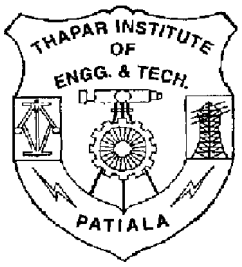
CAPTURING, CONTROLLING AND ANALYZING NETWORK TRAFFIC USING NETWORK PROCESSOR

A Thesis

Submitted in partial fulfillment of the
requirement for the award of degree of

Master of Engineering
In
Software Engineering



Under the Supervision of
Maninder Singh
Assistant Professor
Computer Science and Engineering Department
Batch 2003-2005

Submitted By
Gowni Srinivas
(8033106)

Computer Science & Engineering Department
Thapar Institute Of Engineering & Technology
(Deemed University), Patiala-147004 (India).
May 2005

ABSTRACT

CAPTURING, CONTROLLING AND ANALYZING NETWORK TRAFFIC USING NETWORK PROCESSOR

The fundamental method used for network performance monitoring or for network security implementations is network traffic analysis. Network traffic analysis includes capturing of data from network and inspecting of data at each layer. The exponential growth of Internet traffic, network bandwidth and Internet based applications rise problems of performance, flexibility in network traffic analysis. Flexibility achieved through the programmable devices like General purpose processors (GPP's) and performance can achieve through hardwired solutions like Field Programmable Gate Arrays (FPGA's). With GPP's we can't achieve wire speed performance, with FPGA technology is we can't achieve flexibility.

These situations have given rise to a new breed of programmable microprocessors called network processors. Network processors are new generation technology, which is designed to address the performance and flexibility problems. Network processors, analyzes Lower level layers by hardware and higher level layer by software with parallel and pipelined architecture. With multiple microengines and with parallel and pipelining programming architecture network processors makes network processing at wire speed.

DECLARATION

I hereby certify that the work which is being presented in the thesis entitled, ***"Capturing, Controlling and Analyzing Network traffic using Network Processor"*** *in* partial fulfillment of the requirements for the award degree of Master of Engineering in Software Engineering at Computer Science and Engineering Department of Thapar Institute of Engineering and Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of Maninder Singh.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other University.

Gowni Srinivas.

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Mr. Maninder Singh
Assistant Professor
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology
PATIALA- 147004

Countersigned by

Mr. R.S Salaria
Head
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology
PATIALA- 147004

Dr. D. S. Bawa
Dean Of Academic Affairs
Thapar Institute of Engineering and Technology
PATIALA- 147004

ACKNOWLEDGEMENT

I wish to express my deep gratitude to Mr. Maninder Singh, Assistant Professor, Computer Science and Engineering Department for providing his uncanny guidance and support throughout the Thesis work.

I am also thankful to Mr. R.S.Salaria, Head, Computer Science and Engineering Department and Mr. Rajesh Bhatia, P.G Coordinator, for their excellent guidance and encouragement right from the beginning of this course

I would also like to thank all the staff members and my Co-students who were always there at the need of the hour and provided with all the help and facilities, which I required for the completion of the Thesis.

I wish to express my indebtedness to my parents who have been a constant source of love and encouragement.

Finally I would like to thank God for not letting me down at the time of crisis and showing me the silver lining in the dark clouds.

Gowni Srinivas.
(8033106).

LIST OF FIGURES

LIST OF TABLES

Number *Page*

TABLE OF CONTENTS

ORGANISATION OF THESIS

The First chapter briefly introduces network traffic analysis and provides an insight into the background of network processing functionality. The Second chapter of this thesis is devoted to brief history of network traffic analyzers and problem specification of the thesis and includes no of design alternative such as software solutions using General purpose processors and hardware solutions ASIC's, FPGA's for the problem and the final flexible approach is Network Processors . The Third chapter provides a brief overview of network processors and detailed description of hardware and software architecture of the IXP1200 network processor. The Fourth chapter concerns the design and implementation with source code files of the application with results. Finally, concluding thesis in Fifth chapter with future scope and directions. .

CHAPTER 1 INTRODUCTION

1.1 INTRODUCTION

1.1.1 NETWORK TRAFFIC ANALYSIS

"Network traffic analysis is the process of *capturing, collecting* network traffic and *inspecting (Analyzing*) it closely to determine what is happening on the network, and *controlling* the traffic for security purposes"

1.1.2 REASONS TO INSPECT AND ANALYZE TRAFFIC

- Identify network or communication Issues
- Monitor network performance
- Verify network security
- Track communication transactions
- Log network traffic
- Discover source of unwanted traffic
- Discover compromised workstations

1.1.3 FUNCTIONALITY OF NETWORK ANALYSIS

1.1.3.1 DATA CONTROL

The amount of information that flows on a network is generally quite high with packets corresponding to different protocols and even a simple analysis of this data takes time. This time can be reduced considerably by allowing the user to specify some rules capturing packets selectively .The users should specify rules that capture all packets with given IP address. This would reduce the amount of space required for saving all packets and also required lesser time for analysis. Packet filters provide this facility of specifying such

simple rules. These rules compromise of values corresponding to various fields present the protocol header of a packet. If the protocol headers of a packet contain these values then it is saved else the packet filters drop it

Key to Data control is access control device. There exist different automated tools to implement Data control functionality firewalls such as Checkpoint Firewall, IP Filters.

1.1.3.2 DATA CAPTURE AND DATA COLLECTION

Next function in network traffic analysis is to capture of packets from the network and store its contents. At the lowest level, it requires that the network interface be able to read all packets, irrespective of the destination of the packet this can ensure by configuring interface. This activity of monitoring packets of the network is known as packet sniffing. Packet sniffers are simple monitoring tools that can only dump the network traffic on the storage media. Data can be stored in a buffer until it is full or in a rotation method such as “round robin” where the newest data replaces the oldest data. Buffers can be disk-based or memory-based. Automated tools exist for Data Capture Functionality such as TCP Dump, Ethereal. There exist different layers of information. If one depends on single layer information it may fail into identify all security problems. Data collection takes place of different layers.

1.1.3.3 DATA ANALYSIS

- Ø Checking packet headers to make sure that they are from valid sources.
- Ø Encrypting/decrypting and authenticating VPN packets.
- Ø Assembling packets into content and searching for attacks and banned material.
- Ø Counting packets and measuring flows (traffic shaping)
    - Characterizing all traffic into flows
    - Measuring inter-packet arrival time for all packets
    - Performing network accounting

At different layers the following information of packets is inspected and analyzed Total number of Ethernet frames arrived and processed at Ethernet layer at IP layer based on IP Frame format the Total number of IP packets and based on TCP frame format Total number of TCP packets and total no of UDP packets and finally deep packet inspection includes the Analysis of Application layer protocols in this layer the Classification of packets is according to the data they carry up through network. Like Any IP datagram to xx.x.x.x, TCP control (SYN/FIN/RST) to/from port 21 TCP control (SYN/FIN/RST) to/from port 22, TCP control (SYN/FIN/RST) to/from port 25, TCP control (SYN/FIN/RST) to/from port 80, TCP bulk data, RIP requests ,UDP to port 139, UDP, ICMP , IP with options, ARP packets.

1.2 BACKGROUND

The fundamental model for network data communication is TCP/IP model and underlying functions in network traffic processing depends on the layers and level of analysis required.

1.2.1 THE TCP/IP MODEL

A protocol is a set of rules that governs data communication between devices. It defines what is communicated, how it is communicated and when it is communicated. Due to the complexity of communication between two different devices, network protocols are normally developed in layers, with each layer responsible for a different facet of the communication. Two important benefits result from decomposition into layers. First, the different layers can be designed more or less independently and therefore greatly simplifying network design. Second, compatibility is derived from the independent inner-working of each layer in the sense that each layer can be regarded as a black box with its own defined inputs and outputs. TCP/IP is normally considered to be a 4-layer system, as depicted in Figure 1.1. The layered principle in the TCP/IP model which entails that layer n

at the destination receives the same object sent by layer n at the source Figure 1.2 also illustrates how data are sent from computer A to computer B. As the data travel from computer A to computer B, it may pass through many intermediate nodes, such as routers and switches. These nodes usually involve only the first two lower layers of the TCP/IP model. The transmission first starts at the application layer in computer A and then moves down through the layers of computer A. At each layer n, a header Hn is added to the data unit received from the next higher layer, e.g., H4 is the header added at the layer 4. At the layer 1, a trailer T1 is added as well. Finally, when the formatted data unit passes through the physical network, a stream of bits are sent out. Upon reaching the destination, the data then move upwards through the layers of computer B. At each layer, the previously attached headers and trailers are removed. The different responsibilities and some key characteristics of each layer are discussed as follows:

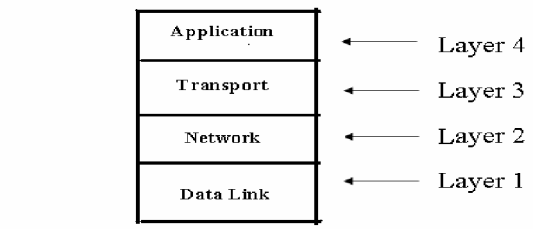


Figure1.1 The Four Layers of the TCP/IP Protocol Suite

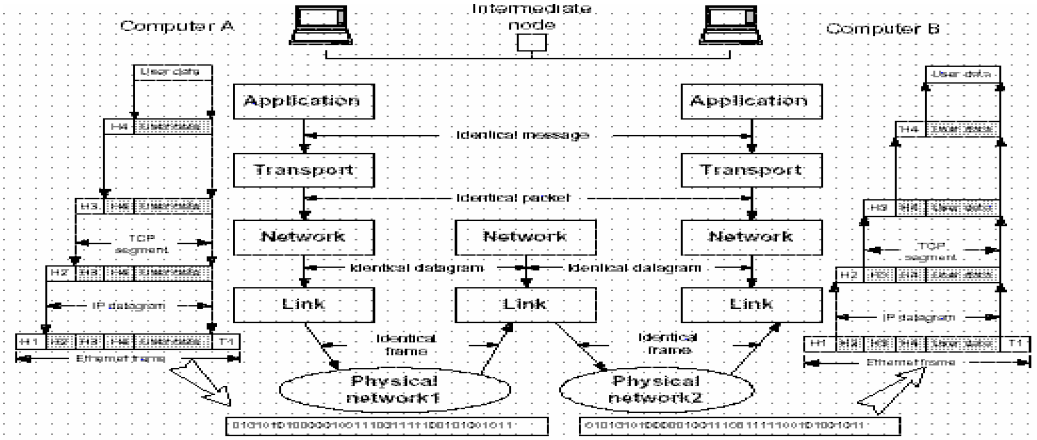


Figure 1.2 Illustration of data transmission From computer A to computer B (Ref 4)

1. Link layer, also called Data-link layer or The Network interface layer

Normally includes the device driver in the operating system and the corresponding network interface card in the computer. The network interface is assigned a unique physical address, e.g., 48-bit physical address for an Ethernet card, by which the computer is recognized. This layer is responsible for sending and receiving data from the network layer, communicating with the network interface in transferring data, and handling all the hardware details of physical interface. TCP/IP supports many different link layers, depending on the type of networking hardware being used: Ethernet, Token ring, Fiber Distributed Data Interface (FDDI), and the like. The data unit generated at the link layer is called a frame by adding a header and a trailer information - Cyclic Redundancy Check (CRC) - to the data received from the network layer. The maximum length of the frame is Maximum Transmission Unit (MTU) which limits the number of bytes of data that can be transferred. For Ethernet, its MTU is 1500 bytes. In this layer, two main protocols are defined: the Address Resolution Protocol (ARP) and the Reverse Address Resolution Protocol (RARP). ARP maps an IP address to a physical address while RARP performs the reverse operation.

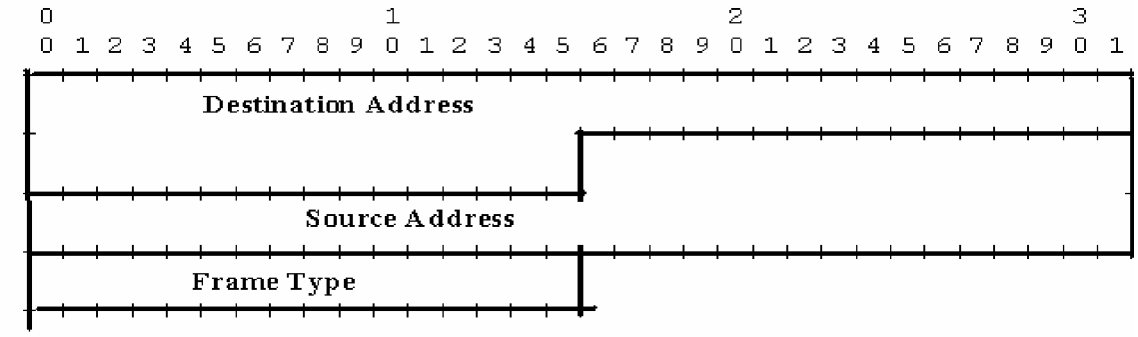


Figure1.3 Ethernet Frame Format

2. The network layer, also called the Internet layer

This layer is responsible for packaging, addressing, and routing functions. The

core protocol of this layer is the Internet Protocol (IP), which is one of the major protocols of the TCP/IP protocol suite (TCP being the other). The basic data object in this layer is the IP datagram, which is a variable-length packet consisting of two parts: a header and a data unit. The header can be from 20 to 60 bytes and contains information, e.g., IP address that is essential to routing and delivery. The most fundamental service provided by IP is a packet delivery. However, this service is unreliable, best effort, connectionless. The term unreliable means that packet may be delayed or duplicated at the destination. Or, the packet may be lost and cannot reach the destination. Best-effort means that IP delivers the packets without detecting problems or informing the sender or receiver about the problems. Connectionless means that IP treats each datagram as an independent entity unrelated to any other datagram's. In other words, IP datagram's can go through different paths to the destination. Therefore, out of order delivery is possible. The network layer also contains two other protocols, Internet Control Message Protocol (ICMP) and Internet Group Management Protocol (IGMP). ICMP is responsible for exchanging error messages and other vital information with the network layer in another host or route. IGMP is responsible for multicasting: sending a UDP datagram to multiple hosts.

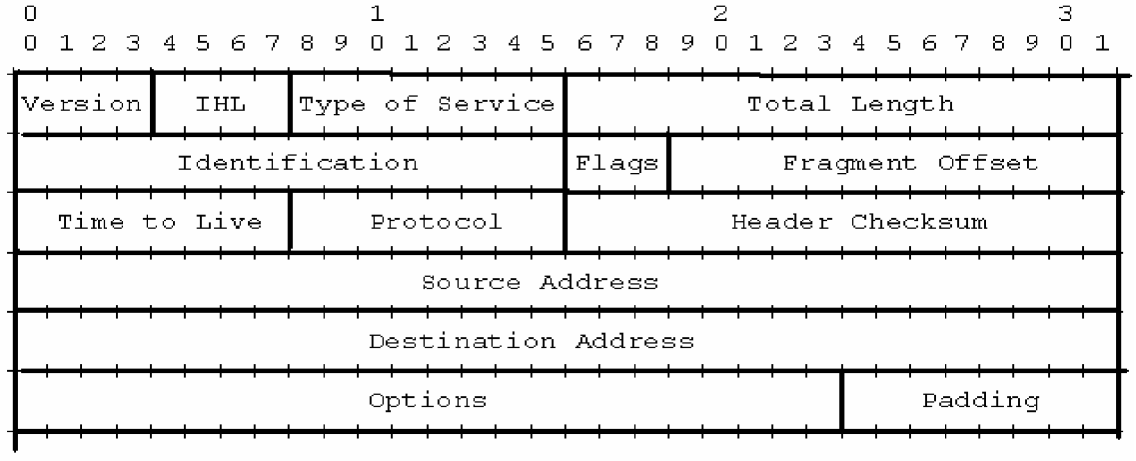


Figure 1.4 IP Frame Format

3. The Transport layer

This layer Provides communication from one application on computer A to another on

computer B, which is called end-to-end communication. This layer contains two main protocols: Transport Control Protocol (TCP) and User Datagram Protocol (UDP). Both TCP and UDP provide a port number to higher layer for applications to identify the endpoints of the communication. The data unit that TCP sends to IP is called a TCP segment. The one that UDP sends to IP is called a UDP datagram. TCP provides a one-to-one, connection-oriented, reliable communication service. The term connection-oriented means the application using TCP must establish a TCP connection between the two end systems before they can exchange data. TCP uses positive acknowledgment (ACK) with retransmission in order to provide a reliable service. Positive ACK means that the receiver will send back an ACK message to the sender when it receives the data. If the ACK is not received by the sender within a timeout interval, the data is retransmitted. At the receiver, the sequence number in the TCP header is used to correctly order the segments that may have been received out of order and to eliminate duplication. Packet damage is handled by adding a checksum to each transmitted TCP segment, checking it at the receiver and discarding it if it is damaged. UDP provides a one-to-one or one-to-many, connectionless, unreliable communication service. UDP is much simpler than TCP for three reasons. First, UDP does not use acknowledgements to ensure packet delivery. Secondly, it does not provide feedback to control the rate at which information flows between the two end systems. Thirdly, it does not order incoming data. Therefore, UDP datagram's can be lost, duplicated or arrive out of order. Furthermore, packets can arrive faster than the receiver can process them. Normally, an application program that utilizes UDP takes full responsibility for reliable packet delivery, which includes message loss, duplication, delay and out-of-order delivery. The header length of UDP is also much smaller than TCP header length. Therefore, UDP is often utilized when the amount of data to be transferred is small (such as the data that would fit into a single packet), when the overhead of establishing a TCP connection is not desired or when the applications or upper layer protocols provide reliable delivery.

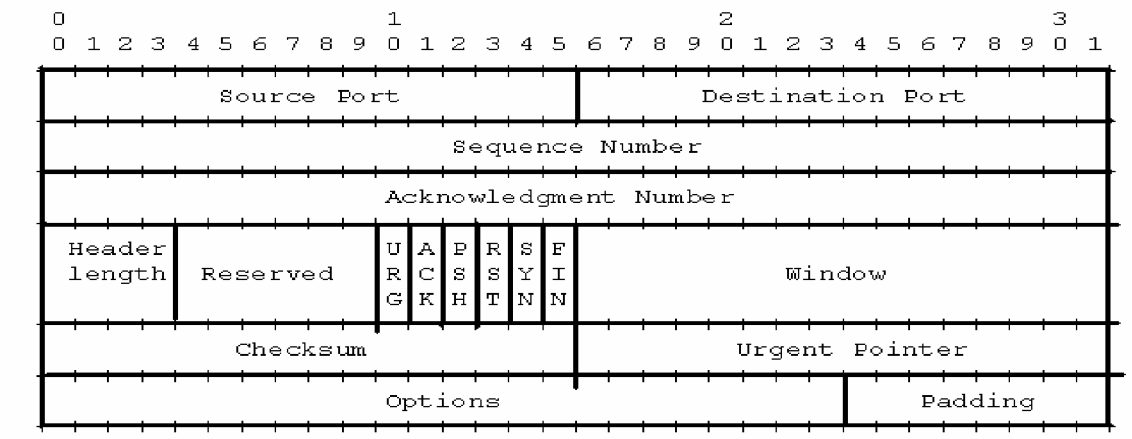


Figure 1.5 TCP Frame Format

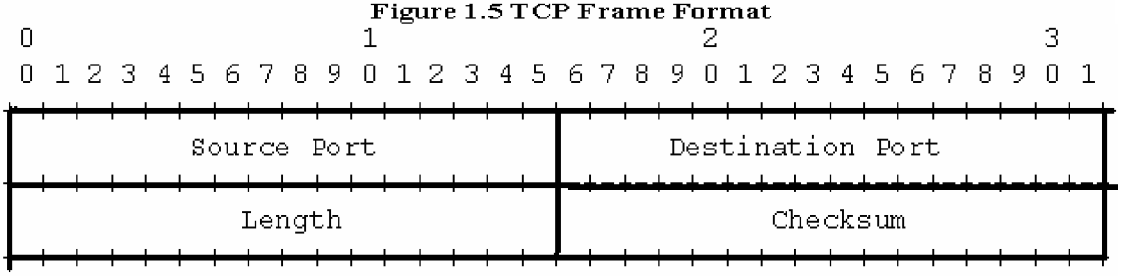


Figure 1.6 UDP Frame Format

4. The Application layer, the highest layer

This layer Contains application programs that provide services across a TCP/IP network. The application includes File Transfer Protocol (FTP), email, Voice over IP (VoIP), video conferencing, etc. An application interacts with one of the transport layer protocols to send and/or receive data.

1.2.2 NETWORK PROCESSING AND UNDERLYING FUNCTIONS.

The main underlying functions in network processing focus on two planes: the data plane and the control plane. In relation to the TCP/IP model, processing in the data plane focuses on layers 1 and 2, while processing in the control plane focuses on layers 3 and 4. Network processing in the data plane concerns analyzing and modifying each incoming packet, managing an input/output queue, scheduling packets in the queue, and forwarding the

packets towards their destinations.

1.2.2.1 THE UNDERLYING FUNCTIONS IN THE DATA PLANE ARE:

a) Media Access Control (MAC)

The term MAC address is often used as a synonym for physical address. Normally, the hosts and routers are recognized by their own physical addresses. However, physical addresses are not adequate in an inter-networking environment where different networks can have different address formats. A universal addressing system, in which each host can be identified uniquely, regardless of the underlying physical network, is needed. The IP address is designed for this purpose. This means that delivery of a packet to a host requires both two types of addressing: a physical address and an IP address. Therefore, the mapping between an IP address and a physical address and vice versa is needed, which is done by ARP and RARP, respectively.

b) Data parsing and classification

Look at a packet header and classify the packet based on a set of rules. Both functions can be found in all layers. For example, in the web server application, a packet to this server is classified by two rules: the destination port in the TCP header must be 80 and the protocol in the IP header must be 6. All packets that do not match the two rules cannot reach the web sever.

c) Data transformation

It entails many different operations: changing the content of a packet by adding additional information to the packet, modifying the packet when passing through different networks and segmenting, fragmenting and reassembling the packet. These operations differ

significantly in complexity. For example, a packet may be encrypted by using some encryption algorithms for security purposes. or, a packet may be encapsulated as a payload in a new packet with a new header when the packet passes from an IP network to an ATM network.

d) Queuing and scheduling

Manage an input buffer and an output buffer, make the decision on how to insert packets to queues and dequeue packets, and schedule packets for different applications. The packets are scheduled by many different policies and their priorities.

1.2.2.2 THE UNDERLYING FUNCTIONS IN THE CONTROL PLANE ARE:

Network processing in the control plane controls and manages device operations, updates routing table, executes routing algorithms, and processes some exception packets passed from the data plane. Unlike processing in the data plane, processing in the control plane will not process all packets. Therefore, it is less time-critical.

a) Topology management and network management

Consist of the following operations: monitoring network activity, distributing a database, generating real-time graphical views of the network topology, and updating a dynamic routing table. A routing table is used to forward a packet to the destination or to the next hop. It has to be updated as soon as there is a change in the Internet. For example, when a router is connected to the Internet for the first time, the routing tables of all routers on the Internet need to be updated by using routing protocols, such as the Routing Information Protocol (RIP) or the Open Shortest Path First (OSPF) protocol.

b) Signaling

Signaling ensures quality of services since the IP-based network only provides best-effort" services. Two groups have proposed signaling standards for IP telephony. The International Telecommunication Union (ITU) has defined a suite of protocols known as H.323, and the Internet Engineering Task Force (IETF) has proposed a signaling protocol known as the Session Initiation Protocol (SIP). Some other signaling protocols, such as Resource Reservation Protocol (RSVP) and Multi-Protocol Label Switching (MPLS), are also utilized in IP networks.

c) Policy application

Policy application is a set of rules which are used to execute routing algorithms and to update routing table. Considering a packet forwarding example, IP will perform a table lookup in order to determine the next hop where the packet should be forward.

A comparison between data plane processing and control plane processing is presented in Table there are five main differences between data plane processing and control plane processing. Because three differences out of five (in the first three rows of Table 1.1) are already discussed above, only the remaining two differences are discussed here. First, the difference in the software is that normally data plane processing is implemented in specific hardware written in VHDL language while control plane processing is implemented in software written in C or C++. Second, the difference in execution is that the data plane processing is launched on a special processor in order to achieve high performance while the control plane processing is launched on a general-purpose processor in order to achieve flexibility. With the increased demand for complex processing, the boundaries between the data plane and the control plane processing have become blurred.

	DATA PLANE	CONTROL PLANE
Main functions	Packet forwarding	Routing and Signaling
Layer	In layer 1-2	In layer 3-4
Performance	Real time processing	Less time critical
software	Fewer than 1500lines of code typically in VHDL or low-level microcode	Million lines of code written in C, C++
Execution	On a special purpose processor accelerated for maximum performance	On a general purpose CPU

Table 1.1 Comparison between Data Plane and Control Plane processing

CHAPTER 2 LITERATURE REVIEW

2.1 NETWORK TRAFFIC ANALYZERS

A network analyzer is a combination of hardware and software. The functions of network analyzers may be implemented on dedicated hardware, standard PC hardware, or a combination of both. The majority of LAN or WAN analyzers is processor-based and falls into two categories:

- Ø Software only and
- Ø Software with processor-based hardware.

Typically, these products attempt to minimize the processing overhead in real-time and capture every frame or cell on the network.

2.1.1 SOFTWARE-ONLY SYSTEMS

Software-only systems generally are low cost and PC-based and use standard off-the-shelf network interface cards (NICs) to connect to the network. All analysis functions are performed in software, and the product performance depends on the PC microprocessor performance. Typically, these products provide protocol decodes and require you to interpret detailed protocol information on a frame-by-frame basis. This is the most inefficient method of troubleshooting and often calls for a high level of protocol expertise. The use of shared RAM as a capture buffer in the PC requires that all captured data to be stored must be transferred across the computer data bus. The microprocessor is interrupted for the transfer of each frame captured by the NIC. For monitoring standard Ethernet (10 Mb/s), this equates to the theoretical maximum of 14,881 interrupts per second. Fast Ethernet (100 Mb/s) increases this by a factor of 10 (to 148,810). These performance limitations mean that software-only products are unsuitable for high-speed networking analysis such as ATM networks operating at 155.52 Mb/s (OC-3) and higher.

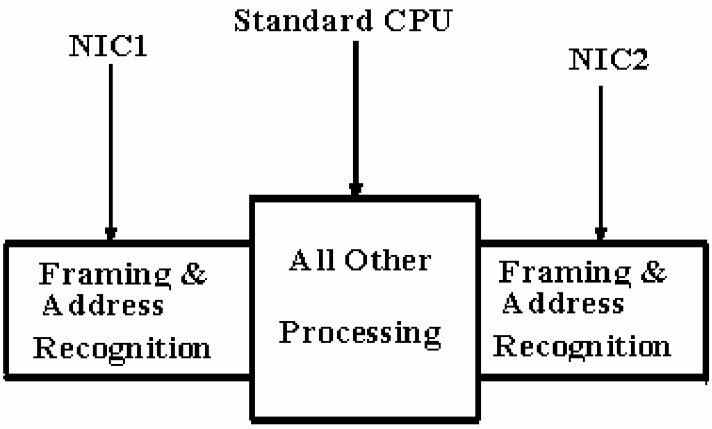


Figure 2.1 Traditional Software based Network Processing

2.1.2 PROCESSOR-ASSISTED ANALYZERS

First Generation Systems

Single CPU architectures are insufficient for higher speed networks. To optimize computation, move operations that account for the most CPU time from software into hardware. Higher speed networks require architectures that include hardware related techniques such as multiprocessor systems, ASIC coprocessor, smart NIC's with onboard processing. In first generation systems Data-capture speed can be increased by adding a dedicated analyzer NIC. A dedicated card can include a RISC processor, dedicated line-interface hardware, and an on-board RAM capture buffer. By using the RISC processor in combination with the on-board RAM buffers on the analyzer, the NIC greatly reduces the number of interrupts to the PC. This off-loading of tasks leaves the PC dedicated to the user interface and generally unaffected by the network loading. The use of direct memory access (DMA) for data transfers from the NIC to the PC will reduce the total number of interrupts. Analyzer performance improvement is directly related to the performance of the RISC processor.

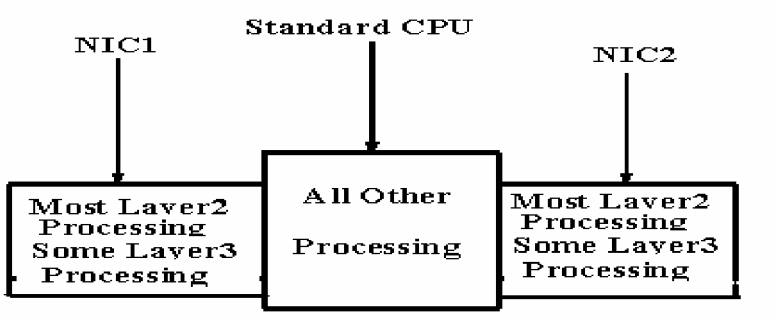


Figure 2.2 Traditional Software based Network Processing with Smart NIC's

Second Generation systems

A technology such as FDDI operates at data rates of 100 Mbps. ATM technologies uses small cells. Higher data rates and smaller packet sizes each increase packet rates, which means less time available to process a packet. The first generation systems are software with demultiflxing hardware could not handle such FDDI and ATM based networks. To accommodate higher packet rates second generation of networking hardware that uses special purpose protocol processing hardware with these features

- Ø The use of classification instead of demultiflexing to handle majority of packets.
- Ø A decentralized architecture with classification and forwarding capability on each interface card.
- Ø A high-speed internal interconnection mechanism that provides a fast data path to move data among interfaces.

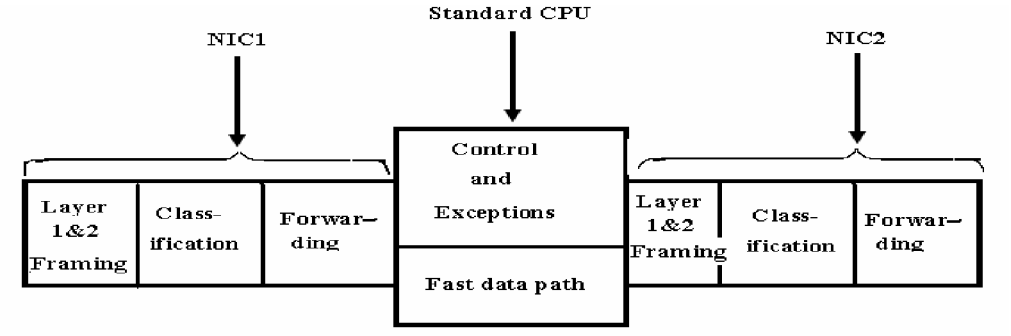


Figure 2.3 NIC's with embedded processor

This generation includes conventional CPU to manage and control system, provide interface to the administrator and run routing propagation protocols in addition to updating tables the CPU handles exceptions and errors including incoming ICMP messages and packets that do not match classification

2.1.3 NEW HARDWARE

Third Generation systems

The disadvantage of a second-generation system arises from the dependency on a GPP for some tasks. Even if the CPU is only used to handle a small percentage of packets the no of packets passed to the CPU increases linearly with the aggregate packet rate. Thus if the system has high aggregate packet rate the central CPU become bottleneck. *High-performance* monitoring is achieved using specialized new hardware. Field-programmable gate array (FPGA), contents addressable memory (CAM), and application-specific integrated circuit (ASIC) technologies are the building blocks that boost the performance of processor-based analyzers to the levels required for broadband network testing such as ATM. While all of these devices perform pattern matching, each technology has unique advantages that influence their use in a hardware design. The FPGA is the most versatile of these devices. It also is the most expensive but allows extensive reprogramming and is suitable for performing much more complex tasks in hardware. The FPGA in a network analyzer can be reprogrammed under software control. These devices perform many tasks traditionally done by processors including pattern-based filtering, cyclic redundancy checking, and frame and cell reassembly. The CAM is specifically designed for pattern matching. It has performance comparable to the FPGA and is programmable but also expensive and limited to pattern-matching applications. It has a unique capability to "learn" patterns as they occur on the monitored line. For example, a CAM could be programmed to continuously detect and remember the first 1,024 media access control addresses on a monitored LAN line and maintain a running count of all occurrences for each detected

(pattern-matched) frame. This could be used by an analyzer to maintain accurate frame counts for all traffic to and from a workstation or server. The ASIC is designed for specific tasks. It is best used in a mature networking technology and offers a high performance at a reasonable price. The ASIC cannot be reprogrammed. It can be used in pattern matching, but it usually is developed only after the standards are fully defined. Current use of ASICs on LAN switches has enabled line-rate frame forwarding using cut-through techniques, such as forwarding messages based on a subset by addresses or without error checking.

Main features

- Ø Functionality partitioned further
- Ø Additional hardware on each NIC
- Ø Onboard…
    - classification
    - forwarding
    - traffic policing
    - monitoring and statistics

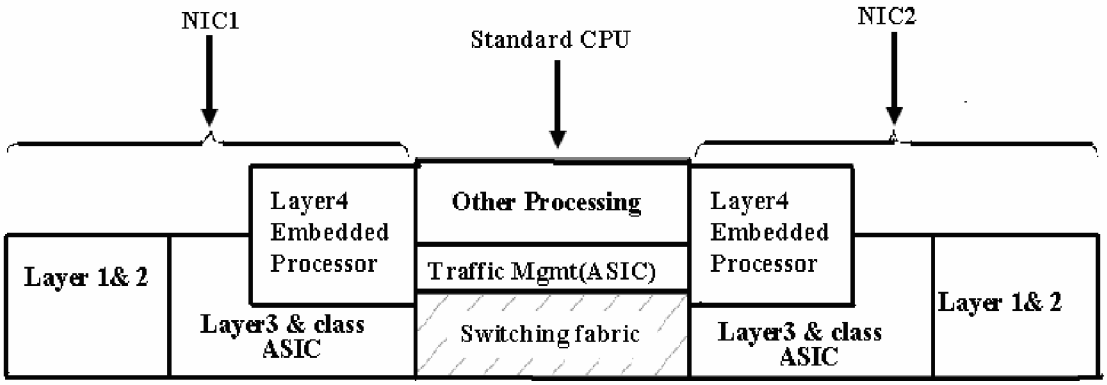


Figure 2.4 Network systems with embedded processor plus ASIC hardware on each NIC

- Ø Almost all packet processing off-loaded from CPU
    - Special-purpose ASICs handle lower layer functions

- Embedded (RISC) processor handles layer 4
- CPU only handles low-demand processing.

2.2 PROBLEM FORMULATION

General purpose processors can't provide wire speed performance for Packet processing and analysis. Even integrating embedded processor in to the NIC cannot handle packet rates for the highest-speed networks. Embedded ASIC hardware has extremely high cost and is difficult to implement and requires many months of fabricate a chip for even small changes. The embedded processor can handle only specified functionality with wire speed performance. Packet processing need's balance between the architecture and network flow. Developing Real time packet Analysis which can receive the packets processes the packets and forwards the packet with wire speed while utilizining the maximum network bandwidth as well as to marinating security in the network is an ideal application.

2.3 NETWORK PROCESSING DESIGN ALTERNATIVES

A complete design methodology - a design process - for embedded computing systems tells us that we need to consider the major goals of the design when creating a product, which may include functionality, performance, power consumption and manufacturing cost. Functionality provides a more detailed description of what the product does, Performance means the processing speed of the product, which may be a combination of soft deadlines such as approximate time to perform a user-level function and hard deadlines by which a particular operation must be completed. Power consumption gives a rough idea of how much power the product can consume. And manufacturing cost primarily defines the cost of the hardware components. Beyond this, we must also consider some other important requirements in system design: time-to-market, design cost and quality. In designing network devices, the latest developments in the Internet give raise two requirements:

performance and flexibility. Traditional network processing mainly focuses on forwarding packet at high speed in order to eliminate the network bottlenecks. Network devices are expected to perform at high speed with low latency. However, as the Internet Protocol keeps maturing, newer protocols have been emerging and will emerge in the future. Such newer protocols include security, signaling, and various network managements, etc.. Network devices are also expected to flexibly support these newer protocols and applications with high performance. In general, there are a number of techniques for designing network processors:

- *General-purpose processor (GP*P) - A programmable processor for general-purpose computing.
- *Application- specific integrated circuit (ASIC)* - An accelerator with specific function units. It interacts with the CPU through the programming model interface. It does not execute instructions.
- *Co-processor* - A separate chip connected to the internals of the CPU or the same chip nowadays. It implements additional instructions, which are not supported by the main processor to which it is attached to.
- *Application-specific instruction processor (ASIP)* - A CPU whose instruction set has been optimized for a particular set of applications.
- *Field-programmable gate array (FPGA)* - A programmable logic chip that can be quickly customized to a particular function. FPGAs come in two basic varieties: one-time-programmable devices, which can be programmed once, and reprogrammable devices, which can be programmed many times with new logic designs.

In order to examine which technique can meet the two requirements, Figure 2.5 maps these categories on two axes: flexibility and performance. In Figure 2.5, we can see that GPP is the most flexible at the cost of the performance and ASIC provides the best performance with the worst flexibility.

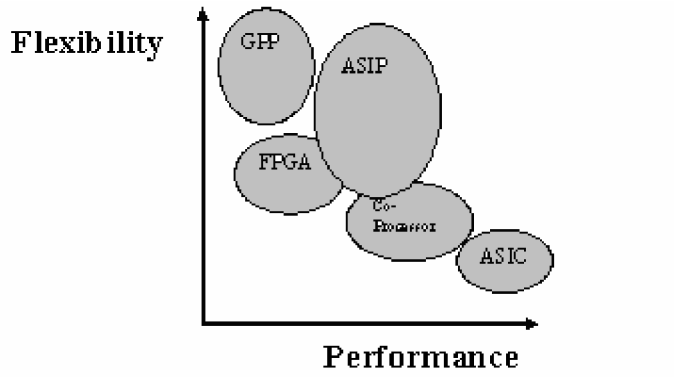


Figure 2.5 The Space of the System

Earlier network devices, such as routers and switches, employ GPPs for network processing. The best advantage of this approach is that it enables flexible support new features required by the new applications since GPP is a programmable processor for general purpose computing. However, the drawback of this approach is its limited ability to scale the performance in order to support the demands for higher bandwidths. With bandwidth increasing from OC-3 (155Mbps) to OC-192 (10Gbps) and up to OC-768 (40Gbps) or higher in the future, design priority shifts from flexibility to speed. GPPs can not have enough performance to process data at wire-speed. Therefore, the market turned to high-performance ASICs for packet processing. An ASIC does accommodate the increasing demand for speed when performing the required packet processing .However it is restricted by its inherent inflexibility. Nowadays, as mentioned before, since security, QoS and multimedia applications become the new networking concerns, ASICs are limited to these functionalities and can not support new features. In conclusion, the flexible combinations of the data plane and the control plane processing with high performance are not possible with either ASIC or GPP solutions alone. Today, the market is turning to a new solution, network processors (NPs): high-performance, programmable devices designed to efficiently execute communications workloads.

A network processor is an ASIP for the networking application domain. Figure 2.5 also depicts that ASIP is the best flexibility-versus-performance trade-off for networking system implementations. Currently, there are a number of network processors available in the market, such as Payload Plus network processor from Agere, Toaster2 from Cisco, and Power NP network processor from IBM, and IXP1200, 2400 network processor from Intel and so on. The features of Network Processors are compared from the following perspectives..

- *Performance* - by executing key computational kernels in hardware, NPs are able to perform many applications at wire speed
- *Flexibility* – having software as a major part of the system allows network equipment to easily adapt to changing standards and applications
- *Fast TTM* – designing software is much faster (and cheaper) than designing hardware of equivalent functionality
- *Power* – while NPs may not be embedded in energy-sensitive devices (like handhelds), their power consumption is important for cost reasons (e.g. implications on packaging).

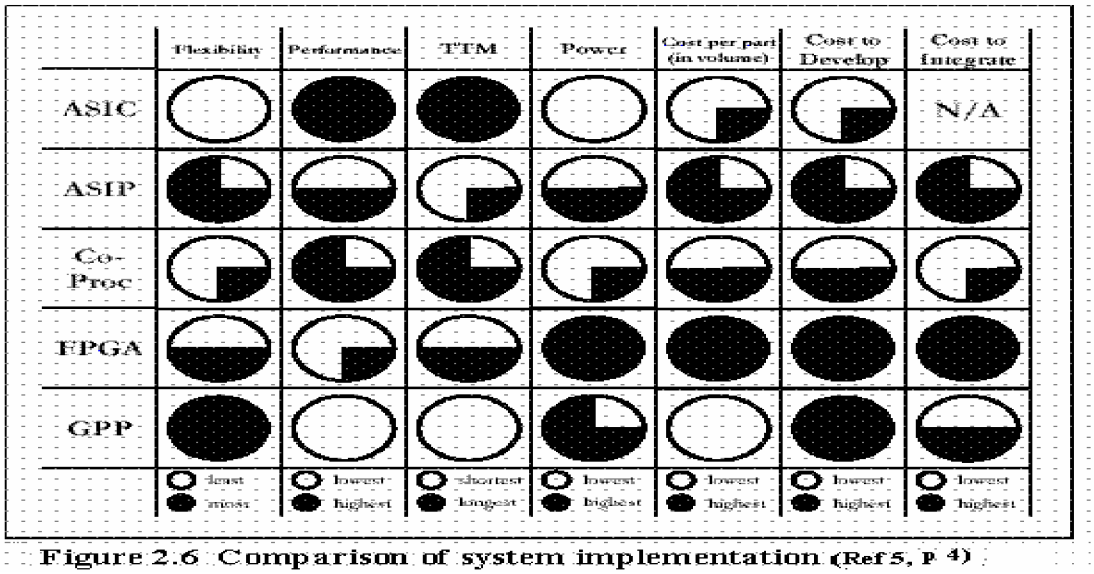


Figure 2.6 Comparison of system implementation (Ref 5, P 4)

CHAPTER 3 NETWORK PROCESSORS

3.1 THE DEFINITION OF A NETWORK PROCESSOR

A Network Processor (NP) is a programmable (processor) integrated as a single semiconductor device which is optimized to primarily handle network processing tasks. These processing tasks include: receiving data packets, processing them, and forwarding them.

3.2 NETWORK PROCESSORS IN GENERAL

A Network Processor's main purpose is to receive data, operate on it, and then send out the data on a network at wire speeds (i.e., only limited by the link's speed). They aim to perform most network specific tasks, in order to replace custom ASIC's in any networking device. A NP plays the same role in a network node as the CPU does in a computer. The fundamental operations for packet processing consist of following operations:

- *Classification*, parsing of (bit) fields in the incoming packet and table lookup to identify the incoming packets, followed by a decision based regarding the destination port of the packet.
- *Modification of the packet*, data fields in the header are modified/ updated. Headers may be added or removed and this usually entails recalculation of CRC or checksum.
- *Queuing and buffering*, packets are placed in an appropriate queue for the outgoing port and temporary buffered for later transmission. The packet may be discarded if the capacity would be exceeded.
- *Other operations*, such as security consideration, policing compression, traffic metrics.

3.3 INTEL IXP1200 NETWORK PROCESSOR

Intel IXP1200 Network Processor family is Intel's alternative for this multi-service network device market. Intel IXP1200 network processors consist of a high-speed processor core and six programmable multithreaded Microengines. The processor family was introduced in 1999 and it now has four models: IXP1200 (the basis of the series), IXP1240 (with CRC error checking built in), IXP1250 (with CRC and error-correcting memory support) and IXP1250 for extended temperature range. Functionally and architecturally they are the same with the mentioned exceptions. Intel's Internet Exchange Architecture (IXA) is a packet processing architecture that focuses on Intel network processors.

3.3.1 IXP1200 ARCHITECTURE

3.3.1 OVERVIEW

The IXP1200 chip contains a Strong-Arm core and six independent 32-bit RISC data engines (Microengines) as well as SRAM, SDRAM, PCI and IX bus controllers (Figure 3.1). The operating frequencies are 166 - 232 MHz. The performance of the processor is

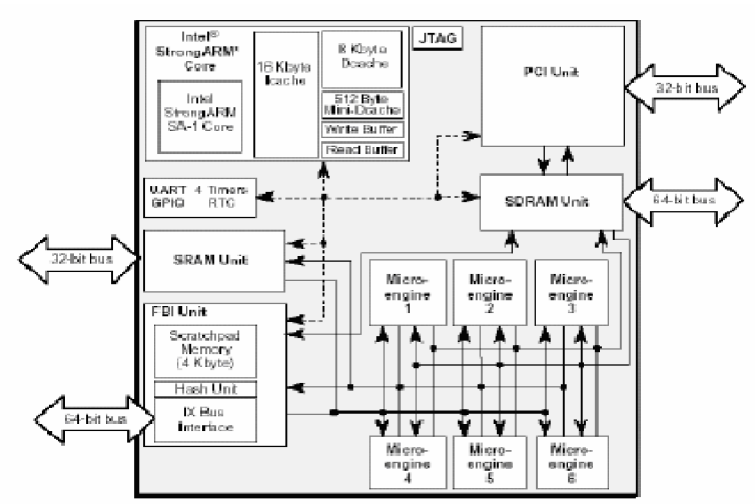


Figure 3.1 IntelIXP1200 Network processor Block Diagram

Said to be 3 million packets per second (3 Mbps), which makes 1.5 Gbps (3000000 * 64 Bytes * 8 bits). Performance can be increased by connecting several processors in parallel .In that way a total capacity of 24 Mbps can be achieved with eight processors.

3.3.2 FUNCTIONAL UNITS

IXP1200 consists of several functional units that communicate with each other in various ways. In general, each unit can work independently and signal other units only when needed. This method helps to eliminate unnecessary waiting for units to complete their work. The units discussed here are Strong-ARM core, six Microengines, FBI bus unit and IX bus.

3.3.2.1 STRONG ARM CORE MICROPROCESSOR

The Strong ARM core is the same industry standard 32-bit RISC processor as used in the Intel Strong ARM SA-1100. It is compatible with the Strong ARM processor family currently used in applications such as network computers, Personal Digital Assistants (PDAs), palmtop computers and portable telephones. It provides high performance in a low-power, compact design. The Strong-ARM core has two caches: a 16-kilobyte cache for instructions and an 8-kilobyte cache for data (Figure 3.2). In addition, the core contains a 512-byte mini cache, which is intended for data that is read once, operated on, and then discarded. The intention is to reduce flushing of the main cache. The Strong ARM core may be used in different ways, depending on the application. If the system already contains a high-level host processor, the Strong-ARM core can run a mini kernel and execute routing protocols, while the Microengines do the fast-path packet processing in designs where there is no other host processor; the Strong ARM is the main processor. As the Microengines are used in actual data transmission, the Strong ARM core is used for running a Real-Time Operating System (RTOS) and more complex tasks such as address

learning, network management and building and maintaining forwarding tables. The Strong-ARM core operates at the nominal frequency of the processor (166 – 232 MHz).

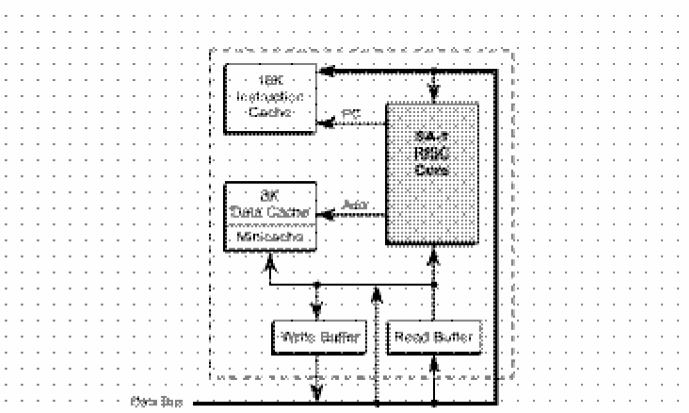


Figure 3.2 The StrongARM block diagram
(Ref 20, p 31)

3.3.2.2 MICROENGINES

Six 32-bit, multithreaded RISC data engines (Microengines) perform data movement and processing without assistance from the Strong-ARM core. The micro-engines are especially designed for data transmission and handling with bit, byte, word and long word operations. The Microengines are programmed with symbolic microcode that is developed with Intel IXA Software Development Kit (SDK). Each Microengine has four independent program counters. That makes a total of 24 microcode contexts in a single IXP1200 processor. The threading is hardware-based and context switching within each Microengine requires no extra instructions.

3.3.2.2.1 INSTRUCTIONS AND EXECUTION

Each Microengine contains its own instruction store (Program Control Store), its own set of 256 registers and other Control/Status Registers (CSRs). All the Microengines are identical so that the functions may be interchanged between them. There are no fixed functions for

any of the Microengines because they are all fully programmable. The Microengines operate at the core clock frequency of the IXP1200 like the Strong-ARM core. All Microengine instructions execute in one clock cycle. The Microengines are implemented as five-stage pipelined processors. In the first stage, the instruction is fetched from the instruction store; in the second stage, the instruction is decoded; in the third stage the operands are fetched from the registers; in the fourth stage the operands precede through the ALU (Arithmetic and Logic Unit) and in the fifth stage, the result is written out to the destination register. As each Microengine has only one instruction store, each four threads running in the same Microengine have basically the same program code. This is not a limitation, however, because each thread can start the execution at different address. The instruction store has 2048 instruction words (earlier stepping of IXP1200 has only 1024 words). Each instruction is a 32-bit long word. The instruction store is loaded by a loader program running in the Strong-ARM core at the boot time.

Deferred Instructions In general, branching inside the program code destroys the idea of pipelining the instruction execution. As told earlier, the processor pipeline is loaded with the instruction a few steps before its actual execution. When a branch instruction is executed, the instructions following the branch instruction may be aborted and their operands have to be discarded. Then the pipeline has to be loaded again from the new point of execution. To avoid this kind of penalties, the IXP1200 implements deferred execution of instructions. Using this method, up to three instructions that are usually placed before the conditional branch instruction, can be placed after the branch instruction, and the Microengine will still load them before branching. While still executing those instructions, the Microengine can already fetch the instructions and operands from the branch destination. 8 For example, the instruction "br[target#], defer[2]" (unconditional branch) changes the execution to label "target" after executing two instructions following the instruction. The microcode assembler does this optimization automatically although the programmer can use this property himself too.

Multithreading many Microengine instructions need to access other units of

IXP1200. Usually those external (from the Microengine point of view) accesses are slower than the operating speed of the Microengines. The hardware-based multithreading enables other threads to run while one thread is waiting for memory or any other external unit access results. There can always be some code running when some other code is stopped. The "ctx_swap" optional token is used to instruct the thread context to be swapped while waiting for data to be complete.

Branch Prediction Branch prediction is another method that is used to optimize the performance. The Microengine can be instructed to fetch next instructions from the location of the branch destination. That can help keeping the execution pipeline filled with valid instructions if the branch is predicted correctly. The "guess_branch" optional token indicates the branch prediction.

3.3.2.2.2 REGISTERS

There are two kinds of registers in the Microengines: general purpose registers (GPRs) and transfer registers. The 128 general purpose registers are in turn divided into two banks, an A bank and a B bank. This is done to enable the ALU to fetch two operands simultaneously. The ALU instructions use one operand from the A bank and one from the B bank (Figure 3.3). The transfer registers are divided into SDRAM and SRAM transfer registers, and they are divided into read and write portions. That makes 32 registers in each portion, and 128 registers in total. Each of these sets of registers can be used at the same time because they have separate data paths in respective functional units. Registers can be accessed in two addressing mode: context-relative and absolute. In context-relative mode each thread gets its own register. In other words, one register name is actually associated with four different physical registers, depending on the thread that is addressing the register. In absolute addressing mode data can be shared between threads because each thread is addressing the same physical registers. The registers are named symbolically. The programmer names the registers he uses and the microcode assembler assigns the names to

appropriate registers. There are some character prefixes that are used to indicate the type of the register (GPR, SDRAM, and SRAM) and the addressing mode

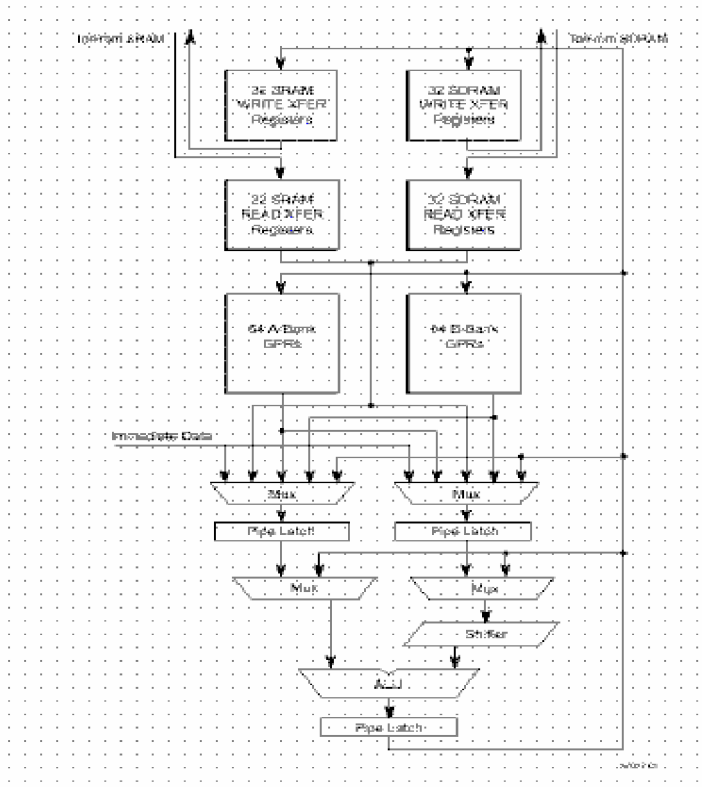


Figure 3.3 Microengine data path (Ref 21, p 20)

3.3.2.3 FBI UNIT AND IX BUS

The IX bus unit (also referred as the FBI unit) controls the IX bus. It transfers data to and from the receive and transmit queues, receive FIFO and transmit FIFO. The IX bus connects the processor to MAC (Media Access Control) devices such as 10/100 Mbps or gigabit Ethernet controllers and also to parallel IXP1200 processors . The Operating frequency is 66 - 104 MHz and the maximum capacity is 4 - 6.26 Gbps depending on the operating frequency of the processor (Figure3.4). IX bus data flow the IX bus can be configured as either a 64-bit bidirectional bus or as two 32-bit unidirectional buses, which

are transferring data independently to different directions. Once the data is in the receive FIFO, it can be transferred to either the Microengines (their transfer registers) or to SDRAM. Data to be transmitted to devices on the IX bus can only come from the transmit FIFO, and then transmitted from there to the devices on the IX bus. The IX bus unit contains (among the receive and transmit FIFO) control and status registers, a 4-kilobyte scratchpad RAM and a hash unit for generating 48- and 64-bit hash 12 keys. There is also a sideband bus operating in parallel to the IX bus. That is called the Ready bus. It consists of eight data pins and five control pins. The Ready bus operates synchronously with the IX bus. The receive and transmit FIFO's are constructed as tables of 16 * 64 bytes, and the software can operate on each component as needed. The FIFO's are shared by all of the Microengine threads. Software has to ensure that the FIFO's are used in a sensible way by all the threads. The IX bus unit is connected to other units as shown in

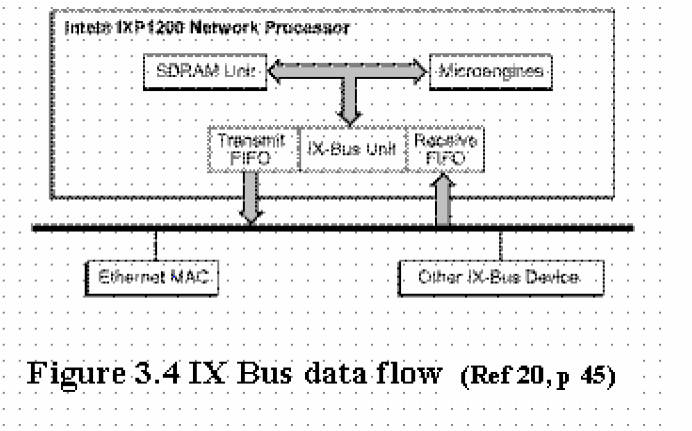


Figure 3.4 IX Bus data flow (Ref 20, p 45)

Unit Access

The Strong-ARM core can access the scratchpad RAM within the IX bus unit, as well as a number of control/status registers within the IX bus unit. The Strong-ARM core cannot access the receive and transmit FIFO's within the IX bus unit. The Strong-Arm core can program the Ready bus controller within the IX bus unit. Microengines Full access, including access to the scratchpad RAM, the hashing unit, programming the Ready bus controller, the receive and transmit FIFO's and all control/status registers. Two direct, unshared, independent busses connect the SDRAM unit with the IX bus unit. The SDRAM

read bus is connected to the receive FIFO, while the write bus is connected to the transmit FIFO. SRAM unit no connection between the SRAM unit and the IX bus unit's FIFOs. Data that needs to be transferred between the IX bus (transmit and receive FIFOs) and SRAM must be transferred by the microengines (by way of their transfer registers). PCI unit No connection between the IX bus unit and the PCI unit. Data that needs to be transferred between the IX bus (transfer and receive FIFOs) and the PCI bus must be transferred by the microengines (by way or their transfer registers).

3.3.2.4 PCI UNIT

The PCI unit provides an industry standard 32-bit PCI (Peripheral Component Interconnect) bus to interface to PCI peripheral devices such as host processors and MAC devices. The PCI unit supports operating speeds up to 66 MHz . Above 33 MHz operation, only two PCI devices are supported: the IXP1200 and a second PCI device. To support more devices on higher frequencies, a PCI-to-PCI bridge can be used. The PCI unit is connected to the SDRAM unit and the Strong ARM core. Devices on the PCI bus have full access to SDRAM. The Strong ARM core also has full access to the PCI interface and can be a bus master. Two DMA controllers are integrated into the PCI unit. Both of them can be used by either the Strong ARM core or the microengines. The DMA controllers are programmed by means of DMA descriptors that reside in SDRAM. The descriptors can be chained so that non-contiguous blocks of data from SDRAM can be transferred to PCI as one block. When transferring data between the PCI unit and IX bus unit, the microengines have to be used. Typically, the data would be transferred from the IX bus unit's FIFOs directly to SDRAM. Then, the microengine thread would use PCI DMA controller to transfer data from SDRAM to the PCI interface.

Unit Resource

The Strong ARM core can access any devices on the PCI bus. The Strong ARM core can program any of the PCI unit's control/status registers. The Microengines can only access

the control/status registers for the DMA controllers within the IX bus unit no other access. The microengines cannot access devices on the PCI bus (and vice versa). IX bus unit No connection between the IX bus unit and the PCI unit. Data that needs to be transferred between the IX bus (receive and transmit FIFOs) and the PCI bus must be transferred by the microengines by way of their transfer registers). SDRAM unit Unshared, direct 32-bit bus connecting the SDRAM unit and the PCI unit. Devices on the PCI bus have access to SDRAM. DMA controllers within the PCI unit can transfer data between SDRAM and devices on the PCI bus. SRAM unit No connection between the SRAM unit and the PCI unit. Devices on the PCI bus cannot access SRAM. DMA controllers within the PCI unit cannot access SRAM.

3.3.3 MEMORY UNITS

3.3.3.1 SDRAM UNIT

The IXP1200 provides an SDRAM unit to access low cost, high bandwidth memory for mass data storage. SDRAM is used for forwarding information and transmit queues. The Strong-ARM core address space allows up to 256 megabytes of SDRAM to be addressed. The SDRAM bus is 64 bits wide. It is accessed using quad word (64 bits, 8 bytes) operations. When byte, word or long word operations occur from the Strong-ARM core or PCI unit, a quad word is read from SDRAM. Only the necessary bytes are modified. Finally the entire quad word is written back to SDRAM. These three steps (read-modify write) are performed automatically. One microcode instruction can initiate a transfer of 16 quad words (128 bytes) at a time. From the Microengines only quad word accesses are supported. Less than 8 bytes can be written using the byte mask within an instruction, but it results in read-modify-write cycles.

Unit Resource

Strong-Arm core can access Full SDRAM. Microengines have Full access capability. IX

bus, unit there is a 32-bit bus connecting the SDRAM unit and the IX bus unit's receive and transmit FIFO. SRAM unit, no connection between the SDRAM unit and the SRAM unit. PCI unit there is a separate and unshared 32-bit bus connecting the SDRAM unit and the PCI bus unit. This allows devices on the PCI bus better access to data buffers within the SDRAM unit. The two DMA controllers in the PCI unit have access to SDRAM data. The SDRAM interface operates at half the core frequency, providing a peak bandwidth of 928 megabytes per second at 232 MHz

3.3.3.2 SRAM UNIT

An SRAM unit is provided for very high bandwidth memory for storing the lookup tables and other data for the packet processing Microengines. The SRAM unit controls the SRAM (up to 8 megabytes), Flatiron (up to 8 megabytes) for booting and 2 megabytes of Slow Port address space for peripheral device access (Figure 3.5). SRAM unit external interfaces In the figure below, the Slow Port address space is used for interfacing to Ethernet MAC access ports for MAC programming and for accessing MAC manageability registers. The SRAM interface is 32 bits wide. That is half of the SDRAM bus width because SRAM is not intended for bulk data storage but for fast lookups. Like for the SDRAM interface, the SRAM interface operating frequency is half the core frequency.

Unit Access

Strong ARM core has Full access capability. Microengines also Full access permission. No connection between the SRAM unit and the SDRAM unit. No connection between the SRAM unit and the IX bus unit.PCI unit No connection between the SRAM unit and the PCI unit.

3.3.3.3 SCRATCHPAD

Scratchpad memory is 1K long-word (4KB) of on chip memory. Scratchpad provides a small low latency memory interface to all microengines. The scratchpad is physically

located within FBI. In addition to random access reads and writes scratchpad also provides atomic bit-test-and-set and increment operations. This atomic operation is unique to scratchpad memory

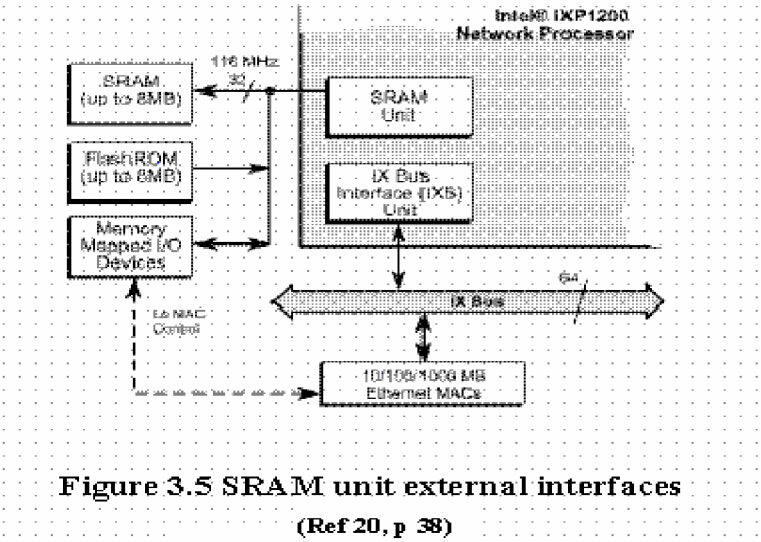


Figure 3.5 SRAM unit external interfaces

(Ref 20, p 38)

3.4 PROGRAMMING PRINCIPLES

3.4.1 PROGRAMMING LANGUAGES

Today, many network processors only have capacity for a few kilobytes of code. Intel still recommends writing in assembly code until their C-compiler has been further developed. Some NPs use functional languages to produce smaller programs with fewer lines of code. These languages are more complex, but programming effort can be saved.

3.4.1.1 ASSEMBLY & MICROCODE

Assembly, or microcode, is the native language for a NP. Although microcode for different

NPs may look the same, there are huge differences. Each network processor has its own architecture and instruction set. Thus programs for the same purpose are quite different between different NPs. Therefore, the NP industry is heading for a serious problem for the future, how to standardize coding, and so programs can be reused in another NP.

3.4.1.2 HIGH-LEVEL LANGUAGES

Most vendors supply code libraries and C-compilers to use for their NP. A code library usually covers basic packet processing code needed for IPv4 forwarding or ATM reassembling. There are significant advantages to use high-level languages such as C instead of microcode:

- C is the most common choice for embedded system and network application developers.
- A high-level language is much more effective at abstracting and hiding details of used Instructions.
- It is easier and faster to write modular code and maintain it in high-level language with support for data types, such as type checking.

One of the upcoming programming techniques is functional programming where the languages describe the protocol rather than a specific series of operations.

3.4.2 THE OPERATING SYSTEM ON IXP1200

The IXP1200 system needs an operating system to be able to run. In general, an operating system is defined as the low-level software that provides the interface to peripheral hardware, allocates storage, schedules tasks and presents an interface to the user. In the IXP1200 system, the operating system does the overall process management work. It has all the necessary routines to manipulate address and forwarding tables so that the code running in Microengines can decide packet forwarding destinations. The operating system

runs software that loads the Microengines' instruction stores and starts and stops the Microengines when necessary. There are two operating systems supported by Intel, VxWorks and an embedded Linux. VxWorks is a real-time operating system developed by Wind River Systems, Inc. It is a very popular operating system in embedded industry. It is the run-time component of Tornado II embedded development platform. VxWorks is interfaced with the embedded target hardware by the Board Support Packages (BSP). Intel ships IXA SDKs with a BSP for IXP1200 Strong-Arm core. The BSP includes the core libraries of the VxWorks operating system ported to that specific processor. Intel also supports the use of an embedded Linux system with IXP1200. The Intel IXA SDK 2.0, which became publicly available in the fall 2001, includes the Embedded Linux IDE (Integrated Development Environment) to develop code for Linux. Intel has released a how-to document of setting up a Linux system on IXP1200. This study only reviews the use of VxWorks in the IXP1200 system. Whichever operating system is selected and used, the operating system will be running on the Strong-Arm core of IXP1200.

3.4.3 APPLICATION STRUCTURE (The Packet Processing Paths)

The IXP2400 network application structures are divided in two logical planes Data plane and Control Plane shown in Figure below:

Data plane, it processes and forwards packet at wire speed. It consists of: a fast path, which handles most of the packets (For example, forwarding IPv4 packets), a slow path used for handling exception packets (For example, handling fragmented packets). Data plane tasks handle time-critical duties in the core design

Control plane, less time critical tasks that fall outside the core processing or forwarding requirements of a network device are called Control Plane tasks it handles protocol messages and is responsible for setup, configure, and update tables used by the data plane. For example, the control plane processes RIP, OSPF packets containing routing information to update IPv4 forwarding tables.

Fast path and slow path The data plane and the control plane are processed over a fast path or a slow path depending on the packet. As a packet enters a networking device, it is first examined and processed further on either the fast path or slow path. The fast path (most data plane tasks) is used for minimal or normal processing of packets and the slow path are used for the unusual packets and control plane tasks that needs more complex processing. After processing, packets from both slow and fast path may leave via the same network interface.

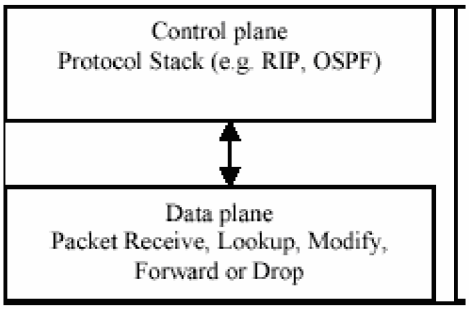


Figure 3.6 Logical planes used in IXP1200

3.4.4 PROGRAMMING FRAMEWORK (Software Architecture)

Divide the application tasks belonging to each functional plane among code modules in any number of ways, depending on the complexity of application. A single control module, for example, might receive messages from multiple data processing modules, or might create separate control modules for different control tasks. Similarly, combine control and management plane tasks into a single code module, or keep them separate (Figure 3.7). Different code modules can be compiled to run on different processors, depending on what is available and appropriate to a module's tasks.

- Data processing and control modules are commonly located on an IXP1200 network processor where they can react to and process packets at high speeds. They can be further optimized by compiling them specifically for different processors on the IXP1200 network processor.

- A manager program might be located on another processor where it has access to user interfaces, storage, and the full range of services provided by the operating system.

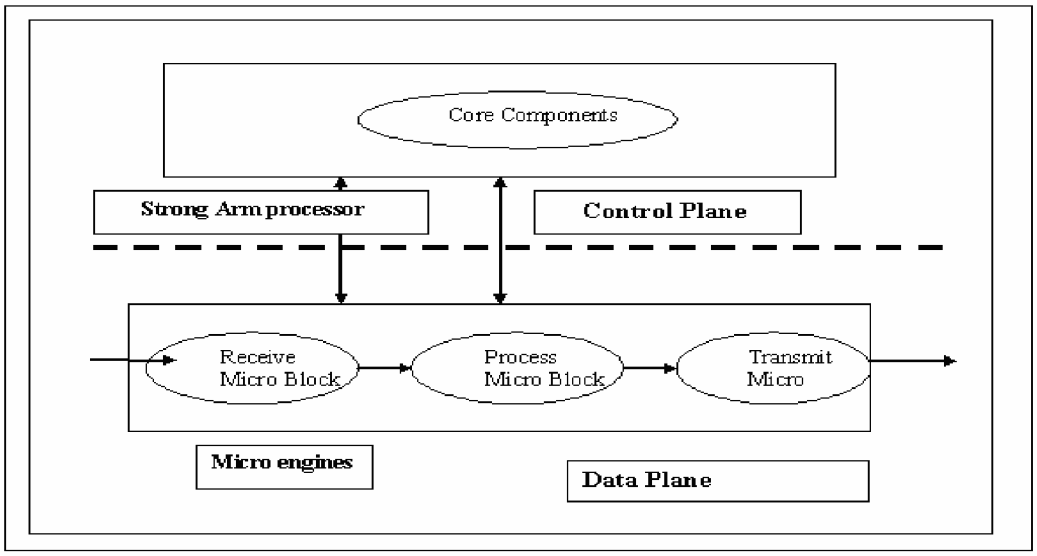


Figure 3.7 Software Architecture in IXP1200

3.4.4.1 THE ACE PROGRAMMING FRAMEWORK:

In order to encapsulate and modularize the unique tasks involved in packet processing, The IXA architecture defines a software construct called an *active computing element (ACE)*. The micro ACE programming model allows a packet-processing application to make use of the IXP1200's microengines for acceleration of fast-path processing. Micro Aces perform specific tasks such as IP forwarding, network address translation (NAT), and IP filtering. A micro ACE has two logical components, running on different processors:

- A *microblock* contains fast-path packet processing logic which you write in Microcode to run on a microengine. Microblocks can be one of three types:
  - *Source* microblocks receive packet buffers from outside the microengine.

- *Transform* microblocks operate on packet buffers and pass them on to other microblocks.
- *Sink* microblocks send packet buffers out of the microengine

Ø Each microblock is associated with a *core component*, which you write in C/C++ and NCL to run on the core processor. The core component is a complete ACE with additional elements that initialize and manage the microblock component. Interface ACE's are implemented as micro ACE's with source (Rx) and sink (Tx) microblocks. The interface micro ACE's are used to receive and transmit packets through the network interface ports.

.

3.4.4.2 MICRO ACE ARCHITECTURAL ELEMENTS:

A micro ACE combines a core component, which runs on the core processor, with a microblock, written in microcode to run on a microengine. A management program loads and configures micro ACE's along with the conventional ACE's for an application. The management program initializes the micro ACE *Resource Manager*, which deals with resource allocation, packet demultiplexing, and other system-level microengine management problems. Several microblocks are typically combined into a single microengine image called a *microblock group*. Each of the microblocks in a group is associated with exactly one core component. A microcode *dispatch loop* for each microengine initializes the microblocks in a group and determines how packets flow into, among, and out of the microblocks. The management program creates static binding among the microACEs to match the packet flow among microblocks in each group, as determined by the dispatch loop. Together, the microblock group and their linked core components form a *micro ACE group*.

3.4.5 DEVELOPING CODE FOR THE STRONG-ARM CORE

The code for the Strong-Arm core is developed using Tornado II by Wind River Systems, Inc. It is an embedded development platform running on 32-bit UNIX and Windows workstations. Tornado II embedded development environment on Windows workstation Tornado comprises the following elements:

- VxWorks,
- A high-performance real-time operating system
- Application-building tools (compilers and associated programs)
- An integrated development environment (IDE) that facilitates managing and building projects, establishing and managing host-target communication, and running, debugging and monitoring VxWorks applications

Tornado environment is used to compile the C/C++ language source files into binary executables that can be downloaded into the IXP1200 system. In particular, a bootable image of the operating system can be stored in the flash memory of IXP1200 system so that the operating system is loaded automatically during the power-up of the system. Other binary images can be loaded afterwards.

3.4.6 DEVELOPING CODE FOR THE MICROENGINES

3.4.6.1 IXP1200 DEVELOPER WORKBENCH

The code for the IXP1200 Microengines is developed using IXP1200 Developer Workbench that is included in the Intel IXA SDK. The code is symbolic microcode that is assembled with an assembler before linking. Figure3.8 shows on overview of the whole development process for the IXP1200. the most important features of Developer Workbench include:

- Source level debugging
- Execution history and statistics

Ø IX bus device and network traffic simulation.

Developer Workbench comprises a preprocessor, assembler and linker to link the microcode with the Strong-Arm code. Also it includes a comprehensive debugger. The preprocessor processes directives, loops, conditional code and expressions and performs macro inline expansion

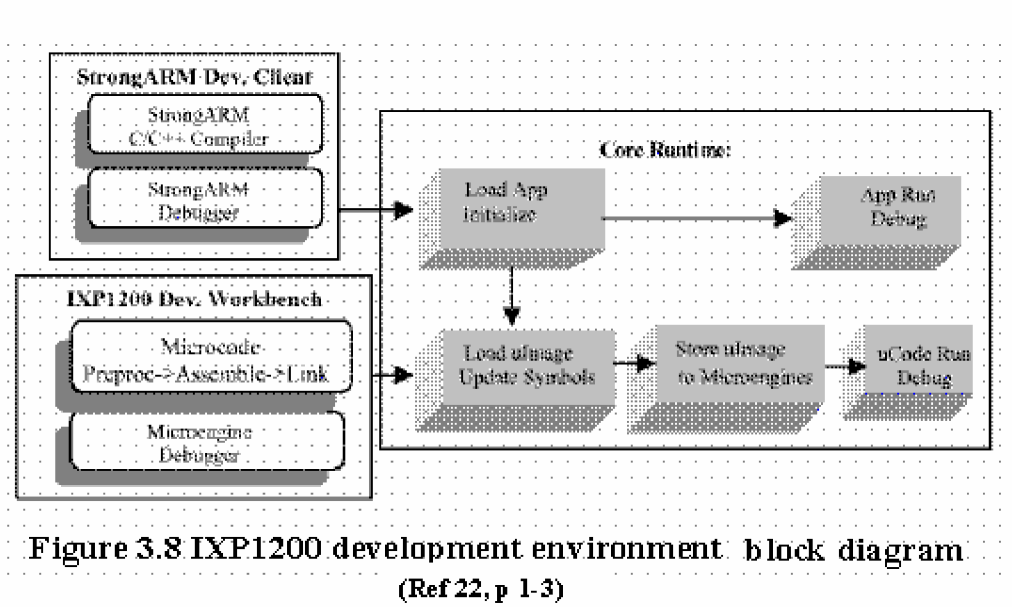


Figure 3.8 IXP1200 development environment block diagram
(Ref 22, p 1-3)

In addition, it does a high-level syntax check. The micro assembler performs low-level syntax checking and branch optimization and assigns symbolic variables to registers while checking that the microcode is legal (the amount of usable registers is limited). Developer Workbench can run the code in a simulation mode or in the hardware if the evaluation kit is present. In the simulation mode, a foreign model can be used. By a foreign model it is meant software modules that can emulate different external hardware devices. A foreign model is an extension to the capabilities of Transactor, which is a cycle and data accurate software model of IXP1200. With a foreign model and the Transactor it is possible to simulate an IXP1200 system very accurately. Figure3.9 shows an example of a simulation session in Developer Workbench. There are typically several windows open while

simulating. In the example, the thread history window shows a timeline presentation of the execution of all the threads running in the Microengines. It indicates different stages of execution in different colors. From that kind of information the programmer can easily see if any piece of code is using plenty of time in waiting for other units to complete, for example. Other information windows contain watches for memory locations and registers, microcode browsers, subroutines or macro definitions and packet buffer status controllers. In conclusion, IXP1200 Developer Workbench offer very sophisticated tools for microcode development.

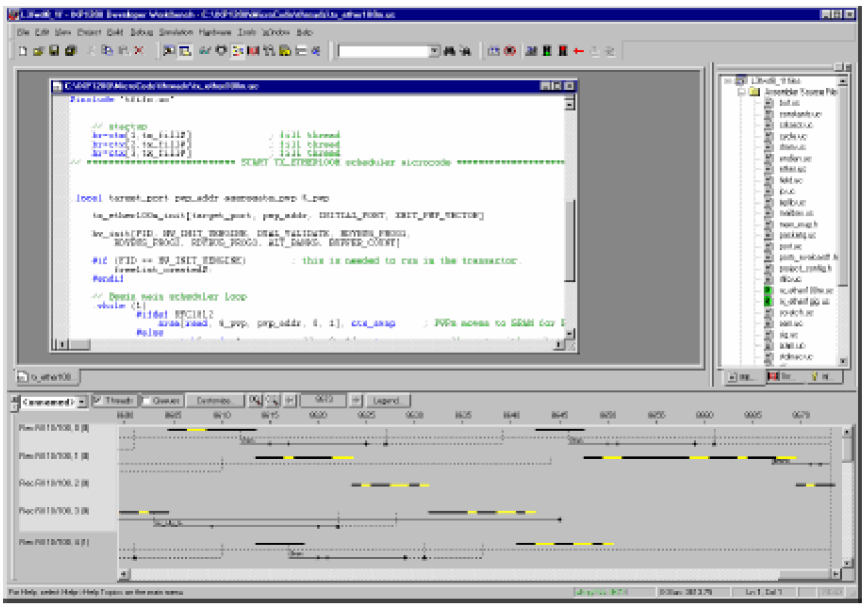


Figure 3.9 IXP1200 Developer Workbench

3.4.6.2 DESIGN PROJECTS

IXP1200 Developer Workbench includes several example microcode design projects. The projects have different port configurations and functioning models. There are projects that work only with 100 Mbps Ethernet ports, projects with only gigabit-port support, and projects that support both types. By function, they are divided in layer 2 bridging and layer

3 routing example designs. Some of the example projects are for simulation only, but most of them can also be used with t he evaluation system. Workbench projects are based on macro building blocks. Small pieces of code are combined into reusable macros. The projects then use the macros to form the programs.

Most of the files in the projects are microcode source files (extension .uc). There are also other files, like header include files (.h) and stream simulator files (.strm). The code is divided for the Microengines depending on the type of ports that are used. A generic method is to divide the threads as receive and transmit threads. Among the transmit threads there are usually some scheduler threads that share the outgoing packets between the actual transmit threads. There is a major difference between gigabit and 10/100 Mbps ports. The processing power of a thread is adequate for 10/100 Mbps ports so that one thread can receive and process the whole incoming data packet with wire speed, and continue with next packet. With gigabit ports, however, data is coming in with such a speed that the same thread cannot receive and process all the packets. Therefore the load is distributed to several threads. The threads read packets and they are assigned identifiers so that the packets can be rearranged when sending them further.

3.5 SYSTEM SETUP

3.5.1 SOFTWARE CONFIGURATION (for WindowsNT/2000 professional Platform)

Windows 2000 Professional or Windows NT (with Service Pack 6) is the supported environment for running the development software on the host computer. The other host requirements are Intel Pentium II 266 MHz processor (or faster), 64 megabytes of RAM, 2 gigabytes of hard disk, a network interface card and a CD-ROM drive. For connecting theIXP12EB console cable, the host computer should have one serial port available. Windows HyperTerminal is used as a terminal emulator to access the console port. The

IXP12EB processor card should be set up according to Hardware Setup in the IXP1200 Ethernet Evaluation System User's Guide (Ref19).

For Linux Software Configuration

The IXP1200 embedded operating system is Linux. The development environment is a combination of the Windows NT operating system and either a Linux platform or Cygwin for a Linux environment on the Windows NT platform, with the following development tools:

- *Microengine tool chain*: Embedded Linux version of the IXP1200 Microengine Development Environment, running under Windows NT. This enables to remotely develop and debug code running on Microengines on the IXP1200.
- *Strong ARM tool chain:* GNU/C++ cross-hosted tool chain. We can invoke the tool chain utilities from a command shell or through the optional Embedded Linux IDE. This enables remotely develop and debug code running on the Strong ARM on the IXP1200.

To boot the IXP1200 Linux from the network, we must set up a trivial file transit protocol (TFTP) server on our Windows NT development machine. For this purpose we should install the TFTP server created a directory called C:\tftp-server.and copied the file tftpd32.exe to C:\tftp-server. Then Created a directory called C:\tftpboot. Copied the files zImage and ramdisk_img.gz double clicking the tftpd32.exe which is in the directory C:\tftp-server folder, it starts the TFTP server. Within the tftpd32 application window, click Settings and changed the base directory to c:\tftpboot.

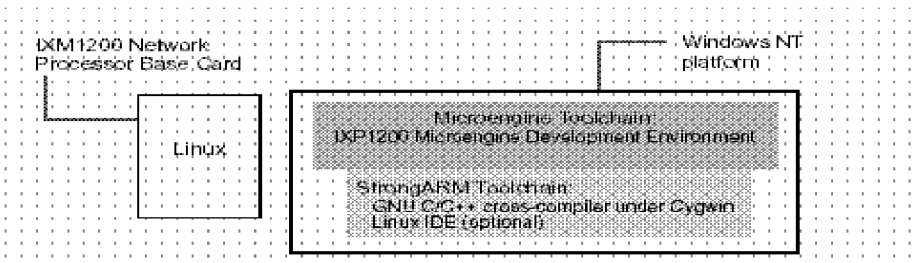


Figure 3.10 Software Configuration for Linux

(Ref 16, p 4)

For Vxworks

The IXP1200 embedded operating system is VxWorks. The RAMdisk and VxWorks kernel are used to load onto the IXP1200 Network Processor Base Card. The tools from Torado Vxworks environment

- *Microengine tool chain*: VxWorks version of the IXP1200 Microengine Development Environment This enables you to remotely develop and debug code running on the IXP1200 Microengines.
- *Microengine C compiler* This allows to use a C-like language instead of structured assembly to write code for the Microengines
- *Strong ARM tool chain*: Wind River Tornado This enables to remotely develop and debug code running on the Strong ARM processor on the IXP1200.

The Tornado installation includes an FTP (File Transfer Protocol) server that is used to load the operating system image to the evaluation system. Any other FTP server software can also be used. The FTP server has to be configured to have a user account so that the evaluation system boot software can download the necessary images from the server. In this case, a user called "ixp" was created with a password "ixpuser". The user's home directory was set to c:\ixp1200\boardsupport\bin\vb\ which is the directory that contains the images when IXP1200 Developer Workbench is installed

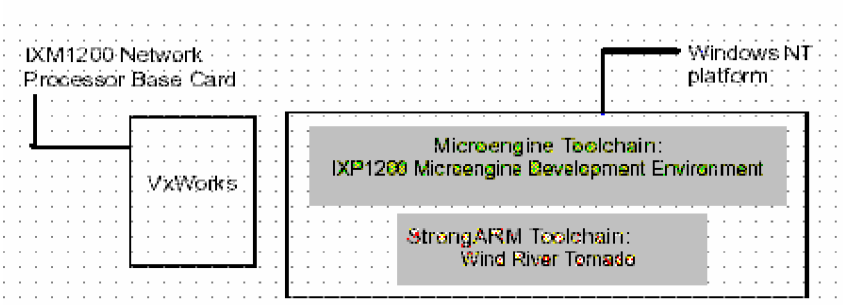


Figure 3.11 Software Configuration (Ref 16, p 5)

3.5.2 HARDWARE AND CABLING CONFIGURATIONS FOR CYGWIN/VXWORKS

- The single-board computer on the Intel IXP1200 Advanced Development Platform with an external keyboard and monitor attached.

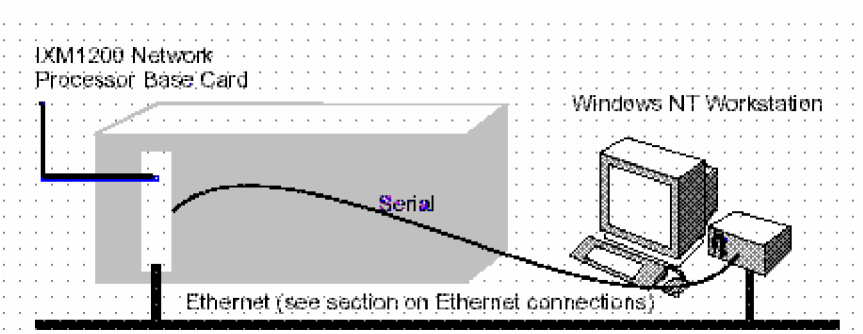


Figure3.12 Hardware Configuration (Ref 17, p 5)

The connections in these configurations include:

- A serial cable from the serial port of the IXP1200 Network Processor Base Card to the com port of the Windows NT platform to provide a remote serial console for the IXP1200 Network Processor Base Card.
- Ethernet connections between the IXP1200 Network Processor Base Card and the Windows NT platform is through a HUB (as next section)
- For remote debugging of the IXP1200 Network Processor Base Card, both configurations can use the Ethernet connection,

3.5.3 ETHERNET CONNECTIONS

For all Ethernet configurations, you can connect any two boards or systems using either:

- A straight Ethernet cable from each system's single Ethernet port to an Ethernet hub or switch.

- An Ethernet cross-over cable that directly connects the two systems' single Ethernet ports. The IXP1200 Network Processor Base Card has only one single Ethernet port, so if we are connecting more than two systems, we must use an Ethernet hub or switch.

3.6 RUNNING APPLICATIONS

3.6.1 STARTING UP LINUX/VXWORKS

Connect the console port of the evaluation system to an available serial port in the host computer, and start the HyperTerminal for that serial port. Correct communication settings are 9600-8-N-1 (9600 bits per second, 8 data bits, no parity, 1 stop bit) with the flow control set to none.

3.6.2 BOOTMGR AND THE BASIC VXWORKS/LINUX CYGWIN LOADER

By default, the system loads BootMgr at the startup. It is a boot manager that can be used to start the basic VxWorks operating system from the flash memory. When the evaluation system is powered up, the auto-boot sequence is shown in the console window. Pressing the space bar will stop the countdown. In the BootMgr prompt, VxWorks can be started with "b 3" command. Cygwin started with b4 option.

For Linux

The following information appears for cygwin Linux kernel

At the cygmon prompt you will need to setup a static ip address for the eval system and also the remote system's ip address by using the following commands:

- s rip ipaddress (remote machine's ip address WindowsNT ip address)

- s lip ipaddress (eval system ip addressIXP1200 address)

For Vxworks

The following information appears for Vxworks kernel

```
BSP IXP Version: 1.3.141, built on Jun 5 2001, 19:18:39
CPU ID: 6901C121
CPU Speed: 200 MHz
Early serial debug initialized
muxLoad failed!
0x
VxWorks System Boot
Copyright 1984-1998 Wind River Systems, Inc.
CPU: Intel ixp1200eb - ARM IXP1200
Version: 5.4
BSP version: 1.2/2
Creation date: Jun 5 2001, 19:18:39
Press any key to stop auto-boot...
```

After pressing a key the [VxWorks Boot] prompt is displayed. Enter command "c" to change the settings:

```
[VxWorks Boot]: c
'.' = clear field; '-' = go to previous field; ^D = quit
boot device : eeE0
processor number : 0
host name : np1200
file name : vxWorks
inet on ethernet (e) : 172.31.5.11
inet on backplane (b):
host inet (h) : 172.31.5.1110
gateway inet (g) :
user (u) : ixp
ftp password (pw) (blank = use rsh): ixpuser
flags (f) : 0x0
target name (tn) : target
startup script (s) :
other (o) :
[VxWorks Boot]:
```

The IP addresses and host names are assigned by the network administrator of the laboratory network. They can be private network addresses, if applicable. Enter the FTP

settings as set up in the FTP server configuration. The full description of the settings an be checked from the Ethernet Evaluation System User's Manual (Ref 19).

3.6.3 LOADING THE IMAGE

For Linux

The linux operating system can be loaded with gl command

Cygmon>gl

Then login prompt appears

At the login prompt type login as root and hit return then password for root is "ixp1200".

At this point the remote shell displays the Linux command-line prompt.

For Vxworks

The Vxworks operating system image can now be loaded with "@" command in the [VxWorks

Boot] prompt:

```
[VxWorks Boot]: @

boot device            : eeE
unit number            : 0
processor number       : 0
host name              : np1200
file name              : vxworks
inet on ethernet (e)   : 172.31.5.11
host inet (h)          : 172.31.5.110
gateway inet (g)       : 172.31.5.1
user (u)               : ixp
ftp password (pw)      : ixpuser
flags (f)              : 0x0
target name (tn)       : target

Attached TCP/IP interface to eeE0.
Attaching network interface lo0... done.
Loading... 712620 + 14184 + 67968
Starting at 0x1000...

BSP IXP Version: 1.3.141, built on Jun  5 2001, 19:18:11
CPU ID: 6901C123
CPU Speed: 200 MHz
Early serial debug initialized
muxDevLoad failed for device entry 0!
Attached TCP/IP interface to eeE unit 0
Attaching network interface lo0... done.
0x7ffe64 (tRootTask): dc0 - Failed to read ethernet address
NFS client support not included.


                    VxWorks

Copyright 1984-1998  Wind River Systems, Inc.

            CPU: Intel ixp1200eb - ARM IXP1200
        VxWorks: 5.4
    BSP version: 1.2/2
  Creation date: Jun  5 2001
            WDB: Ready.
```

The last line indicates that the operating system is now loaded and that the target server can be started from the Tornado system.

3.6.4 TO START NETWORK

For Linux Connecting to the target server

To start the network, run the ifup script:

- Ø $ ifup <local_ipaddr> <subnet_mask> <broadcast> <gateway_ipaddr>
    - local_ipaddr is the static IP address of the IXP1200 network processor
    - subnet_mask is the subnet mask.
    - broadcast is the broadcast address.
    - gateway_ipaddr is the gateway IP address.

Example ifup 10.3.19. <xx> 255.255.255.0 10.3.19.255 10.3.19.251. This also starts the portmapper and the microcode debugger daemons ifup

For vxworks

a) Configuring and Launching the Target Server

The target server is a part of the Wind River Tornado II development environment. It is used to interface the host system with the target system.

To configure the target system started the tornado (*Start -Programs - Tornado2 - Tornado*), and then selected *Target Server - Configure...* from the *Tools* menu. From *Target Server Properties* drop-down list select *Back End.* Target server configuration dialog t settings that we have done are *Target Server Name* and *IP Address*. Selected *Core File and Symbols* from *Target Server Properties* list. Checked *File Path From Target (If Available)*. Selected *Memory Cache Size* from *Target Server Properties* list. As Check *Specify* input value is *8192*. Selected *Console and Redirection* from *Target Server Properties* list. Checked *Redirect Target IO*. The command line in the window is "tgtsvr.exe 172.31.5.11 -n

targert -V -m 8388608 -B wdbrpc -redirectIO". By Clicking on *Launch* the target server is started.

b) Opening the Tornado Shell

The Tornado Shell is a command line interface to the target VxWorks operating system. It is started by selecting the proper target and clicking on Launch Shell. The shell window opened is as shown in the Fig.3.13

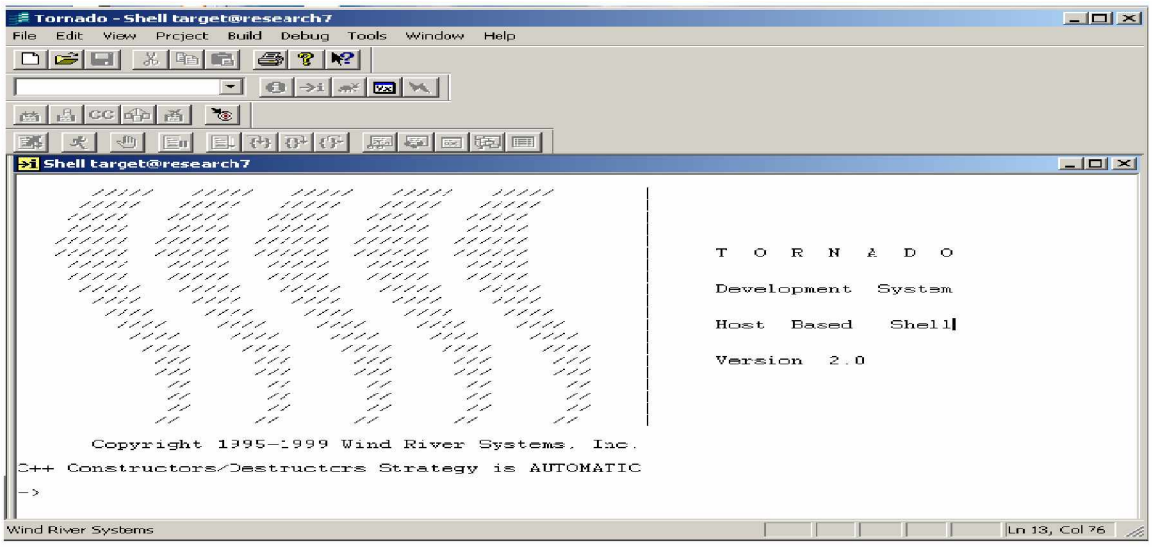


Figure 3.13 Tornado Shell diagram

c) Loading the VxWorks NetApp Image

After launching the target server and Tornado Shell, the VxWorks NetApp image is loaded and initialized in the following way. The NetApp image provides the ability to test the Developer Workbench connection and to run projects. By clicking on the shell window and entering the command “ld < path of the vxworks_gig.o file from vxworks_lib". This

command loads the image from the specified directory. The image is then initialized with command "NetApp_GigInit" (this is case sensitivity command).

d) Setting up the Routing Table

Then we have to give the ip address of the systems which are connected to the network processor. We can set the address using Tornado Shell window .One command to set the routes is route_add (). It is used to set up a configuration, where two computers are Connected to 10/100 Mbps ports of the evaluation system, in ports 0 &7

- route_add ("192.168.3.1", "255.255.255.255", "", 0, 0x0000863c, 0xb408)
- route_add ("192.168.3.2", "255.255.255.255", "", 7, 0x0004762f, 0xb113)

The last parameters are the MAC addresses of the network cards.

3.6.5 COMPILING AND LOADING AN EXAMPLE PROJECT FOR CYGWIN /VXWORKS

The Microengine files are compiled and loaded using IXP1200 Developer Workbench The steps are *File - Open Project...* command in IXP1200 Developer Workbench. Open c:\ixp1200\microcode\workbench_projects directory. L3fwd8_1f project is selected from corresponding directory. Project files have names ending .dwp. L3fwd8_1f is a project based on macro building blocks, supporting Layer 3 routing with eight 10/100 Mbps ports and one gigabit port. When running the project on hardware, it has to be selected from Debug menu. Hardware - Options... menu item is checked so that in Connections page, Connect via Ethernet is selected and the node name is specified as the same as configured while loaded Vxworks. In Ports page, all ports are unchecked.

To compile Strong ARM code

For linux

The steps to be followed are Change the directory where ace files source code exists for example

- change to the directory where the Strong-Arm code files exists
- Use the Linux make file to compile and generate the ace files.
- Copy the configuration file into the directory from where u can mount the files into ixp1200
- The compiled source files also copied into the directory that is mounted into the IXP processor

For Vxworks

Follow the steps as Tornado development tools user guide to compile the Strong ARM code and load the .o file using ld command into the IXP1200board

3.6.6 RUNNING THE PROJECT CYGWIN/VXWORKS.

Stared the debugging session using IXP1200 Developer Workbench with Debug - Start Debugging command (or by pressing F12). Started the running session by pressing F5 (Debug - Run Control - Go). Now the application is running.

For Vxworks

Specified ARP tables using the ARP -s ip address Mac address at two systems which are connected to th IXP.

For Linux

Stared the running session by the command. ./ixstart passing the configuration file as parameter.

CHAPTER 4 DESIGN, IMPLEMENTATION AND RESULTS

4.1 DESIGNING THE APPLICATION

The Intel IXA SDK ACE Programming Framework is used to the design network traffic analysis application. The main steps in designing the Application are

- Ø Defining the MicroACEs and Packet Flow
- Ø Defining the Microblock and Core Component of the Count Micro ACE
- Ø Defining Microblock Groups
- Ø Allocating Microengines

4.1.1 DEFINING THE MICROACES AND PACKET FLOW

The first step in designing an application using the micro ACE programming framework is to define the function of the different ACE's and micro Aces and the packet flow between them. The IXA SDK provides predefined microACEs for input from and output to the network ports. This application uses these two predefined microACEs for handling input and output, and we defined two user-defined micro ACE's Analysis ACE and Exceptional Micro ACE to perform the Analysis and exceptional packet handling functions. The input micro ACE receives packets from the various ports and sends them to the next micro ACE to be Analysis. After Analyzed the packets, the Analysis micro ACE sends the exceptional packets to the Exceptional ACE and normal packets to the output micro ACE to forward the packets into the network through output port.

The lists of Micro ACE's are

1).Input Micro ACE

2).The Output Micro ACE

3).Analysis Micro ACE

4). Exceptional Micro ACE

The packet flow of application is as the following diagram

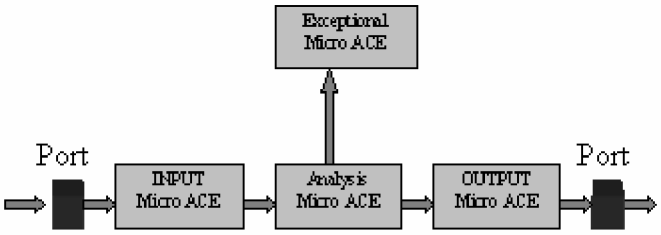


Figure 4.1 Packet Flow Diagram

4.1.2 DEFINING THE MICRO BLOCK AND CORE COMPONENT OF THE ANALYSIS MICRO ACE

Each micro ACE functionality in the application is divided in to two components one is microblock component functionality which runs on the microengines and another is core component functionality which runs on the core processor.

The input and output microACEs are pre-defined microACEs which are provided as part of the IXA SDK. Like all microACEs, they have a micro block and a core component which controls configuration information using Resource Manager cross-calls. The core components of the microACEs are bind to mirror the packet flow of the micro blocks.

User-Defined micro ACE's are the Analysis Micro ACE, the Exceptional Micro ACE. The Analysis micro ACE has the following functions:

- Ø To count the number of packets which are input for each port
- Ø To display the packet count to on monitor.
- Ø To perform packet analysis based on IP Frame format, TCP and Ethernet frame format
- Ø To perform exception handling

Micro block of the Analysis micro ACE to maintain the count of packets received for each port and it checks the packet header to see if the packet is TCP or UDP or IP or ICMP packets and if it is an ICMP packet, it marks the packet as an exception packet and routes it to the core component of the Analysis micro ACE. The core component of Micro ACE will do the printing function. In addition, the core component receives ICMP packets from the analysis micro block and then sends the packet to its default target. In this application, the default target for the Analysis micro ACE core component is the Exceptional Micro ACE. Exceptional micro ACE is not bound to any other target ACE so, by definition, the packets are dropped. The micro block running on the IXP1200 Microengines, maintains the packet count in shared scratch memory. The core component accesses the shared scratch memory using the Resource Manager. The Resource Manager is system software that provides resources to use in managing the Microengine memory and shared state of the micro block and core component of a micro ACE.

The following diagram shows t micro block components and core components and the packet flow between the components.

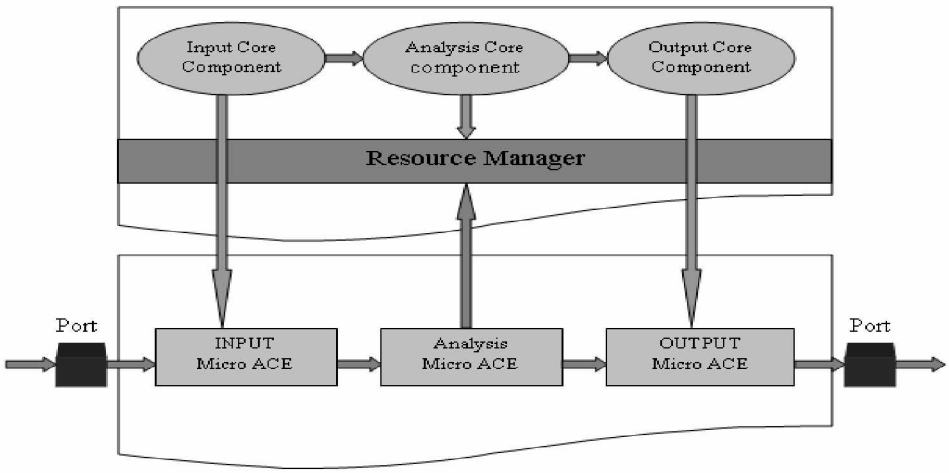


Figure 4.2 Software Components of The Application

4.1.3 DEFINING MICROBLOCK GROUPS

A microblock group consists of one or more microblocks which run on a single microengine. For each group microcode *dispatch loop is used* to implement the packet flow within the group. The dispatch loop configures the microblocks in the group into a processing pipeline. The microblocks and dispatch loop are compiled into a single microcode image (.uof) file which is downloaded onto the microengines to run the application In this application, we define two microblock groups. The first microblock group includes the Input micro ACE and the Analysis micro ACE. The second microblock group contains only the Output micro ACE.

The following diagram illustrates the packet flow through the microblock groups.

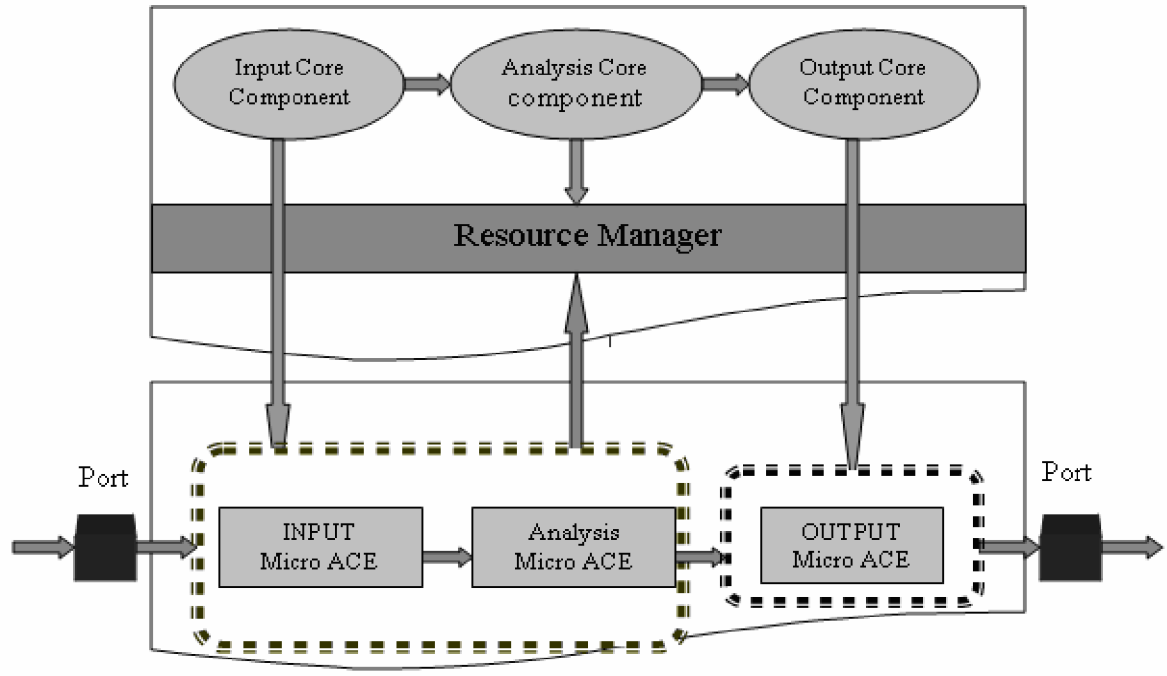


Figure 4.3 Micro Block Structure Diagram

4.1.4 ALLOCATING MICROENGINES

A single microcode image can be loaded onto more than one Microengine… Each microengine runs four threads. Each 10/100 port requires one thread, so each Microengine can support four ports of this type. Each gigabit port requires eight threads, so it requires two Microengines to support a port of this type.

This IXP Evaluation Board consists of eight 10/100 ports and one gigabit port. The First micro block (Input Micro ACE and Analysis micro ACE) image is loaded on four Microengines. The second micro block group (Output Micro block) image is loaded on the other two microengines.

4.1.5 DETAILED DESIGN OF THE APPLICATION

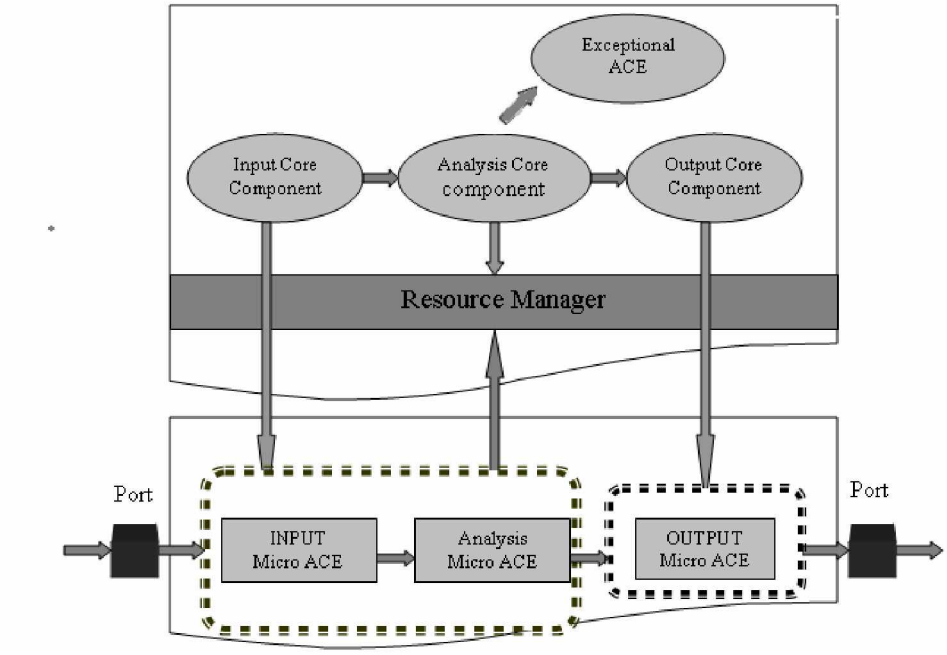


Figure 4.4 Complete Design of The Application

4.2 SOURCE CODE FILES OF THE APPLICATION

The source code files that are developed to implement this application include the sources or all of the micro ACE's.

1. The Workbench project file's
2. Microcode source files for the Analysis ACE microblock
3. The dispatch loop for the microblock group containing the Input ACE microblock and Analysis ACE microblock
4. Micro code source files for Analysis ACE Micro Block
5. System macro Library files.
6. C source files for the Analysis ACE core component
7. C source files for the Exceptional ACE's.
8. Startup and configuration scripts and

4.2.1 DEVELOPER WORKBENCH PROJECT FILES

The IXP1200 Microengine Development Environment Workbench defines projects to manage the microcode source files and build parameters for micro ACEs. A project consists of one or more IXP1200 network processor chips, microcode and C language source files, debug script files, and assembler and linker settings used to build the microcode image files. This project configuration information is maintained in a Workbench project file with the .dwp extension.

The source files developed and included using this environment is Application's project file is Analysis.dwp**.** The source files developed and included from the Microsoft SDK are Predefined Input micro block from SDK(.uc file), predefined output micro block from SDK (.uc file) , Dispatch loop for Input and Analysis micro ACE's , Dispatch loop for Output micro ACE , and Analysis micro block and included SDK library files.

4.2.2 MICROCODE SOURCE FILES

Developed all microcode files using the IXP1200 Microengine Development Environment Development Workbench. The source files developed in this environment three types of Microcode source files

- Ø Dispatch loop source files
- Ø MicroACE microblock source and header files
- Ø System macro library source files

4.2.3 DISPATCH LOOP SOURCES

For each microblock group a dispatch loop is developed .In our application we have two micro block groups. The first group consists Input Micro ACE micro block and Analysis micro ACE micro block. .The Second group is only Output Micro ACE micro block.. Developed one dispatch loop for the first group and no need to develop dispatch loop for the second since there is no packet flow (one micro block).

The source file is **Analysis_IngressDispatch.uc**

4.2.4 MICRO ACE MICRO BLOCK SOURCES

Microcode source files define the micro ACE's micro block. The Micro code source files for Analysis Micro ACE micro block are **Analysis.uc** and the macros are defined in separate file is **Analysis_internal_h.uc.**

Header files define imported variables that are used in the microcode files. **AnalysisControlBlock.h** Defines the Analysis control block data structure. This file is also included by the C source files which define the core component of the micro ACE

Analysis_ImportVars.h Declares the imported linker variables used in the Analysis transform microblock. This file is also included by the C source files which define the core component of the micro ACE

4.2.5 SYSTEM MACRO LIBRARY SOURCES

The system header files are included in the microcode source file Analysis.uc. These source files define macros commonly used in the IXA API. The following table shows the macros defined in these files that are used in Analysis.uc.

FILE NAME	DESCRIPTION
Buf_h.uc	Packet buffer macros for microACEs
Debug_h.uc	Debugging macros
endian.uc	IXP block byte-order macros
field.uc	IXP block byte-order field extract, compare, and branch macros
ip.uc	Macros to verify, modify and extract fields from IP headers
stdmac.uc	Standard library macros
xbuf.uc	Macros for ethernet protocol header

Table 4.1 Library Source Files

4.2.6 C SOURCE FILES CORE COMPONENTS

The core component of the Analysis micro ACE is developed in C-language. These files

can edit using text editor, including the Developer's Workbench (although they are not part of the Workbench project). We are used the GNU tool chain in the development environment to compile, and debug these files. In Windows NT development system, we can access these Linux GNU tools through Cygwin.

The source and header files that make up the Analysis micro ACE core component are **init.c, action.c** and **AnalysisMicroACE.h.**

To initialization and termination of the Analysis core component init.c is defined. And the functionality of Analysis core component is defined in action.c the declaration of data structure which are used by the init.c and action.c are defined in AnalysisMicroACE.h

4.2.7 MANAGEMENT SCRIPTS

Management Scripts are used to start the system software and the application ACEs automatically as part of the boot procedure for the IXP1200 platform. System startup script (**ixstart.sh**) starts the system software and calls the system configuration scripts (**ixsys.config.sh**)

4.3 COMPILING AND RUNNING THE APPLICATION

Windows NT Development Environment is used to develop, compile and run the programs. The steps to compile and run the analysis application are section includes the following topics:

- Compiling the Microcode Source Files
- Compiling the Core Component of the Micro ACE
- Starting and Running the Count Application

4.3.1 COMPILING THE MICROCODE SOURCE FILES

The microcode source files can be compiled using Developer Workbench and the extension of compiled object files are .uof

Using the IXP1200 Microengine Development Environment for Building and Downloading Microcode

Develop and Open the Analysis project (Analysis dwp). The Project include all micro code blocks for Input Microblock , Analysis Microblock, Output Microblock , dispatch loop and all header files library files which are used by microblock files. Used build option from tools menu to compile all the microcode source files and from the options of hardware menu IXP1200 ip address is entered to connect the IXP1200.

4.3.2 COMPILING THE CORE COMPONENTS OF THE MICRO ACE

Use the following steps to compile and link the core component of the Analysis micro ACE. . Used linux environment in the Windows NT system using Cygwin Bash Shell Stared the Cygwin bash shell and followed the steps given the IXA SDK from Tutorial. The compiled core components give the **AnalasysACE** image. For the input and out put SDK provided ACE files are used in this application.

4.3.3 RUNNING THE APPLICATION

The Steps followed to start the Analysis application are

1. Booted the IXP1200 from Windows NT computer,
   Program->Accessories->HyperTerminal->HyperTerminal

2. Started the network using ifup script command.
   $ ifup *local_ipaddr subnet_mask broadcast gateway_ipaddr*
   local_ipaddr is the static IP address of the IXP1200.
   subnet_mask is the subnet mask.
   broadcast is the broadcast address.
   gateway_ipaddr is the gateway IP address.
   This script also starts the port mapper and microcode debugger daemons.
3. Mounted the directory where the compiled Micro Ace files and system configuration files exist.
   mount -t nfs <NT IP address>: compiled ACE's and configuration files /mnt (/mnt is the directory created in the IXP1200 Board)
5. In the cd /mnt directory started the running session by the command
   /ixstart ixsys_Analysis.config
7. Used Iperf to generate packets from port 0.

4.4. RESULTS

The Results are taken when the 100Mbps Input port is in promiscuous mode .we are Send different types of data to 100Mbpst Ethernet port 0. The analysis function classifies the data depending on the data type ARP traffic, UDP traffic, Web Traffic , SSH traffic over TCP over IP and NON-IP traffic and forwards the data to different ports depending on the type (ARP traffic to port 1, UDP/IP port 2, TCP/IP port3, SSH traffic port4, Web traffic port 5)

In the first test case we send 100 packets through port 0. And they are forwarded to port 0 and 1 here no classification is implemented. In the figure the no of packets are displayed of each port and the packets count. In the second test case we send packets through port 0.and here we implemented the Analysis function based on rules such as ARP, UDP/IP, TCP/IP,

SSH and HTTP Web traffic. In this case the classified packets are sent through different ports as shown in figure below.

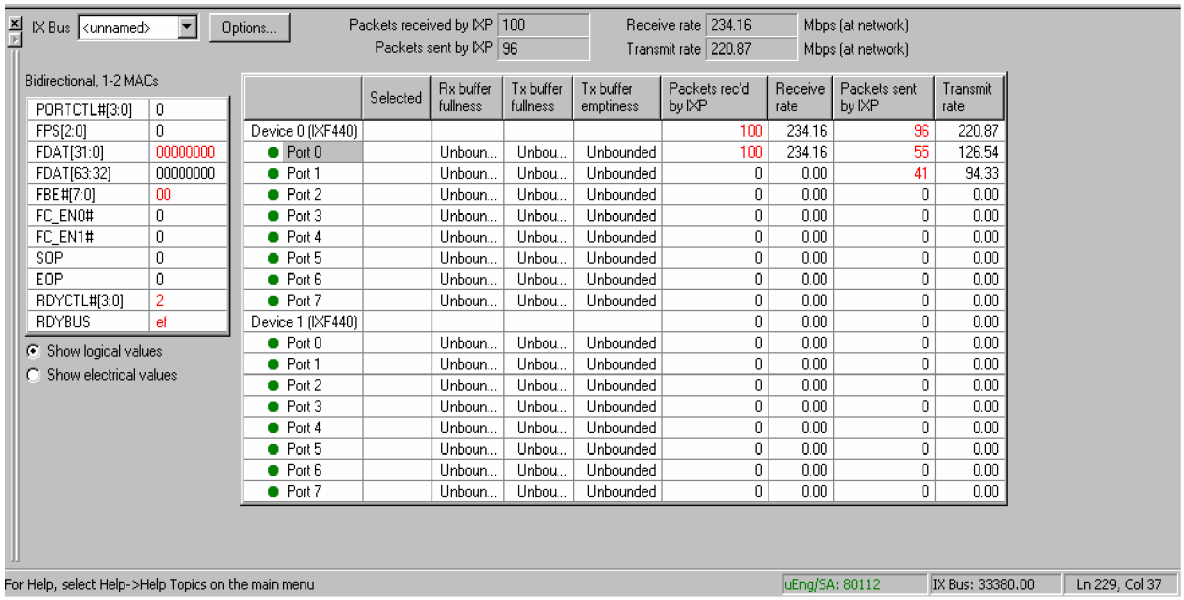


Figure 4.5 Packet flow without Analysis function

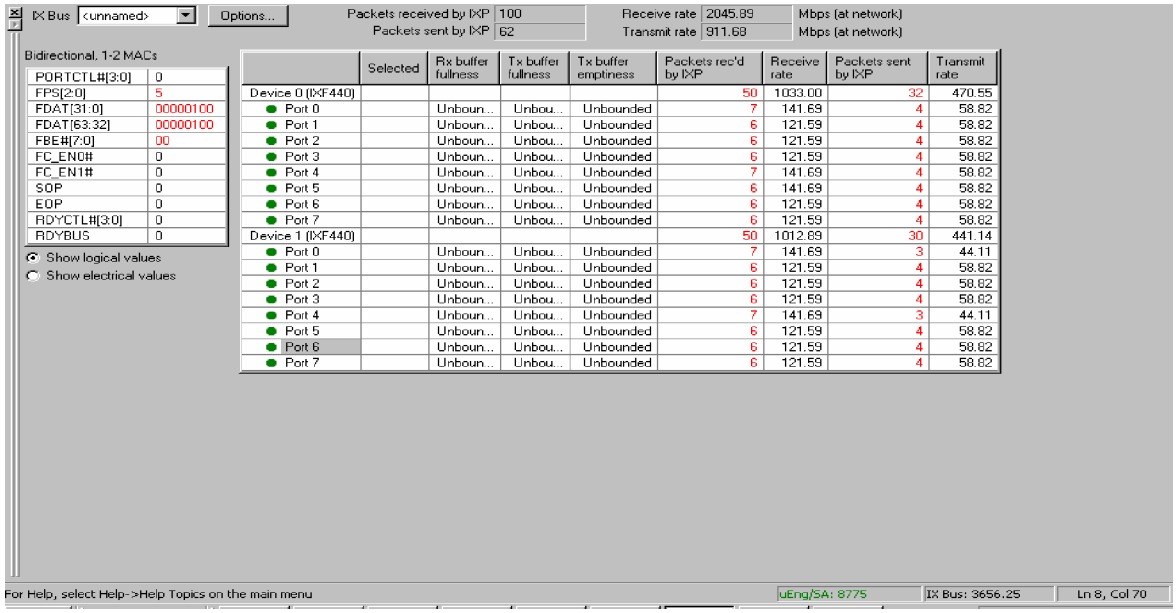


Figure 4.6 Packet flow with Analysis function

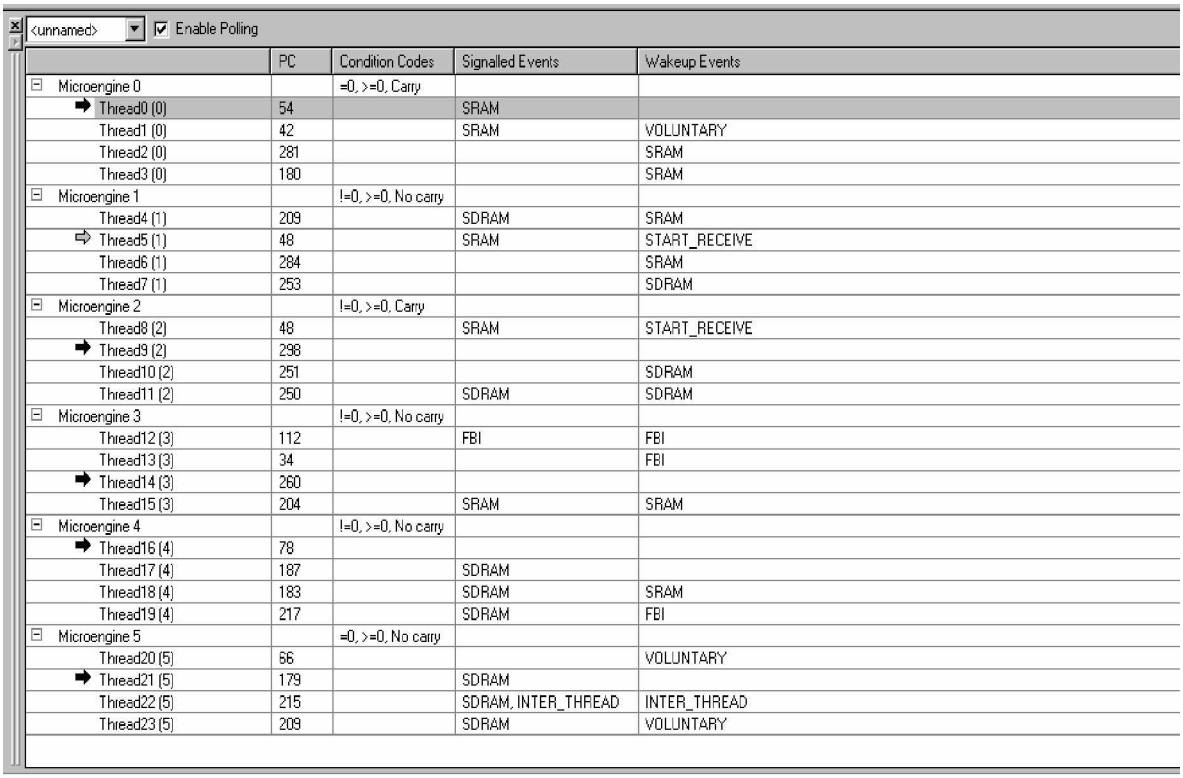
Comparison: From the two figures the packet drop rate is 4% without any analysis functions. And with analysis it is forwarded only 62 packets from the below Figure.

And the number packets are displayed on the HyperTerminal (IXP command processor) window are like Packets on port 1 = 4, Packets on port 2 = 4, Packets on port 3 = 4 Packets on port 4 = 4, Packets on port 5 = 4.

We can see Thread Status of each microengine as in following figure and the difference without analysis and with analysis function.

	PC	Condition Codes	Signalled Events	Wakeup Events
Microengine 0		=0, >=0, Carry		
Thread0 (0)	54		SRAM	
Thread1 (0)	42		SRAM	VOLUNTARY
Thread2 (0)	281			SRAM
Thread3 (0)	180			SRAM
Microengine 1		!=0, >=0, No carry		
Thread4 (1)	209		SDRAM	SRAM
Thread5 (1)	48		SRAM	START_RECEIVE
Thread6 (1)	284			SRAM
Thread7 (1)	253			SDRAM
Microengine 2		!=0, >=0, Carry		
Thread8 (2)	48		SRAM	START_RECEIVE
Thread9 (2)	298			
Thread10 (2)	251			SDRAM
Thread11 (2)	250		SDRAM	SDRAM
Microengine 3		!=0, >=0, No carry		
Thread12 (3)	112		FBI	FBI
Thread13 (3)	34			FBI
Thread14 (3)	260			
Thread15 (3)	204		SRAM	SRAM
Microengine 4		!=0, >=0, No carry		
Thread16 (4)	78			
Thread17 (4)	187		SDRAM	
Thread18 (4)	183		SDRAM	SRAM
Thread19 (4)	217		SDRAM	FBI
Microengine 5		=0, >=0, No carry		
Thread20 (5)	66			VOLUNTARY
Thread21 (5)	179		SDRAM	
Thread22 (5)	215		SDRAM, INTER_THREAD	INTER_THREAD
Thread23 (5)	209		SDRAM	VOLUNTARY

Figure 4.7 Thread Status without Analysis function



	PC	Condition Codes	Signalled Events	Wakeup Events
Microengine 0		=0, >=0, Carry		
Thread0 (0)	54		SRAM	
Thread1 (0)	42		SRAM	VOLUNTARY
Thread2 (0)	281			SRAM
Thread3 (0)	180			SRAM
Microengine 1		!=0, >=0, No carry		
Thread4 (1)	209		SDRAM	SRAM
Thread5 (1)	48		SRAM	START_RECEIVE
Thread6 (1)	284			SRAM
Thread7 (1)	253			SDRAM
Microengine 2		!=0, >=0, Carry		
Thread8 (2)	48		SRAM	START_RECEIVE
Thread9 (2)	298			
Thread10 (2)	251			SDRAM
Thread11 (2)	250		SDRAM	SDRAM
Microengine 3		!=0, >=0, No carry		
Thread12 (3)	112		FBI	FBI
Thread13 (3)	34			FBI
Thread14 (3)	260			
Thread15 (3)	204		SRAM	SRAM
Microengine 4		!=0, >=0, No carry		
Thread16 (4)	78			
Thread17 (4)	187		SDRAM	
Thread18 (4)	183		SDRAM	SRAM
Thread19 (4)	217		SDRAM	FBI
Microengine 5		=0, >=0, No carry		
Thread20 (5)	66			VOLUNTARY
Thread21 (5)	179		SDRAM	
Thread22 (5)	215		SDRAM, INTER_THREAD	INTER_THREAD
Thread23 (5)	209		SDRAM	VOLUNTARY

Figure 4.8 Thread Status with Analysis function

CHAPTER 5 CONCLUSION

5.1 CONCLUSION

- Working on the IXP kits gave insight to the fact that there is vast potential in network processor technology for real-time traffic monitoring purposes that is essential for applications such as intrusion detection
- With the help of customized hardware and a highly parallel architecture enable support for a wide variety of network services at gigabit line speeds.
- The programmability of these processors results in a high degree of application flexibility and makes them ideal for purposes of network measurement.
- Network Processors have a complex architecture with multiple processing engines and multiple threads per processing engine. With this we can achieve packet processing at sustained gigabit data rates which is not possible using general-purpose processors.
- Developing Flexible Network Traffic Analysis system Using Network Processors is an ideal application.

5.2 FUTURE SCOPE

Network processor architectures make s possible to develop and implement Wire sped network applications .Next generation network processors are have improvements in Architecture (Functional and context Pipelining, Multithreading) and Improvements in Hardware(Next neighbor registers, Distributed Cache (embedded memory), No of Microengines) These Architectural improvements makes possible to implement advanced network traffic analysis techniques like deep packet inspection Passive fingerprinting and Forensics analysis with wire speed.

REFERENCES

BOOKS

[1]. Douglas E. Comer Network Systems Design using Network Processors Pearson Education 2003.

[2]. Johnson, E. and A. Kunze IXP1200 Programming: The Microengine Coding Guide for the Intel IXP1200 Network Processor Family, Intel Press.

[3]. Richard Stevens W, TCP/IP Illustrated, Volume 1: The Protocols, Addison Wesley, Massachusetts, 1994.

[4]. Richard Stevens W, TCP/IP Illustrated, Volume 2: The Implementations, Addison-Wesley, Massachusetts, 1995.

REPORTS

[5]. NirajShah, "Understanding Network Processors"; version 1.0, University of California, Berkeley, September 2001.

[6]. T. Spalink, S. Karlin, and L. Peterson. Evaluating Network Processors in IP Forwarding. Technical report, Computer Science dep, Princeton University, Nov 15, 2000.

WHITE PAPERS

[7]. "7layers packet processing A Performance Analysis" White Paper Ezchp technologies.

[8]. "Building Next Generation Network Processors" April 2001 White Paper Agere Systems Proprietary.

[9]. "The Role of Memory in NPU system design White Paper" Ezchip Technologies.

[10]. "Using the Intel IXP1200 network processor to optimize packet processing application development" White paper radisys.

CONFERENCE AND WORKSHOP PROCEEDINGS

[11]. Charitakis I, D Pnevrnatikatos, E. Markatos and K. Anagnostakis Code Generation for Packet Header Intrusion Analysis on the IXP1200 Network Processor 7th International Workshop on Software and compilers for Embedded Systems (SCOPES 2003), Vienna, Austria, September 2003.

[12]. Niraj Shah, William Plishker, Kurt Keutzer NP-Click: A Programming Model for the Intel IXP1200 *Appears in 2nd Workshop on Network Processors (NP-2), 9th International Symposium on High Performance Computer Architectures (HPCA), Feb 2003*

[13]. Tammo Spalink, Scott Karlin, Larry Peterson and Yitzchak Gottlieb: "Building a robust software-based router using Network Processor", 2001.

MANUALS AND OTHERS

[14]. Intel [2001a], Intel IXA SDK ACE Programming Framework: IXA SDK2.0 Reference, Intel Corporation Part no A71582-001.

[15]. Intel [2001b], Intel IXA SDK ACE Programming Framework: IXA SDK2.0 Developer's Guide, Intel Corporation part no: A71582-001.

[16]. Intel [2001], Intel IXA SDK for the IXP1200 Network Processor Family: IXA SDK Getting started, Intel Corporation.

[17]. Intel [2001], Intel IXA SDK for the IXP1200 Network Processor Family: IXA SDK Installation and setup Guide, Intel Corporation.

[18]. Intel [2001], IXP1200 Development Tools User's Guide, Intel Corporation. Part no 278302-007.

[19]. Intel [2001], Intel IXP1200 Network Processor Family IXP1200 Ethernet Evaluation System User's Manual, , Intel Corporation, part no A14987-007.

[20]. Intel [2001c], IXP1200 Network Processor Family: Hardware Reference Manual", Intel Corporation part no 278303-008.

[21]. Intel [2001d] Intel IXP1200 Network Processor Family: Microcode Programmers Reference Manual, Intel Corporation, Part no278303-009

[22]. Intel [2001], IXP1200 Network Processor Family: Microcode Software Reference Manual,, Intel Corporation, part no 278306-007.

[23]. Intel [2001], Intel Microengine C Compiler Language Support Reference Manual,, Intel Corporation.

[24]. Intel [2001] Intel Microengine C Networking Library for the IXP1200 Network Processor Reference Guide,, Intel Corporation.

[25]. Tornado User's Guide, 2.0 (Windows Version), Edition 1, 9 Apr 1999, part no DOC-12612-8D-01, Wind River Systems Inc.

INTERNET RELATED LINKS

[26]. Intel IXP1200 Network Processor Family, Intel Corporation, http://developer.intel.com/design/network/products/npfamily/ixp1200.htm, 10th September 2001.

[27]. "IXP1200 Linux How To", Intel corporation http://www.liacs.nl/~herbertb /projects/studprojects/tnguyen/IXP1200_Linux.htm.

[28]. IXP1200 mailing list https://lists.cs.princeton.edu/mailman/listinfo/ixp1200.

[29]. Strong-ARM Microengine interaction lab http://www-2.cs.cmu.edu/afs /cs.cmu.edu/project/cmcl-prs/www/cap00/str_ueng//str-ueng-interaction.htm.

[30]. The homepage of Wind River Systems Inc., http://www.windriver.com/, 28 October 2001.

[31]. Vxwork/TornadoII faqs http://www.xs4all.nl/~borkhuis/vxworks/vxworks.html

PAPERS COMMUNICATED /ACCEPTED

[1]. G. Srinivas and Maninder Singh “An Efficient & Flexible approach to Network Traffic analysis using IXP1200 Network Processor” Paper No 10 in BVCON 2005, Proc. National Conference on “Cyber Crime: Past, Present and Future” held in Institute of Management and Rural Development Administration, Sangali.