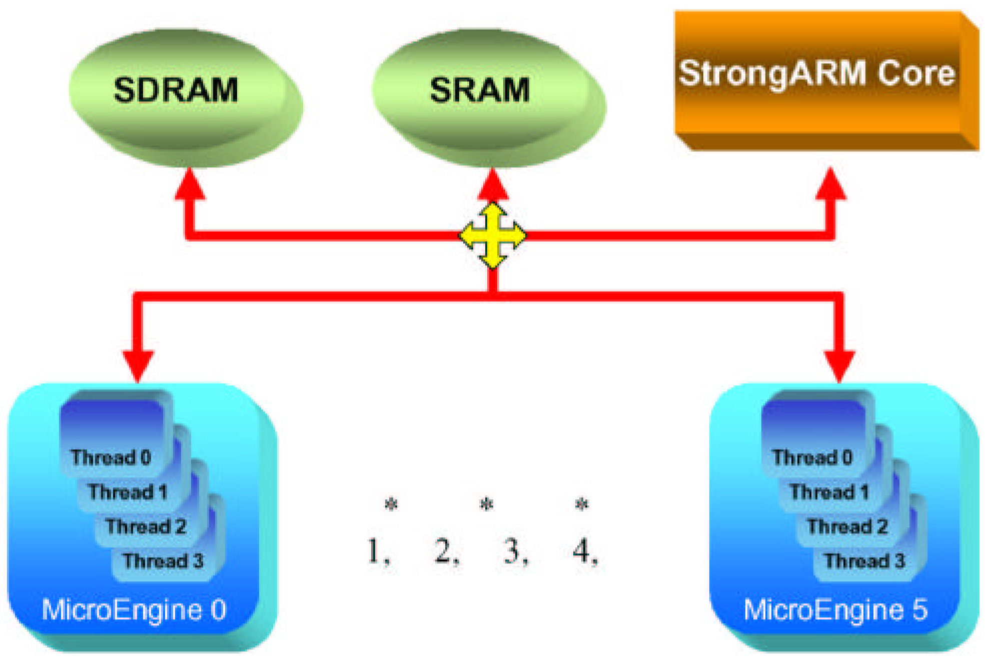

Figure 2. Scalability Achievement.

## 3. Pipelined Processing

Pipelining is one method of achieving scalability. With pipelining, instead of making processors faster, more instructions are executed with the same clock frequency.

The main goal of pipelining is to keep processors as busy as possible. Without pipelining, instructions are executed one-by-one. Instructions wait for completion of previous instruction execution cycle. Pipelining allows several instructions to be issued per clock cycle instead just one [8].

Pipelining increases throughput, at the expense of latency [8]. Latency in pipelined architecture is larger because every instruction must pass through all stages. If there are five stages, every stage block has a separate register block. At any time, several instructions are in the execution queue, having just completed some stages of their execution cycle, and waiting to pop from the queue. Unfortunately, dependencies among instructions in a pipeline can be cause delay. For example, there is branch delay for branch instructions. Branch delay is discussed in the next chapter.

The six RISC MicroEngines in IXP1200 have five stage-pipelined structure. Table 2 shows each stage and its functionality. This allows all instructions to execute in one clock cycle [9].

| Pipeline Stage | Description |
|---|---|
| P0 | Lookup of instruction |
| P1 | Initial instruction decode and formation of the source register address |
| P2 | Reads operand from source registers |
| P3 | Perform ALU, shift, or compare operations and generate the condition codes |
| P4 | Write result to the destination register |

Table 2. MicroEngine Execution Pipeline (From Ref. 9.).

For Table 2, in stage P0, the instruction is fetched from the instruction store. In P1, the instruction is decoded and the operation to be performed is determined. In P2, the operands of the instruction are read from registers. In P3, operands are passed through the ALU [9]. In P4, the result from the fourth stage is written to the destination registers.

This design allows a complete instruction to be executed each clock cycle, except for branch instructions.

### 4. Parallel Processing

Network processors employ more than one MicroEngine RISC processor in parallel to increase the packet rate. At any point in time, every RISC inside of the Network Processor can be computing a different networking function.

There must be a control mechanism to control the synchronization and communications between the RISCs. It can be another RISC processor or a simple control unit. The functions of each parallel processor can be determined by programming the network processor. Since the various RISCs process packets at the same time, this increases the packet rate.

Intel's Network architecture uses this concept. The IXP1200 network processor has six RISC processors or MicroEngines (Figure 3) that can run in parallel. Each MicroEngine has four separate threads running in the MicroEngine concurrently. When a thread in a MicroEngine processes an instruction to access memory, it can permit another thread to run, while the previous thread performs its memory access.

Every MicroEngine has a separate program counter for each thread. One can partition memory into blocks for each thread or let them share memory with each other.

As shown in Figure 3, an IXP1200 Network Processor actually has six Microengines and each of these has four separate threads, and so it can have a total of 24 different threads in parallel. This approach allows the IXP1200 to handle high data rates.

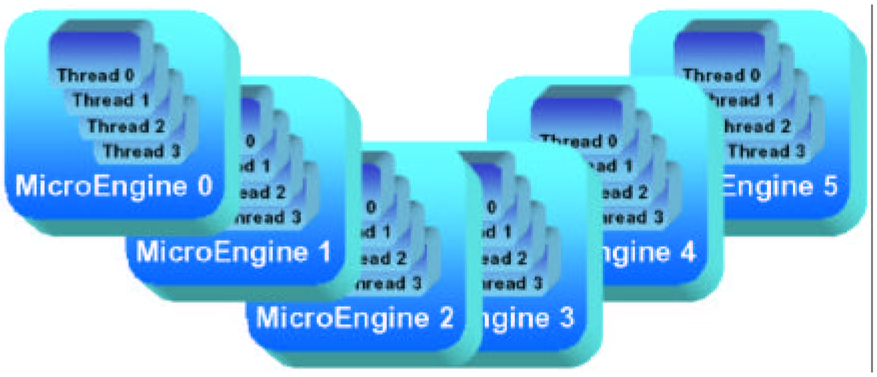

Figure 3. Parallelism in the IXP1200 NPU.

## 5. Memory Management

The most time-consuming process in computing is memory access. Reading and writing or transferring data from memory takes more time than most other processing jobs.

SRAM has low latency, but its cost is very high. SDRAM is cheaper than SRAM, but it has higher latency. Therefore, a trade-off between latency and cost must be considered when designing a new chip.

As shown in Figure 2, the INTEL IXP1200 has 8 Mbytes SRAM and 256 Mbytes SDRAM. It uses SRAM to store the routing table for lookups where low latency is important, and SDRAM to store packet data, payload, or very large tables where latency is not important [9].

As mentioned in the previous section, while a thread in the same Microengine executes a memory reference command, the MicroEngine can swap threads that are ready to process. So, no one has to wait for a memory cycle to be completed. Therefore, a process does not have to be blocked while waiting for memory access to be completed. This eliminates the unused time frames. It keeps the Network Processors busy as much as possible.

## D. INTEL EXCHANGE ARCHITECTURE

Networks and Internet have become a core component of daily operations [10]. The applications are getting bigger and more complex. The Intel Internet Exchange Architecture was designed to address the new application requirements.

IXA has all the features of the network processors mentioned in the previous section. The programmability and high packet rate performance allow systems to be developed in a short time.

There are four important features of the IXA [11]:

- Flexibility,
- High Performance (ability to process high packet and data rates),
- Scalability, and
- Software Portability.

To meet these requirements, Intel created the Intel IXPxxxx series Network Processors and Intel software portability framework. They are the components of the Intel Internet Exchange Architecture (IXA) [11].

There are three basic task levels in the IXA software architecture [10].

#### 1. Data Plane

The data plane takes care of packet processing. The forwarding of incoming packets is done at high speed. The data plane receives packets from a network interface, makes a classification, and determines the required action. Rules determine which action is taken on the packet.

The data plane handles the fast data path and is controlled by the MicroEngines.

#### 2. Control Plane

This part of the IXA application is the controller. It handles the time-consuming and complex tasks that are encountered while packet processing [11].

The control plane is implemented in the StrongARM core component. When the data plane encounters an unexpected packet or a packet that does not have any forwarding table entry, it passes that packet to the control plane where that packet is processed.

#### 3. Management Plane

The management plane performs the managing functions. It is a manager program at the top of the hierarchy. The manager can be an application off the chip or a Linux application running on the chip [11]. The management program of the IXA application is part of the main system, and can have a user interface to interact with the user.

## E. SUMMARY

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. ASIC played an important role in network systems. After different Internet and networking applications were introduced, the drawbacks of ASICs became apparent. Designing and producing ASICs to address the requirements of a new application was very expensive and time consuming.

Finally, vendors introduced network processors. Network processors are now in every networking system. They contain more than one RISC processor to increase the packet processing performance using parallelism and pipelining, and they are programmable. This allows software reusability, and the product can be produced quickly.

Intel introduced its Internet Exchange Architecture for networking systems. With IXA, Intel designed IX technologies, which included the IXPxxxx network processor family and the ACE programming structure.

IXP1200 is one of the Intel's network processors. The next chapter examines the IXP1200 hardware and its concepts. IXP1200 has six multithreaded programmable RISC MicroEngines, StrongARM core processor, memory interfaces and buses.

# III. INTEL IXP1200 ARHITECTURE

## A. INTRODUCTION

The previous chapter described the main concepts of network processors, including which features they have to address new bandwidth requirements.

Network processor functions are managed through system software; therefore, they provide the programmability and reusability features of software together with high-performance processing of the hardware in aiding system design. As a result, network processors enable designers to design and to manufacture more intelligent and compact network systems. Intel's network processor is part of their Internet Exchange Architecture. The IXPxxxx is a family of network processors produced by Intel. Currently, the second-generation IXP2xxx network processors are being developed.

The IXP1200 is one of the first network processors of the Intel IXA technology. This chapter presents an in-depth examination of Intel's IXP1200 structure and concepts with hardware.

Because of the complexity of Intel processors, some internal units unrelated to this thesis will not be discussed.

## B. OVERVIEW OF IXP1200

IXP1200 addresses the requirements of today's networking technology. To achieve high-speed data manipulation, and high packet rate, the IXP1200 includes programmability, pipelining, parallel processing, and memory management. These give the IXP1200 flexibility, scalability, high performance, and low power consumption [9].

Figure 4 shows the internal blocks, the external, and internal interfaces of IXP1200. The IXP1200 contains one RISC StrongARM core processor, six multithreaded programmable RISC MicroEngines, memory interfaces, and system bus interfaces.

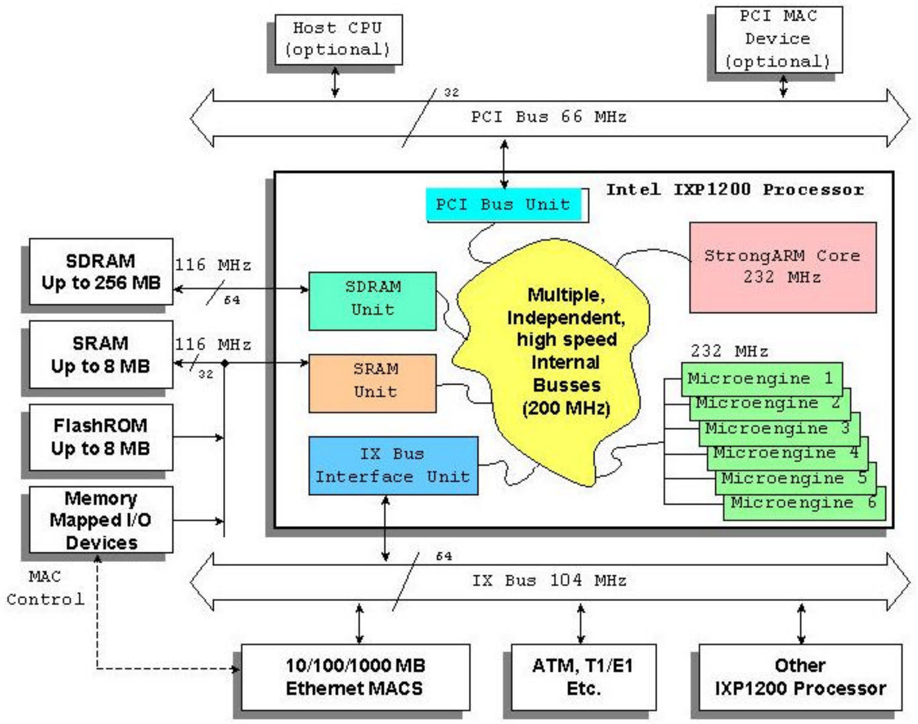

Figure 4. IXP1200 Block Diagram (From Ref. 9.).

Unlike ASICs, the IXP1200 allows the implementation of networking systems with software without considering the hardware structure. Within the software development environment, it is easy to develop, debug, and modify networking systems.

## C. IXP1200 COMPONENTS

### 1. StrongARM Core

The StrongARM is a 32-bit RISC microprocessor. It runs at 232 MHz. The StrongARM RISC core is a 5-stage pipelined processor. Figure 5 shows that it has 16 Kbytes of instruction cache, and 8 Kbytes of data cache. It also has 512 bytes of mini-cache to decrease transfers to and from the main data cache.

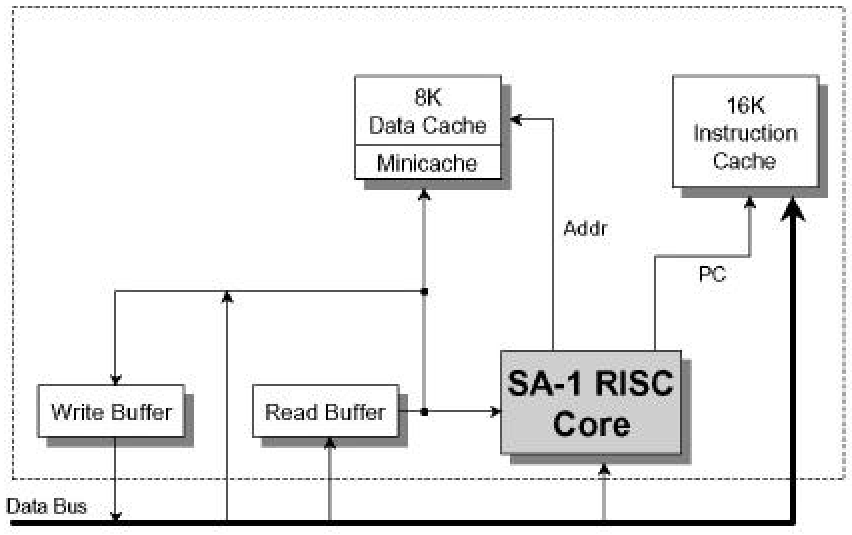

Figure 5. StrongARM Block Diagram (After Ref. 9.).

Depending on the system architecture, StrongARM may or may not be used. If the system has a main host CPU, the host CPU can maintain the system, upload the software, and operate the system. The StrongARM can do the exception handling and be the higher layer processor [9]. The StrongARM leaves the packet forwarding to the MicroEngines, but runs the routing protocols. It controls the IXP1200 system, MicroEngines and interfaces between components. If there is not any host CPU, StrongARM can assume the role of a host processor and perform system maintenance.

### 2. MicroEngines

IXP1200 contains six 32-bit multithreaded RISC Microengines. The Microengines can handle packet processing at a high rate. The six MicroEngines run in parallel to increase the total throughput of the system. The Microengines are fully programmable.

The MicroEngines, shown in Figure 6, have four hardware threads. Each of them has its own Program Counter to execute different instruction parts of the MicroEngine.

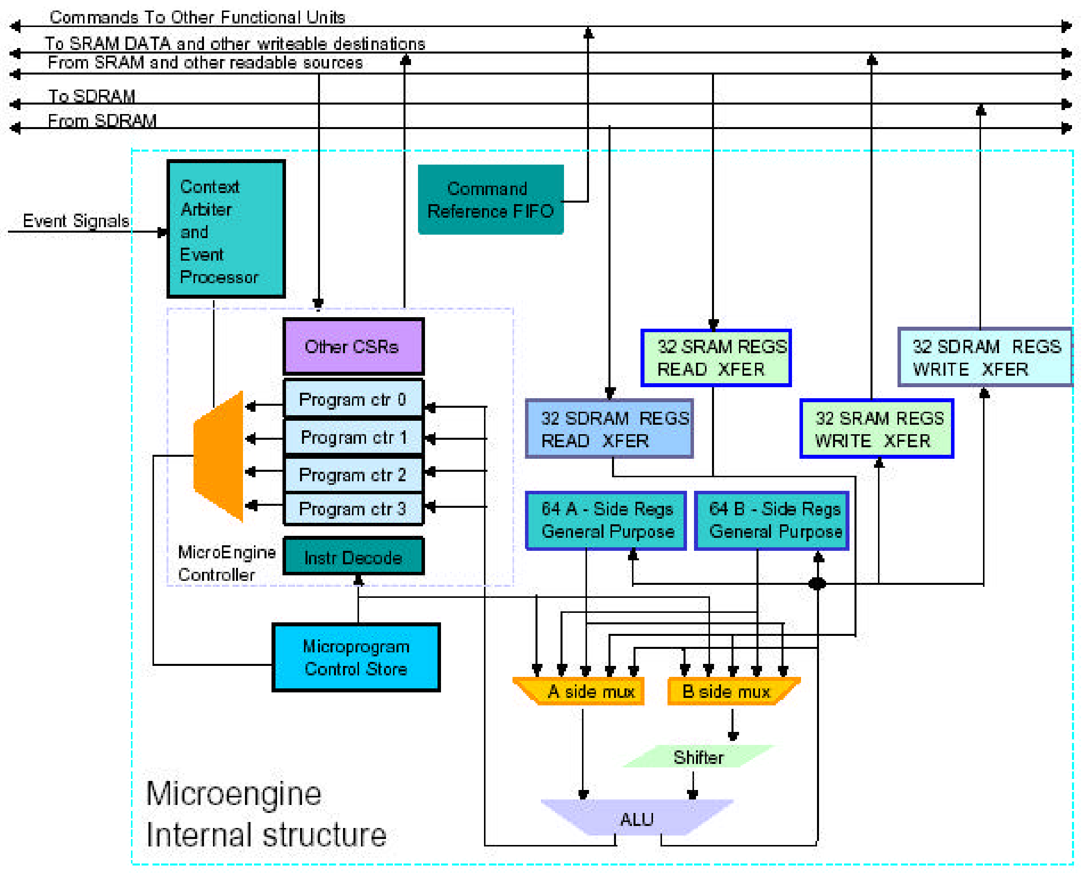

Figure 6. MicroEngine Internal Structure (From Ref. 9.).

The MicroEngines operate at 233 MHz. They are implemented as a 5-stage pipelined RISC processor. This pipelined structure makes the IXP1200 faster and scalable to higher rates in the future. A non-pipelined architecture would execute an instruction with, for example, five clock cycles. However, because of pipelining, the IXP1200 executes an instruction in each clock cycle. Table 1 in Chapter 1 shows the IXP1200 pipelined execution stages.

Branch instructions are a problem with pipelining architectures [7,8]. The instructions after the branch instruction may have already been inside the pipelining queue stages with their operands, but if the branch is taken, execution of these instructions will have to be aborted, consuming extra time and sacrificing performance. There are several solutions for this problem. The instructions can be pushed into a pipelining queue with the probability of not executing the branch instruction. By putting bubbles after the branch instructions, this problem can be solved, but this reduces the performance of the

MicroEngine. Bubbles are no operation instructions. Instead of doing this, instructions before the branch would be put after the branch to maintain performance [9]. The instructions before the branch will be executed anyway whether the branch is taken or not. Another solution is using the guess-branch-taken instruction. When the guess-branch-taken instruction is used, the instruction at the branch destination will be started [9].

While a processor executes instructions, memory accesses consume a lot of time. To reduce this problem, the IXP1200 has several memory access reduction features.

The first is to store instructions in a separate memory near the MicroEngine. This special memory stores 1K x 32 bits instructions. Each instruction is 32 bits long.

The second feature is hardware thread context swapping. The IXP1200 has 8 Mbytes of SRAM, and 256 Mbytes of SDRAM. SRAM is used to store table lookups where low latency is an important issue [9]. SDRAM is used to store packet data, payload, and very large tables where latency is not very an important issue [9]. To access data that is external to the Microengine, like SRAM or SDRAM, the Microengine executes memory access and transaction commands. These commands are called Reference Commands.

Every MicroEngine has four hardware threads (Figure 7). Each thread has its own program counter. Four threads can be executing the same code or different code pieces of the MicroEngine's instruction store at any one time.

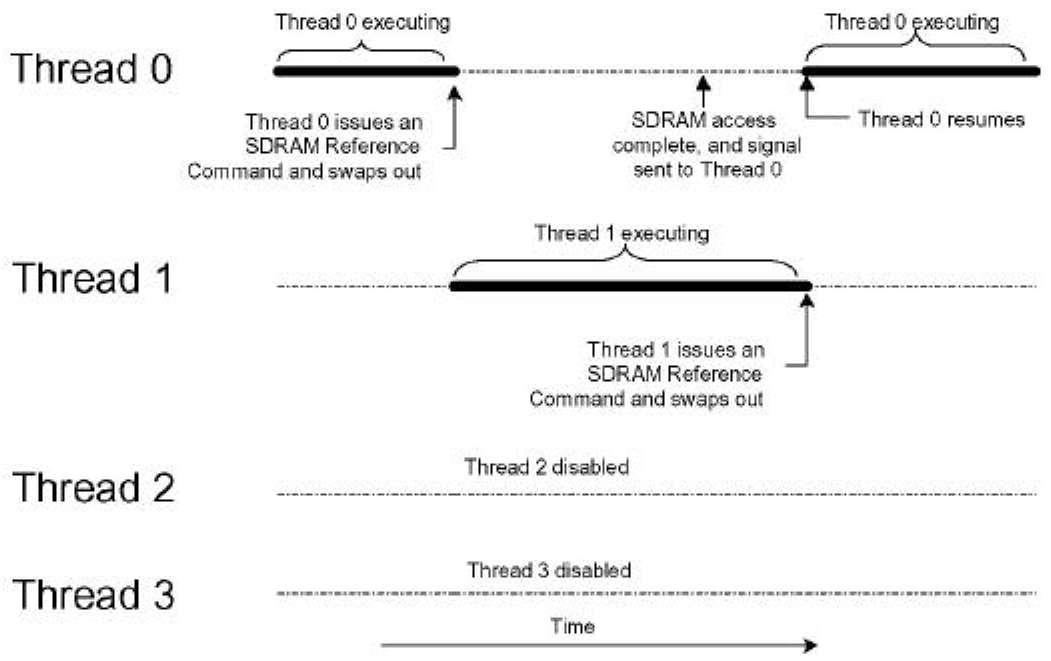
Thread 0
Thread 0 executing
Thread 0 executing
Thread 0 issues an SDRAM Reference Command and swaps out
SDRAM access complete, and signal sent to Thread 0
Thread 0 resumes
Thread 1
Thread 1 executing
Thread 1 issues an SDRAM Reference Command and swaps out
Thread 2
Thread 2 disabled
Thread 3
Thread 3 disabled
Time

Figure 7. Thread Context Swapping (From Ref. 9.).

When Thread 0 executes a reference command to access memory, the Microengine does not wait for that thread to complete its memory access. The control unit gives execution priority to the other thread, Thread 1, and the Thread 0 swaps out. Thread 0 goes to sleep until Thread 1 completes its cycle. This is called Hardware Multithreading. There is a difference between hardware and software threading. With hardware threading each thread has its own program counter, register and memory block, if relative addressing is used. With software threading, there is only one program counter and register set. The operating system makes all threads of processes share execution time according to priorities. In the IXP1200, all multithreading coordination and context swapping are handled by hardware only. The programmer need not worry about programming the threads. In software threading, all thread processing is coordinated by the operating system.

Every MicroEngine has 256 32-bit registers. Of these 128 are general purpose registers (GPR) and the other 128 are transfer registers.

The MicroEngines use two types of addressing of registers, context relative addressing and absolute addressing.

With context relative addressing, each thread in the same Microengine uses its registers block. It is assured that none of the threads overwrites another's registers. GPRs and transfer registers are divided into equal size blocks for each thread. If one or more threads require sharing some registers or communicating with each other, absolute addressing is used. With absolute addressing, the threads do not have to go beyond the MicroEngine to communicate with each other. Relative and absolute addressing is controlled on an instruction-by-instruction basis [9].

General-purpose registers are divided into two banks, the A bank and the B bank. This structure allows the IXP1200 to fetch two separate operands in the same clock cycle (one from SRAM and one from SDRAM). Each bank supports a port and a write port as shown in Figure 8.

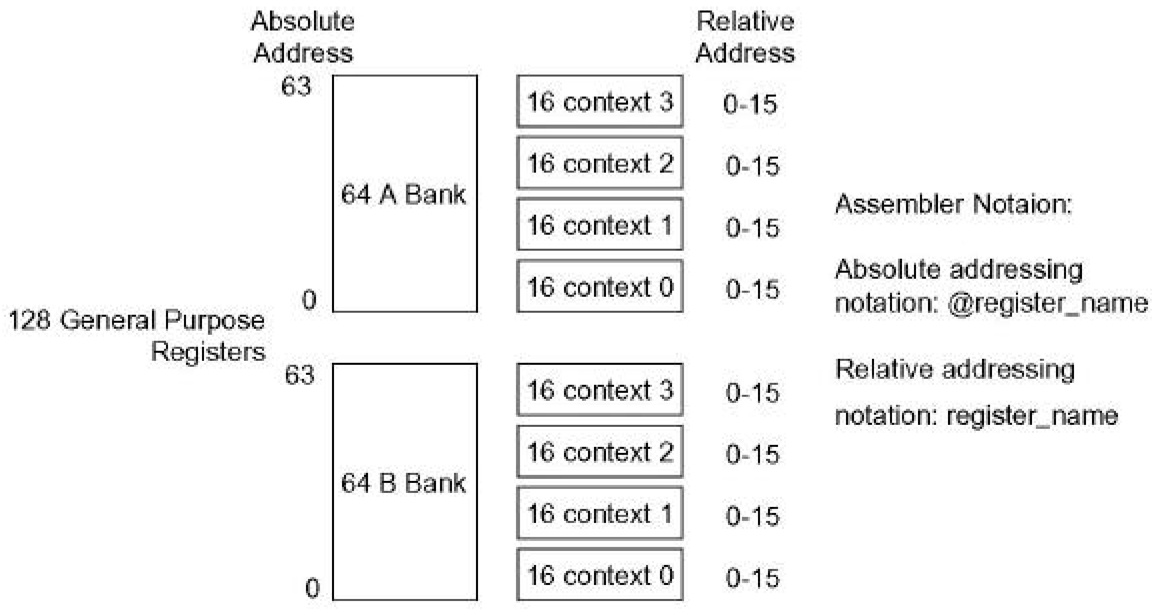

Figure 8. GPR Addressing (From Ref. 9.).

GPRs are divided into four logical register regions. Each region has 32 registers. This structure eliminates the overhead of switching among threads [9]. Absolute addressing in GPRs allows sharing registers between threads in a MicroEngine.

Each MicroEngine has 128 32-bit transfer registers (Figure 9). Transfer registers are intended for transferring data to and from memory components.

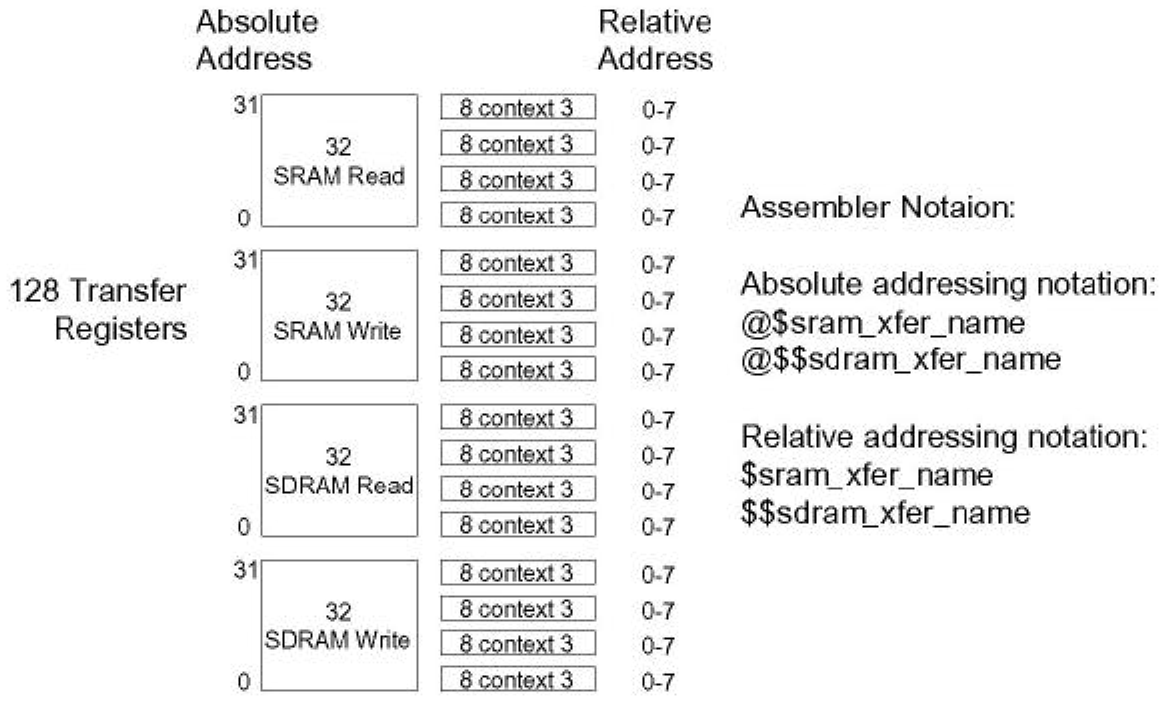

Figure 9. Transfer Register Addressing (From Ref. 9.).

As shown in Figure 9, transfer registers are divided into two-memory type blocks. Sixty-four SDRAM transfer registers and sixty-four SRAM transfer registers are connected to the SRAM/SDRAM memory busses. The two blocks are divided into 32 read and write registers blocks.

### 3. SRAM and Internal SRAM Interface Unit

The IXP1200 has 8 Mbytes of SRAM to quickly store any data needed, such as lookup tables, free buffer lists, and data buffer queue. It is important to lookup these data with low latency. For example, in a routing table, the lookup table must be done quickly. This is because all processing depends on that lookup. So, lookup tables are stored in SRAM because SRAM is faster than SDRAM. While the lookup tables are stored in SRAM, SDRAM is for a large data structure like a Routing Table.

The SRAM interface is 32 bits wide. It supports either pipelined or flow-through SRAMs. Recall that SRAM is not for bulk data, rather, it is used for fast access.

#### 4. SDRAM and Internal SDRAM Interface Unit

The IXP1200 has 256 Mbytes of SDRAM to store bulk data like routing tables. The SDRAM unit, like a SRAM unit takes Reference Commands from the MicroEngines and StrongARM and fetches data in an optimal fashion. The SDRAM has a 64-bit data bus and a 14-bit address bus.

#### 5. PCI Unit

The PCI Unit is a standard 32-bit PCI 2.1 interface. It can run at 33 MHz with the standard number of loads [9]. It also runs at 66 MHz with a point-to-point configuration [9].

The main purpose of the PCI Unit is to communicate with the host system and make the IXP1200 reachable by the user who can modify it.

## D. SUMMARY

The IXP1200 addressed current networking requirements. It succeeded in this by using parallelism, pipelining and programmability. To manage these features, the IXP1200 network processor contains six hardware multithreading RISC MicroEngines, one core StrongARM processor, memory interfaces and fast data buses. The MicroEngines work in parallel, and every one of them is a 5-stage pipeling RISC processor.

Each MicroEngine contains four hardware threads with separate program counters for each thread. Multi-threading allows each MicroEngine to increase its performance by not wasting time while waiting for memory access instructions.

Common packets are handled by MicroEngines while exception packets are handled by StrongARM, because MicroEngines perform fast data processing. Exceptions reduce the performance of the system. So, exceptions are handled by the StrongARM core processor.

Every MicroEngine and StrongARM is a programmable processor. The next chapter explains programming of the IXP1200. First, the basic programming concepts and structures are introduced. Second, advanced programming models, such as IXA API, and ACE, are explained.

THIS PAGE INTENTIONALLY LEFT BLANK

# IV. IXP1200 PROGRAMMING AND MICROACE STRUCTURE

## A. INTRODUCTION

In this chapter, the software of the IXP1200 network processor will be explained. This chapter includes two sections, IXP1200 programming and the MicroACE structure. In the first section, the IXP1200 Software Development Kit, instructions and programming structures such as assembly directives and macro will be explained. In the second section, advanced IXP1200 programming and MicroACE structure will be discussed.

Network processors are software-based processors. The functions and services depend on the program written for that processor. The software-based approach provides reusability, low-cost, rapid production, and easy maintenance. Each processor has its own assembly language programming structure, and concepts.

Network processor vendors offer a reference platform or an evaluation testbed [2]. These platforms help designers to produce, test, and evaluate their software and hardware implementations before adapting a network processor into a new network system. A reference platform includes the following five items [2]:

- Hardware testbed
- Development software
- Simulator and emulator
- Download and bootstrap software
- Reference implementations

With these tools, designers can write, test, simulate, and run the software without a current network system. Development software includes libraries and some sample codes to help the designer create his own code.

## B. IXP1200 PROGRAMMING

### 1. Software Development Kit

The Intel Corporation also provides a reference platform for its network processors [2,13]. This platform is divided into two sub-systems, a hardware testbed and a Software Development Kit (SDK) [2,13].

The hardware testbed is a PCI card that can be plugged into the PCI bus of a PC. The card has four 100-Mbps ports. After the software is written and tested with the IPX1200 SDK, it can be downloaded into the hardware testbed and run.

With an average personal computer and an IXP1200 hardware testbed, an inexpensive router can be built that forwards minimum-sized packets at a rate of 3.47 Mbps [14].

The Intel IXP1200 SDK has some software components to support IXP1200 software development. These components are shown in Table 3.

| **Software** | **Purpose** |
|---|---|
| C complier | Compile C programs for the StrongARM |
| Network Classification Language (NCL) complier | Compile NCL programs for the StrongARM |
| MicroC complier | Compile C programs for the MicroEngines |
| Assembler | Assemble programs for the MicroEngines |
| Simulator | Simulate an IXP1200 code to debug |
| Downloader | Load software into the network processor |
| Monitor | Communicate with the network processor and interact with running software |
| Bootstrap | Start the network processor running |
| Reference Code | Example programs for the IXP1200 that show how to implement basic functions |

Table 3. IXP1200 SDK Items (From Ref. 2.).

To work with this system, two operating systems are needed. Linux runs on the StrongARM, and Windows NT runs on the MicroEngines [2,16]. One does not need two separate PCs to work two operating systems. One main operating system and one emulator program that runs inside the main are sufficient to work with the SDK. For example, the main system can be Linux. For the secondary system a Windows emulator such as Wine can be used [2].

### 2. Instruction Set

Because each IXP1200 MicroEngine is a RISC processor, it has few basic opcodes; the MicroEngine instruction set has 32 basic opcodes (Table 4) [15]. All instructions are 32-bits in length.

| Instruction | Description |
|---|---|
| **Arithmetic, Rotate, And Shift Instructions** | |
| ALU | Perform an arithmetic operation |
| ALU_SHF | Perform an arithmetic operation and shift |
| DBL SHIFT | Concatenate and shift two longwords |
| **Branch and Jump Instructions** | |
| BR, BR=O, BR!=O, BR>O, BR>=O, BR<O, BR<=O, BR=count, BR!=count | Branch or branch conditional |
| BR_BSET, BR_BCLR | Branch if bit set or clear |
| BR=BYTE, BR!=BYTE | Branch if byte equal or not equal |
| BR=CTX, BR!=CTX | Branch on current context |
| BR_INP _STATE | Branch on event state |
| BR_!SIGNAL | Branch if signal deasserted |
| JUMP | Jump to label |
| RTN | Return from branch or jump |
| **Reference Instructions** | |
| CSR | CSR reference |
| FAST_WR | Write immediate data to thd_done CSRs |
| LOCAL_CSR_RD, LOCAL_CSR_WR | Read and write CSRs |
| RJIFO_RD | Read the receive FIFO |
| PCI_DMA | Issue a request on the PCI bus |
| SCRATCH | Scratch pad memory request |
| SDRAM | SDRAM reference |
| SRAM | SRAM reference |
| T FIFO WR | Write to transmit FIFO |
| **Local Register Instructions** | |
| FIND_BST, FIND_BSET_WITH_MASK | Find first 1 bit in a value |
| IMMED | Load immediate value and sign extend |
| IMMED_BO, IMMED_B1, IMMED_B2, IMMED_B3 | Load immediate byte to a field |
| IMMED_WO, IMMED_W1 | Load immediate word to a field |
| LDJIELD, LDJIELD_W_CLR | Load byte(s) into specified field(s) |
| LOAD_ADDR | Load instruction address |
| LOAD BSET RESULT1, LOAD BSET RESULT2 | Load the result of find bset |
| **Miscellaneous Instructions** | |
| CTX_ARB | Perform context swap and wake on event |
| NOP | Skip to next instruction |
| HASH1- 48, HASH2- 48, HASH3- 48 | Perform 48-bit hash function 1, 2, or 3 |
| HASH1 64, HASH2 64, HASH3 64 | Perform 64-bit hash function 1, 2, or 3 |

Table 4. MicroEngine Basic Inst. Set (From Ref. 15.).

Detailed explanations and descriptions of instructions are found in the Reference 15.

### 3. MicroEngine Assembly Syntax

The general syntax is the same for all assembly languages. That is, they have labels, operators and operands [2]. MicroEngine assembly conforms to this format.

*label: operator operand token*

The label specifies the beginning of that code piece and is used by the reference instructions to branch or jump to that piece. The operator denotes the instruction to be executed. The operand specifies the data that will be processed with that operator. The token is optional [2].

For example, for the *alu* instructions, the format is

alu[*dest_op*,*$src_1$_op*,*operation*,*$src_2$_op*].

This is an arithmetic and logic unit instruction [15]. The *dest_op* refers to the destination, which is usually a register, to store the result of this operation. The *$src_1$_op* and *$src_2$_op* specifies the operands of this alu operation. The *operation* refers the alu operation. It can be +, -, AND, OR, etc.

Registers are important for RISC processors because every instruction refers to registers to process an operation. Referring and naming registers is a problem, especially if there are many registers. The MicroEngine assembler allows the programmer to name registers manually or to leave this task to the assembler [15]. There are two kinds of register assignments [15]. Table 5 shows the assembly directives used for manual assignment.

| **Directive** | **Register Type** |
|---|---|
| .areg | GPRS A-Bank |
| .breg | GPRS B-Bank |
| .$reg | SRAM Transfer Register |
| .$$reg | SDRAM Transfer Register |

Table 5. Manual Register Assignment Directives (After Ref. 15.).

There are two basic types of addressing modes. These are context relative and absolute modes. Chapter 3 describes the hardware difference between them. Table 6 shows the different naming syntax of the addressing modes.

| Register Type | Context Relative Addressing Syntax | Absolute Addressing Syntax |
|---|---|---|
| GPRS | reg_name | @reg_name |
| SRAM transfer | $reg_name | @$reg_name |
| SDRAM transfer | $$reg_name | @$$reg_name |

Table 6. Register Addressing (After Refs. 2,15.).

The IXP1200 assembler responds to the assembler's directives. These directives make the programmer's job easier. The assembler understands each directive and replaces each with an actual machine code or links the code with the other programs. These directives include an assembler loop, assembler macro, conditional assembly, error reporting, structured assembly and subroutine directives.

The IXP1200 assembler also allows macros. Macros help the programmer reuse the software pieces. The IXP1200 hardware testbed and SDK are supplied with macros. For example, SDK has layer-3 forwarder software, macros and subroutines. Macros can be defined and invoked at any point in a program [15]. Macro directives are used to define and invoke the macros. Macros already written as a separate file must be included by using a file inclusion assembly directive into the program.

Figure 10 shows a macro used in layer-3 forwarder software. Macros are used to make programming easy. For example, it is hard to remember the exact alu instructions for addition, but they can be written as shown in Figure 10. Instead of "alu_op[out_dst, in_src_a, +, in_src_b]", it can be used in a program as an instruction like add(out_dst, in_src_a, in_src_b).

```
// add
//              description: 32 bit add in_src_a + in_src_b
//              outputs:
//                      out_dst         GPR
//              inputs:
//                      in_src_a        register or constant
//                      in_src_b        register or constant
//              size: 1-5 instructions
//              example: add(output, 0x1234, 0x12345678);
//
#macro add(out_dst, in_src_a, in_src_b)
        alu_op[out_dst, in_src_a, +, in_src_b]
#endm
```

Figure 10. IXP1200 Macro Example (From Ref. 18.).

## 4. Simple Packet Data Flow in IXP1200

This packet data flow [9] is given in the introduction to the advanced IXP1200 programming. Figure 11 shows the simple packet flow hardware diagram, and Table 7 explains each step that occurs in the diagram.

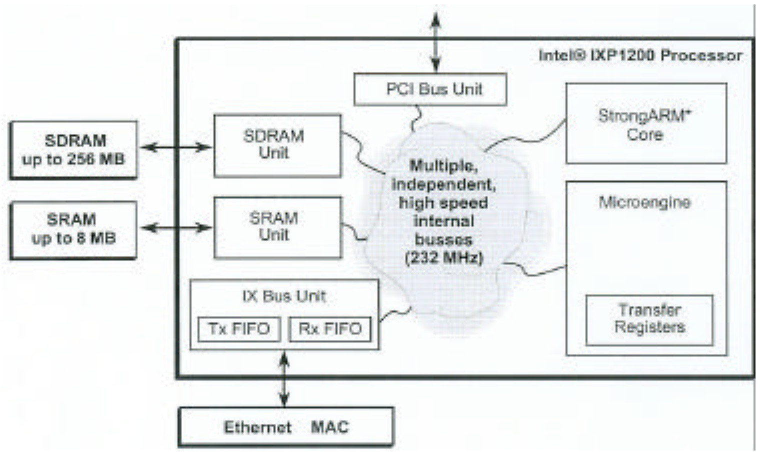

Figure 11. Simple Data Flow Hardware (From Ref. 9.).

| Step: | Packet Data Flow Steps: |
|---|---|
| 1. | The data is received by the Ethernet Media Access Control (MAC) unit from the incoming Ethernet port. |
| 2. | A MicroEngine executes an IX bus unit reference command to get data from MAC to store it in the receive FIFO queue. The IX bus unit does transfer, independent of the MicroEngine. |
| 3. | The IX bus unit signals a MicroEngine thread that the data has been transferred. |
| 4. | The MicroEngine instructs the SDRAM unit to transfer the data from the receive FIFO to SDRAM. The SDRAM unit, independent of the MicroEngine, does the transfer. |
| 5. | The MicroEngine transfers the first few bytes (header) into its transfer registers. |
| 6. | The MicroEngine Thread processes the header and determines what to do about that packet. It can make use of tables stored in SRAM to do lookups. |
| 7. | If necessary, it modifies and adds bytes to the packet header and writes the new header to SDRAM. |
| 8. | The MicroEngine instructs the SDRAM unit to write the packet data out to the transmit FIFO of IX bus unit. When the transfer is complete, the SDRAM unit notifies the MicroEngine. |
| 9. | The MicroEngine instructs the IX bus unit to transfer ready data to the appropriate MAC. |

Table 7. Data Flow Steps (After Ref. 9.).

## C. ACE PROGRAMMING MODEL

### 1. IXA Application Programming Interface

Network applications for the Intel Exchange Programming Interface can be designed by using the IXA application programming interface (API). The IXA API consists of basic modules for the programmer to create the application. The IXA software development kit provides tools and libraries that allow the designer to quickly develop IXA applications [16].

Since the goal is to design a network system, the main component of the application is to handle the data packets. The application processes the packet, as shown in Figure 12, and outputs the packet or an error message. The specific action depends on the goal of the system. The system can be a simple router, firewall, network address translator (NAT) or intrusion detection system.

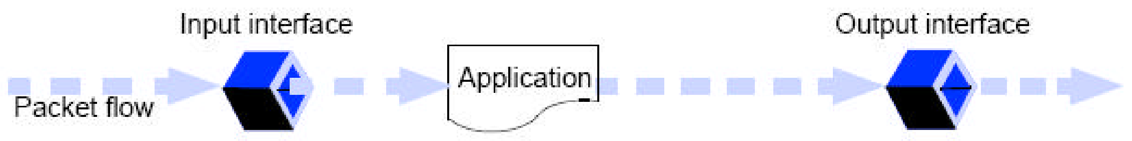

Figure 12. Packet Flow (From Ref. 16.).

Applications manipulate packets, independent of the protocol layer [16]. The application can be divided into many jobs. We view each job as a single process. Each process in turn can be divided into simple task modules. This division is derived from the concept of object-oriented programming. Each simple task can be handled by a single module [16]. MicroEngines or threads in a MicroEngine can be assigned one or more modules to perform. For example, a single module can receive packets from the IX bus and put them into memory.

### 2. Active Computing Element

The active computing element (ACE) encapsulates and modularizes the unique tasks involved in packet processing [16]. ACEs are fundamental software blocks used to construct packet-processing systems [2].

Each ACE usually performs one type of packet handling or data control task [16]. Tasks can be distributed among ACEs.

There are three types of ACEs [16]: user ACEs, library ACEs and system ACEs. User ACEs are those the user developed. Library ACEs are supplied by Intel for common application functionality. System ACEs implement hardware network interfaces or protocol stacks.

An ACE can take advantage of the speed of a processor [16]. Such an ACE is called an accelerated ACE [16,17]. An ACE that does not use a hardware acceleration component is called a conventional ACE [16,17].

### 3. MicroACE

The microACE programming model provides a framework for designing a packet processing application that makes use of the IXP1200's MicroEngines and StrongARM processor [2,16].

A microACE has two main components that run on different processors [2,16]:

- Core Component: This is a conventional ACE written in a C/C++ network classification language. The Core component runs in the StrongARM, the core processor of the IXP1200.
- Microblock: This is the acceleration component of the microACE. It is hardware specific and runs on MicroEngines. Microblocks perform fast-path processing. They can exchange and communicate with the core component to handle exception packets.

Figure 13 shows the pseudo-code of a microACE application. This microACE application is responsible for [17]:

- Initializing the resource manager.
- Setting up the port configuration.
- Loading the microcode image onto each MicroEngine according to the MicroEngine configuration.
- Launching microACEs and any other conventional StrongARM ACEs.
- Configuring the microACE's and the conventional StrongARM ACE's using the IDL interface defined for these ACEs.
- Binding the microACEs with each other and with other conventional StrongARM ACE's.
- Enabling the MicroEngine.

```
int main(int argc, char**argv)
{
        // Initialize the resource manager
        RmInit()

        // Set the port configuration
        RmSetPortConfiguration()

        // Set the microcode image onto each microengine
        for (i = 0; i < MAX_NUM_UENG; i++)
                RmUengSetUcode()

        // Launch microACES. This call is synchronous. So by the time it returns, //the ix_init() of the microACE has
        been called. The application assumes //that all imported variables were patched in during the ix_init()
        for (i = 0; i < numberOfMicroaces; i++)
                RmCreateMicroAce()

        // Launch conventional ACES
        for (i = 0; i < numberOfRegularAces; i++)
                ix_res_create_ace();

        // Configure the microACEs and aces with IDL
        ConfigureACEs()

        // bind the microACE's together with static targets
        for (i = 0; i < numberOfStaticBinds; i++)
                RmBindMicroAce()

        // bind microACEs and other ACEs with regular targets
        for (i = 0; i < numberOfRegularBinds; i++)
                ix_res_bind()

        // Load the microcode
        RmUengLoad()

        // Enable the microengines
        RmUengEnable()

        // Loop for ever
        while (1);
}
```

Figure 13. MicroACE App. Using Pseudo-code (From Ref. 17.).

### 4. An Example of MicroACE Processing

Figure 14 shows a system for doing IP (layer- 3) forwarding. There are microblocks that run in MicroEngines and core components that run in StrongARM core processor.

The reason for dividing MicroACE into two components is to maintain a high packet throughput [2]. The MicroEngines have a fast data path and process the typical packets. When a MicroEngine encounters a packet that is not expected (Figure 14), the microblock does not try to handle the packet. Instead it passes it to the core component to process [2].

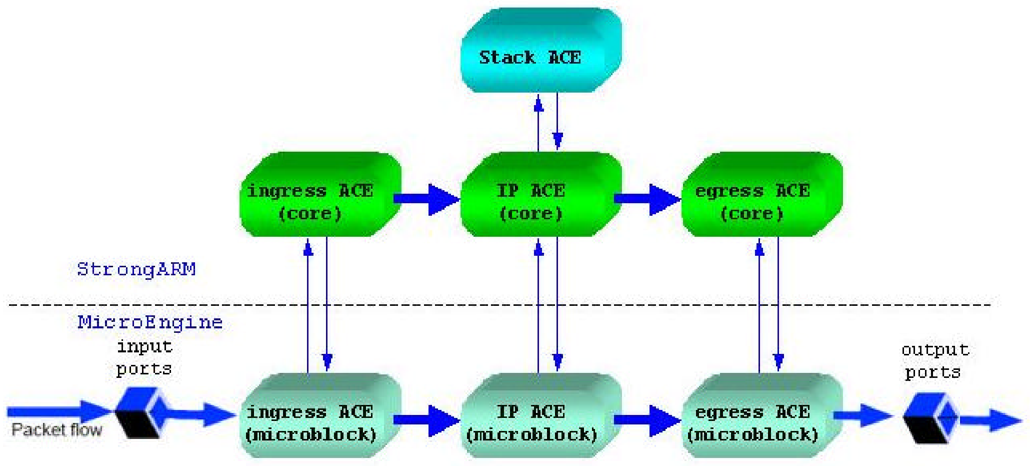

Figure 14. IP Forwarding MicroACE (After Refs. 2,16.).

The Ingress ACE microblock takes packets from input ports, and examines the packet header. If it encounters an unexpected condition or an error, it passes it to the core ingress ACE to handle the situation. This scenario repeats at every step until the packet exits the loop.

### 5. The Dispatch Loop

There must be a system to control packet flow among microblocks (every packet does not follow the same path [2]). When an unexpected condition occurs, the packet can follow a path different from the usual one.

To control packet flow between microblocks, a dispatch loop is used. The dispatch loop is a small program code segment that contains an infinite loop (Figure 15). Each hardware thread executes the dispatch loop [2].

```
// Example Dispatch Loop Algorithm
Allocate global registers;
Initialize dispatch loop;
Initialize Ethernet devices;
Initialize ingress microblock;
Initialize IP microblock;
while (1) {
        Get next packet from input device(s);
        Invoke ingress microblock;
        if (return code == 0) {
                Drop the packet;
        } else if (return code == 1 ) {
                Send packet to ingress core component;
        } else { // IP packet
                Invoke IP microblock;
                if (return code == 0) {
                        Drop packet;
                } else if ( return code == 1 ) {
                        Send packet to IP core component;
                } else {
                        Send packet to egress microblock;
                }
        }
}
```

Figure 15. Dispatch Loop (From Ref. 2,17.).

The first loop invokes the microblocks. When a microblock finishes its job, it returns a return code. The return code tells the dispatch loop what to do next with that packet. For example, after it invokes the ingress microblock, the ingress microblock chooses a return code and gives control to the dispatch loop. If the return code is “0”, the dispatch loop discards the packet. If it is “1”, it passes the packet to the core component. If the return code is anything else, the dispatch loop continues with the next microblock to process.

## D. SUMMARY

Intel provides a software development kit with its network processor system test-bed and platforms. This includes compilers and debuggers for various languages used to program the processor.

The assembly language for MicroEngines consists of 32 32-bit instructions. There are two kind of addressing modes. The context relative mode allows each MicroEngine thread to use its own register block, and the absolute mode allows threads to communicate with each other by sharing registers.

Using macros makes the code easier to write and understand. There are also assembler directives for macro usage.

Intel's SDK provides a designer application programming interface. This API contains library codes to allow software reusability. Intel IXA API defines a software construct called "active computing element" (ACE) to encapsulate and modularize the unique tasks involved in packet processing [16]. ACEs are program blocks used by MicroEngines and StrongARM core. ACEs provide modularity to divide tasks into modules. Each module performs a job and communicates with each other.

ACEs for MicroEngines and StrongARM are called MicroACE. MicroACE has two main components: Microblocks for MicroEngines and core components for StrongARM.

SDK provides several sample microACEs for the designers. Layer -3 forwarder microACE is one of them. It is examined and used to produce a multicasting forwarder described  in the next chapter.

# V. A MULTICASTING FORWARDER DESIGN USING MICROACE

## A. INTRODUCTION

Unicast forwarding involves one sender and one receiver. The packet is only delivered to the intended destination. But, a number of network applications are required for delivering information to more than one receiver. With unicast, if the sender wants to send a packet to more than one receiver, the sender must send the same packet many times, one to each receiver. For example, if there are five receiver hosts, the sender must send it five times. To deliver information with more efficient bandwidth usage to a group of receivers, multicast services have been proposed. A multicasting system includes one sender and more than one receiver. The sender only sends one packet, and the system duplicates it and sends it to the receivers in a multicasting group. The multicast conserves bandwidth because the sender does not have to send the same packet more than once.

Multicast is used in group communication applications where more than one receiver is involved. For example, to transmit audio, video, and information of a live lecture to a group of receivers, multicasting transmission is preferred over unicast.

In this chapter, the design of a multicasting forwarder using the IXP1200 network processor with microACE is proposed.

## B. MULTICASTING FROM UNICAST FORWARDING

Routers are the main components in networks. Routers support the packet switching architecture, whereas packets are forwarded based on the IP destination address in the IP header.

In unicast forwarding, each packet is routed according to the routing algorithm running on the routers. The routing tables are defined by an algorithm that specifies the next hop for a packet to follow.

Several IP multicasting protocols have been proposed and standardized [DVMRP-RFC1075, PIM-RFC2362]. Class D IP addresses are used to identify a group of receivers. The sender sends the packet with a class D IP address as the destination address and the routers send the packet to the receivers that are registered in that class D address group. Class D addresses range from 224.0.0.0 to 239.255.255.255.

Conventional IP multicasting is a useful service abstraction for many applications, but, for various reasons, its wide scale deployment has been slow [19]. Problems with IP multicasting include the complexity and variety of routing protocols used and application-to-abstraction mismatches [19].

The IXP1200 is a fully programmable network processor that can implement IP multicasting. In addition, its programmability makes the IXP1200 a good platform to implement other multicast designs that avoid problems of IP multicasting. In this thesis, we investigated the implementation of a multicasting forwarder that uses unicast IP forwarding to send packets to the destinations. The design of such a multicasting forwarder using the layer-3 forwarding microACE on the IXP1200 network processor is examined.

In the unicast-based multicast design [19], routers are installed with a small piece of code that duplicates packets. The code is enabled when a receiver wants to join a multicast group. In the example shown in Figure 16, the edge multicast router to which the four hosts are connected maintains an additional code to implement the unicast-based multicast.

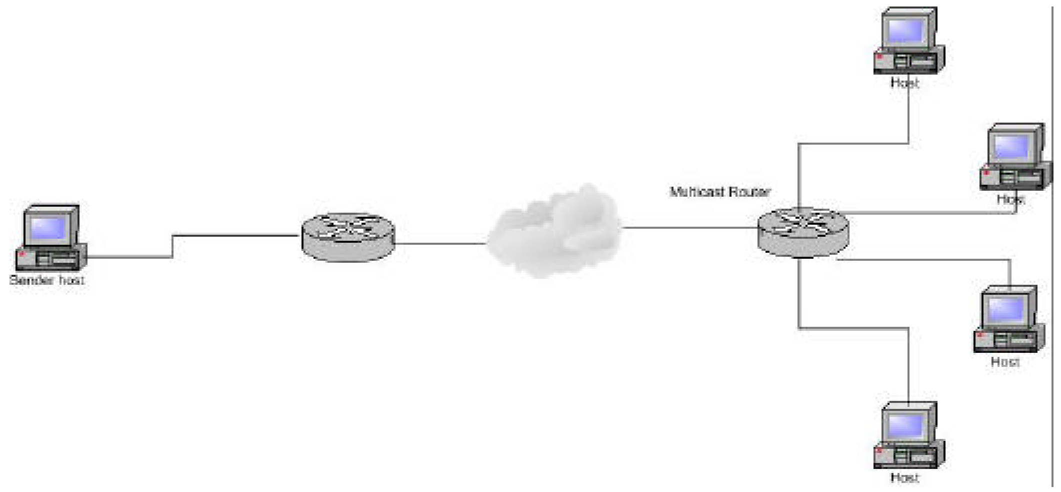

Figure 16. Basic Multicasting System.

Every multicast group has a group identifier to specify the group. When the sender sends the packets for multicasting, it puts the multicast group identifier into the option field of the IP header. Group management is done by the hosts by communicating with the router. Routers maintain a list of unicast addresses that correspond to a multicast group.

This approach eliminates the multicast routing algorithms for the class-D addresses. The unicast-based approach is flexible because maintenance of the multicast tree is left to the end hosts. This can achieve a balance of flexibility and efficiency.

A layer-3 forwarder in the IXP1200 works as follows. When it gets a packet, it lookups the IP destination address in the header. Based on the destination address in the routing table, the layer -3 forwarder determines the interface to which that packet should be sent. According to the routing table, the router forwards the packet to the appropriate interface.

When the router first gets the packet, it examines the packet's header to learn whether the packet is intended for multicasting. A multicast packet is a unicast packet carrying a multicasting identifier. If it is a usual unicast packet, the router does its usual forwarding process. If the packet is a multicast packet, according to the number of IP addresses in the multicast group table, it duplicates the packet and changes the destination address of each duplicated packet based on the addresses in the group table.

The next section describes in detail the design of multicasting forwarder with MicroACE on the IXP1200.

## C. MULTICASTING FORWARDER MICROACE

Every multicasting group must have a group identifier. In the unicast-based design, the destination addresses of multicast packets are still unicast IP addresses. Therefore, something is needed to specify them as multicasting packets. The IP header has an option portion to specify application specific information. We proposed to write the multicast group identifier into the options field of the IP header. The sender application program must then know where to write the multicast identifier to the IP header. Every group identifier corresponds to one or more IP addresses and must be long enough to prevent a collision. The derivation of a multicast identifier is beyond the scope of this thesis. In an IXP1200-based router, the IP addresses of a multicasting group can be stored as a table in SRAM along with the group identifier. An example is shown in Table 8.

| Identifier | IP Addresses |
|---|---|
| 1 | 131.120.41.56<br>131.120.34.32<br>131.120.42.67 |
| 5 | 131.120.41.45<br>131.120.45.43 |

Table 8. Multicast Identifier Examples.

The SRAM in IXP1200 is smaller in size but has faster access time compared to SDRAM. So, we believe it is appropriate to store a reasonably small-multicast table in SRAM.

To implement the multicast forwarder on the IXP1200, we want to use as much of the existing software design as possible. The IXP1200 SDK has a unicast forwarder microACE. In the unicast forwarder, there are three main microblocks. The ingress ACE microblock takes the packets from input port, examines them, and sends them to the IP ACE microblock. The IP ACE microblock does the IP forwarding processing. The egress is responsible for delivering the packets to the appropriate ports. Our design is to make only a modification in the ingress and egress microblocks of the unicast forwarder and leave the IP forwarder microACE unchanged.

To explain our design, as shown in Figure 17, the process algorithm (Figure 18) is described below.

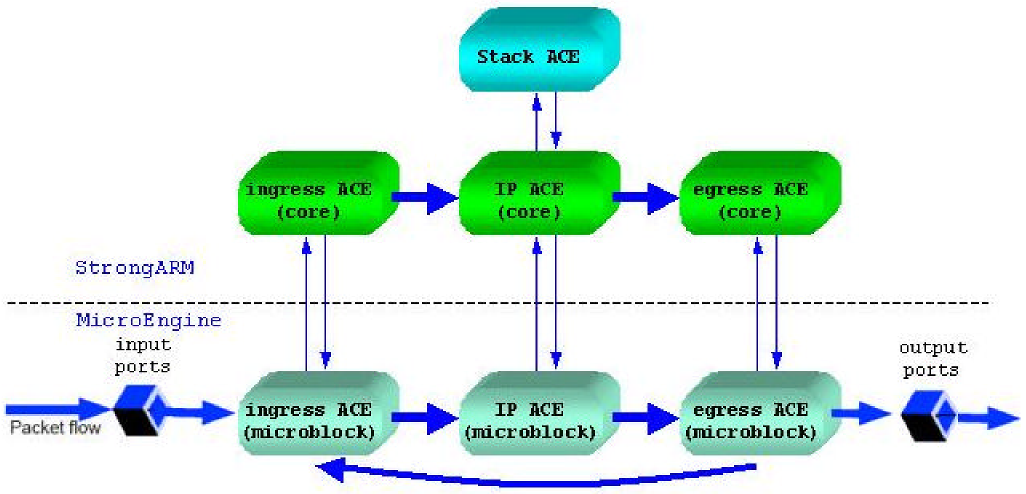

Figure 17. Multicasting MicroACE (After Ref.2,16.).

The ingress ACE takes the packets from the receiver FIFO buffer. It examines them to determine which is unicast and which is multicast. If an IP header field has an option field with multicasting identifier, it checks whether it is a multicasting packet. That identifier specifies the multicasting group table to lookup the IP addresses.

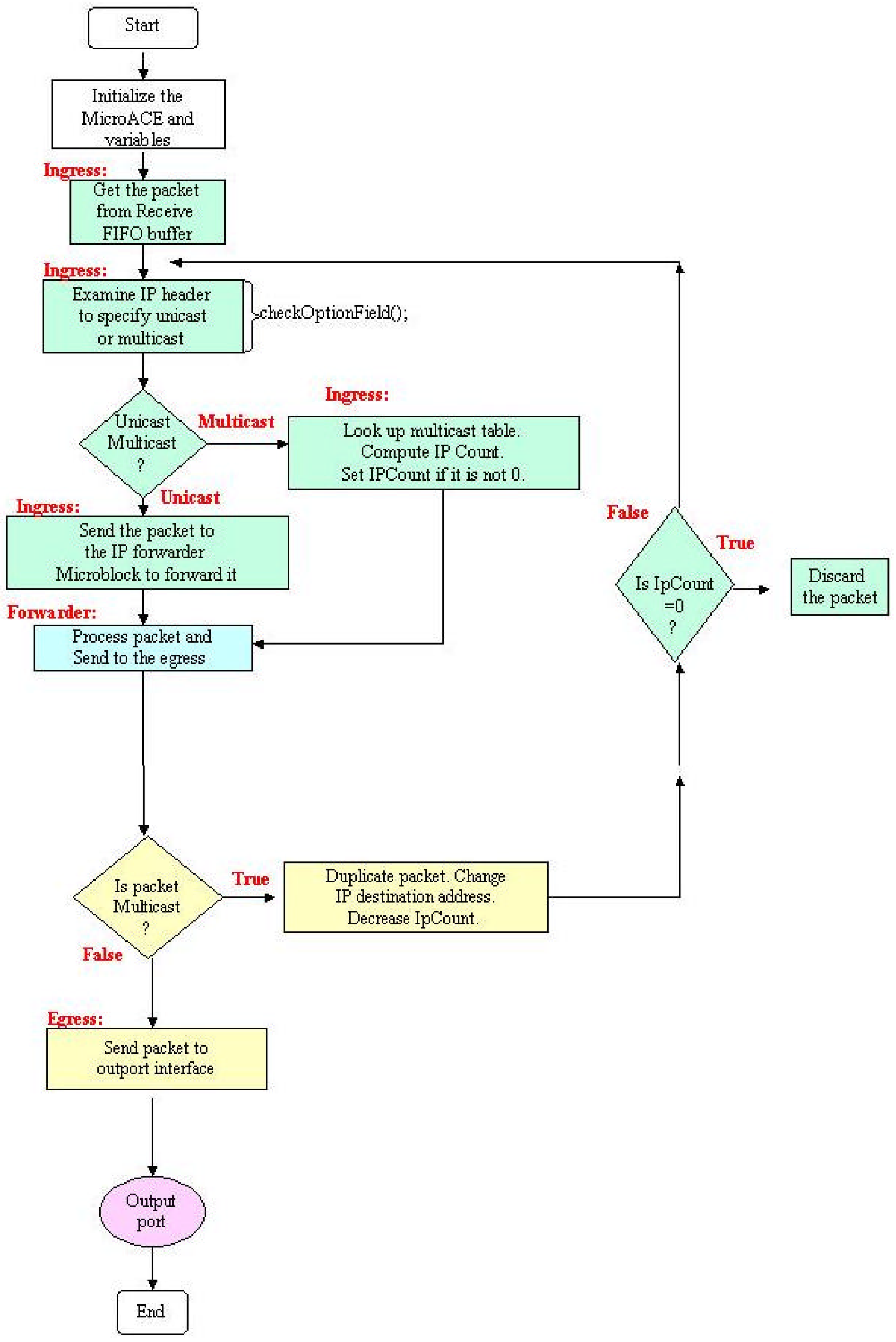
Start
Initialize the MicroACE and variables
Ingress:
Get the packet from Receive FIFO buffer
Ingress:
Examine IP header to specify unicast or multicast
checkOptionField();
Unicast Multicast ?
Multicast
Ingress:
Look up multicast table. Compute IP Count. Set IPCount if it is not 0.
Unicast
Ingress:
Send the packet to the IP forwarder Microblock to forward it
Forwarder:
Process packet and Send to the egress
False
Is IpCount =0 ?
True
Discard the packet
Is packet Multicast ?
True
Duplicate packet. Change IP destination address. Decrease IpCount.
False
Egress:
Send packet to outport interface
Output port
End

Figure 18. Multicasting MicroACE Algorithm.

If a packet is a unicast packet, ingress simply sends it to the next target IP forwarder microblock. If it is a multicasting packet, ingress lookups the table to determine how many destinations IP addresses there are. It sets the IpCount variable to the number of destination addresses for that multicast packet. The IpCount variable is used to count the packet duplication times. Since every multicast packet must have an IpCount variable, it must be stored with the multicast packet to specify the packet needs to be duplicated. Packets are duplicated and sent according to the IpCount variable.

The ingress ACE sends the multicast packet to the IP forwarder microblock. Then, the egress ACE sends each packet to the appropriate output port. If it is a unicast packet, it sends it to the output port. If it is a multicasting packet, it takes the destination IP address from the table, changes it, and decreases the IpCount by one. Egress duplicates the multicasting packet untill the IpCount is zero. It sends one of the duplicated packets to egress to forward it output port, and it sends the other to the ingress again to duplicate that packet for all receivers. This process goes on until the IpCount variable goes to the zero.

For each multicast packets, an IpCount variable is created with the multicast group identifier

```
setIpCount(groupID);
```

With this method call, IpCount is created with the group identifier. This variable is controlled by the ingress and egress microblocks. Figure 19 shows the initialization program block.

```
ix_error ix_init(int argc,char** argv,ix ace **acepp)
{
        // Allocate an ix_ace structure
        *acepp = AllocateAceStructure()
        // Get the name of the ACE
        name = argv [1]
        // call the standard ASL ix_ace_init() to set up //the ix ace structure
        ix_ace_init(*acepp, name)
        //Initialize the resource manager
        RmInit ()
        // Allocate SRAM for control block
        controlBlock = RmMalloc()
        // Allocate SRAM/SDRAM/SCRATCH for other data //structures
        otherMemory = RmMalloc()
        //Set up these data structures
        SetupDataStructures()
        // Register an exception handler with the Re //source Manager and get a unique tag for the
        mi //croblock. This tag can be used to send pack//ets to the microblock
        RmRegister ()
        //Get the microengine mask for this ACE
        meMask = atoi(argv[2])
        // For each microengine in the mask patch vari//ables for this ACE
        for (i = 0;i < numberOfImportedVariables; i++)
                RmUengpatchSymbol()
        // Do other ACE specific stuff
        //like creating targets
        return 0;
}
```

Figure 19. Initialization Routine.

While in these processes, if any unexpected condition occurs, the microblocks send the packet to the core component running in the StrongARM to solve the problem. Processing exceptions in the core component is more time consuming.

## D. SUMMARY

In this chapter, unicast-based multicasting design is discussed. Multicasting is used to send information to more than one receiver hosts. Teleconferencing and lecturing are using multicasting.

Conventional IP multicasting uses extra protocols and multicasting forwarding algorithms. Our design is using unicast IP addresses to send multicast packets. The edge multicasting router takes every packet and examines them to determine whether they are unicast-based multicasting packets or unicast packets.

Every unicast-based multicast packet contains a multicast group identifier to specify the receiver hosts.

Our design is effective and reliable, because there is no extra protocol and routing algorithms. The receiver hosts handle the group creation and modifications.

THIS PAGE INTENTIONALLY LEFT BLANK

# VI. CONCLUSIONS AND RECOMMENDATIONS

Our design takes advantage of the programmability of IXP1200. Using the proposed design, an IXP1200 router can still run unicast forwarding and IP multicasting and unicast-based multicasting. The new functions can be added to the system by just reprogramming certain components.

The ACE programming module design allows the components to be reused, modified, or added. It is much easier than re-implementing the entire system. We used a unicast forwarder to design a unicast-based multicast forwarder. However, the IXP1200 is a very complex system.

Our unicast-based multicasting forwarder design is effective and reliable. This is done by hosts directly connected to multicasting edge router. Hosts are responsible to configure multicasting groups. This eliminates extra protocol overhead for multicasting.

This thesis is the first thesis at the Naval Postgraduate School for the implementation of the multicasting forwarder with the Intel IXP1200 network processor.

Therefore, for further research, I would recommend that students consider the following. My research began with an in-depth study of the processor. With few sources or documents to follow, I found that designing a system required an enormous amount of time. I would advise future researchers to focus on learning the programming rather than conducting an extensive survey of the hardware architecture. The basic concepts can be learned while using the IXP1200 for programming.

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF REFERENCES

[1] W. Slater, “Internet History and Growth”, *Internet Society*, Chicago Chapter of the Internet Society, September 2002 http://www.isoc.org/internet/history/, last accessed 2/22/2004

[2] D. E. Comer, *Network Systems Design Using Network Processors*, Pearson Prentice Hall, Upper Saddle River, New Jersey, 2004

[3] S. Hari, “Next Generation Network Processors”, *Network Magazine India*, http://www.networkmagazineindia.com/200107/process1.htm, last accessed 2/20/2004

[4] Agere Systems Corp., “The Challenge For Next Generation Network Processors”, April 2001,http://www.agere.com/ enterprise_metro_access/ network_processors.html, last accessed 2/19/2004

[5] Agere Systems Corp., “Building Next Generation Network Processors”, April 2001, http://www.agere.com/ enterprise_metro_access/ network_processors.html, last accessed 2/19/2004

[6] S. Lawson, “Analysis: Network Processors Enter New Generation”, *Info World*, San Francisco, June 19, 2002, http://archive.infoworld.com/articles/hn/xml/02/06/19/020619hnnetworkchip.xml, last accessed 2/19/2004

[7] E. Krapf, “A New Generation Of Network Processors Appears”, *Business Communications Review*, pp. 10, June 1999

[8] V. P. Heuring and H. F. Jordan, *Computer Systems Design and Architecture*, Prentice Hall, Upper Saddle River, New Jersey, 1997

[9] INTEL Corporation, *Intel IXP1200 Network Processor Family Hardware Reference Manual*, 2001

[10] INTEL Corporation, *Intel Internet Exchange Architecture*, 2001 http://www.intel.com/design/ network/ixa.htm, last accessed 01/15/2004

[11] B. Carlson, *Intel Internet Exchange Architecture and Applications*, Intel Press, USA, 2003

[12] Intel Corporation, *IXP1200 Network Processor Datasheet*, 2001

[13] Intel Corporation, *Intel SDK for the IXP1200*, 2001

[14] T. Spalink, S. Karlin, L. Peterson, Y. Gottlieb, *Building a Robust Software-Based Router Using Network Processors*, Department of Computer Science, Princeton University

[15] Intel Corporation, *Intel IXP1200 Network Processor Family Microcode Programmer's Refrence Manual*, December 2001

[16] Intel Corporation, *Intel IXA SDK ACE Programming Framework*, December 2001

[17] Intel Corporation, *MicroAce Design Document*, 2001

[18] Intel Corporation, *L3 Forwarder MicroACE Design Document,* 2001

[19] S. Wen, J. Grifioen and K.L. Calvert, "Building Multicast Services From Unicast Forwarding and Ephemeral State", *Computer Networks: the International Journal of Computer and Telecommunications Networking*, Vol. 38, Issue 3, pp. 327-345 February 2002

## INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Su Wen, Code CS
   Naval Postgraduate School
   Monterey, California

4. Professor Jon Butler, Code EC/BU
   Naval Postgraduate School
   Monterey, California

5. Deniz Kuvvetleri Komutanligi
   Personel Baskanligi, 06410 Bakanliklar
   Ankara, TURKEY

6. Deniz Harp Okulu Komutanligi
   Kutuphanesi, 81704 Tuzla
   Istanbul, TURKEY

7. Arastirma Merkezi Komutanligi
   Teknik Gelistirme Grup Baskanligi, 81504 Pendik
   Istanbul, TURKEY

8. Timur Diler
   Muhittin Mah. Celal Bayar Bul. Ece Sitesi A-Blok D= 27
   59860, Corlu, Tekirdag, TURKEY