10-01-02 A

JC970 U.S. PTO 09/27/60

UTILITY PATENT APPLICATION TRANSMITTAL (New Nonprovisional Applications Under 37 CFR § 1.53(b))	Attorney Docket No. ALA-006E

JC970 U.S. PTO 10/260878 09/27/02

TO THE COMMISSIONER FOR PATENTS:

Transmitted herewith is a patent application identified as follows:
First-named inventor: Laurence B. Boucher
Assignee: Alacritech, Inc.
Filing Date: September 27, 2002
Title: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

This application claims the benefit under 35 USC §120 (prior application not abandoned) of:

Prior Application: "Fast-Path Apparatus For Receiving Data Corresponding to a TCP Connection"
Serial No.: 10/092,967
Filing Date: March 6, 2002
Atty. Docket: ALA-006C
Examiner: Zarni Maung

This application claims the benefit under 35 USC §120 of Application Serial No. 10/092,967, filed March 6, 2002, which in turn claims the benefit under 35 USC §120 of Application Serial No. 10/023,240, filed December 15, 2001, which in turn claims the benefit under 35 USC §120 of Application Serial No. 09/464,283, filed December 15, 1999, which in turn claims the benefit under 35 USC §120 of Application Serial No. 09/439,603, filed November 12, 1999, which in turn claims the benefit under 35 USC §120 of Application Serial No. 09/067,544, filed April 27, 1998, which in turn claims the benefit under 35 USC §119 of Provisional Application Serial No. 60/061,809, filed October 14, 1997.

This application also claims the benefit under 35 USC §120 of Application Serial No. 09/384,792, filed August 27, 1999, which in turn claims the benefit under 35 USC §120 of Application Serial No. 09/141,713, filed August 28, 1998, which in turn claims the benefit under 35 USC §119 of Provisional Application Serial No. 60/098,296, filed August 27, 1998.

This application also claims the benefit under 35 U.S.C. §120 of the following:
U.S. Patent Application Serial No. 09/416,925 (ALA-005), filed October 13, 1999;
U.S. Patent Application Serial No. 09/514,425 (ALA-007), filed February 28, 2000;
U.S. Patent Application Serial No. 09/675,484 (ALA-010A), filed September 29, 2000;
U.S. Patent Application Serial No. 09/675,700 (ALA-010B), filed September 29, 2000;
U.S. Patent Application Serial No. 09/789,366 (ALA-013), filed February 20, 2001;
U.S. Patent Application Serial No. 09/801,488 (ALA-011), filed March 7, 2001;
U.S. Patent Application Serial No. 09/802,551 (ALA-012), filed March 9, 2001;
U.S. Patent Application Serial No. 09/802,426 (ALA-014), filed March 9, 2001;
U.S. Patent Application Serial No. 09/802,550 (ALA-015), filed March 9, 2001;
U.S. Patent Application Serial No. 09/855,979 (ALA-016), filed March 14, 2001; and
U.S. Patent Application Serial No. 09/970,124 (ALA-020), filed October 2, 2001.

(X) The specification contains a statement claiming priority under 35 USC § 120 and claiming the benefit under 35 U.S.C. §119.
(X) The entire disclosure of each of the prior applications (10/092,967; 10/023,240; 09/464,283; 09/439,603; 09/067,544; 09/384,792; 09/141,713; 09/416,925; 09/514,425; 09/675,484; 09/675,700; 09/789,366; 09/801,488; 09/802,551; 09/802,426; 09/802,550; 09/855,979; 09/970,124) is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.
(X) The entire disclosure of each of the prior provisional applications (60/061,809; 60/098,296) is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 001

Enclosed are:

2	pages Application Transmittal Letter
145	pages Specification
5	pages Claims
1	page Abstract
89	pages Drawings
4	pages Declaration/Power of Attorney from prior application 10/092,967 (signed - copy)
4	pages Declaration/Power of Attorney from prior application 10/092,967 (signed - copy)
2	page CD Appendix Transmittal Letter
X	CD Appendix (two copies)
1	page Terminal Disclaimer Over A Prior Patent
X	A check for filing fee ($ 922.00)
X	Return Receipt Postcard

Newly Executed Declaration Not Required:

A newly executed declaration is not filed in this application because, under 37 CFR 1.63(d)(1), a newly executed declaration is not required because: the prior application contained a declaration as prescribed by 37 CFR 1.63; the continuation application (this application) is filed by all of the inventors named in the prior application; the specification and drawings in the continuation application (this application) contain no matter that would have been new matter in the prior application; and a copy of the executed declaration (there were two) in the prior application is being submitted in the continuation application (this application).

The filing fee is calculated as follows:

CLAIMS AS FILED				
FOR	NO. FILED	NO. EXTRA	RATE	FEE
Total Claims	24	4	$18.00	$ 72.00
Independent Claims	3	0	$84.00	$ 0.00
Multiple Dependent Claims (if applicable)				$0.00
Assignment Recording Fee				$0.00
Terminal Disclaimer Fee (37 CFR 1.20(d))				$110.00
Basic Filing Fee				$740.00
			Total Filing Fee	$922.00

I hereby certify that this is being deposited with the U.S. Postal Service "Express Mail Post Office to Addressee" service under 37 CFR § 1.10 on the date indicated below and is addressed to:

Box Patent Application
Assistant Commissioner for Patents
Washington, D.C. 20231

By: _____

Typed Name: Mark Lauer

Express Mail Label No.: EL928548779US.

Date of Deposit: 9-27-02

Respectfully submitted,

By: _____

Mark Lauer
Attorney for Applicants
Reg. No. 36,578
Customer No. 24,501

Date: 9-27-02

Correspondence Address:
Mark Lauer, Patent Attorney
7041 Koll Center Parkway, Suite 280
Pleasanton, California 94566
Phone: (925) 484-9295
Fax: (925) 484-9291

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 002

TO THE ASSISTANT COMMISSIONER FOR PATENTS:

Inventors:	Laurence B. Boucher, et al.	Atty Docket:	ALA-006E
Filing Date:	September 27, 2002	Serial No.:	Unknown

Title: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

Compact Disk Transmittal Letter per 37 CFR 1.52(e)3(ii))

Sir:

Transmitted herewith are:

Two Labeled Compact Discs – Recordable (CD-R) – "Copy 1" and "Copy 2," each in a CD case and contained in a padded envelope.

The content on the two discs is identical
The machine format is: IBM-PC
The operating system is: MS-Windows
The creation date of the CDs is: September 26, 2002
The name, date and size of the files on the CDs are listed below:

There are three folders on each disc: 1) CD Appendix A,
2) CD Appendix B, and
3) CD Appendix C.

Folder Appendix A contains two files:

CD Appendix A Title Page.txt. Its size is 370 bytes. It was created 9/26/02.

Rcv.v. Its size is 84.4KB. It was created 1/7/99.

Folder Appenidix B contains two files:

CD Appendix B Title Page.txt. Its size is 495 bytes. It was created 9/26/02.

Microcode.txt. Its size is 105 KB. It was created 10/1/99.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 003

Folder Appendix C contains three files:

CD Appendix C Title Page.txt. Its size is 416 bytes. It was created 9/26/02.

atcpsource.wrd.txt. Its size is 778 KB. It was created (written to disc) 2/19/02.

simbasource.wrd.txt. Its size is 262 KB. It was created (written to disc) 2/19/02.

Respectfully submitted,

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as Express Mail Label No. EL928365779US in an envelope addressed to: Box PATENT APPLICATION, Assistant Commissioner for Patents, Washington, D.C. 20231, on September 27, 2002.

Date: 9-27-02

Mark Lauer

Mark Lauer
Reg. No. 36,578
7041 Koll Center Parkway
Suite 280
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 004

ALA-006E

TERMINAL DISCLAIMER OVER A PRIOR PATENT

10/03/2002 DTESSEM1 00000026 10260878
03 FC:148
110.00 OP

In re Application of: Laurence B. Boucher et al.

Application No.: Unknown

Filed: September 27, 2002

Title: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

Express Mail No.: EL928365779US

The owner, Alacritech, Inc., of a one hundred percent interest in the instant application hereby disclaims, except as provided below, the terminal part of the statutory term of any patent granted on the instant application, which would extend beyond the expiration date of the full statutory term defined in 35 U.S.C. 154 to 156 and 173, as presently shortened by any terminal disclaimer, of prior U.S. Patent Nos. 6,226,680 and 6,247,060. The owner hereby agrees that any patent so granted on the instant application shall be enforceable only for and during such period that it and the prior patents are commonly owned. This agreement runs with any patent granted on the instant application and is binding upon the grantee, its successors or assigns.

In making the above disclaimer, the owner does not disclaim the terminal part of any patent granted on the instant application that would extend to the expiration date of the full statutory term as defined in 35 U.S.C. 154 to 156 and 173 of the prior patents, as presently shortened by any terminal disclaimer, in the event that they later: expire for failure to pay a maintenance fee, are held unenforceable, are found invalid by a court of competent jurisdiction, are statutorily disclaimed in whole or terminally disclaimed under 37 CFR 1.321, have all claims canceled by a reexamination certificate, are reissued, or are in any manner terminated prior to the expiration of its full statutory term as presently shortened by any terminal disclaimer.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

The undersigned is an attorney or agent of record.

Date: 9-27-02

Mark Lauer
Registration No. 36,578

The terminal disclaimer fee under 37 CFR 1.20(d) is included.

[redacted]

ALA-006E

FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

Laurence B. Boucher
Stephen E. J. Blightman
Peter K. Craft
David A. Higgen
Clive M. Philbrick
Daryl D. Starr

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 10/092,967, entitled "FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION," filed March 6, 2002, by Laurence B. Boucher et al., which in turn claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 10/023,240 (Attorney Docket No. ALA-006A), entitled "TRANSMIT FAST-PATH PROCESSING ON TCP/IP OFFLOAD NETWORK INTERFACE DEVICE," filed December 15, 2001, by Laurence B. Boucher et al., which in turn claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/464,283 (Attorney Docket No. ALA-006), entitled "INTELLIGENT NETWORK INTERFACE DEVICE AND SYSTEM FOR ACCELERATED COMMUNICATION", filed December 15, 1999, by Laurence B. Boucher et al., which in turn claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/439,603 (Attorney Docket No. ALA-009), entitled "INTELLIGENT NETWORK INTERFACE SYSTEM AND METHOD FOR ACCELERATED PROTOCOL PROCESSING", filed November 12, 1999, by Laurence B. Boucher et al., which in turn claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/067,544 (Attorney Docket No. ALA-002), entitled "INTELLIGENT NETWORK INTERFACE SYSTEM AND METHOD FOR ACCELERATED PROTOCOL PROCESSING", filed April 27, 1998, which in turn claims the benefit under 35 U.S.C. § 119(e)(1) of the Provisional Application filed under 35 U.S.C. §111(b) entitled "INTELLIGENT NETWORK

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 006

INTERFACE CARD AND SYSTEM FOR PROTOCOL PROCESSING," Serial No. 60/061,809 (Attorney Docket No. ALA-001), filed on October 14, 1997.

This application also claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/384,792 (Attorney Docket No. ALA-008), entitled "INTELLIGENT NETWORK INTERFACE DEVICE AND SYSTEM FOR ACCELERATED COMMUNICATION," filed August 27, 1999, which in turn claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/141,713 (Attorney Docket No. ALA-003), entitled "INTELLIGENT NETWORK INTERFACE DEVICE AND SYSTEM FOR ACCELERATED PROTOCOL PROCESSING", filed August 28, 1998, which both claim the benefit under 35 U.S.C. § 119(e)(1) of the Provisional Application filed under 35 U.S.C. §111(b) entitled "INTELLIGENT NETWORK INTERFACE DEVICE AND SYSTEM FOR ACCELERATED COMMUNICATION," Serial No. 60/098,296 (Attorney Docket No. ALA-004), filed August 27, 1998.

This application also claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/416,925 (Attorney Docket No. ALA-005), entitled "QUEUE SYSTEM FOR MICROPROCESSORS," filed October 13, 1999, U.S. Patent Application Serial No. 09/514,425 (Attorney Docket No. ALA-007), entitled "PROTOCOL PROCESSING STACK FOR USE WITH INTELLIGENT NETWORK INTERFACE CARD," filed February 28, 2000, U.S. Patent Application Serial No. 09/675,484 (Attorney Docket No. ALA-010A), entitled "INTELLIGENT NETWORK STORAGE INTERFACE SYSTEM," filed September 29, 2000, U.S. Patent Application Serial No. 09/675,700 (Attorney Docket No. ALA-010B), entitled "INTELLIGENT NETWORK STORAGE INTERFACE DEVICE," filed September 29, 2000, U.S. Patent Application Serial No. 09/789,366 (Attorney Docket No. ALA-013), entitled "OBTAINING A DESTINATION ADDRESS SO THAT A NETWORK INTERFACE DEVICE CAN WRITE NETWORK DATA WITHOUT HEADERS DIRECTLY INTO HOST MEMORY," filed February 20, 2001, U.S. Patent Application Serial No. 09/801,488 (Attorney Docket No. ALA-011), entitled "PORT AGGREGATION FOR NETWORK CONNECTIONS THAT ARE OFFLOADED TO NETWORK INTERFACE DEVICES," filed March 7, 2001, U.S. Patent Application Serial No. 09/802,551 (Attorney Docket No. ALA-012), entitled "INTELLIGENT NETWORK STORAGE INTERFACE SYSTEM," filed March 9, 2001, U.S. Patent Application Serial No. 09/802,426 (Attorney Docket No. ALA-014), entitled "REDUCING DELAYS ASSOCIATED WITH

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 007

INSERTING A CHECKSUM INTO A NETWORK MESSAGE," filed March 9, 2001, U.S. Patent Application Serial No. 09/802,550 (Attorney Docket No. ALA-015), entitled "INTELLIGENT INTERFACE CARD AND METHOD FOR ACCELERATED PROTOCOL PROCESSING," filed March 9, 2001, U.S. Patent Application Serial No. 09/855,979 (Attorney Docket No. ALA-016), entitled "NETWORK INTERFACE DEVICE EMPLOYING DMA COMMAND QUEUE," filed March 14, 2001, U.S. Patent Application Serial No. 09/970,124 (Attorney Docket No. ALA-020), entitled "NETWORK INTERFACE DEVICE THAT FAST-PATH PROCESSES SOLICITED SESSION LAYER READ COMMANDS," filed October 2, 2001.

The subject matter of all of the above-identified patent applications (including the subject matter in the Microfiche Appendix of U.S. Application Serial No. 09/464,283), and of the two above-identified provisional applications, is incorporated by reference herein.

REFERENCE TO COMPACT DISC APPENDIX

The Compact Disc Appendix (CD Appendix), which is a part of the present disclosure, includes three folders, designated CD Appendix A, CD Appendix B, and CD Appendix C on the compact disc. CD Appendix A contains a hardware description language (verilog code) description of an embodiment of a receive sequencer. CD Appendix B contains microcode executed by a processor that operates in conjunction with the receive sequencer of CD Appendix A. CD Appendix C contains a device driver executable on the host as well as ATCP code executable on the host. A portion of the disclosure of this patent document contains material (other than any portion of the "free BSD" stack included in CD Appendix C) which is subject to copyright protection. The copyright owner of that material has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights.

TECHNICAL FIELD

The present invention relates generally to computer or other networks, and more particularly to processing of information communicated between hosts such as computers connected to a network.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 008

BACKGROUND

The advantages of network computing are increasingly evident. The convenience and efficiency of providing information, communication or computational power to individuals at their personal computer or other end user devices has led to rapid growth of such network computing, including internet as well as intranet devices and applications.

As is well known, most network computer communication is accomplished with the aid of a layered software architecture for moving information between host computers connected to the network. The layers help to segregate information into manageable segments, the general functions of each layer often based on an international standard called Open Systems Interconnection (OSI). OSI sets forth seven processing layers through which information may pass when received by a host in order to be presentable to an end user. Similarly, transmission of information from a host to the network may pass through those seven processing layers in reverse order. Each step of processing and service by a layer may include copying the processed information. Another reference model that is widely implemented, called TCP/IP (TCP stands for transport control protocol, while IP denotes internet protocol) essentially employs five of the seven layers of OSI.

Networks may include, for instance, a high-speed bus such as an Ethernet connection or an internet connection between disparate local area networks (LANs), each of which includes multiple hosts, or any of a variety of other known means for data transfer between hosts. According to the OSI standard, physical layers are connected to the network at respective hosts, the physical layers providing transmission and receipt of raw data bits via the network. A data link layer is serviced by the physical layer of each host, the data link layers providing frame division and error correction to the data received from the physical layers, as well as processing acknowledgment frames sent by the receiving host. A network layer of each host is serviced by respective data link layers, the network layers primarily controlling size and coordination of subnets of packets of data.

A transport layer is serviced by each network layer and a session layer is serviced by each transport layer within each host. Transport layers accept data from their respective session layers and split the data into smaller units for transmission to the other host's transport layer, which concatenates the data for presentation to respective presentation layers. Session layers allow for enhanced communication control between the hosts. Presentation layers are serviced by their respective session layers, the presentation layers translating between data semantics

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 009

and syntax which may be peculiar to each host and standardized structures of data representation. Compression and/or encryption of data may also be accomplished at the presentation level. Application layers are serviced by respective presentation layers, the application layers translating between programs particular to individual hosts and standardized programs for presentation to either an application or an end user. The TCP/IP standard includes the lower four layers and application layers, but integrates the functions of session layers and presentation layers into adjacent layers. Generally speaking, application, presentation and session layers are defined as upper layers, while transport, network and data link layers are defined as lower layers.

The rules and conventions for each layer are called the protocol of that layer, and since the protocols and general functions of each layer are roughly equivalent in various hosts, it is useful to think of communication occurring directly between identical layers of different hosts, even though these peer layers do not directly communicate without information transferring sequentially through each layer below. Each lower layer performs a service for the layer immediately above it to help with processing the communicated information. Each layer saves the information for processing and service to the next layer. Due to the multiplicity of hardware and software architectures, devices and programs commonly employed, each layer is necessary to insure that the data can make it to the intended destination in the appropriate form, regardless of variations in hardware and software that may intervene.

In preparing data for transmission from a first to a second host, some control data is added at each layer of the first host regarding the protocol of that layer, the control data being indistinguishable from the original (payload) data for all lower layers of that host. Thus an application layer attaches an application header to the payload data and sends the combined data to the presentation layer of the sending host, which receives the combined data, operates on it and adds a presentation header to the data, resulting in another combined data packet. The data resulting from combination of payload data, application header and presentation header is then passed to the session layer, which performs required operations including attaching a session header to the data and presenting the resulting combination of data to the transport layer. This process continues as the information moves to lower layers, with a transport header, network header and data link header and trailer attached to the data at each of those layers, with each step typically including data moving and copying, before sending the data as bit packets over the network to the second host.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 010

The receiving host generally performs the converse of the above-described process, beginning with receiving the bits from the network, as headers are removed and data processed in order from the lowest (physical) layer to the highest (application) layer before transmission to a destination of the receiving host. Each layer of the receiving host recognizes and manipulates only the headers associated with that layer, since to that layer the higher layer control data is included with and indistinguishable from the payload data. Multiple interrupts, valuable central processing unit (CPU) processing time and repeated data copies may also be necessary for the receiving host to place the data in an appropriate form at its intended destination.

The above description of layered protocol processing is simplified, as college-level textbooks devoted primarily to this subject are available, such as Computer Networks, Third Edition (1996) by Andrew S. Tanenbaum, which is incorporated herein by reference. As defined in that book, a computer network is an interconnected collection of autonomous computers, such as internet and intranet devices, including local area networks (LANs), wide area networks (WANs), asynchronous transfer mode (ATM), ring or token ring, wired, wireless, satellite or other means for providing communication capability between separate processors. A computer is defined herein to include a device having both logic and memory functions for processing data, while computers or hosts connected to a network are said to be heterogeneous if they function according to different operating devices or communicate via different architectures.

As networks grow increasingly popular and the information communicated thereby becomes increasingly complex and copious, the need for such protocol processing has increased. It is estimated that a large fraction of the processing power of a host CPU may be devoted to controlling protocol processes, diminishing the ability of that CPU to perform other tasks. Network interface cards have been developed to help with the lowest layers, such as the physical and data link layers. It is also possible to increase protocol processing speed by simply adding more processing power or CPUs according to conventional arrangements. This solution, however, is both awkward and expensive. But the complexities presented by various networks, protocols, architectures, operating devices and applications generally require extensive processing to afford communication capability between various network hosts.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 011

SUMMARY OF THE INVENTION

The current invention provides a device for processing network communication that greatly increases the speed of that processing and the efficiency of transferring data being communicated. The invention has been achieved by questioning the long-standing practice of performing multilayered protocol processing on a general-purpose processor. The protocol processing method and architecture that results effectively collapses the layers of a connection-based, layered architecture such as TCP/IP into a single wider layer which is able to send network data more directly to and from a desired location or buffer on a host. This accelerated processing is provided to a host for both transmitting and receiving data, and so improves performance whether one or both hosts involved in an exchange of information have such a feature.

The accelerated processing includes employing representative control instructions for a given message that allow data from the message to be processed via a fast-path which accesses message data directly at its source or delivers it directly to its intended destination. This fast-path bypasses conventional protocol processing of headers that accompany the data. The fast-path employs a specialized microprocessor designed for processing network communication, avoiding the delays and pitfalls of conventional software layer processing, such as repeated copying and interrupts to the CPU. In effect, the fast-path replaces the states that are traditionally found in several layers of a conventional network stack with a single state machine encompassing all those layers, in contrast to conventional rules that require rigorous differentiation and separation of protocol layers. The host retains a sequential protocol processing stack which can be employed for setting up a fast-path connection or processing message exceptions. The specialized microprocessor and the host intelligently choose whether a given message or portion of a message is processed by the microprocessor or the host stack.

One embodiment is a method of generating a fast-path response to a packet received onto a network interface device where the packet is received over a TCP/IP network connection and where the TCP/IP network connection is identified at least in part by a TCP source port, a TCP destination port, an IP source address, and an IP destination address. The method comprises: 1) Examining the packet and determining from the packet the TCP source port, the TCP destination port, the IP source address, and the IP destination address; 2) Accessing an appropriate template header stored on the network interface device. The template header has TCP fields and IP fields; 3) Employing a finite state machine that implements both TCP

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 012

protocol processing and IP protocol processing to fill in the TCP fields and IP fields of the template header; and 4) Transmitting the fast-path response from the network interface device. The fast-path response includes the filled in template header and a payload. The finite state machine does not entail a TCP protocol processing layer and a discrete IP protocol processing layer where the TCP and IP layers are executed one after another in sequence. Rather, the finite state machine covers both TCP and IP protocol processing layers.

In one embodiment, buffer descriptors that point to packets to be transmitted are pushed onto a plurality of transmit queues. A transmit sequencer pops the transmit queues and obtains the buffer descriptors. The buffer descriptors are then used to retrieve the packets from buffers where the packets are stored. The retrieved packets are then transmitted from the network interface device. In one embodiment, there are two transmit queues, one having a higher transmission priority than the other. Packets identified by buffer descriptors on the higher priority transmit queue are transmitted from the network interface device before packets identified by the lower priority transmit queue.

Other structures and methods are disclosed in the detailed description below. This summary does not purport to define the invention. The invention is defined by the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a plan view diagram of a device of the present invention, including a host computer having a communication-processing device for accelerating network communication.

FIG. 2 is a diagram of information flow for the host of FIG. 1 in processing network communication, including a fast-path, a slow-path and a transfer of connection context between the fast and slow-paths.

FIG. 3 is a flow chart of message receiving according to the present invention.

FIG. 4A is a diagram of information flow for the host of FIG. 1 receiving a message packet processed by the slow-path.

FIG. 4B is a diagram of information flow for the host of FIG. 1 receiving an initial message packet processed by the fast-path.

FIG. 4C is a diagram of information flow for the host of FIG. 4B receiving a subsequent message packet processed by the fast-path.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 013

FIG. 4D is a diagram of information flow for the host of FIG. 4C receiving a message packet having an error that causes processing to revert to the slow-path.

FIG. 5 is a diagram of information flow for the host of FIG. 1 transmitting a message by either the fast or slow-paths.

FIG. 6 is a diagram of information flow for a first embodiment of an intelligent network interface card (INIC) associated with a client having a TCP/IP processing stack.

FIG. 7 is a diagram of hardware logic for the INIC embodiment shown in FIG. 6, including a packet control sequencer and a fly-by sequencer.

FIG. 8 is a diagram of the fly-by sequencer of FIG. 7 for analyzing header bytes as they are received by the INIC.

FIG. 9 is a diagram of information flow for a second embodiment of an INIC associated with a server having a TCP/IP processing stack.

FIG. 10 is a diagram of a command driver installed in the host of FIG. 9 for creating and controlling a communication control block for the fast-path.

FIG. 11 is a diagram of the TCP/IP stack and command driver of FIG. 10 configured for NetBios communications.

FIG. 12 is a diagram of a communication exchange between the client of FIG. 6 and the server of FIG. 9.

FIG. 13 is a diagram of hardware functions included in the INIC of FIG. 9.

FIG. 14 is a diagram of a trio of pipelined microprocessors included in the INIC of FIG. 13, including three phases with a processor in each phase.

FIG. 15A is a diagram of a first phase of the pipelined microprocessor of FIG. 14.

FIG. 15B is a diagram of a second phase of the pipelined microprocessor of FIG. 14.

FIG. 15C is a diagram of a third phase of the pipelined microprocessor of FIG. 14.

FIG. 16 is a diagram of a plurality of queue storage units that interact with the microprocessor of FIG. 14 and include SRAM and DRAM.

FIG. 17 is a diagram of a set of status registers for the queues storage units of FIG. 16.

FIG. 18 is a diagram of a queue manager, which interacts, with the queue storage units and status registers of FIG. 16 and FIG. 17.

FIGs. 19A-D are diagrams of various stages of a least-recently-used register that is employed for allocating cache memory.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 014

FIG. 20 is a diagram of the devices used to operate the least-recently-used register of FIGs. 19A-D.

FIG. 21 is another diagram of Intelligent Network Interface Card (INIC) 200 of Figure 13.

FIG. 22 is a diagram of the receive sequencer of FIG. 21.

FIG. 23 is a diagram illustrating a "fast-path" transfer of data of a multi-packet message from INIC 200 to a destination 2311 in host 20.

FIGS. 24-107 are associated with the description below entitled "Disclosure From Provisional Application 60/061,809."

DETAILED DESCRIPTION

FIG. 1 shows a host 20 of the present invention connected by a network 25 to a remote host 22. The increase in processing speed achieved by the present invention can be provided with an intelligent network interface card (INIC) that is easily and affordably added to an existing host, or with a communication processing device (CPD) that is integrated into a host, in either case freeing the host CPU from most protocol processing and allowing improvements in other tasks performed by that CPU. The host 20 in a first embodiment contains a CPU 28 and a CPD 30 connected by a host bus 33. The CPD 30 includes a microprocessor designed for processing communication data and memory buffers controlled by a direct memory access (DMA) unit. Also connected to the host bus 33 is a storage device 35, such as a semiconductor memory or disk drive, along with any related controls.

Referring additionally to FIG. 2, the host CPU 28 controls a protocol processing stack 44 housed in storage 35, the stack including a data link layer 36, network layer 38, transport layer 40, upper layer 46 and an upper layer interface 42. The upper layer 46 may represent a session, presentation and/or application layer, depending upon the particular protocol being employed and message communicated. The upper layer interface 42, along with the CPU 28 and any related controls can send or retrieve a file to or from the upper layer 46 or storage 35, as shown by arrow 48. A connection context 50 has been created, as will be explained below, the context summarizing various features of the connection, such as protocol type and source and destination addresses for each protocol layer. The context may be passed between an interface for the session layer 42 and the CPD 30, as shown by arrows 52 and 54, and stored as a communication control block (CCB) at either CPD 30 or storage 35.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 015

When the CPD 30 holds a CCB defining a particular connection, data received by the CPD from the network and pertaining to the connection is referenced to that CCB and can then be sent directly to storage 35 according to a fast-path 58, bypassing sequential protocol processing by the data link 36, network 38 and transport 40 layers. Transmitting a message, such as sending a file from storage 35 to remote host 22, can also occur via the fast-path 58, in which case the context for the file data is added by the CPD 30 referencing a CCB, rather than by sequentially adding headers during processing by the transport 40, network 38 and data link 36 layers. The DMA controllers of the CPD 30 perform these transfers between CPD and storage 35.

The CPD 30 collapses multiple protocol stacks each having possible separate states into a single state machine for fast-path processing. As a result, exception conditions may occur that are not provided for in the single state machine, primarily because such conditions occur infrequently and to deal with them on the CPD would provide little or no performance benefit to the host. Such exceptions can be CPD 30 or CPU 28 initiated. An advantage of the invention includes the manner in which unexpected situations that occur on a fast-path CCB are handled. The CPD 30 deals with these rare situations by passing back or flushing to the host protocol stack 44 the CCB and any associated message frames involved, via a control negotiation. The exception condition is then processed in a conventional manner by the host protocol stack 44. At some later time, usually directly after the handling of the exception condition has completed and fast-path processing can resume, the host stack 44 hands the CCB back to the CPD.

This fallback capability enables the performance-impacting functions of the host protocols to be handled by the CPD network microprocessor, while the exceptions are dealt with by the host stacks, the exceptions being so rare as to negligibly effect overall performance. The custom designed network microprocessor can have independent processors for transmitting and receiving network information, and further processors for assisting and queuing. A preferred microprocessor embodiment includes a pipelined trio of receive, transmit and utility processors. DMA controllers are integrated into the implementation and work in close concert with the network microprocessor to quickly move data between buffers adjacent to the controllers and other locations such as long term storage. Providing buffers logically adjacent to the DMA controllers avoids unnecessary loads on the PCI bus.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 016

FIG. 3 diagrams the general flow of messages received according to the current invention. A large TCP/IP message such as a file transfer may be received by the host from the network in a number of separate, approximately 64 KB transfers, each of which may be split into many, approximately 1.5 KB frames or packets for transmission over a network. Novell NetWare protocol suites running Sequenced Packet Exchange Protocol (SPX) or NetWare Core Protocol (NCP) over Internetwork Packet Exchange (IPX) work in a similar fashion. Another form of data communication which can be handled by the fast-path is Transaction TCP (hereinafter T/TCP or TTCP), a version of TCP which initiates a connection with an initial transaction request after which a reply containing data may be sent according to the connection, rather than initiating a connection via a several-message initialization dialogue and then transferring data with later messages. In any of the transfers typified by these protocols, each packet conventionally includes a portion of the data being transferred, as well as headers for each of the protocol layers and markers for positioning the packet relative to the rest of the packets of this message.

When a message packet or frame is received 47 from a network by the CPD, it is first validated by a hardware assist. This includes determining the protocol types of the various layers, verifying relevant checksums, and summarizing 57 these findings into a status word or words. Included in these words is an indication whether or not the frame is a candidate for fast-path data flow. Selection 59 of fast-path candidates is based on whether the host may benefit from this message connection being handled by the CPD, which includes determining whether the packet has header bytes indicating particular protocols, such as TCP/IP or SPX/IPX for example. The small percent of frames that are not fast-path candidates are sent 61 to the host protocol stacks for slow-path protocol processing. Subsequent network microprocessor work with each fast-path candidate determines whether a fast-path connection such as a TCP or SPX CCB is already extant for that candidate, or whether that candidate may be used to set up a new fast-path connection, such as for a TTCP/IP transaction. The validation provided by the CPD provides acceleration whether a frame is processed by the fast-path or a slow-path, as only error free, validated frames are processed by the host CPU even for the slow-path processing.

All received message frames which have been determined by the CPD hardware assist to be fast-path candidates are examined 53 by the network microprocessor or INIC comparator circuits to determine whether they match a CCB held by the CPD. Upon confirming such a

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 017

match, the CPD removes lower layer headers and sends 69 the remaining application data from the frame directly into its final destination in the host using direct memory access (DMA) units of the CPD. This operation may occur immediately upon receipt of a message packet, for example when a TCP connection already exists and destination buffers have been negotiated, or it may first be necessary to process an initial header to acquire a new set of final destination addresses for this transfer. In this latter case, the CPD will queue subsequent message packets while waiting for the destination address, and then DMA the queued application data to that destination.

A fast-path candidate that does not match a CCB may be used to set up a new fast-path connection, by sending 65 the frame to the host for sequential protocol processing. In this case, the host uses this frame to create 51 a CCB, which is then passed to the CPD to control subsequent frames on that connection. The CCB, which is cached 67 in the CPD, includes control and state information pertinent to all protocols that would have been processed had conventional software layer processing been employed. The CCB also contains storage space for per-transfer information used to facilitate moving application-level data contained within subsequent related message packets directly to a host application in a form available for immediate usage. The CPD takes command of connection processing upon receiving a CCB for that connection from the host.

As shown more specifically in FIG. 4A, when a message packet is received from the remote host 22 via network 25, the packet enters hardware receive logic 32 of the CPD 30, which checksums headers and data, and parses the headers, creating a word or words which identify the message packet and status, storing the headers, data and word temporarily in memory 60. As well as validating the packet, the receive logic 32 indicates with the word whether this packet is a candidate for fast-path processing. FIG. 4A depicts the case in which the packet is not a fast-path candidate, in which case the CPD 30 sends the validated headers and data from memory 60 to data link layer 36 along an internal bus for processing by the host CPU, as shown by arrow 56. The packet is processed by the host protocol stack 44 of data link 36, network 38, transport 40 and session 42 layers, and data (D) 63 from the packet may then be sent to storage 35, as shown by arrow 65.

FIG. 4B, depicts the case in which the receive logic 32 of the CPD determines that a message packet is a candidate for fast-path processing, for example by deriving from the packet's headers that the packet belongs to a TCP/IP, TTCP/IP or SPX/IPX message. A

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 018

processor 55 in the CPD 30 then checks to see whether the word that summarizes the fast-path candidate matches a CCB held in a cache 62. Upon finding no match for this packet, the CPD sends the validated packet from memory 60 to the host protocol stack 44 for processing. Host stack 44 may use this packet to create a connection context for the message, including finding and reserving a destination for data from the message associated with the packet, the context taking the form of a CCB. The present embodiment employs a single specialized host stack 44 for processing both fast-path and non-fast-path candidates, while in an embodiment described below fast-path candidates are processed by a different host stack than non-fast-path candidates. Some data (D1) 66 from that initial packet may optionally be sent to the destination in storage 35, as shown by arrow 68. The CCB is then sent to the CPD 30 to be saved in cache 62, as shown by arrow 64. For a traditional connection-based message such as typified by TCP/IP, the initial packet may be part of a connection initialization dialogue that transpires between hosts before the CCB is created and passed to the CPD 30.

Referring now to FIG. 4C, when a subsequent packet from the same connection as the initial packet is received from the network 25 by CPD 30, the packet headers and data are validated by the receive logic 32, and the headers are parsed to create a summary of the message packet and a hash for finding a corresponding CCB, the summary and hash contained in a word or words. The word or words are temporarily stored in memory 60 along with the packet. The processor 55 checks for a match between the hash and each CCB that is stored in the cache 62 and, finding a match, sends the data (D2) 70 via a fast-path directly to the destination in storage 35, as shown by arrow 72, bypassing the session layer 42, transport layer 40, network layer 38 and data link layer 36. The remaining data packets from the message can also be sent by DMA directly to storage, avoiding the relatively slow protocol layer processing and repeated copying by the CPU stack 44.

FIG. 4D shows the procedure for handling the rare instance when a message for which a fast-path connection has been established, such as shown in FIG. 4C, has a packet that is not easily handled by the CPD. In this case the packet is sent to be processed by the protocol stack 44, which is handed the CCB for that message from cache 62 via a control dialogue with the CPD, as shown by arrow 76, signaling to the CPU to take over processing of that message. Slow-path processing by the protocol stack then results in data (D3) 80 from the packet being sent, as shown by arrow 82, to storage 35. Once the packet has been processed and the error situation corrected, the CCB can be handed back via a control dialogue to the cache 62, so that

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 019

payload data from subsequent packets of that message can again be sent via the fast-path of the CPD 30. Thus the CPU and CPD together decide whether a given message is to be processed according to fast-path hardware processing or more conventional software processing by the CPU.

Transmission of a message from the host 20 to the network 25 for delivery to remote host 22 also can be processed by either sequential protocol software processing via the CPU or accelerated hardware processing via the CPD 30, as shown in FIG. 5. A message (M) 90 that is selected by CPU 28 from storage 35 can be sent to session layer 42 for processing by stack 44, as shown by arrows 92 and 96. For the situation in which a connection exists and the CPD 30 already has an appropriate CCB for the message, however, data packets can bypass host stack 44 and be sent by DMA directly to memory 60, with the processor 55 adding to each data packet a single header containing all the appropriate protocol layers, and sending the resulting packets to the network 25 for transmission to remote host 22. This fast-path transmission can greatly accelerate processing for even a single packet, with the acceleration multiplied for a larger message.

A message for which a fast-path connection is not extant thus may benefit from creation of a CCB with appropriate control and state information for guiding fast-path transmission. For a traditional connection-based message, such as typified by TCP/IP or SPX/IPX, the CCB is created during connection initialization dialogue. For a quick-connection message, such as typified by TTCP/IP, the CCB can be created with the same transaction that transmits payload data. In this case, the transmission of payload data may be a reply to a request that was used to set up the fast-path connection. In any case, the CCB provides protocol and status information regarding each of the protocol layers, including which user is involved and storage space for per-transfer information. The CCB is created by protocol stack 44, which then passes the CCB to the CPD 30 by writing to a command register of the CPD, as shown by arrow 98. Guided by the CCB, the processor 55 moves network frame-sized portions of the data from the source in host memory 35 into its own memory 60 using DMA, as depicted by arrow 99. The processor 55 then prepends appropriate headers and checksums to the data portions, and transmits the resulting frames to the network 25, consistent with the restrictions of the associated protocols. After the CPD 30 has received an acknowledgement that all the data has reached its destination, the CPD will then notify the host 35 by writing to a response buffer.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 020

Thus, fast-path transmission of data communications also relieves the host CPU of per-frame processing. A vast majority of data transmissions can be sent to the network by the fast-path. Both the input and output fast-paths attain a huge reduction in interrupts by functioning at an upper layer level, i.e., session level or higher, and interactions between the network microprocessor and the host occur using the full transfer sizes which that upper layer wishes to make. For fast-path communications, an interrupt only occurs (at the most) at the beginning and end of an entire upper-layer message transaction, and there are no interrupts for the sending or receiving of each lower layer portion or packet of that transaction.

A simplified intelligent network interface card (INIC) 150 is shown in FIG. 6 to provide a network interface for a host 152. Hardware logic 171 of the INIC 150 is connected to a network 155, with a peripheral bus (PCI) 157 connecting the INIC and host. The host 152 in this embodiment has a TCP/IP protocol stack, which provides a slow-path 158 for sequential software processing of message frames received from the network 155. The host 152 protocol stack includes a data link layer 160, network layer 162, a transport layer 164 and an application layer 166, which provides a source or destination 168 for the communication data in the host 152. Other layers which are not shown, such as session and presentation layers, may also be included in the host stack 152, and the source or destination may vary depending upon the nature of the data and may actually be the application layer.

The INIC 150 has a network processor 170 which chooses between processing messages along a slow-path 158 that includes the protocol stack of the host, or along a fast-path 159 that bypasses the protocol stack of the host. Each received packet is processed on the fly by hardware logic 171 contained in INIC 150, so that all of the protocol headers for a packet can be processed without copying, moving or storing the data between protocol layers. The hardware logic 171 processes the headers of a given packet at one time as packet bytes pass through the hardware, by categorizing selected header bytes. Results of processing the selected bytes help to determine which other bytes of the packet are categorized, until a summary of the packet has been created, including checksum validations. The processed headers and data from the received packet are then stored in INIC storage 185, as well as the word or words summarizing the headers and status of the packet. For a network storage configuration, the INIC 150 may be connected to a peripheral storage device such as a disk drive which has an IDE, SCSI or similar interface, with a file cache for the storage device

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 021

residing on the memory 185 of the INIC 150. Several such network interfaces may exist for a host, with each interface having an associated storage device.

The hardware processing of message packets received by INIC 150 from network 155 is shown in more detail in FIG. 7. A received message packet first enters a media access controller 172, which controls INIC access to the network and receipt of packets and can provide statistical information for network protocol management. From there, data flows one byte at a time into an assembly register 174, which in this example is 128 bits wide. The data is categorized by a fly-by sequencer 178, as will be explained in more detail with regard to FIG. 8, which examines the bytes of a packet as they fly by, and generates status from those bytes that will be used to summarize the packet. The status thus created is merged with the data by a multiplexor 180 and the resulting data stored in SRAM 182. A packet control sequencer 176 oversees the fly-by sequencer 178, examines information from the media access controller 172, counts the bytes of data, generates addresses, moves status and manages the movement of data from the assembly register 174 to SRAM 182 and eventually DRAM 188. The packet control sequencer 176 manages a buffer in SRAM 182 via SRAM controller 183, and also indicates to a DRAM controller 186 when data needs to be moved from SRAM 182 to a buffer in DRAM 188. Once data movement for the packet has been completed and all the data has been moved to the buffer in DRAM 188, the packet control sequencer 176 will move the status that has been generated in the fly-by sequencer 178 out to the SRAM 182 and to the beginning of the DRAM 188 buffer to be prepended to the packet data. The packet control sequencer 176 then requests a queue manager 184 to enter a receive buffer descriptor into a receive queue, which in turn notifies the processor 170 that the packet has been processed by hardware logic 171 and its status summarized.

FIG. 8 shows that the fly-by sequencer 178 has several tiers, with each tier generally focusing on a particular portion of the packet header and thus on a particular protocol layer, for generating status pertaining to that layer. The fly-by sequencer 178 in this embodiment includes a media access control sequencer 191, a network sequencer 192, a transport sequencer 194 and a session sequencer 195. Sequencers pertaining to higher protocol layers can additionally be provided. The fly-by sequencer 178 is reset by the packet control sequencer 176 and given pointers by the packet control sequencer that tell the fly-by sequencer whether a given byte is available from the assembly register 174. The media access control sequencer 191 determines, by looking at bytes 0-5, that a packet is addressed to host 152 rather than or in

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 022

addition to another host. Offsets 12 and 13 of the packet are also processed by the media access control sequencer 191 to determine the type field, for example whether the packet is Ethernet or 802.3. If the type field is Ethernet those bytes also tell the media access control sequencer 191 the packet's network protocol type. For the 802.3 case, those bytes instead indicate the length of the entire frame, and the media access control sequencer 191 will check eight bytes further into the packet to determine the network layer type.

For most packets the network sequencer 192 validates that the header length received has the correct length, and checksums the network layer header. For fast-path candidates the network layer header is known to be IP or IPX from analysis done by the media access control sequencer 191. Assuming for example that the type field is 802.3 and the network protocol is IP, the network sequencer 192 analyzes the first bytes of the network layer header, which will begin at byte 22, in order to determine IP type. The first bytes of the IP header will be processed by the network sequencer 192 to determine what IP type the packet involves. Determining that the packet involves, for example, IP version 4, directs further processing by the network sequencer 192, which also looks at the protocol type located ten bytes into the IP header for an indication of the transport header protocol of the packet. For example, for IP over Ethernet, the IP header begins at offset 14, and the protocol type byte is offset 23, which will be processed by network logic to determine whether the transport layer protocol is TCP, for example. From the length of the network layer header, which is typically 20-40 bytes, network sequencer 192 determines the beginning of the packet's transport layer header for validating the transport layer header. Transport sequencer 194 may generate checksums for the transport layer header and data, which may include information from the IP header in the case of TCP at least.

Continuing with the example of a TCP packet, transport sequencer 194 also analyzes the first few bytes in the transport layer portion of the header to determine, in part, the TCP source and destination ports for the message, such as whether the packet is NetBios or other protocols. Byte 12 of the TCP header is processed by the transport sequencer 194 to determine and validate the TCP header length. Byte 13 of the TCP header contains flags that may, aside from ack flags and push flags, indicate unexpected options, such as reset and fin, that may cause the processor to categorize this packet as an exception. TCP offset bytes 16 and 17 are the checksum, which is pulled out and stored by the hardware logic 171 while the rest of the frame is validated against the checksum.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 023

Session sequencer 195 determines the length of the session layer header, which in the case of NetBios is only four bytes, two of which tell the length of the NetBios payload data, but which can be much larger for other protocols. The session sequencer 195 can also be used to categorize the type of message as read or write, for example, for which the fast-path may be particularly beneficial. Further upper layer logic processing, depending upon the message type, can be performed by the hardware logic 171 of packet control sequencer 176 and fly-by sequencer 178. Thus hardware logic 171 intelligently directs hardware processing of the headers by categorization of selected bytes from a single stream of bytes, with the status of the packet being built from classifications determined on the fly. Once the packet control sequencer 176 detects that all of the packet has been processed by the fly-by sequencer 178, the packet control sequencer 176 adds the status information generated by the fly-by sequencer 178 and any status information generated by the packet control sequencer 176, and prepends (adds to the front) that status information to the packet, for convenience in handling the packet by the processor 170. The additional status information generated by the packet control sequencer 176 includes media access controller 172 status information and any errors discovered, or data overflow in either the assembly register or DRAM buffer, or other miscellaneous information regarding the packet. The packet control sequencer 176 also stores entries into a receive buffer queue and a receive statistics queue via the queue manager 184. An advantage of processing a packet by hardware logic 171 is that the packet does not, in contrast with conventional sequential software protocol processing, have to be stored, moved, copied or pulled from storage for processing each protocol layer header, offering dramatic increases in processing efficiency and savings in processing time for each packet. The packets can be processed at the rate bits are received from the network, for example 100 megabits/second for a 100 baseT connection. The time for categorizing a packet received at this rate and having a length of sixty bytes is thus about 5 microseconds. The total time for processing this packet with the hardware logic 171 and sending packet data to its host destination via the fast-path may be about 16 microseconds or less, assuming a 66 MHz PCI bus, whereas conventional software protocol processing by a 300 MHz Pentium II® processor may take as much as 200 microseconds in a busy device. More than an order of magnitude decrease in processing time can thus be achieved with fast-path 159 in comparison with a high-speed CPU employing conventional sequential software protocol processing, demonstrating the dramatic acceleration provided by processing the protocol headers by the

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 024

hardware logic 171 and processor 170, without even considering the additional time savings afforded by the reduction in CPU interrupts and host bus bandwidth savings.

The processor 170 chooses, for each received message packet held in storage 185, whether that packet is a candidate for the fast-path 159 and, if so, checks to see whether a fast-path has already been set up for the connection that the packet belongs to. To do this, the processor 170 first checks the header status summary to determine whether the packet headers are of a protocol defined for fast-path candidates. If not, the processor 170 commands DMA controllers in the INIC 150 to send the packet to the host for slow-path 158 processing. Even for a slow-path 158 processing of a message, the INIC 150 thus performs initial procedures such as validation and determination of message type, and passes the validated message at least to the data link layer 160 of the host.

For fast-path 159 candidates, the processor 170 checks to see whether the header status summary matches a CCB held by the INIC. If so, the data from the packet is sent along fast-path 159 to the destination 168 in the host. If the fast-path 159 candidate's packet summary does not match a CCB held by the INIC, the packet may be sent to the host 152 for slow-path processing to create a CCB for the message. Employment of the fast-path 159 may also not be needed or desirable for the case of fragmented messages or other complexities. For the vast majority of messages, however, the INIC fast-path 159 can greatly accelerate message processing. The INIC 150 thus provides a single state machine processor 170 that decides whether to send data directly to its destination, based upon information gleaned on the fly, as opposed to the conventional employment of a state machine in each of several protocol layers for determining the destiny of a given packet.

In processing an indication or packet received at the host 152, a protocol driver of the host selects the processing route based upon whether the indication is fast-path or slow-path. A TCP/IP or SPX/IPX message has a connection that is set up from which a CCB is formed by the driver and passed to the INIC for matching with and guiding the fast-path packet to the connection destination 168. For a TTCP/IP message, the driver can create a connection context for the transaction from processing an initial request packet, including locating the message destination 168, and then passing that context to the INIC in the form of a CCB for providing a fast-path for a reply from that destination. A CCB includes connection and state information regarding the protocol layers and packets of the message. Thus a CCB can include source and destination media access control (MAC) addresses, source and destination

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 025

IP or IPX addresses, source and destination TCP or SPX ports, TCP variables such as timers, receive and transmit windows for sliding window protocols, and information indicating the session layer protocol.

Caching the CCBs in a hash table in the INIC provides quick comparisons with words summarizing incoming packets to determine whether the packets can be processed via the fast-path 159, while the full CCBs are also held in the INIC for processing. Other ways to accelerate this comparison include software processes such as a B-tree or hardware assists such as a content addressable memory (CAM). When INIC microcode or comparator circuits detect a match with the CCB, a DMA controller places the data from the packet in the destination 168, without any interrupt by the CPU, protocol processing or copying. Depending upon the type of message received, the destination of the data may be the session, presentation or application layers, or a file buffer cache in the host 152.

FIG. 9 shows an INIC 200 connected to a host 202 that is employed as a file server. This INIC provides a network interface for several network connections employing the 802.3u standard, commonly known as Fast Ethernet. The INIC 200 is connected by a PCI bus 205 to the server 202, which maintains a TCP/IP or SPX/IPX protocol stack including MAC layer 212, network layer 215, transport layer 217 and application layer 220, with a source/destination 222 shown above the application layer, although as mentioned earlier the application layer can be the source or destination. The INIC is also connected to network lines 210, 240, 242 and 244, which are preferably Fast Ethernet, twisted pair, fiber optic, coaxial cable or other lines each allowing data transmission of 100 Mb/s, while faster and slower data rates are also possible. Network lines 210, 240, 242 and 244 are each connected to a dedicated row of hardware circuits which can each validate and summarize message packets received from their respective network line. Thus line 210 is connected with a first horizontal row of sequencers 250, line 240 is connected with a second horizontal row of sequencers 260, line 242 is connected with a third horizontal row of sequencers 262 and line 244 is connected with a fourth horizontal row of sequencers 264. After a packet has been validated and summarized by one of the horizontal hardware rows it is stored along with its status summary in storage 270.

A network processor 230 determines, based on that summary and a comparison with any CCBs stored in the INIC 200, whether to send a packet along a slow-path 231 for processing by the host. A large majority of packets can avoid such sequential processing and have their

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 026

data portions sent by DMA along a fast-path 237 directly to the data destination 222 in the server according to a matching CCB. Similarly, the fast-path 237 provides an avenue to send data directly from the source 222 to any of the network lines by processor 230 division of the data into packets and addition of full headers for network transmission, again minimizing CPU processing and interrupts. For clarity only horizontal sequencer 250 is shown active; in actuality each of the sequencer rows 250, 260, 262 and 264 offers full duplex communication, concurrently with all other sequencer rows. The specialized INIC 200 is much faster at working with message packets than even advanced general-purpose host CPUs that processes those headers sequentially according to the software protocol stack.

One of the most commonly used network protocols for large messages such as file transfers is server message block (SMB) over TCP/IP. SMB can operate in conjunction with redirector software that determines whether a required resource for a particular operation, such as a printer or a disk upon which a file is to be written, resides in or is associated with the host from which the operation was generated or is located at another host connected to the network, such as a file server. SMB and server/redirector are conventionally serviced by the transport layer; in the present invention SMB and redirector can instead be serviced by the INIC. In this case, sending data by the DMA controllers from the INIC buffers when receiving a large SMB transaction may greatly reduce interrupts that the host must handle. Moreover, this DMA generally moves the data to its final destination in the file device cache. An SMB transmission of the present invention follows essentially the reverse of the above described SMB receive, with data transferred from the host to the INIC and stored in buffers, while the associated protocol headers are prepended to the data in the INIC, for transmission via a network line to a remote host. Processing by the INIC of the multiple packets and multiple TCP, IP, NetBios and SMB protocol layers via custom hardware and without repeated interrupts of the host can greatly increase the speed of transmitting an SMB message to a network line.

As shown in FIG. 10, for controlling whether a given message is processed by the host 202 or by the INIC 200, a message command driver 300 may be installed in host 202 to work in concert with a host protocol stack 310. The command driver 300 can intervene in message reception or transmittal, create CCBs and send or receive CCBs from the INIC 200, so that functioning of the INIC, aside from improved performance, is transparent to a user. Also shown is an INIC memory 304 and an INIC miniport driver 306, which can direct message packets received from network 210 to either the conventional protocol stack 310 or the

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 027

command protocol stack 300, depending upon whether a packet has been labeled as a fast-path candidate. The conventional protocol stack 310 has a data link layer 312, a network layer 314 and a transport layer 316 for conventional, lower layer processing of messages that are not labeled as fast-path candidates and therefore not processed by the command stack 300. Residing above the lower layer stack 310 is an upper layer 318, which represents a session, presentation and/or application layer, depending upon the message communicated. The command driver 300 similarly has a data link layer 320, a network layer 322 and a transport layer 325.

The driver 300 includes an upper layer interface 330 that determines, for transmission of messages to the network 210, whether a message transmitted from the upper layer 318 is to be processed by the command stack 300 and subsequently the INIC fast-path, or by the conventional stack 310. When the upper layer interface 330 receives an appropriate message from the upper layer 318 that would conventionally be intended for transmission to the network after protocol processing by the protocol stack of the host, the message is passed to driver 300. The INIC then acquires network-sized portions of the message data for that transmission via INIC DMA units, prepends headers to the data portions and sends the resulting message packets down the wire. Conversely, in receiving a TCP, TTCP, SPX or similar message packet from the network 210 to be used in setting up a fast-path connection, miniport driver 306 diverts that message packet to command driver 300 for processing. The driver 300 processes the message packet to create a context for that message, with the driver 302 passing the context and command instructions back to the INIC 200 as a CCB for sending data of subsequent messages for the same connection along a fast-path. Hundreds of TCP, TTCP, SPX or similar CCB connections may be held indefinitely by the INIC, although a least recently used (LRU) algorithm is employed for the case when the INIC cache is full. The driver 300 can also create a connection context for a TTCP request which is passed to the INIC 200 as a CCB, allowing fast-path transmission of a TTCP reply to the request. A message having a protocol that is not accelerated can be processed conventionally by protocol stack 310.

FIG. 11 shows a TCP/IP implementation of command driver software for Microsoft® protocol messages. A conventional host protocol stack 350 includes MAC layer 353, IP layer 355 and TCP layer 358. A command driver 360 works in concert with the host stack 350 to process network messages. The command driver 360 includes a MAC layer 363, an IP layer

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 028

366 and an Alacritech TCP (ATCP) layer 373. The conventional stack 350 and command driver 360 share a network driver interface specification (NDIS) layer 375, which interacts with the INIC miniport driver 306. The INIC miniport driver 306 sorts receive indications for processing by either the conventional host stack 350 or the ATCP driver 360. A TDI filter driver and upper layer interface 380 similarly determines whether messages sent from a TDI user 382 to the network are diverted to the command driver and perhaps to the fast-path of the INIC, or processed by the host stack.

FIG. 12 depicts a typical SMB exchange between a client 190 and server 290, both of which have communication devices of the present invention, the communication devices each holding a CCB defining their connection for fast-path movement of data. The client 190 includes INIC 150, 802.3 compliant data link layer 160, IP layer 162, TCP layer 164, NetBios layer 166, and SMB layer 168. The client has a slow-path 157 and fast-path 159 for communication processing. Similarly, the server 290 includes INIC 200, 802.3 compliant data link layer 212, IP layer 215, TCP layer 217, NetBios layer 220, and SMB 222. The server is connected to network lines 240, 242 and 244, as well as line 210 which is connected to client 190. The server also has a slow-path 231 and fast-path 237 for communication processing. Assuming that the client 190 wishes to read a 100KB file on the server 290, the client may begin by sending a Read Block Raw (RBR) SMB command across network 210 requesting the first 64 KB of that file on the server 290. The RBR command may be only 76 bytes, for example, so the INIC 200 on the server will recognize the message type (SMB) and relatively small message size, and send the 76 bytes directly via the fast-path to NetBios of the server. NetBios will give the data to SMB, which processes the Read request and fetches the 64KB of data into server data buffers. SMB then calls NetBios to send the data, and NetBios outputs the data for the client. In a conventional host, NetBios would call TCP output and pass 64 KB to TCP, which would divide the data into 1460 byte segments and output each segment via IP and eventually MAC (slow-path 231). In the present case, the 64KB data goes to the ATCP driver along with an indication regarding the client-server SMB connection, which indicates a CCB held by the INIC. The INIC 200 then proceeds to DMA 1460 byte segments from the host buffers, add the appropriate headers for TCP, IP and MAC at one time, and send the completed packets on the network 210 (fast-path 237). The INIC 200 will repeat this until the whole 64KB transfer has been sent. Usually after receiving acknowledgement from the client

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 029

that the 64KB has been received, the INIC will then send the remaining 36KB also by the fast-path 237.

With INIC 150 operating on the client 190 when this reply arrives, the INIC 150 recognizes from the first frame received that this connection is receiving fast-path 159 processing (TCP/IP, NetBios, matching a CCB), and the ATCP may use this first frame to acquire buffer space for the message. This latter case is done by passing the first 128 bytes of the NetBios portion of the frame via the ATCP fast-path directly to the host NetBios; that will give NetBios/SMB all of the frame's headers. NetBios/SMB will analyze these headers, realize by matching with a request ID that this is a reply to the original RawRead connection, and give the ATCP a 64K list of buffers into which to place the data. At this stage only one frame has arrived, although more may arrive while this processing is occurring. As soon as the client buffer list is given to the ATCP, it passes that transfer information to the INIC 150, and the INIC 150 starts DMAing any frame data that has accumulated into those buffers.

FIG. 13 provides a simplified diagram of the INIC 200, which combines the functions of a network interface controller and a protocol processor in a single ASIC chip 400. The INIC 200 in this embodiment offers a full-duplex, four channel, 10/100-Megabit per second (Mbps) intelligent network interface controller that is designed for high speed protocol processing for server applications. Although designed specifically for server applications, the INIC 200 can be connected to personal computers, workstations, routers or other hosts anywhere that TCP/IP, TTCP/IP or SPX/IPX protocols are being utilized.

The INIC 200 is connected with four network lines 210, 240, 242 and 244, which may transport data along a number of different conduits, such as twisted pair, coaxial cable or optical fiber, each of the connections providing a media independent interface (MII) via commercially available physical layer chips, such as model 80220/80221 Ethernet Media Interface Adapter from SEEQ Technology Incorporated, 47200 Bayside Parkway, Fremont, CA 94538. The lines preferably are 802.3 compliant and in connection with the INIC constitute four complete Ethernet nodes, the INIC supporting 10Base-T, 10Base-T2, 100Base-TX, 100Base-FX and 100Base-T4 as well as future interface standards. Physical layer identification and initialization is accomplished through host driver initialization routines. The connection between the network lines 210, 240, 242 and 244 and the INIC 200 is controlled by MAC units MAC-A 402, MAC-B 404, MAC-C 406 and MAC-D 408 which contain logic circuits for performing the basic functions of the MAC sublayer, essentially controlling when

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 030

the INIC accesses the network lines 210, 240, 242 and 244. The MAC units 402-408 may act in promiscuous, multicast or unicast modes, allowing the INIC to function as a network monitor, receive broadcast and multicast packets and implement multiple MAC addresses for each node. The MAC units 402-408 also provide statistical information that can be used for simple network management protocol (SNMP).

The MAC units 402, 404, 406 and 408 are each connected to a transmit and receive sequencer, XMT & RCV-A 418, XMT & RCV-B 420, XMT & RCV-C 422 and XMT & RCV-D 424, by wires 410, 412, 414 and 416, respectively. Each of the transmit and receive sequencers can perform several protocol processing steps on the fly as message frames pass through that sequencer. In combination with the MAC units, the transmit and receive sequencers 418-422 can compile the packet status for the data link, network, transport, session and, if appropriate, presentation and application layer protocols in hardware, greatly reducing the time for such protocol processing compared to conventional sequential software engines. The transmit and receive sequencers 410-414 are connected, by lines 426, 428, 430 and 432 to an SRAM and DMA controller 444, which includes DMA controllers 438 and SRAM controller 442. Static random access memory (SRAM) buffers 440 are coupled with SRAM controller 442 by line 441. The SRAM and DMA controllers 444 interact across line 446 with external memory control 450 to send and receive frames via external memory bus 455 to and from dynamic random access memory (DRAM) buffers 460, which is located adjacent to the IC chip 400. The DRAM buffers 460 may be configured as 4 MB, 8 MB, 16 MB or 32 MB, and may optionally be disposed on the chip. The SRAM and DMA controllers 444 are connected via line 464 to a PCI Bus Interface Unit (BIU) 468, which manages the interface between the INIC 200 and the PCI interface bus 257. The 64-bit, multiplexed BIU 468 provides a direct interface to the PCI bus 257 for both slave and master functions. The INIC 200 is capable of operating in either a 64-bit or 32-bit PCI environment, while supporting 64-bit addressing in either configuration.

A microprocessor 470 is connected by line 472 to the SRAM and DMA controllers 444, and connected via line 475 to the PCI BIU 468. Microprocessor 470 instructions and register files reside in an on chip control store 480, which includes a writable on-chip control store (WCS) of SRAM and a read only memory (ROM), and is connected to the microprocessor by line 477. The microprocessor 470 offers a programmable state machine which is capable of processing incoming frames, processing host commands, directing network traffic and

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 031

directing PCI bus traffic. Three processors are implemented using shared hardware in a three level pipelined architecture that launches and completes a single instruction for every clock cycle. A receive processor 482 is primarily used for receiving communications while a transmit processor 484 is primarily used for transmitting communications in order to facilitate full duplex communication, while a utility processor 486 offers various functions including overseeing and controlling PCI register access.

The instructions for the three processors 482, 484 and 486 reside in the on-chip control-store 480. Thus the functions of the three processors can be easily redefined, so that the microprocessor 470 can adapted for a given environment. For instance, the amount of processing required for receive functions may outweigh that required for either transmit or utility functions. In this situation, some receive functions may be performed by the transmit processor 484 and/or the utility processor 486. Alternatively, an additional level of pipelining can be created to yield four or more virtual processors instead of three, with the additional level devoted to receive functions.

The INIC 200 in this embodiment can support up to 256 CCBs which are maintained in a table in the DRAM 460. There is also, however, a CCB index in hash order in the SRAM 440 to save sequential searching. Once a hash has been generated, the CCB is cached in SRAM, with up to sixteen cached CCBs in SRAM in this example. Allocation of the sixteen CCBs cached in SRAM is handled by a least recently used register, described below. These cache locations are shared between the transmit 484 and receive 486 processors so that the processor with the heavier load is able to use more cache buffers. There are also eight header buffers and eight command buffers to be shared between the sequencers. A given header or command buffer is not statically linked to a specific CCB buffer, as the link is dynamic on a per-frame basis.

FIG. 14 shows an overview of the pipelined microprocessor 470, in which instructions for the receive, transmit and utility processors are executed in three alternating phases according to Clock increments I, II and III, the phases corresponding to each of the pipeline stages. Each phase is responsible for different functions, and each of the three processors occupies a different phase during each Clock increment. Each processor usually operates upon a different instruction stream from the control store 480, and each carries its own program counter and status through each of the phases.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 032

In general, a first instruction phase 500 of the pipelined microprocessors completes an instruction and stores the result in a destination operand, fetches the next instruction, and stores that next instruction in an instruction register. A first register set 490 provides a number of registers including the instruction register, and a set of controls 492 for first register set provides the controls for storage to the first register set 490. Some items pass through the first phase without modification by the controls 492, and instead are simply copied into the first register set 490 or a RAM file register 533. A second instruction phase 560 has an instruction decoder and operand multiplexer 498 that generally decodes the instruction that was stored in the instruction register of the first register set 490 and gathers any operands which have been generated, which are then stored in a decode register of a second register set 496. The first register set 490, second register set 496 and a third register set 501, which is employed in a third instruction phase 600, include many of the same registers, as will be seen in the more detailed views of FIGs. 15A-C. The instruction decoder and operand multiplexer 498 can read from two address and data ports of the RAM file register 533, which operates in both the first phase 500 and second phase 560. A third phase 600 of the processor 470 has an arithmetic logic unit (ALU) 602 which generally performs any ALU operations on the operands from the second register set, storing the results in a results register included in the third register set 501. A stack exchange 608 can reorder register stacks, and a queue manager 503 can arrange queues for the processor 470, the results of which are stored in the third register set. The instructions continue with the first phase then following the third phase, as depicted by a circular pipeline 505. Note that various functions have been distributed across the three phases of the instruction execution in order to minimize the combinatorial delays within any given phase. With a frequency in this embodiment of 66 MHz, each Clock increment takes 15 nanoseconds to complete, for a total of 45 nanoseconds to complete one instruction for each of the three processors. The rotating instruction phases are depicted in more detail in FIGs. 15A-C, in which each phase is shown in a different figure.

More particularly, FIG. 15A shows some specific hardware functions of the first phase 500, which generally includes the first register set 490 and related controls 492. The controls for the first register set 492 includes an SRAM control 502, which is a logical control for loading address and write data into SRAM address and data registers 520. Thus the output of the ALU 602 from the third phase 600 may be placed by SRAM control 502 into an address register or data register of SRAM address and data registers 520. A load control 504 similarly provides

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 033

controls for writing a context for a file to file context register 522, and another load control 506 provides controls for storing a variety of miscellaneous data to flip-flop registers 525. ALU condition codes, such as whether a carried bit is set, get clocked into ALU condition codes register 528 without an operation performed in the first phase 500. Flag decodes 508 can perform various functions, such as setting locks, that get stored in flag registers 530.

The RAM file register 533 has a single write port for addresses and data and two read ports for addresses and data, so that more than one register can be read from at one time. As noted above, the RAM file register 533 essentially straddles the first and second phases, as it is written in the first phase 500 and read from in the second phase 560. A control store instruction 510 allows the reprogramming of the processors due to new data in from the control store 480, not shown in this figure, the instructions stored in an instruction register 535. The address for this is generated in a fetch control register 511, which determines which address to fetch, the address stored in fetch address register 538. Load control 515 provides instructions for a program counter 540, which operates much like the fetch address for the control store. A last-in first-out stack 544 of three registers is copied to the first register set without undergoing other operations in this phase. Finally, a load control 517 for a debug address 548 is optionally included, which allows correction of errors that may occur.

FIG. 15B depicts the second microprocessor phase 560, which includes reading addresses and data out of the RAM file register 533. A scratch SRAM 565 is written from SRAM address and data register 520 of the first register set, which includes a register that passes through the first two phases to be incremented in the third. The scratch SRAM 565 is read by the instruction decoder and operand multiplexer 498, as are most of the registers from the first register set, with the exception of the stack 544, debug address 548 and SRAM address and data register mentioned above. The instruction decoder and operand multiplexer 498 looks at the various registers of set 490 and SRAM 565, decodes the instructions and gathers the operands for operation in the next phase, in particular determining the operands to provide to the ALU 602 below. The outcome of the instruction decoder and operand multiplexer 498 is stored to a number of registers in the second register set 496, including ALU operands 579 and 582, ALU condition code register 580, and a queue channel and command 587 register, which in this embodiment can control thirty-two queues. Several of the registers in set 496 are loaded fairly directly from the instruction register 535 above without substantial decoding by the decoder 498, including a program control 590, a literal field 589, a test select 584 and a

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 034

flag select 585. Other registers such as the file context 522 of the first phase 500 are always stored in a file context 577 of the second phase 560, but may also be treated as an operand that is gathered by the multiplexer 572. The stack registers 544 are simply copied in stack register 594. The program counter 540 is incremented 568 in this phase and stored in register 592. Also incremented 570 is the optional debug address 548, and a load control 575 may be fed from the pipeline 505 at this point in order to allow error control in each phase, the result stored in debug address 598.

FIG. 15C depicts the third microprocessor phase 600, which includes ALU and queue operations. The ALU 602 includes an adder, priority encoders and other standard logic functions. Results of the ALU are stored in registers ALU output 618, ALU condition codes 620 and destination operand results 622. A file context register 616, flag select register 626 and literal field register 630 are simply copied from the previous phase 560. A test multiplexer 604 is provided to determine whether a conditional jump results in a jump, with the results stored in a test results register 624. The test multiplexer 604 may instead be performed in the first phase 500 along with similar decisions such as fetch control 511. A stack exchange 608 shifts a stack up or down by fetching a program counter from stack 594 or putting a program counter onto that stack, results of which are stored in program control 634, program counter 638 and stack 640 registers. The SRAM address may optionally be incremented in this phase 600. Another load control 610 for another debug address 642 may be forced from the pipeline 505 at this point in order to allow error control in this phase also. A QRAM & QALU 606, shown together in this figure, read from the queue channel and command register 587, store in SRAM and rearrange queues, adding or removing data and pointers as needed to manage the queues of data, sending results to the test multiplexer 604 and a queue flags and queue address register 628. Thus the QRAM & QALU 606 assume the duties of managing queues for the three processors, a task conventionally performed sequentially by software on a CPU, the queue manager 606 instead providing accelerated and substantially parallel hardware queuing.

FIG. 16 depicts two of the thirty-two hardware queues that are managed by the queue manager 606, with each of the queues having an SRAM head, an SRAM tail and the ability to queue information in a DRAM body as well, allowing expansion and individual configuration of each queue. Thus FIFO 700 has SRAM storage units, 705, 707, 709 and 711, each containing eight bytes for a total of thirty-two bytes, although the number and capacity of these units may vary in other embodiments. Similarly, FIFO 702 has SRAM storage units

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 035

713, 715, 717 and 719. SRAM units 705 and 707 are the head of FIFO 700 and units 709 and 711 are the tail of that FIFO, while units 713 and 715 are the head of FIFO 702 and units 717 and 719 are the tail of that FIFO. Information for FIFO 700 may be written into head units 705 or 707, as shown by arrow 722, and read from tail units 711 or 709, as shown by arrow 725. A particular entry, however, may be both written to and read from head units 705 or 707, or may be both written to and read from tail units 709 or 711, minimizing data movement and latency. Similarly, information for FIFO 702 is typically written into head units 713 or 715, as shown by arrow 733, and read from tail units 717 or 719, as shown by arrow 739, but may instead be read from the same head or tail unit to which it was written.

The SRAM FIFOS 700 and 702 are both connected to DRAM 460, which allows virtually unlimited expansion of those FIFOS to handle situations in which the SRAM head and tail are full. For example a first of the thirty-two queues, labeled Q-zero, may queue an entry in DRAM 460, as shown by arrow 727, by DMA units acting under direction of the queue manager, instead of being queued in the head or tail of FIFO 700. Entries stored in DRAM 460 return to SRAM unit 709, as shown by arrow 730, extending the length and fall-through time of that FIFO. Diversion from SRAM to DRAM is typically reserved for when the SRAM is full, since DRAM is slower and DMA movement causes additional latency. Thus Q-zero may comprise the entries stored by queue manager 606 in both the FIFO 700 and the DRAM 460. Likewise, information bound for FIFO 702, which may correspond to Q-twenty-seven, for example, can be moved by DMA into DRAM 460, as shown by arrow 735. The capacity for queuing in cost-effective albeit slower DRAM 460 is user-definable during initialization, allowing the queues to change in size as desired. Information queued in DRAM 460 is returned to SRAM unit 717, as shown by arrow 737.

Status for each of the thirty-two hardware queues is conveniently maintained in and accessed from a set 740 of four, thirty-two bit registers, as shown in FIG. 17, in which a specific bit in each register corresponds to a specific queue. The registers are labeled Q-Out_Ready 745, Q-In_Ready 750, Q-Empty 755 and Q-Full 760. If a particular bit is set in the Q-Out_Ready register 750, the queue corresponding to that bit contains information that is ready to be read, while the setting of the same bit in the Q-In_Ready 752 register means that the queue is ready to be written. Similarly, a positive setting of a specific bit in the Q-Empty register 755 means that the queue corresponding to that bit is empty, while a positive setting of a particular bit in the Q-Full register 760 means that the queue corresponding to that bit is full.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 036

Thus Q-Out_Ready 745 contains bits zero 746 through thirty-one 748, including bits twenty-seven 752, twenty-eight 754, twenty-nine 756 and thirty 758. Q-In_Ready 750 contains bits zero 762 through thirty-one 764, including bits twenty-seven 766, twenty-eight 768, twenty-nine 770 and thirty 772. Q-Empty 755 contains bits zero 774 through thirty-one 776, including bits twenty-seven 778, twenty-eight 780, twenty-nine 782 and thirty 784, and Q-full 760 contains bits zero 786 through thirty-one 788, including bits twenty-seven 790, twenty-eight 792, twenty-nine 794 and thirty 796.

Q-zero, corresponding to FIFO 700, is a free buffer queue, which holds a list of addresses for all available buffers. This queue is addressed when the microprocessor or other devices need a free buffer address, and so commonly includes appreciable DRAM 460. Thus a device needing a free buffer address would check with Q-zero to obtain that address. Q-twenty-seven, corresponding to FIFO 702, is a receive buffer descriptor queue. After processing a received frame by the receive sequencer the sequencer looks to store a descriptor for the frame in Q-twenty-seven. If a location for such a descriptor is immediately available in SRAM, bit twenty-seven 766 of Q-In_Ready 750 will be set. If not, the sequencer must wait for the queue manager to initiate a DMA move from SRAM to DRAM, thereby freeing space to store the receive descriptor.

Operation of the queue manager, which manages movement of queue entries between SRAM and the processor, the transmit and receive sequencers, and also between SRAM and DRAM, is shown in more detail in FIG. 18. Requests which utilize the queues include Processor Request 802, Transmit Sequencer Request 804, and Receive Sequencer Request 806. Other requests for the queues are DRAM to SRAM Request 808 and SRAM to DRAM Request 810, which operate on behalf of the queue manager in moving data back and forth between the DRAM and the SRAM head or tail of the queues. Determining which of these various requests will get to use the queue manager in the next cycle is handled by priority logic Arbiter 815. To enable high frequency operation the queue manager is pipelined, with Register A 818 and Register B 820 providing temporary storage, while Status Register 822 maintains status until the next update. The queue manager reserves even cycles for DMA, receive and transmit sequencer requests and odd cycles for processor requests. Dual ported QRAM 825 stores variables regarding each of the queues, the variables for each queue including a Head Write Pointer, Head Read Pointer, Tail Write Pointer and Tail Read Pointer

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 037

corresponding to the queue's SRAM condition, and a Body Write Pointer and Body Read Pointer corresponding to the queue's DRAM condition and the queue's size.

After Arbiter 815 has selected the next operation to be performed, the variables of QRAM 825 are fetched and modified according to the selected operation by a QALU 828, and an SRAM Read Request 830 or an SRAM Write Request 840 may be generated. The variables are updated and the updated status is stored in Status Register 822 as well as QRAM 825. The status is also fed to Arbiter 815 to signal that the operation previously requested has been fulfilled, inhibiting duplication of requests. The Status Register 822 updates the four queue registers Q-Out_Ready 745, Q-In_Ready 750, Q-Empty 755 and Q-Full 760 to reflect the new status of the queue that was accessed. Similarly updated are SRAM Addresses 833, Body Write Request 835 and Body Read Requests 838, which are accessed via DMA to and from SRAM head and tails for that queue. Alternatively, various processes may wish to write to a queue, as shown by Q Write Data 844, which are selected by multiplexor 846, and pipelined to SRAM Write Request 840. The SRAM controller services the read and write requests by writing the tail or reading the head of the accessed queue and returning an acknowledge. In this manner the various queues are utilized and their status updated.

FIGs. 19A-C show a least-recently-used register 900 that is employed for choosing which contexts or CCBs to maintain in INIC cache memory. The INIC in this embodiment can cache up to sixteen CCBs in SRAM at a given time, and so when a new CCB is cached an old one must often be discarded, the discarded CCB usually chosen according to this register 900 to be the CCB that has been used least recently. In this embodiment, a hash table for up to two hundred fifty-six CCBs is also maintained in SRAM, while up to two hundred fifty-six full CCBs are held in DRAM. The least-recently-used register 900 contains sixteen four-bit blocks labeled R0-R15, each of which corresponds to an SRAM cache unit. Upon initialization, the blocks are numbered 0-15, with number 0 arbitrarily stored in the block representing the least recently used (LRU) cache unit and number 15 stored in the block representing the most recently used (MRU) cache unit. FIG. 19A shows the register 900 at an arbitrary time when the LRU block R0 holds the number 9 and the MRU block R15 holds the number 6.

When a different CCB than is currently being held in SRAM is to be cached, the LRU block R0 is read, which in FIG. 19A holds the number 9, and the new CCB is stored in the SRAM cache unit corresponding to number 9. Since the new CCB corresponding to number 9 is now the most recently used CCB, the number 9 is stored in the MRU block, as shown in

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 038

FIG. 19B. The other numbers are all shifted one register block to the left, leaving the number 1 in the LRU block. The CCB that had previously been cached in the SRAM unit corresponding to number 9 has been moved to slower but more cost-effective DRAM.

FIG. 19C shows the result when the next CCB used had already been cached in SRAM. In this example, the CCB was cached in an SRAM unit corresponding to number 10, and so after employment of that CCB, number 10 is stored in the MRU block. Only those numbers which had previously been more recently used than number 10 (register blocks R9-R15) are shifted to the left, leaving the number 1 in the LRU block. In this manner the INIC maintains the most active CCBs in SRAM cache.

In some cases a CCB being used is one that is not desirable to hold in the limited cache memory. For example, it is preferable not to cache a CCB for a context that is known to be closing, so that other cached CCBs can remain in SRAM longer. In this case, the number representing the cache unit holding the decacheable CCB is stored in the LRU block R0 rather than the MRU block R15, so that the decacheable CCB will be replaced immediately upon employment of a new CCB that is cached in the SRAM unit corresponding to the number held in the LRU block R0. FIG. 19D shows the case for which number 8 (which had been in block R9 in FIG. 19C) corresponds to a CCB that will be used and then closed. In this case number 8 has been removed from block R9 and stored in the LRU block R0. All the numbers that had previously been stored to the left of block R9 (R1-R8) are then shifted one block to the right.

FIG. 20 shows some of the logical units employed to operate the least-recently-used register 900. An array of sixteen, three or four input multiplexors 910, of which only multiplexors MUX0, MUX7, MUX8, MUX9 and MUX15 are shown for clarity, have outputs fed into the corresponding sixteen blocks of least-recently-used register 900. For example, the output of MUX0 is stored in block R0, the output of MUX7 is stored in block R7, etc. The value of each of the register blocks is connected to an input for its corresponding multiplexor and also into inputs for both adjacent multiplexors, for use in shifting the block numbers. For instance, the number stored in R8 is fed into inputs for MUX7, MUX8 and MUX9. MUX0 and MUX15 each have only one adjacent block, and the extra input for those multiplexors is used for the selection of LRU and MRU blocks, respectively. MUX15 is shown as a four-input multiplexor, with input 915 providing the number stored on R0.

An array of sixteen comparators 920 each receives the value stored in the corresponding block of the least-recently-used register 900. Each comparator also receives a signal from

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 039

processor 470 along line 935 so that the register block having a number matching that sent by processor 470 outputs true to logic circuits 930 while the other fifteen comparators output false. Logic circuits 930 control a pair of select lines leading to each of the multiplexors, for selecting inputs to the multiplexors and therefore controlling shifting of the register block numbers. Thus select lines 939 control MUX0, select lines 944 control MUX7, select lines 949 control MUX8, select lines 954 control MUX9 and select lines 959 control MUX15.

When a CCB is to be used, processor 470 checks to see whether the CCB matches a CCB currently held in one of the sixteen cache units. If a match is found, the processor sends a signal along line 935 with the block number corresponding to that cache unit, for example number 12. Comparators 920 compare the signal from that line 935 with the block numbers and comparator C8 provides a true output for the block R8 that matches the signal, while all the other comparators output false. Logic circuits 930, under control from the processor 470, use select lines 959 to choose the input from line 935 for MUX15, storing the number 12 in the MRU block R15. Logic circuits 930 also send signals along the pairs of select lines for MUX8 and higher multiplexors, aside from MUX15, to shift their output one block to the left, by selecting as inputs to each multiplexor MUX8 and higher the value that had been stored in register blocks one block to the right (R9-R15). The outputs of multiplexors that are to the left of MUX8 are selected to be constant.

If processor 470 does not find a match for the CCB among the sixteen cache units, on the other hand, the processor reads from LRU block R0 along line 966 to identify the cache corresponding to the LRU block, and writes the data stored in that cache to DRAM. The number that was stored in R0, in this case number 3, is chosen by select lines 959 as input 915 to MUX15 for storage in MRU block R15. The other fifteen multiplexors output to their respective register blocks the numbers that had been stored each register block immediately to the right.

For the situation in which the processor wishes to remove a CCB from the cache after use, the LRU block R0 rather than the MRU block R15 is selected for placement of the number corresponding to the cache unit holding that CCB. The number corresponding to the CCB to be placed in the LRU block R0 for removal from SRAM (for example number 1, held in block R9) is sent by processor 470 along line 935, which is matched by comparator C9. The processor instructs logic circuits 930 to input the number 1 to R0, by selecting with lines 939 input 935 to MUX0. Select lines 954 to MUX9 choose as input the number held in register

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 040

block R8, so that the number from R8 is stored in R9. The numbers held by the other register blocks between R0 and R9 are similarly shifted to the right, whereas the numbers in register blocks to the right of R9 are left constant. This frees scarce cache memory from maintaining closed CCBs for many cycles while their identifying numbers move through register blocks from the MRU to the LRU blocks.

Figure 21 is another diagram of Intelligent Network Interface Card (INIC) 200 of Figure 13. INIC card 200 includes a Physical Layer Interface (PHY) chip 2100, ASIC chip 400 and Dynamic Random Access Memory (DRAM) 460. PHY chip 2100 couples INIC card 200 to network line 210 via a network connector 2101. INIC card 200 is coupled to the CPU of the host (for example, CPU 28 of host 20 of Figure 1) via card edge connector 2107 and PCI bus 257. ASIC chip 400 includes a Media Access Control (MAC) unit 402, a sequencers block 2103, SRAM control 442, SRAM 440, DRAM control 450, a queue manager 2103, a processor 470, and a PCI bus interface unit 468. Structure and operation of queue manager 2103 is described above in connection with Figure 18 and in U.S. Patent Application Serial Number 09/416,925, entitled "Queue System For Microprocessors", attorney docket no. ALA-005, filed October 13, 1999, by Daryl D. Starr and Clive M. Philbrick (the subject matter of which is incorporated herein by reference). Sequencers block 2102 includes a transmit sequencer 2104, a receive sequencer 2105, and configuration registers 2106. A MAC destination address is stored in configuration register 2106. Part of the program code executed by processor 470 is contained in ROM (not shown) and part is located in a writeable control store SRAM (not shown). The program is downloaded into the writeable control store SRAM at initialization from the host 20.

Figure 22 is a more detailed diagram of receive sequencer 2105. Receive sequencer 2105 includes a data synchronization buffer 2200, a packet synchronization sequencer 2201, a data assembly register 2202, a protocol analyzer 2203, a packet processing sequencer 2204, a queue manager interface 2205, and a Direct Memory Access (DMA) control block 2206. The packet synchronization sequencer 2201 and data synchronization buffer 2200 utilize a network-synchronized clock of MAC 402, whereas the remainder of the receive sequencer 2105 utilizes a fixed-frequency clock. Dashed line 2221 indicates the clock domain boundary.

CD Appendix A contains a complete hardware description (verilog code) of an embodiment of receive sequencer 2105. Signals in the verilog code are named to designate their functions. Individual sections of the verilog code are identified and labeled with comment lines. Each of

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 041

these sections describes hardware in a block of the receive sequencer 2105 as set forth below in Table 1.

SECTION OF VERILOG CODE	BLOCK OF FIG. 22
Synchronization Interface	2201
Sync-Buffer Read-Ptr Synchronizers	2201
Packet-Synchronization Sequencer	2201
Data Synchronization Buffer	2201 and 2200
Synchronized Status for Link-Destination-Address	2201
Synchronized Status-Vector	2201
Synchronization Interface	2204
Receive Packet Control and Status	2204
Buffer-Descriptor	2201
Ending Packet Status	2201
AssyReg shift-in. Mac -> AssyReg.	2202 and 2204
Fifo shift-in. AssyReg -> Sram Fifo	2206
Fifo ShiftOut Burst. SramFifo -> DramBuffer	2206
Fly-By Protocol Analyzer; Frame, Network and Transport Layers	2203
Link Pointer	2203
Mac address detection	2203
Magic pattern detection	2203
Link layer and network layer detection	2203
Network counter	2203
Control Packet analysis	2203
Network header analysis	2203
Transport layer counter	2203
Transport header analysis	2203
Pseudo-header stuff	2203
Free-Descriptor Fetch	2205
Receive-Descriptor Store	2205
Receive-Vector Store	2205
Queue-manager interface-mux	2205

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 042

Pause Clock Generator	2201
Pause Timer	2204

TABLE 1

Operation of receive sequencer 2105 of Figures 21 and 22 is now described in connection with the receipt onto INIC card 200 of a TCP/IP packet from network line 210. At initialization time, processor 470 partitions DRAM 460 into buffers. Receive sequencer 2105 uses the buffers in DRAM 460 to store incoming network packet data as well as status information for the packet. Processor 470 creates a 32-bit buffer descriptor for each buffer. A buffer descriptor indicates the size and location in DRAM of its associated buffer. Processor 470 places these buffer descriptors on a "free-buffer queue" 2108 by writing the descriptors to the queue manager 2103. Queue manager 2103 maintains multiple queues including the "free-buffer queue" 2108. In this implementation, the heads and tails of the various queues are located in SRAM 440, whereas the middle portion of the queues are located in DRAM 460.

Lines 2229 comprise a request mechanism involving a request line and address lines. Similarly, lines 2230 comprise a request mechanism involving a request line and address lines. Queue manager 2103 uses lines 2229 and 2230 to issue requests to transfer queue information from DRAM to SRAM or from SRAM to DRAM.

The queue manager interface 2205 of the receive sequencer always attempts to maintain a free buffer descriptor 2207 for use by the packet processing sequencer 2204. Bit 2208 is a ready bit that indicates that free-buffer descriptor 2207 is available for use by the packet processing sequencer 2204. If queue manager interface 2205 does not have a free buffer descriptor (bit 2208 is not set), then queue manager interface 2205 requests one from queue manager 2103 via request line 2209. (Request line 2209 is actually a bus which communicates the request, a queue ID, a read/write signal and data if the operation is a write to the queue.)

In response, queue manager 2103 retrieves a free buffer descriptor from the tail of the "free buffer queue" 2108 and then alerts the queue manager interface 2205 via an acknowledge signal on acknowledge line 2210. When queue manager interface 2205 receives the acknowledge signal, the queue manager interface 2205 loads the free buffer descriptor 2207 and sets the ready bit 2208. Because the free buffer descriptor was in the tail of the free buffer queue in SRAM 440, the queue manager interface 2205 actually receives the free buffer

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 043

descriptor 2207 from the read data bus 2228 of the SRAM control block 442. Packet processing sequencer 2204 requests a free buffer descriptor 2207 via request line 2211. When the queue manager interface 2205 retrieves the free buffer descriptor 2207 and the free buffer descriptor 2207 is available for use by the packet processing sequencer, the queue manager interface 2205 informs the packet processing sequencer 2204 via grant line 2212. By this process, a free buffer descriptor is made available for use by the packet processing sequencer 2204 and the receive sequencer 2105 is ready to processes an incoming packet.

Next, a TCP/IP packet is received from the network line 210 via network connector 2101 and Physical Layer Interface (PHY) 2100. PHY 2100 supplies the packet to MAC 402 via a Media Independent Interface (MII) parallel bus 2109. MAC 402 begins processing the packet and asserts a "start of packet" signal on line 2213 indicating that the beginning of a packet is being received. When a byte of data is received in the MAC and is available at the MAC outputs 2215, MAC 402 asserts a "data valid" signal on line 2214. Upon receiving the "data valid" signal, the packet synchronization sequencer 2201 instructs the data synchronization buffer 2200 via load signal line 2222 to load the received byte from data lines 2215. Data synchronization buffer 2200 is four bytes deep. The packet synchronization sequencer 2201 then increments a data synchronization buffer write pointer. This data synchronization buffer write pointer is made available to the packet processing sequencer 2204 via lines 2216. Consecutive bytes of data from data lines 2215 are clocked into the data synchronization buffer 2200 in this way.

A data synchronization buffer read pointer available on lines 2219 is maintained by the packet processing sequencer 2204. The packet processing sequencer 2204 determines that data is available in data synchronization buffer 2200 by comparing the data synchronization buffer write pointer on lines 2216 with the data synchronization buffer read pointer on lines 2219.

Data assembly register 2202 contains a sixteen-byte long shift register 2217. This register 2217 is loaded serially a single byte at a time and is unloaded in parallel. When data is loaded into register 2217, a write pointer is incremented. This write pointer is made available to the packet processing sequencer 2204 via lines 2218. Similarly, when data is unloaded from register 2217, a read pointer maintained by packet processing sequencer 2204 is incremented. This read pointer is available to the data assembly register 2202 via lines 2220. The packet

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 044

processing sequencer 2204 can therefore determine whether room is available in register 2217 by comparing the write pointer on lines 2218 to the read pointer on lines 2220.

If the packet processing sequencer 2204 determines that room is available in register 2217, then packet processing sequencer 2204 instructs data assembly register 2202 to load a byte of data from data synchronization buffer 2200. The data assembly register 2202 increments the data assembly register write pointer on lines 2218 and the packet processing sequencer 2204 increments the data synchronization buffer read pointer on lines 2219. Data shifted into register 2217 is examined at the register outputs by protocol analyzer 2203 which verifies checksums, and generates "status" information 2223.

DMA control block 2206 is responsible for moving information from register 2217 to buffer 2114 via a sixty-four byte receive FIFO 2110. DMA control block 2206 implements receive FIFO 2110 as two thirty-two byte ping-pong buffers using sixty-four bytes of SRAM 440. DMA control block 2206 implements the receive FIFO using a write-pointer and a read-pointer. When data to be transferred is available in register 2217 and space is available in FIFO 2110, DMA control block 2206 asserts an SRAM write request to SRAM controller 442 via lines 2225. SRAM controller 442 in turn moves data from register 2217 to FIFO 2110 and asserts an acknowledge signal back to DMA control block 2206 via lines 2225. DMA control block 2206 then increments the receive FIFO write pointer and causes the data assembly register read pointer to be incremented.

When thirty-two bytes of data has been deposited into receive FIFO 2110, DMA control block 2206 presents a DRAM write request to DRAM controller 450 via lines 2226. This write request consists of the free buffer descriptor 2207 ORed with a "buffer load count" for the DRAM request address, and the receive FIFO read pointer for the SRAM read address. Using the receive FIFO read pointer, the DRAM controller 450 asserts a read request to SRAM controller 442. SRAM controller 442 responds to DRAM controller 450 by returning the indicated data from the receive FIFO 2110 in SRAM 440 and asserting an acknowledge signal. DRAM controller 450 stores the data in a DRAM write data register, stores a DRAM request address in a DRAM address register, and asserts an acknowledge to DMA control block 2206. The DMA control block 2206 then decrements the receive FIFO read pointer. Then the DRAM controller 450 moves the data from the DRAM write data register to buffer 2114. In this way, as consecutive thirty-two byte chunks of data are stored in SRAM 440, DRAM control block 2206 moves those thirty-two byte chunks of data one at a time from

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 045

SRAM 440 to buffer 2214 in DRAM 460. Transferring thirty-two byte chunks of data to the DRAM 460 in this fashion allows data to be written into the DRAM using the relatively efficient burst mode of the DRAM.

Packet data continues to flow from network line 210 to buffer 2114 until all packet data has been received. MAC 402 then indicates that the incoming packet has completed by asserting an "end of frame" (i.e., end of packet) signal on line 2227 and by presenting final packet status (MAC packet status) to packet synchronization sequencer 2204. The packet processing sequencer 2204 then moves the status 2223 (also called "protocol analyzer status") and the MAC packet status to register 2217 for eventual transfer to buffer 2114. After all the data of the packet has been placed in buffer 2214, status 2223 and the MAC packet status is transferred to buffer 2214 so that it is stored prepended to the associated data as shown in Figure 22.

After all data and status has been transferred to buffer 2114, packet processing sequencer 2204 creates a summary 2224 (also called a "receive packet descriptor") by concatenating the free buffer descriptor 2207, the buffer load-count, the MAC ID, and a status bit (also called an "attention bit"). If the attention bit is a one, then the packet is not a "fast-path candidate"; whereas if the attention bit is a zero, then the packet is a "fast-path candidate". The value of the attention bit represents the result of a significant amount of processing that processor 470 would otherwise have to do to determine whether the packet is a "fast-path candidate". For example, the attention bit being a zero indicates that the packet employs both TCP protocol and IP protocol. By carrying out this significant amount of processing in hardware beforehand and then encoding the result in the attention bit, subsequent decision making by processor 470 as to whether the packet is an actual "fast-path packet" is accelerated. A complete logical description of the attention bit in verilog code is set forth in CD Appendix A in the lines following the heading "Ending Packet Status".

Packet processing sequencer 2204 then sets a ready bit (not shown) associated with summary 2224 and presents summary 2224 to queue manager interface 2205. Queue manager interface 2205 then requests a write to the head of a "summary queue" 2112 (also called the "receive descriptor queue"). The queue manager 2103 receives the request, writes the summary 2224 to the head of the summary queue 2212, and asserts an acknowledge signal back to queue manager interface via line 2210. When queue manager interface 2205 receives the acknowledge, queue manager interface 2205 informs packet processing sequencer 2204

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 046

that the summary 2224 is in summary queue 2212 by clearing the ready bit associated with the summary. Packet processing sequencer 2204 also generates additional status information (also called a "vector") for the packet by concatenating the MAC packet status and the MAC ID. Packet processing sequencer 2204 sets a ready bit (not shown) associated with this vector and presents this vector to the queue manager interface 2205. The queue manager interface 2205 and the queue manager 2103 then cooperate to write this vector to the head of a "vector queue" 2113 in similar fashion to the way summary 2224 was written to the head of summary queue 2112 as described above. When the vector for the packet has been written to vector queue 2113, queue manager interface 2205 resets the ready bit associated with the vector.

Once summary 2224 (including a buffer descriptor that points to buffer 2114) has been placed in summary queue 2112 and the packet data has been placed in buffer 2144, processor 470 can retrieve summary 2224 from summary queue 2112 and examine the "attention bit".

If the attention bit from summary 2224 is a digital one, then processor 470 determines that the packet is not a "fast-path candidate" and processor 470 need not examine the packet headers. Only the status 2223 (first sixteen bytes) from buffer 2114 are DMA transferred to SRAM so processor 470 can examine it. If the status 2223 indicates that the packet is a type of packet that is not to be transferred to the host (for example, a multicast frame that the host is not registered to receive), then the packet is discarded (i.e., not passed to the host). If status 2223 does not indicate that the packet is the type of packet that is not to be transferred to the host, then the entire packet (headers and data) is passed to a buffer on host 20 for "slow-path" transport and network layer processing by the protocol stack of host 20.

If, on the other hand, the attention bit is a zero, then processor 470 determines that the packet is a "fast-path candidate". If processor 470 determines that the packet is a "fast-path candidate", then processor 470 uses the buffer descriptor from the summary to DMA transfer the first approximately 96 bytes of information from buffer 2114 from DRAM 460 into a portion of SRAM 440 so processor 470 can examine it. This first approximately 96 bytes contains status 2223 as well as the IP source address of the IP header, the IP destination address of the IP header, the TCP source address of the TCP header, and the TCP destination address of the TCP header. The IP source address of the IP header, the IP destination address of the IP header, the TCP source address of the TCP header, and the TCP destination address of the TCP header together uniquely define a single connection context (TCB) with which the packet is associated. Processor 470 examines these addresses of the TCP and IP headers and

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 047

determines the connection context of the packet. Processor 470 then checks a list of connection contexts that are under the control of INIC card 200 and determines whether the packet is associated with a connection context (TCB) under the control of INIC card 200.

If the connection context is not in the list, then the "fast-path candidate" packet is determined not to be a "fast-path packet." In such a case, the entire packet (headers and data) is transferred to a buffer in host 20 for "slow-path" processing by the protocol stack of host 20.

If, on the other hand, the connection context is in the list, then software executed by processor 470 including software state machines 2231 and 2232 checks for one of numerous exception conditions and determines whether the packet is a "fast-path packet" or is not a "fast-path packet". These exception conditions include: 1) IP fragmentation is detected; 2) an IP option is detected; 3) an unexpected TCP flag (urgent bit set, reset bit set, SYN bit set or FIN bit set) is detected; 4) the ACK field in the TCP header is before the TCP window, or the ACK field in the TCP header is after the TCP window, or the ACK field in the TCP header shrinks the TCP window; 5) the ACK field in the TCP header is a duplicate ACK and the ACK field exceeds the duplicate ACK count (the duplicate ACK count is a user settable value); and 6) the sequence number of the TCP header is out of order (packet is received out of sequence). If the software executed by processor 470 detects one of these exception conditions, then processor 470 determines that the "fast-path candidate" is not a "fast-path packet." In such a case, the connection context for the packet is "flushed" (the connection context is passed back to the host) so that the connection context is no longer present in the list of connection contexts under control of INIC card 200. The entire packet (headers and data) is transferred to a buffer in host 20 for "slow-path" transport layer and network layer processing by the protocol stack of host 20.

If, on the other hand, processor 470 finds no such exception condition, then the "fast-path candidate" packet is determined to be an actual "fast-path packet". The receive state machine 2232 then processes of the packet through TCP. The data portion of the packet in buffer 2114 is then transferred by another DMA controller (not shown in Figure 21) from buffer 2114 to a host-allocated file cache in storage 35 of host 20. In one embodiment, host 20 does no analysis of the TCP and IP headers of a "fast-path packet". All analysis of the TCP and IP headers of a "fast-path packet" is done on INIC card 20.

Figure 23 is a diagram illustrating the transfer of data of "fast-path packets" (packets of a 64k-byte session layer message 2300) from INIC 200 to host 20. The portion of the diagram

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 048

to the left of the dashed line 2301 represents INIC 200, whereas the portion of the diagram to the right of the dashed line 2301 represents host 20. The 64k-byte session layer message 2300 includes approximately forty-five packets, four of which (2302, 2303, 2304 and 2305) are labeled on Figure 23. The first packet 2302 includes a portion 2306 containing transport and network layer headers (for example, TCP and IP headers), a portion 2307 containing a session layer header, and a portion 2308 containing data. In a first step, portion 2307, the first few bytes of data from portion 2308, and the connection context identifier 2310 of the packet 2300 are transferred from INIC 200 to a 256-byte buffer 2309 in host 20. In a second step, host 20 examines this information and returns to INIC 200 a destination (for example, the location of a file cache 2311 in storage 35) for the data. Host 20 also copies the first few bytes of the data from buffer 2309 to the beginning of a first part 2312 of file cache 2311. In a third step, INIC 200 transfers the remainder of the data from portion 2308 to host 20 such that the remainder of the data is stored in the remainder of first part 2312 of file cache 2311. No network, transport, or session layer headers are stored in first part 2312 of file cache 2311. Next, the data portion 2313 of the second packet 2303 is transferred to host 20 such that the data portion 2313 of the second packet 2303 is stored in a second part 2314 of file cache 2311. The transport layer and network layer header portion 2315 of second packet 2303 is not transferred to host 20. There is no network, transport, or session layer header stored in file cache 2311 between the data portion of first packet 2302 and the data portion of second packet 2303. Similarly, the data portion 2316 of the next packet 2304 of the session layer message is transferred to file cache 2311 so that there is no network, transport, or session layer headers between the data portion of the second packet 2303 and the data portion of the third packet 2304 in file cache 2311. In this way, only the data portions of the packets of the session layer message are placed in the file cache 2311. The data from the session layer message 2300 is present in file cache 2311 as a block such that this block contains no network, transport, or session layer headers.

In the case of a shorter, single-packet session layer message, portions 2307 and 2308 of the session layer message are transferred to 256-byte buffer 2309 of host 20 along with the connection context identifier 2310 as in the case of the longer session layer message described above. In the case of a single-packet session layer message, however, the transfer is completed at this point. Host 20 does not return a destination to INIC 200 and INIC 200 does not transfer subsequent data to such a destination.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 049

CD Appendix B includes a listing of software executed by processor 470 that determines whether a "fast-path candidate" packet is or is not a "fast-path packet". An example of the instruction set of processor 470 is found starting on page 79 of the Provisional U.S. Patent Application Serial No. 60/061,809, entitled "Intelligent Network Interface Card And System For Protocol Processing", filed October 14, 1997 (the subject matter of this provisional application is incorporated herein by reference).

CD Appendix C includes device driver software executable on host 20 that interfaces the host 20 to INIC card 200. There is also ATCP code that executes on host 20. This ATCP code includes: 1) a "free BSD" stack (available from the University of California, Berkeley) that has been modified slightly to make it run on the NT4 operating system (the "free BSD" stack normally runs on a UNIX machine), and 2) code added to the free BSD stack between the session layer above and the device driver below that enables the BSD stack to carry out "fast-path" processing in conjunction with INIC 200.

TRANSMIT FAST-PATH PROCESSING: The following is an overview of one embodiment of a transmit fast-path flow once a command has been posted (for additional information, see provisional application 60/098,296, filed August 27, 1998). The transmit request may be a segment that is less than the MSS, or it may be as much as a full 64K session layer packet. The former request will go out as one segment, the latter as a number of MSS-sized segments. The transmitting CCB must hold on to the request until all data in it has been transmitted and ACKed. Appropriate pointers to do this are kept in the CCB. To create an output TCP/IP segment, a large DRAM buffer is acquired from the Q_FREEL queue. Then data is DMAd from host memory into the DRAM buffer to create an MSS-sized segment. This DMA also checksums the data. The TCP/IP header is created in SRAM and DMAd to the front of the payload data. It is quicker and simpler to keep a basic frame header (i.e., a template header) permanently in the CCB and DMA this directly from the SRAM CCB buffer into the DRAM buffer each time. Thus the payload checksum is adjusted for the pseudo-header (i.e., the template header) and placed into the TCP header prior to DMAing the header from SRAM. Then the DRAM buffer is queued to the appropriate Q_UXMT transmit queue. The final step is to update various window fields etc in the CCB. Eventually either the entire request will have been sent and ACKed, or a retransmission timer will expire in which case the context is flushed to the host. In either case, the INIC will place a command response in the

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 050

response queue containing the command buffer from the original transmit command and appropriate status.

The above discussion has dealt with how an actual transmit occurs. However the real challenge in the transmit processor is to determine whether it is appropriate to transmit at the time a transmit request arrives, and then to continue to transmit for as long as the transport protocol permits. There are many reasons not to transmit: the receiver's window size is less than or equal to zero, the persist timer has expired, the amount to send is less than a full segment and an ACK is expected/outstanding, the receiver's window is not half-open, etc. Much of transmit processing will be in determining these conditions.

The fast-path is implemented as a finite state machine (FSM) that covers at least three layers of the protocol stack, i.e., IP, TCP, and Session. The following summarizes the steps involved in normal fast-path transmit command processing: 1) get control of the associated CCB (gotten from the command): this involves locking the CCB to stop other processing (e.g. Receive) from altering it while this transmit processing is taking place. 2) Get the CCB into an SRAM CCB buffer. There are sixteen of these buffers in SRAM and they are not flushed to DRAM until the buffer space is needed by other CCBs. Acquisition and flushing of these CCB buffers is controlled by a hardware LRU mechanism. Thus getting into a buffer may involve flushing another CCB from its SRAM buffer. 3) Process the send command (EX_SCMD) event against the CCB's FSM.

Each event and state intersection provides an action to be executed and a new state. The following is an example of the state/event transition, the action to be executed and the new state for the SEND command while in transmit state IDLE (SX_IDLE). The action from this state/event intersection is AX_NUCMD and the next state is XMIT COMMAND ACTIVE (SX_XMIT). To summarize, a command to transmit data has been received while transmit is currently idle. The action performs the following steps: 1) Store details of the command into the CCB. 2) Check that it is okay to transmit now (e.g. send window is not zero). 3) If output is not possible, send the Check Output event to Q_EVENT1 queue for the Transmit CCB's FSM and exit. 4) Get a DRAM 2K-byte buffer from the Q-FREEL queue into which to move the payload data. 5) DMA payload data from the addresses in the scatter/gather lists in the command into an offset in the DRAM buffer that leaves space for the frame header. These DMAs will provide the checksum of the payload data. 6) Concurrently with the above DMA, fill out variable details in the frame header template in the CCB. Also get the IP and TCP

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 051

header checksums while doing this. Note that base IP and TCP headers checksums are kept in the CCB, and these are simply updated for fields that vary per frame, viz. IP Id, IP length, IP checksum, TCP sequence and ACK numbers, TCP window size, TCP flags and TCP checksum. 7) When the payload is complete, DMA the frame header from the CCB to the front of the DRAM buffer. 8) Queue the DRAM buffer (i.e., queue a buffer descriptor that points to the DRAM buffer) to the appropriate Q_UXMT queue for the interface for this CCB. 9) Determine if there is more payload in the command. If so, save the current command transfer address details in the CCB and send a CHECK OUTPUT event via the Q_EVENT1 queue to the Transmit CCB. If not, send the ALL COMMAND DATA SENT (EX_ACDS) event to the Transmit CCB. 10) Exit from Transmit FSM processing.

Code that implements an embodiment of the Transmit FSM (transmit software state machine 2231 of Figure 21) is found in CD Appendix B. In one embodiment, fast-path transmit processing is controlled using write only transmit configuration register (XmtCfg). Register XmtCfg has the following portions: 1) Bit 31 (name: Reset). Writing a one (1) will force reset asserted to the transmit sequencer of the channel selected by XcvSel. 2) Bit 30 (name: XmtEn). Writing a one (1) allows the transmit sequencer to run. Writing a zero (0) causes the transmit sequencer to halt after completion of the current packet. 3) Bit 29 (name: PauseEn). Writing a one (1) allows the transmit sequencer to stop packet transmission, after completion of the current packet, whenever the receive sequencer detects an 802.3X pause command packet. 4) Bit 28 (name: LoadRng). Writing a one (1) causes the data in RcvAddrB[10:00] to be loaded in to the Mac's random number register for use during collision back-offs. 5) Bits 27:20 (name: Reserved). 6) Bits 19:15 (name: FreeQId). Selects the queue to which the freed buffer descriptors will be written once the packet transmission has been terminated, either successfully or unsuccessfully. 7) Bits 14:10 (name: XmtQId). Selects the queue from which the transmit buffer descriptors will be fetched for data packets. 8) Bits 09:05 (name: CtrlQId). Selects the queue from which the transmit buffer descriptors will be fetched for control packets. These packets have transmission priority over the data packets and will be exhausted before data packets will be transmitted. 9) Bits 04:00 (name: VectQId). Selects the queue to which the transmit vector data is written after the completion of each packet transmit. In some embodiments, transmit sequencer 2104 of Figure 21 retrieves buffer descriptors from two transmit queues, one of the queues having a higher transmission priority than the other. The higher transmission priority transmit queue is used for the

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 052

transmission of TCP ACKs, whereas the lower transmission priority transmit queue is used for the transmission of other types of packets. ACKs may be transmitted in accordance with techniques set forth in U.S. Patent Application Serial No. 09/802,426 (the subject matter of which is incorporated herein by reference). In some embodiments, the processor that executes the Transmit FSM, the receive and transmit sequencers, and the host processor that executes the protocol stack are all realized on the same printed circuit board. The printed circuit board may, for example, be a card adapted for coupling to another computer.

All told, the above-described devices and systems for processing of data communication result in dramatic reductions in the time and host resources required for processing large, connection-based messages. Protocol processing speed and efficiency is tremendously accelerated by specially designed protocol processing hardware as compared with a general purpose CPU running conventional protocol software, and interrupts to the host CPU are also substantially reduced. These advantages can be provided to an existing host by addition of an intelligent network interface card (INIC), or the protocol processing hardware may be integrated with the CPU. In either case, the protocol processing hardware and CPU intelligently decide which device processes a given message, and can change the allocation of that processing based upon conditions of the message.

DISCLOSURE FROM PROVISIONAL APPLICATION 60/061,809.

BACKGROUND OF THE INVENTION.

Network processing as it exists today is a costly and inefficient use of system resources. A 200 MHz Pentium-Pro is typically consumed simply processing network data from a 100Mb/second-network connection. The reasons that this processing is so costly are described here.

TOO MANY DATA MOVES.

When network packet arrives at a typical network interface card (NIC), the NIC moves the data into pre-allocated network buffers in system main memory. From there the data is read into the CPU cache so that it can be checksummed (assuming of course that the protocol in use requires checksums. Some, like IPX, do not.). Once the data has been fully processed by the protocol stack, it can then be moved into its final destination in memory. Since the

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 053

CPU is moving the data, and must read the destination cache line in before it can fill it and write it back out, this involves at a minimum two more trips across the system memory bus. In short, the best one can hope for is that the data will get moved across the system memory bus four times before it arrives in its final destination. It can, and does, get worse. If the data happens to get invalidated from system cache after it has been checksummed, then it must get pulled back across the memory bus before it can be moved to its final destination. Finally, on some systems, including Windows NT 4.0, the data gets copied yet another time while being moved up the protocol stack. In NT 4.0, this occurs between the miniport driver interface and the protocol driver interface. This can add up to a whopping eight trips across the system memory bus (the four trips described above, plus the move to replenish the cache, plus three more to copy from the miniport to the protocol driver). That's enough to bring even today's advanced memory busses to their knees.

TOO MUCH PROCESSING BY THE CPU.

In all but the original move from the NIC to system memory, the system CPU is responsible for moving the data. This is particularly expensive because while the CPU is moving this data it can do nothing else. While moving the data the CPU is typically stalled waiting for the relatively slow memory to satisfy its read and write requests. A CPU, which can execute an instruction every 5 nanoseconds, must now wait as long as several hundred nanoseconds for the memory controller to respond before it can begin its next instruction. Even today's advanced pipelining technology doesn't help in these situations because that relies on the CPU being able to do useful work while it waits for the memory controller to respond. If the only thing the CPU has to look forward to for the next several hundred instructions is more data moves, then the CPU ultimately gets reduced to the speed of the memory controller.

Moving all this data with the CPU slows the system down even after the data has been moved. Since both the source and destination cache lines must be pulled into the CPU cache when the data is moved, more than 3k of instructions and or data resident in the CPU cache must be flushed or invalidated for every 1500 byte frame. This is of course assuming a combined instruction and data second level cache, as is the case with the Pentium processors. After the data has been moved, the former resident of the cache will likely need to be pulled back in, stalling the CPU even when we are not performing network processing. Ideally a

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 054

system would never have to bring network frames into the CPU cache, instead reserving that precious commodity for instructions and data that are referenced repeatedly and frequently.

But the data movement is not the only drain on the CPU. There is also a fair amount of processing that must be done by the protocol stack software. The most obvious expense is calculating the checksum for each TCP segment (or UDP datagram). Beyond this, however, there is other processing to be done as well. The TCP connection object must be located when a given TCP segment arrives, IP header checksums must be calculated, there are buffer and memory management issues, and finally there is also the significant expense of interrupt processing which we will discuss in the following section.

TOO MANY INTERRUPTS.

A 64k SMB request (write or read-reply) is typically made up of 44 TCP segments when running over Ethernet (1500 byte MTU). Each of these segments may result in an interrupt to the CPU. Furthermore, since TCP must acknowledge all of this incoming data, it's possible to get another 44 transmit-complete interrupts as a result of sending out the TCP acknowledgements. While this is possible, it is not terribly likely. Delayed ACK timers allow us to acknowledge more than one segment at a time. And delays in interrupt processing may mean that we are able to process more than one incoming network frame per interrupt. Nevertheless, even if we assume four incoming frames per input, and an acknowledgement for every two segments (as is typical per the ACK-every-other-segment property of TCP), we are still left with 33 interrupts per 64k SMB request.

Interrupts tend to be very costly to the system. Often when a system is interrupted, important information must be flushed or invalidated from the system cache so that the interrupt routine instructions, and needed data can be pulled into the cache. Since the CPU will return to its prior location after the interrupt, it is likely that the information flushed from the cache will immediately need to be pulled back into the cache.

What's more, interrupts force a pipeline flush in today's advanced processors. While the processor pipeline is an extremely efficient way of improving CPU performance, it can be expensive to get going after it has been flushed.

Finally, each of these interrupts results in expensive register accesses across the peripheral bus (PCI). This is discussed more in the following section.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 055

INEFFICIENT USE OF THE PERIPHERAL BUS (PCI).

We noted earlier that when the CPU has to access system memory, it may be stalled for several hundred nanoseconds. When it has to read from PCI, it may be stalled for many microseconds. This happens every time the CPU takes an interrupt from a standard NIC. The first thing the CPU must do when it receives one of these interrupts is to read the NIC Interrupt Status Register (ISR) from PCI to determine the cause of the interrupt. The most troubling thing about this is that since interrupt lines are shared on PC-based systems, we may have to perform this expensive PCI read even when the interrupt is not meant for us.

There are other peripheral bus inefficiencies as well. Typical NICs operate using descriptor rings. When a frame arrives, the NIC reads a receive descriptor from system memory to determine where to place the data. Once the data has been moved to main memory, the descriptor is then written back out to system memory with status about the received frame. Transmit operates in a similar fashion. The CPU must notify that NIC that it has a new transmit. The NIC will read the descriptor to locate the data, read the data itself, and then write the descriptor back with status about the send. Typically on transmits the NIC will then read the next expected descriptor to see if any more data needs to be sent. In short, each receive or transmit frame results in 3 or 4 separate PCI reads or writes (not counting the status register read).

SUMMARY OF THE INVENTION.

Alacritech was formed with the idea that the network processing described above could be offloaded onto a cost-effective Intelligent Network Interface Card (INIC). With the Alacritech INIC, we address each of the above problems, resulting in the following advancements:

1. The vast majority of the data is moved directly from the INIC into its final destination. A single trip across the system memory bus.

2. There is no header processing, little data copying, and no checksumming required by the CPU. Because of this, the data is never moved into the CPU cache, allowing the system to keep important instructions and data resident in the CPU cache.

3. Interrupts are reduced to as little as 4 interrupts per 64k SMB read and 2 per 64k SMB write.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 056

4. There are no CPU reads over PCI and there are fewer PCI operations per receive or transmit transaction.

In the remainder of this document we will describe how we accomplish the above.

PERFORM TRANSPORT LEVEL PROCESSING ON THE INIC.

In order to keep the system CPU from having to process the packet headers or checksum the packet, we must perform this task on the INIC. This is a daunting task. There are more than 20,000 lines of C code that make up the FreeBSD TCP/IP protocol stack. Clearly this is more code than could be efficiently handled by a competitively priced network card. Furthermore, as noted above, the TCP/IP protocol stack is complicated enough to consume a 200 MHz Pentium-Pro. Clearly in order to perform this function on an inexpensive card, we need special network processing hardware as opposed to simply using a general purpose CPU.

ONLY SUPPORT TCP/IP.

In this section we introduce the notion of a "context". A context is required to keep track of information that spans many, possibly discontiguous, pieces of information. When processing TCP/IP data, there are actually two contexts that must be maintained. The first context is required to reassemble IP fragments. It holds information about the status of the IP reassembly as well as any checksum information being calculated across the IP datagram (UDP or TCP). This context is identified by the IP_ID of the datagram as well as the source and destination IP addresses. The second context is required to handle the sliding window protocol of TCP. It holds information about which segments have been sent or received, and which segments have been acknowledged, and is identified by the IP source and destination addresses and TCP source and destination ports.

If we were to choose to handle both contexts in hardware, we would have to potentially keep track of many pieces of information. One such example is a case in which a single 64k SMB write is broken down into 44 1500 byte TCP segments, which are in turn broken down into 131 576 byte IP fragments, all of which can come in any order (though the maximum window size is likely to restrict the number of outstanding segments considerably).

Fortunately, TCP performs a Maximum Segment Size negotiation at connection establishment time, which should prevent IP fragmentation in nearly all TCP connections. The

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 057

only time that we should end up with fragmented TCP connections is when there is a router in the middle of a connection which must fragment the segments to support a smaller MTU. The only networks that use a smaller MTU than Ethernet are serial line interfaces such as SLIP and PPP. At the moment, the fastest of these connections only run at 128k (ISDN) so even if we had 256 of these connections, we would still only need to support 34Mb/sec, or a little over three 10bT connections worth of data. This is not enough to justify any performance enhancements that the INIC offers. If this becomes an issue at some point, we may decide to implement the MTU discovery algorithm, which should prevent TCP fragmentation on all connections (unless an ICMP redirect changes the connection route while the connection is established).

With this in mind, it seems a worthy sacrifice to not attempt to handle fragmented TCP segments on the INIC. UDP is another matter. Since UDP does not support the notion of a Maximum Segment Size, it is the responsibility of IP to break down a UDP datagram into MTU sized packets. Thus, fragmented UDP datagrams are very common. The most common UDP application running today is NFSV2 over UDP. While this is also the most common version of NFS running today, the current version of Solaris being sold by Sun Microsystems runs NFSV3 over TCP by default. We can expect to see the NFSV2/UDP traffic start to decrease over the coming years. In summary, we will only offer assistance to non-fragmented TCP connections on the INIC.

DON'T HANDLE TCP "EXCEPTIONS".

As noted above, we won't provide support for fragmented TCP segments on the INIC. We have also opted to not handle TCP connection and breakdown. Here is a list of other TCP "exceptions" which we have elected to not handle on the INIC:

Fragmented Segments –Discussed above.

Retransmission Timeout – Occurs when we do not get an acknowledgement for previously sent data within the expected time period.

Out of order segments – Occurs when we receive a segment with a sequence number other than the next expected sequence number.

FIN segment – Signals the close of the connection.

Since we have now eliminated support for so many different code paths, it might seem hardly worth the trouble to provide any assistance by the card at all. This is not the case.

[redacted]

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 058

According to W. Richard Stevens and Gary Write in their book "TCP/IP Illustrated Volume 2", TCP operates without experiencing any exceptions between 97 and 100 percent of the time in local area networks. As network, router, and switch reliability improve this number is likely to only improve with time.

TWO MODES OF OPERATION.

So the next question is what to do about the network packets that do not fit our criteria. The answer shown in Fig. 24 is to use two modes of operation: One in which the network frames are processed on the INIC through TCP and one in which the card operates like a typical dumb NIC. We call these two modes fast-path, and slow-path. In the slow-path case, network frames are handed to the system at the MAC layer and passed up through the host protocol stack like any other network frame. In the fast path case, network data is given to the host after the headers have been processed and stripped.

The transmit case works in much the same fashion. In slow-path mode the packets are given to the INIC with all of the headers attached. The INIC simply sends these packets out as if it were a dumb NIC. In fast-path mode, the host gives raw data to the INIC which it must carve into MSS sized segments, add headers to the data, perform checksums on the segment, and then send it out on the wire.

THE TCB CACHE.

Consider a situation in which a TCP connection is being handled by the card and a fragmented TCP segment for that connection arrives. In this situation, it will be necessary for the card to turn control of this connection over to the host.

This introduces the notion of a Transmit Control Block (TCB) cache. A TCB is a structure that contains the entire context associated with a connection. This includes the source and destination IP addresses and source and destination TCP ports that define the connection. It also contains information about the connection itself such as the current send and receive sequence numbers, and the first-hop MAC address, etc. The complete set of TCBs exists in host memory, but a subset of these may be "owned" by the card at any given time. This subset is the TCB cache. The INIC can own up to 256 TCBs at any given time.

TCBs are initialized by the host during TCP connection setup. Once the connection has achieved a "steady-state" of operation, its associated TCB can then be turned over to the INIC,

[redacted]

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 059

putting us into fast-path mode. From this point on, the INIC owns the connection until either a FIN arrives signaling that the connection is being closed, or until an exception occurs which the INIC is not designed to handle (such as an out of order segment). When any of these conditions occur, the INIC will then flush the TCB back to host memory, and issue a message to the host telling it that it has relinquished control of the connection, thus putting the connection back into slow-path mode. From this point on, the INIC simply hands incoming segments that are destined for this TCB off to the host with all of the headers intact.

Note that when a connection is owned by the INIC, the host is not allowed to reference the corresponding TCB in host memory as it will contain invalid information about the state of the connection.

TCP HARDWARE ASSISTANCE.

When a frame is received by the INIC, it must verify it completely before it even determines whether it belongs to one of its TCBs or not. This includes all header validation (is it IP, IPV4 or V6, is the IP header checksum correct, is the TCP checksum correct, etc). Once this is done it must compare the source and destination IP address and the source and destination TCP port with those in each of its TCBs to determine if it is associated with one of its TCBs. This is an expensive process. To expedite this, we have added several features in hardware to assist us. The header is fully parsed by hardware and its type is summarized in a single status word. The checksum is also verified automatically in hardware, and a hash key is created out of the IP addresses and TCP ports to expedite TCB lookup. For full details on these and other hardware optimizations, refer to the INIC Hardware Specification sections (Heading 8).

With the aid of these and other hardware features, much of the work associated with TCP is done essentially for free. Since the card will automatically calculate the checksum for TCP segments, we can pass this on to the host, even when the segment is for a TCB that the INIC does not own.

TCP SUMMARY.

By moving TCP processing down to the INIC we have offloaded the host of a large amount of work. The host no longer has to pull the data into its cache to calculate the TCP

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 060

checksum. It does not have to process the packet headers, and it does not have to generate TCP ACKs. We have achieved most of the goals outlined above, but we are not done yet.

TRANSPORT LAYER INTERFACE.

This section defines the INIC's relation to the hosts transport layer interface (Called TDI or Transport Driver Interface in Windows NT). For full details on this interface, refer to the Alacritech TCP (ATCP) driver specification (Heading 4).

RECEIVE.

Simply implementing TCP on the INIC does not allow us to achieve our goal of landing the data in its final destination. Somehow the host has to tell the INIC where to put the data. This is a problem in that the host cannot do this without knowing what the data actually is. Fortunately, NT has provided a mechanism by which a transport driver can "indicate" a small amount of data to a client above it while telling it that it has more data to come. The client, having then received enough of the data to know what it is, is then responsible for allocating a block of memory and passing the memory address or addresses back down to the transport driver, which is in turn responsible for moving the data into the provided location.

We will make use of this feature by providing a small amount of any received data to the host, with a notification that we have more data pending. When this small amount of data is passed up to the client, and it returns with the address in which to put the remainder of the data, our host transport driver will pass that address to the INIC which will DMA the remainder of the data into its final destination.

Clearly there are circumstances in which this does not make sense. When a small amount of data (500 bytes for example), with a push flag set indicating that the data must be delivered to the client immediately, it does not make sense to deliver some of the data directly while waiting for the list of addresses to DMA the rest. Under these circumstances, it makes more sense to deliver the 500 bytes directly to the host, and allow the host to copy it into its final destination. While various ranges are feasible, it is currently preferred that anything less than a segment's (1500 bytes) worth of data will be delivered directly to the host, while anything more will be delivered as a small piece which may be128 bytes, while waiting until receiving the destination memory address before moving the rest.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 061

The trick then is knowing when the data should be delivered to the client or not. As we've noted, a push flag indicates that the data should be delivered to the client immediately, but this alone is not sufficient. Fortunately, in the case of NetBIOS transactions (such as SMB), we are explicitly told the length of the session message in the NetBIOS header itself. With this we can simply indicate a small amount of data to the host immediately upon receiving the first segment. The client will then allocate enough memory for the entire NetBIOS transaction, which we can then use to DMA the remainder of the data into as it arrives. In the case of a large (56k for example) NetBIOS session message, all but the first couple hundred bytes will be DMA'd to their final destination in memory.

But what about applications that do not reside above NetBIOS? In this case we can not rely on a session level protocol to tell us the length of the transaction. Under these circumstances we will buffer the data as it arrives until A) we have receive some predetermined number of bytes such as 8k, or B) some predetermined period of time passes between segments or C) we get a push flag. If after any of these conditions occur we will then indicate some or all of the data to the host depending on the amount of data buffered. If the data buffered is greater than about 1500 bytes we must then also wait for the memory address to be returned from the host so that we may then DMA the remainder of the data.

TRANSMIT.

The transmit case is much simpler. In this case the client (NetBIOS for example) issues a TDI Send with a list of memory addresses which contain data that it wishes to send along with the length. The host can then pass this list of addresses and length off to the INIC. The INIC will then pull the data from its source location in host memory, as it needs it, until the complete TDI request is satisfied.

AFFECTS ON INTERRUPTS.

Note that when we receive a large SMB transaction, for example, that there are two interactions between the INIC and the host. The first in which the INIC indicates a small amount of the transaction to the host, and the second in which the host provides the memory location(s) in which the INIC places the remainder of the data. This results in only two interrupts from the INIC. The first when it indicates the small amount of data and the second after it has finished filling in the host memory given to it. A drastic reduction from the 33/64k

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 062

SMB request that we estimate at the beginning of this section. On transmit, we actually only receive a single interrupt when the send command that has been given to the INIC completes.

TRANSPORT LAYER INTERFACE SUMMARY.

Having now established our interaction with Microsoft's TDI interface, we have achieved our goal of landing most of our data directly into its final destination in host memory. We have also managed to transmit all data from its original location on host memory. And finally, we have reduced our interrupts to 2 per 64k SMB read and 1 per 64k SMB write. The only thing that remains in our list of objectives is to design an efficient host (PCI) interface.

HOST (PCI) INTERFACE.

In this section we define the host interface. For a more detailed description, refer to the "Host Interface Strategy for the Alacritech INIC" section (Heading 3).

AVOID PCI READS.

One of our primary objectives in designing the host interface of the INIC was to eliminate PCI reads in either direction. PCI reads are particularly inefficient in that they completely stall the reader until the transaction completes. As noted above, this could hold a CPU up for several microseconds, a thousand times the time typically required to execute a single instruction. PCI writes on the other hand, are usually buffered by the memory-bus⇔PCI-bridge allowing the writer to continue on with other instructions. This technique is known as "posting".

MEMORY-BASED STATUS REGISTER.

The only PCI read that is required by most NICs is the read of the interrupt status register. This register gives the host CPU information about what event has caused an interrupt (if any). In the design of our INIC we have elected to place this necessary status register into host memory. Thus, when an event occurs on the INIC, it writes the status register to an agreed upon location in host memory. The corresponding driver on the host reads this local register to determine the cause of the interrupt. The interrupt lines are held high until the host clears the interrupt by writing to the INIC's Interrupt Clear Register. Shadow registers are maintained on the INIC to ensure that events are not lost.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 063

BUFFER ADDRESSES ARE PUSHED TO THE INIC.

Since it is imperative that our INIC operate as efficiently as possible, we must also avoid PCI reads from the INIC. We do this by pushing our receive buffer addresses to the INIC. As mentioned at the beginning of this section, most NICs work on a descriptor queue algorithm in which the NIC reads a descriptor from main memory in order to determine where to place the next frame. We will instead write receive buffer addresses to the INIC as receive buffers are filled. In order to avoid having to write to the INIC for every receive frame, we instead allow the host to pass off a pages worth (4k) of buffers in a single write.

SUPPORT SMALL AND LARGE BUFFERS ON RECEIVE.

In order to reduce further the number of writes to the INIC, and to reduce the amount of memory being used by the host, we support two different buffer sizes. A small buffer contains roughly 200 bytes of data payload, as well as extra fields containing status about the received data bringing the total size to 256 bytes. We can therefore pass 16 of these small buffers at a time to the INIC. Large buffers are 2k in size. They are used to contain any fast or slow-path data that does not fit in a small buffer. Note that when we have a large fast-path receive, a small buffer will be used to indicate a small piece of the data, while the remainder of the data will be DMA'd directly into memory. Large buffers are never passed to the host by themselves, instead they are always accompanied by a small buffer which contains status about the receive along with the large buffer address. By operating in the manner, the driver must only maintain and process the small buffer queue. Large buffers are returned to the host by virtue of being attached to small buffers. Since large buffers are 2k in size they are passed to the INIC 2 buffers at a time.

COMMAND AND RESPONSE BUFFERS.

In addition to needing a manner by which the INIC can pass incoming data to us, we also need a manner by which we can instruct the INIC to send data. Plus, when the INIC indicates a small amount of data in a large fast-path receive, we need a method of passing back the address or addresses in which to put the remainder of the data. We accomplish both of these with the use of a command buffer. Sadly, the command buffer is the only place in which we must violate our rule of only pushing data across PCI. For the command buffer, we write

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 064

the address of command buffer to the INIC. The INIC then reads the contents of the command buffer into its memory so that it can execute the desired command. Since a command may take a relatively long time to complete, it is unlikely that command buffers will complete in order. For this reason we also maintain a response buffer queue. Like the small and large receive buffers, a page worth of response buffers is passed to the INIC at a time. Response buffers are only 32 bytes, so we have to replenish the INIC's supply of them relatively infrequently. The response buffers only purpose is to indicate the completion of the designated command buffer, and to pass status about the completion.

EXAMPLES.

In this section we will provide a couple of examples describing some of the differing data flows that we might see on the Alacritech INIC.

FAST-PATH 56K NETBIOS SESSION MESSAGE.

Let's say a 56k NetBIOS session message is received on the INIC. The first segment will contain the NetBIOS header, which contains the total NetBIOS length. A small chunk of this first segment is provided to the host by filling in a small receive buffer, modifying the interrupt status register on the host, and raising the appropriate interrupt line. Upon receiving the interrupt, the host will read the ISR, clear it by writing back to the INIC's Interrupt Clear Register, and will then process its small receive buffer queue looking for receive buffers to be processed. Upon finding the small buffer, it will indicate the small amount of data up to the client to be processed by NetBIOS. It will also, if necessary, replenish the receive buffer pool on the INIC by passing off a pages worth of small buffers. Meanwhile, the NetBIOS client will allocate a memory pool large enough to hold the entire NetBIOS message, and will pass this address or set of addresses down to the transport driver. The transport driver will allocate an INIC command buffer, fill it in with the list of addresses, set the command type to tell the INIC that this is where to put the receive data, and then pass the command off to the INIC by writing to the command register. When the INIC receives the command buffer, it will DMA the remainder of the NetBIOS data, as it is received, into the memory address or addresses designated by the host. Once the entire NetBIOS transaction is complete, the INIC will complete the command by writing to the response buffer with the appropriate status and command buffer identifier.

[redacted]

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 065

In this example, we have two interrupts, and all but a couple hundred bytes are DMA'd directly to their final destination. On PCI we have two interrupt status register writes, two interrupt clear register writes, a command register write, a command read, and a response buffer write.

With a standard NIC this would result in an estimated 30 interrupts, 30 interrupt register reads, 30 interrupt clear writes, and 58 descriptor reads and writes. Plus the data will get moved anywhere from 4 to 8 times across the system memory bus.

SLOW-PATH RECEIVE.

If the INIC receives a frame that does not contain a TCP segment for one of its TCB's, it simply passes it to the host as if it were a dumb NIC. If the frame fits into a small buffer (~200 bytes or less), then it simply fills in the small buffer with the data and notifies the host. Otherwise it places the data in a large buffer, writes the address of the large buffer into a small buffer, and again notifies the host. The host, having received the interrupt and found the completed small buffer, checks to see if the data is contained in the small buffer, and if not, locates the large buffer. Having found the data, the host will then pass the frame upstream to be processed by the standard protocol stack. It must also replenish the INIC's small and large receive buffer pool if necessary.

With the INIC, this will result in one interrupt, one interrupt status register write and one interrupt clear register write as well as a possible small and or large receive buffer register write. The data will go through the normal path although if it is TCP data then the host will not have to perform the checksum.

With a standard NIC this will result in a single interrupt, an interrupt status register read, an interrupt clear register write, and a descriptor read and write. The data will get processed as it would by the INIC, except for a possible extra checksum.

FAST-PATH 400 BYTE SEND.

In this example, lets assume that the client has a small amount of data to send. It will issue the TDI Send to the transport driver which will allocate a command buffer, fill it in with the address of the 400 byte send, and set the command to indicate that it is a transmit. It will then pass the command off to the INIC by writing to the command register. The INIC will then DMA the 400 bytes into its own memory, prepare a frame with the appropriate

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 066

checksums and headers, and send the frame out on the wire. After it has received the acknowledgement it will then notify the host of the completion by writing to a response buffer.

With the INIC, this will result in one interrupt, one interrupt status register write, one interrupt clear register write, a command buffer register write a command buffer read, and a response buffer write. The data is DMA'd directly from the system memory.

With a standard NIC this will result in a single interrupt, an interrupt status register read, an interrupt clear register write, and a descriptor read and write. The data would get moved across the system bus a minimum of 4 times. The resulting TCP ACK of the data, however, would add yet another interrupt, another interrupt status register read, interrupt clear register write, a descriptor read and write, and yet more processing by the host protocol stack.

HOST INTERFACE STRATEGY FOR THE ALACRITECH INIC.

This section describes the host interface strategy for the Alacritech Intelligent Network Interface Card (INIC). The goal of the Alacritech INIC is to not only process network data through TCP, but also to provide zero-copy support for the SMP upper-layer protocol. It achieves this by supporting two paths for sending and receiving data, the fast-path and the slow-path. The fast path data flow corresponds to connections that are maintained on the NIC, while slow-path traffic corresponds to network data for which the NIC does not have a connection. The fast-path flow works by passing a header to the host and subsequently holding further data for that connection on the card until the host responds via an INIC command with a set of buffers into which to place the accumulated data. In the slow-path data flow, the INIC will be operating as a "dumb" NIC, so that these packets are simply dumped into frame buffers on the host as they arrive. To do either path requires a pool of smaller buffers to be used for headers and a pool of data buffers for frames/data that are too large for the header buffer, with both pools being managed by the INIC. This section discusses how these two pools of data are managed as well as how buffers are associated with a given context.

RECEIVE INTERFACE.

The varying requirements of the fast and slow paths and a desire to save PCI bandwidth are the driving forces behind the host interface that is described herein. As mentioned above, the fast-path flow puts a header into a header buffer that is then forwarded to the host. The host uses the header to determine what further data is following, allocates the necessary host

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 067

buffers, and these are passed back to the INIC via a command to the INIC. The INIC then fills these buffers from data it was accumulating on the card and notifies the host by sending a response to the command. Alternatively, the fast-path may receive a header and data that is a complete request, but that is also too large for a header buffer. This results in a header and data buffer being passed to the host. This latter flow is identical to the slow-path flow, which also puts all the data into the header buffer or, if the header is too small, uses a large (2K) host buffer for all the data. This means that on the unsolicited receive path, the host will only see either a header buffer or a header and at most, one data buffer. Note that data is never split between a header and a data buffer.

Fig. 25 illustrates both situations. Since we want to fill in the header buffer with a single DMA, the header must be the last piece of data to be written to the host for any received transaction.

RECEIVE INTERFACE DETAILS.

HEADER BUFFERS.

Header buffers in host memory are 256 bytes long, and are aligned on 256 byte boundaries. There will be a field in the header buffer indicating it has valid data. This field will initially be reset by the host before passing the buffer descriptor to the INIC. A set of header buffers are passed from the host to the INIC by the host writing to the "Header Buffer Address Register" on the INIC. This register is defined as follows:

Bits 31-8 Physical address in host memory of the first of a set of contiguous header buffers.

Bits 7-0 Number of header buffers passed.

In this way the host can, say, allocate 16 buffers in a 4K page, and pass all 16 buffers to the INIC with one register write. The INIC will maintain a queue of these header descriptors in the SmallHType queue in it's own local memory, adding to the end of the queue every time the host writes to the Header Buffer Address Register. Note that the single entry is added to the queue; the eventual dequeuer will use the count after extracting that entry.

The header buffers, will be used and returned to the host in the same order that they were given to the INIC. The valid field will be set by the INIC before returning the buffer to the host. In this way a PCI interrupt, with a single bit in the interrupt register, may be generated to indicate that there is a header buffer for the host to process. When servicing this

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 068

interrupt, the host will look at its queue of header buffers, reading the valid field to determine how many header buffers are to be processed.

RECEIVE DATA BUFFERS.

Receive data buffers in host memory are aligned to page boundaries, assumed here to be 2K bytes long and aligned on 4K page boundaries, 2 buffers per page. In order to pass receive data buffers to the INIC, the host must write to two registers on the INIC. The first register to be written is the "Data Buffer Handle Register." The buffer handle is not significant to the INIC, but will be copied back to the host to return the buffer to the host. The second register written is the Data Buffer Address Register. This is the physical address of the data buffer. When both registers have been written, the INIC will add the contents of these two registers to FreeType queue of data buffer descriptors. Note that the INIC host driver sets the handle register first, then the address register. There needs to be some mechanism put in place to ensure the reading of these registers does not get out of sync with writing them. Effectively the INIC can read the address register first and save its contents, then read the handle register. It can then lock the register pair in some manner such that another write to the handle register is not permitted until the current contents have been saved. Both addresses extracted from the registers are to be written to the FreeType queue. The INIC will extract 2 entries each time when dequeuing.

Data buffers will be allocated and used by the INIC as needed. For each data buffer used by a slow-path transaction, the data buffer handle will be copied into a header buffer. Then the header buffer will be returned to the host.

TRANSMIT INTERFACE.

TRANSMIT INTERFACE OVERVIEW.

The transmit interface shown in Fig. 26, like the receive interface, has been designed to minimize the amount of PCI bandwidth and latencies. In order to transmit data, the host will transfer a command buffer to the INIC. This command buffer will include a command buffer handle, a command field, possibly a TCP context identification, and a list of physical data pointers. The command buffer handle is defined to be the first word of the command buffer and is used by the host to identify the command. This word will be passed back to the host in a response buffer, since commands may complete out of order, and the host will need to know

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 069

which command is complete. Commands will be used for many reasons, but primarily to cause the INIC to transmit data, or to pass a set of buffers to the INIC for input data on the fast-path as previously discussed.

Response buffers are physical buffers in host memory. They are used by the INIC in the same order as they were given to it by the host. This enables the host to know which response buffer(s) to next look at when the INIC signals a command completion.

TRANSMIT INTERFACE DETAILS.

COMMAND BUFFERS.

Command buffers in host memory are a multiple of 32 bytes, up to a maximum of 1K bytes, and are aligned on 32 byte boundaries. A command buffer is passed to the INIC by writing to one of five "Command Buffer Address Registers." These registers are defined as follows:

Bits 31-5 Physical address in host memory of the command buffer.

Bits 4-0 Length of command buffer in bytes / 32 (i.e. number of multiples of 32 bytes).

This is the physical address of the command buffer. The register to which the command is written predetermines the XMT interface number, or if the command is for the RCV CPU; hence there will be 5 of them, 0 – 3 for XMT and 4 for RCV. When one of these registers has been written, the INIC will add the contents of the register to it's own internal queue of command buffer descriptors. The first word of all command buffers is defined to be the command buffer handle. It is the job of the utility CPU to extract a command from its local queue, DMA the command into a small INIC buffer (from the FreeSType queue), and queue that buffer into the Xmit#Type queue, where # is 0 – 3 depending on the interface, or the appropriate RCV queue. The receiving CPU will service the queues to perform the commands. When that CPU has completed a command, it extracts the command buffer handle and passes it back to the host via a response buffer.

RESPONSE BUFFERS.

Response buffers in host memory are 32 bytes long and aligned on 32 byte boundaries. They are handled in a very similar fashion to header buffers. There will be a field in the response buffer indicating it has valid data. This field will initially be reset by the host before

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 070

passing the buffer descriptor to the INIC. A set of response buffers are passed from the host to the INIC by the host writing to the "Response Buffer Address Register" on the INIC. This register is defined as follows:

Bits 31-8 Physical address in host memory of the first of a set of contiguous response buffers.

Bits 7-0 Number of response buffers passed.

In this way the host can, say, allocate 128 buffers in a 4K page, and pass all 128 buffers to the INIC with one register write. The INIC will maintain a queue of these header descriptors in it's ResponseType queue, adding to the end of the queue every time the host writes to the "Response Buffer Address Register". The INIC writes the extracted contents including the count, to the queue in exactly the same manner as for the header buffers.

The response buffers can be used and returned to the host in the same order that they were given to the INIC. The valid field will be set by the INIC before returning the buffer to the host. In this way a PCI interrupt, with a single bit in the interrupt register, may be generated to indicate that there is a response buffer for the host to process. When servicing this interrupt, the host will look at its queue of response buffers, reading the valid field to determine how many response buffers are to be processed.

INTERRUPT STATUS REGISTER / INTERRUPT MASK REGISTER.

Fig. 27 shows the general format of this register. The setting of any bits in the ISR will cause an interrupt, provided the corresponding bit in the Interrupt Mask Register is set. The default setting for the IMR is 0.

The INIC is configured so that the host should never need to directly read the ISR from the INIC. To support this, it is important for the host/INIC to arrange a buffer area in host memory into which the ISR is dumped. The address and size of that area ca be passed to the INIC via a command on the XMT interface. That command will also specify the setting for the IMR. Until the INIC receives this command, it will not DMA the ISR to host memory, and no events will cause an interrupt. The host could if necessary, read the ISR directly from the INIC in this case.

For the host to never have to actually read the register from the INIC itself, it is necessary for the INIC to update this host copy of the register whenever anything in it changes. The host will Ack (or deassert) events in the register by writing the register with 0's in

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 071

appropriate bit fields. So that the host does not miss events, the following scheme has been developed:

The INIC keeps a local copy of the register whenever it DMAs it to the host i.e. after some event(s). Call this COPYA Then the INIC starts accumulating any new events not reflected in the host copy in a separate word. Call this NEWA. As the host clears bits by writing the register back with those bits set to zero, the INIC clears these bits in COPYA (or the host write-back goes directly to COPYA). If there are new events in NEWA, it ORs them with COPYA, and DMAs this new ISR to the host. This new ISR then replaces COPYA, NEWA is cleared and the cycle then repeats.

REGISTER ADDRESS.

For the sake of simplicity, in this example of Fig. 28 the registers are at 4-byte increments from whatever the base address is.

ALACRITECH TCP (ATCP) DESIGN SPECIFICATION.

This section outlines the design specification for the Alacritech TCP (ATCP) transport driver. The ATCP driver consists of three components:

1. The bulk of the protocol stack is based on the FreeBSD TCP/IP protocol stack. This code performs the Ethernet, ARP, IP, ICMP, and (slow path) TCP processing for the driver.

2. At the top of the protocol stack we introduce an NT filter driver used to intercept TDI requests destined for the Microsoft TCP driver.

3. At the bottom of the protocol stack we include an NDIS protocol-driver interface which allows us to communicate with the INIC miniport NDIS driver beneath the ATCP driver.

This section covers each of these topics, as well as issues common to the entire ATCP driver.

CODING STYLE.

In order to ensure that our ATCP driver is written in a consistent manner, we have adopted a set of coding guidelines. These guidelines are introduced with the philosophy that we should write code in a Microsoft style since we are introducing an NT-based product. The

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 072

guidelines below apply to all code that we introduce into our driver. Since a very large portion of our ATCP driver will be based on FreeBSD, and since we are somewhat time-constrained on our driver development, the ported FreeBSD code will be exempt from these guidelines.

1. Global symbols – All function names and global variables in the ATCP driver should begin with the "ATK" prefix (ATKSend() for instance).

2. Variable names – Microsoft seems to use capital letters to separate multi-word variable names instead of underscores (VariableName instead of variable_name). We should adhere to this style.

3. Structure pointers – Microsoft typedefs all of their structures. The structure types are always capitals and they typedef a pointer to the structure as "P"<name> as follows:

```
typedef  struct _FOO {
        INT     bar;
} FOO, *PFOO;
```

We will adhere to this style.

4. Function calls – Microsoft separates function call arguments on separate lines:

```
X = foobar(
        argument1,
        argument2,
        );
```

We will adhere to this style.

5. Comments – While Microsoft seems to alternatively use // and /* */ comment notation, we will exclusively use the /* */ notation.

6. Function comments – Microsoft includes comments with each function that describe the function, its arguments, and its return value. We will also include these comments, but will move them from within the function itself to just prior to the function for better readability.

7. Function arguments – Microsoft includes the keywords IN and OUT when defining function arguments. These keywords denote whether the function argument is used as an input parameter, or alternatively as a placeholder for an output parameter. We will include these keywords.

[redacted]

8. Function prototypes – We will include function prototypes in the most logical header file corresponding to the .c file. For example, the prototype for function foo() found in foo.c will be placed in foo.h.

9. Indentation – Microsoft code fairly consistently uses a tabstop of 4. We will do likewise.

10. Header file #ifndef – each header file should contain a #ifndef/#define/#endif which is used to prevent recursive header file includes. For example, foo.h would include:

```
#ifndef __FOO_H__
#define __FOO_H__
<foo.h contents..>
#endif /* __FOO_H__ */
```

Note the __NAME_H__ format.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 074

11. Each file must contain a comment at the beginning which includes the Id as follows:

```
/*
 * $Id$
 */
```

CVS (RCS) will expand this keyword to denote RCS revision, timestamps, author, etc.

SMP

This section describes the process by which we will make the ATCP driver SMP safe. The basic rule for SMP kernel code is that any access to a memory variable must be protected by a lock that prevents a competing access by code running on another processor. Spinlocks are the normal locking method for code paths which do not take a long time to execute (and which do not sleep.)

In general each instance of a structure will include a spinlock, which must be acquired before members of that structure are accessed, and held while a function is accessing that instance of the structure. Structures which are logically grouped together may be protected by a single spinlock: for example, the 'in_pcb' structure, 'tcpcb' structure, and 'socket' structure which together constitute the administrative information for a TCP connection will probably be collectively managed by a single spinlock in the 'socket' structure.

In addition, every global data structure such as a list or hash table must also have a protecting spinlock which must be held while the structure is being accessed or modified. The NT DDK in fact provides a number of convenient primitives for SMP-safe list manipulation, and it is recommended that these be used for any new lists. Existing list manipulations in the FreeBSD code can probably be left as-is to minimize code disturbance, except of course that the necessary spinlock acquisition and release must be added around them.

Spinlocks should not be held for long periods of time, and most especially, must not be held during a sleep, since this will lead to deadlocks. There is a significant deficiency in the NT kernel support for SMP systems: it does not provide an operation which allows a spinlock to be exchanged atomically for a sleep lock. This would be a serious problem in a UNIX environment where much of the processing occurs in the context of the user process which initiated the operation. (The spinlock would have to be explicitly released, followed by a separate acquisition of the sleep lock: creating an unsafe window.)

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 075

The NT approach is more asynchronous, however: IRPs are simply marked as 'PENDING' when an operation cannot be completed immediately. The calling thread does NOT sleep at that point: it returns, and may go on with other processing. Pending IRPs are later completed, not by waking up the thread which initiated them, but by an "IoCompleteRequest" call which typically runs at DISPATCH level in an arbitrary context.

Thus we have not in fact used sleep locks anywhere in the design of the ATCP driver, hoping the above issue will not arise.

DATA FLOW OVERVIEW.

The ATCP driver supports two paths for sending and receiving data, the fast-path and the slow-path. The fast-path data flow corresponds to connections that are maintained on the INIC, while slow-path traffic corresponds to network data for which the INIC does not have a connection. In order to set some groundwork for the rest of this section, these two data paths are summarized here.

FAST-PATH INPUT DATA FLOW.

There are 2 different cases to consider:

1. NETBIOS traffic (identifiable by port number.)
2. Everything else.

NETBIOS INPUT.

As soon as the INIC has received a segment containing a NETBIOS header, it will forward it up to the TCP driver, along with the NETBIOS length from the header. (In principle the host could get this from the header itself, but since the INIC has already done the decode, it seem reasonable to just pass it.)

From the TDI spec, the amount of data in the buffer actually sent must be at least 128 bytes. For small SMBs, all of the received SMB should be forwarded; it will be absorbed directly by the TDI client without any further MDL exchange. Experiments tracing the TDI data flow show that the NETBIOS client directly absorbs up to 1460 bytes: the amount of payload data in a single Ethernet frame. Thus the initial system specifies that the INIC will indicate anything up to a complete segment to the ATCP driver. [See note (1)].

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 076

Once the INIC has passed up an indication with an NETBIOS length greater than the amount of data in the packet it passed, it will continue to accumulate further incoming data in DRAM on the INIC. Overflow of INIC DRAM buffers will be avoided by using a receive window on the INIC at this point, which can be 8K.

On receiving the indicated packet, the ATCP driver will call the receive handler registered by the TDI client for the connection, passing the actual size of the data in the packet from the INIC as "bytes indicated" and the NETBIOS length as "bytes available." [See note (2)].

In the "large data input" case, where "bytes available" exceeds the packet length, the TDI client will then provide an MDL, associated with an IRP, which must be completed when this MDL is filled. (This IRP/MDL may come back either in the response to TCP's call of the receive handler, or as an explicit TDI_RECEIVE request.)

The ATCP driver will build a "receive request" from the MDL information, and pass this to the INIC. This request will contain:

1) The TCP context identifier; 2) Size and offset information; 3) A list of physical addresses corresponding to the MDL pages; 4) A context field to allow the ATCP driver to identify the request on completion; and 5) "Piggybacked" window update information.

Note: the ATCP driver must copy any remaining data (which was not taken by the receive handler) from the segment indicated by the INIC to the start of the MDL, and must adjust the size & offset information in the request passed to the INIC to account for this.

The INIC will fill the given page(s) with incoming data up to the requested amount, and respond to the ATCP driver when this is done [See note (3)]. If the MDL is large, the INIC may open up its advertised receive window for improved throughput while filling the MDL. On receiving the response from the INIC, the ATCP driver will complete the IRP associated with this MDL, to tell the TDI client that the data is available. At this point the cycle of events is complete, and the ATCP driver is now waiting for the next header indication.

OTHER TCP INPUT.

In the general case we do not have a higher-level protocol header to enable us to predict that more data is coming. So on non-NETBIOS connections, the INIC will just accumulate incoming data in INIC DRAM up to a quantity of 8K in this example. Again, a

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 077

maximum advertised window size, which may be 16K, will be used to prevent overflow of INIC DRAM buffers.

When the prescribed amount has been accumulated, or when a PSH flag is seen, the INIC will indicate a small packet which may be 128 bytes of the data to the ATCP driver, along with the total length of the data accumulated in INIC DRAM.

On receiving the indicated packet, the ATCP driver will call the receive handler registered by the TDI client for the connection, passing the actual size of the data in the packet from the INIC as "bytes indicated" and the total INIC-buffer length as "bytes available."

As in the NETBIOS case, if "bytes available" exceeds "bytes indicated", the TDI client will provide an IRP with an MDL. The ATCP driver will pass the MDL to the INIC to be filled, as before. The INIC will reply to the ATCP driver, which in turn will complete the IRP to the TDI client.

Using an MDL from the client avoids a copy step. However, if we can only buffer 8K and delay indicating to the ATCP driver until we have done so, a question arises regarding further segments coming in, since INIC DRAM is a scarce resource. We do not want to ACK with a zero-size window advertisement: this would cause the transmitting end to go into persist state, which is bad for throughput. If the transmitting end is also our INIC, this results in having to implement the persist timer on the INIC, which we do not wish to do. Instead for large transfers (i.e. no PSH flag seen) we will not send an ACK until the host has provided the MDL, and also, to avoid stopping the transmitting end, we will use a receive window of twice the amount we will buffer before calling the host. Since the host comes back with the MDL quite quickly (measured at < 100 microseconds), we do not expect to experience significant overruns.

INIC RECEIVE WINDOW UPDATES.

If the INIC "owns" an MDL provided by the TDI client (sent by ATCP as a receive request), it will treat this as a "promise" by the TDI client to accept the data placed in it, and may therefore ACK incoming data as it is filling the pages.

However, for small requests, there will be no MDL returned by the TDI client: it absorbs all of the data directly in the receive callback function. We need to update the INIC's view of data which has been accepted, so that it can update its receive window. In order to be

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 078

able to do this, the ATCP driver will accumulate a count of data which has been accepted by the TDI client receive callback function for a connection.

From the INIC's point of view, though, segments sent up to the ATCP driver are just "thrown over the wall"; there is no explicit reply path. We will therefore "piggyback" the update on requests sent out to the INIC. Whenever the ATCP driver has outgoing data for that connection, it will place this count in a field in the send request (and then clear the counter.) Any receive request (passing a receive MDL to the INIC) may also be used to transport window update info in the same way.

Note: we will probably also need to design a message path whereby the ATCP driver can explicitly send an update of this "bytes consumed" information (either when it exceeds a preset threshold or if there are no requests going out to the INIC for more than a given time interval), to allow for possible scenarios in which the data stream is entirely one-way.

NOTES.

1) The PSH flag can help to identify small SMB requests that fit into one segment.

2) Actually, the observed "bytes available" from the NT TCP driver to its client's callback in this case is always 1460. The NETBIOS-aware TDI client presumably calculates the size of the MDL it will return from the NETBIOS header. So strictly speaking we do not need the NETBIOS header length at this point: just an indication that this is a header for a "large" size. However, we *do* need an actual "bytes available" value for the non-NETBIOS case, so we may as well pass it.

3) We observe that the PSH flag is set in the segment completing each NETBIOS transfer. The INIC can use this to determine when the current transfer is complete and the MDL should be returned. It can, at least in a debug mode, sanity check the amount of received data against what is expected, though.

FAST-PATH OUTPUT DATA FLOW.

The fast-path output data flow is similar to the input data-flow, but simpler. In this case the TDI client will provide a MDL to the ATCP driver along with an IRP to be completed when the data is sent. The ATCP driver will then give a request (corresponding to the MDL) to the INIC. This request will contain:

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 079

1) The TCP context identifier; 2) Size and offset information; 3) A list of physical addresses corresponding to the MDL pages; 4) A context field to allow the ATCP driver to identify the request on completion; 5) "Piggybacked" window update information (as discussed in section 6.1.3.)

The INIC will copy the data from the given physical location(s) as it sends the corresponding network frames onto the network. When all of the data is sent, the INIC will notify the host of the completion, and the ATCP driver will complete the IRP.

Note that there may be multiple output requests pending at any given time, since SMB allows multiple SMB requests to be simultaneously outstanding.

SLOW-PATH DATA FLOW.

For data for which there is no connection being maintained on the INIC, we will have to perform all of the TCP, IP, and Ethernet processing ourselves. To accomplish this we will port the FreeBSD protocol stack. In this mode, the INIC will be operating as a "dumb NIC"; the packets which pass over the NDIS interface will just contain MAC-layer frames.

The MBUFs in the incoming direction will in fact be managing NDIS-allocated packets. In the outgoing direction, we need protocol-allocated MBUFs in which to assemble the data and headers. The MFREE macro must be cognizant of the various types of MBUFs, and "do the right thing" for each type.

We will retain a (modified) socket structure for each connection, containing the socket buffer fields expected by the FreeBSD code. The TCP code that operates on socket buffers (adding/removing MBUFs to & from queues, indicating acknowledged & received data etc) will remain essentially unchanged from the FreeBSD base (though most of the socket functions & macros used to do this will need to be modified; these are the functions in kern/uipc_socket2.c)

The upper socket layer (kern/uipc_socket.c), where the overlying OS moves data in and out of socket buffers, must be entirely re-implemented to work in TDI terms. Thus, instead of sosend(), there will be a function that copies data from the MDL provided in a TDI_SEND call into socket buffer MBUFs. Instead of soreceive(), there will be a handler that calls the TDI client receive callback function, and also copies data from socket buffer MBUFs into any MDL provided by the TDI client (either explicitly with the callback response or as a separate TDI_RECEIVE call.)

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 080

We must note that there is a semantic difference between TDI_SEND and a write() on a BSD socket. The latter may complete back to its caller as soon as the data has been copied into the socket buffer. The completion of a TDI_SEND, however, implies that the data has actually been sent on the connection. Thus we will need to keep the TDI_SEND IRPs (and associated MDLs) in a queue on the socket until the TCP code indicates that the data from them has been ACK'd.

DATA PATH NOTES:

1. There might be input data on a connection object for which there is no receive handler function registered. This has not been observed, but we can probably just ASSERT for a missing handler for the moment. If it should happen, however, we must assume that the TDI client will be doing TDI_RECEIVE calls on the connection. If we can't make a callup at the time that the indication from the INIC appears, we can queue the data and handle it when a TDI_RECEIVE does appear.

2. NT has a notion of "canceling" IRPs. It is possible for us to get a "cancel" on an IRP corresponding to an MDL which has been "handed" to the INIC by a send or receive request. We can handle this by being able to force the context back off the INIC, since IRPs will only get cancelled when the connection is being aborted.

CONTEXT PASSING BETWEEN ATCP AND INIC.

FROM ATCP TO INIC.

There is a synchronization problem that must be addressed here. The ATCP driver will make a decision on a given connection that this connection should now be passed to the INIC. It builds and sends a command identifying this connection to the INIC.

Before doing so, it must ensure that no slow-path outgoing data is outstanding. This is not difficult; it simply pends and queues any new TDI_SEND requests and waits for any unacknowledged slow path output data to be acknowledged before initiating the context pass operation.

The problem arises with incoming slow-path data. If we attempt to do the context-pass in a single command handshake, there is a window during which the ATCP driver has send the context command, but the INIC has not yet seen this (or has not yet completed setting up its context.) During this time, slow-path input data frames could arrive and be fed into the slow-

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 081

path ATCP processing code. Should that happen, the context information which the ATCP driver passed to the INIC is no longer correct. We can simply abort the outward pass of the context in this event, but it seems better to have a reliable handshake.

Therefore, the command to pass context from ATCP driver to INIC will be split into two halves, and there will be a two-exchange handshake.

The initial command from ATCP to INIC expresses an "intention" to hand out the context. It will include the source and destination IP addresses and ports, which will allow the INIC to establish a "provisional" context. Once it has this "provisional" context in place, the INIC will not send any more slow-path input frames for that src/dest IP/port combination (it will queue them, if any are received.)

When the ATCP driver receives the response to this initial "intent" command, it knows that the INIC will send no more slow-path input. The ATCP driver then waits for any remaining unconsumed slow-path input data for this connection to be consumed by the client. (Generally speaking there will be none, since the ATCP driver will not initiate a context pass while there is unconsumed slow-path input data; the handshake is simply to close the crossover window.)

Once any such data has been consumed, we know things are in a quiescent state. The ATCP driver can then send the second, "commit" command to hand out the context, with confidence that the TCB values it is handing out (sequence numbers etc) are reliable.

Note 1: it is conceivable that there might be situations in which the ATCP driver decides, after having sent the original "intention" command, that the context is not to be passed after all. (E.g. the local client issues a close.) So we must allow for the possibility that the second command may be a "abort", which should cause the INIC to deallocate and clear up its "provisional" context.

Note 2: to simplify the logic, the ATCP driver will guarantee that only one context may be in process of being handed out at a time: in other words, it will never issue another initial "intention" command until it has completed the second half of the handshake for the first one.

FROM INIC TO ATCP.

There are two possible cases for this: a context transfer may be initiated either by the ATCP driver or by the INIC. However the machinery will be very similar in the two cases. If the ATCP driver wishes to cause context to be flushed from INIC to host, it will send a "flush"

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 082

message to the INIC specifying the context number to be flushed. Once the INIC receives this, it will proceed with the same steps as for the case where the flush is initiated by the INIC itself:

1) The INIC will send an error response to any current outstanding receive request it is working on (corresponding to an MDL into which data is being placed.) Before sending the response, it updates the receive command "length" field to reflect the amount of data which has actually been placed in the MDL buffers at the time of the flush.

2) Likewise it will send an error response for any current send request, again reporting the amount of data actually sent from the request.

3) The INIC will DMA the TCB for the context back to the host. (Note: part of the information provided with a context must be the address of the TCB in the host.)

4) The INIC will send a "flush" indication to the host (very preferably via the regular input path as a special type of frame) identifying the context which is being flushed. Sending this indication via the regular input path ensures that it will arrive before any following slow-path frames.

At this point, the INIC is no longer doing fast-path processing, and any further incoming frames for the connection will simply be sent to the host as raw frames for the slow input path. The ATCP driver may not be able to complete the cleanup operations needed to resume normal slow path processing immediately on receipt of the "flush frame", since there may be outstanding send and receive requests to which it has not yet received a response. If this is the case, the ATCP driver must set a "pend incoming TCP frames" flag in its per-connection context. The effect of this is to change the behavior of tcp_input(). This runs as a function call in the context of ip_input(), and normally returns only when incoming frames have been processed as far as possible (queued on the socket receive buffer or out-of-sequence reassembly queue.) However, if there is a flush pending and we have not yet completed resynchronization, we cannot do TCP processing and must instead queue input frames for TCP on a "holding queue" for the connection, to be picked up later when context flush is complete and normal slow path processing resumes. (This is why we want to send the "flush" indication via the normal input path: so that we can ensure it is seen before any following frames of slow-path input.)

Next we need to wait for any outstanding "send" requests to be errored off:

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 083

1) The INIC maintains its context for the connection in a "zombie" state. As "send" requests for this connection come out of the INIC queue, it sends error responses for them back to the ATCP driver. (It is apparently difficult for the INIC to identify all command requests for a given context; simpler for it to just continue processing them in order, detecting ones that are for a "zombie" context as they appear.)

2) The ATCP driver has a count of the number of outstanding requests it has sent to the INIC. As error responses for these are received, it decrements this count, and when it reaches zero, the ATCP driver sends a "flush complete" message to the INIC.

3) When the INIC receives the "flush complete" message, it dismantles its "zombie" context. From the INIC perspective, the flush is now completed.

4) When the ATCP driver has received error responses for all outstanding requests, it has all the information needed to complete its cleanup. This involves completing any IRPs corresponding to requests which have entirely completed and adjusting fields in partially-completed requests so that send and receive of slow path data will resume at the right point in the byte streams.

5) Once all this cleanup is complete, the ATCP driver will loop pulling any "pended" TCP input frames off the "pending queue" mentioned above and feeding them into the normal TCP input processing. Once all input frames on this queue have been cleared off, the "pend incoming TCP frames" flag can be cleared for the connection, and we are back to normal slow-path processing.

FREEBSD PORTING SPECIFICATION.

The largest portion of the ATCP driver is either derived, or directly taken from the FreeBSD TCP/IP protocol stack. This section defines the issues associated with porting this code, the FreeBSD code itself, and the modifications required for it to suit our needs.

PORTING PHILOSOPHY.

FreeBSD TCP/IP (current version referred to as Net/3) is a general purpose TCP/IP driver. It contains code to handle a variety of interface types and many different kinds of protocols. To meet this requirement the code is often written in a sometimes confusing, over-complex manner. General-purpose structures are overlaid with other interface-specific structures so that different interface types can coexist using the same general-purpose code.

[redacted]

For our purposes much of this complexity is unnecessary since we are only supporting a single interface type and a few specific protocols. It is therefore tempting to modify the code and data structures in an effort to make it more readable, and perhaps a bit more efficient. There are, however, some problems with doing this. First, the more we modify the original FreeBSD, the more changes we will have to make. This is especially true with regard to data structures. If we collapse two data structures into one we might improve the cleanliness of the code a bit, but we will then have to modify every reference to that data structure in the entire protocol stack. Another problem with attempting to "clean up" the code is that we might later discover that we need something that we had previously thrown away. Finally, while we might gain a small performance advantage in cleaning up the FreeBSD code, the FreeBSD TCP code will mostly only run in the slow-path connections, which are not our primary focus. Our priority is to get the slow-path code functional and reliable as quickly as possible.

For the reasons above we have adopted the philosophy that we should initially keep the data structures and code at close to the original FreeBSD implementation as possible. The code will be modified for the following reasons:

1) As required for NT interaction – Obviously we can't expect to simply "drop-in" the FreeBSD code as is. The interface of this code to the NT system will require some significant code modifications. This will mostly occur at the topmost and bottommost portions of the protocol stack, as well as the "ioctl" sections of the code. Modifications for SMP issues are also needed.

2) Unnecessary code can be removed – While we will keep the code as close to the original FreeBSD as possible, we will nonetheless remove code that will never be used (UDP is a good example of this).

UNIX ↔ NT CONVERSION.

The FreeBSD TCP/IP protocol stack makes use of many Unix system services. These include bcopy to copy memory, malloc to allocate memory, timestamp functions, etc. These will not be itemized in detail since the conversion to the corresponding NT calls is a fairly trivial and mechanical operation.

An area which will need non-trivial support redesign is MBUFs.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 085

NETWORK BUFFERS.

Under FreeBSD, network buffers are mapped using mbufs. Under NT network buffers are mapped using a combination of packet descriptors and buffer descriptors (the buffer descriptors are really MDLs). There are a couple of problems with the Microsoft method. First it does not provide the necessary fields which allow us to easily strip off protocol headers. Second, converting all of the FreeBSD protocol code to speak in terms of buffer descriptors is an unnecessary amount of overhead. Instead, in our port we will allocate our own mbuf structures and remap the NT packets as shown in Fig. 29.

The mbuf structure will provide the standard fields provided in the FreeBSD mbuf including the data pointer, which points to the current location of the data, data length fields and flags. In addition each mbuf will point to the packet descriptor which is associated with the data being mapped. Once an NT packet is mapped, our transport driver should never have to refer to the packet or buffer descriptors for any information except when we are finished and are preparing to return the packet.

There are a couple of things to note here. We have designed our INIC such that a packet header should never be split across multiple buffers. Thus, we should never require the equivalent of the "m_pullup" routine included in Unix. Also note that there are circumstances in which we will be accepting data that will also be accepted by the Microsoft TCP/IP. One such example of this is ARP frames. We will need to build our own ARP cache by looking at ARP replies as they come off the network. Under these circumstances, it is absolutely imperative that we do not modify the data, or the packet and buffer descriptors. We will discuss this further in the following sections.

We will allocate a pool of mbuf headers at ATCP initialization time. It is important to remember that unlike other NICs, we can not simply drop data if we run out of the system resources required to manage/map the data. The reason for this is that we will be receiving data from the card that has already been acknowledged by TCP. Because of this it is essential that we never run out of mbuf headers. To solve this problem we will statically allocate mbuf headers for the maximum number of buffers that we will ever allow to be outstanding. By doing so, the card will run out of buffers in which to put the data before we will run out of mbufs, and as a result, the card will be forced to drop data at the link layer instead of us dropping it at the transport layer. DhXXX: as we've discussed, I don't think this is really true anymore. The INIC won't ACK data until either it's gotten a window update from ATCP to

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 086

tell it the data's been accepted, or it's got an MDL. Thus it seems workable, though undesirable, if we can't accept a frame from the INIC & return an error to it saying it was not taken.

We will also require a pool of actual mbufs (not just headers). These mbufs are required in order to build transmit protocol headers for the slow-path data path, as well as other miscellaneous purposes such as for building ARP requests. We will allocate a pool of these at initialization time and we will add to this pool dynamically as needed. Unlike the mbuf headers described above, which will be used to map acknowledged TCP data coming from the card, the full mbufs will contain data that can be dropped if we can not get an mbuf.

THE CODE.

In this section we describe each section of the FreeBSD TCP/IP port. These sections include Interface Initialization, ARP, Route, IP, ICMP, and TCP.

INTERFACE INITIALIZATION.

STRUCTURES.

There are a variety of structures, which represent a single interface in FreeBSD. These structures include: ifnet, arpcom, ifaddr, in_ifaddr, sockaddr, sockaddr_in, and sockaddr_dl. Fig. 30 shows the relationship between all of these structures:

In the example of Fig. 30 we show a single interface with a MAC address of 00:60:97:DB:9B:A6 configured with an IP address of 192.100.1.2. As illustrated above, the in_ifaddr is actually an ifaddr structure with some extra fields tacked on to the end. Thus the ifaddr structure is used to represent both a MAC address and an IP address. Similarly the sockaddr structure is recast as a sockaddr_dl or a sockaddr_in depending on its address type. An interface can be configured to multiple IP addresses by simply chaining in_ifaddr structures after the in_ifaddr structure shown in Fig. 30.

As mentioned in the Porting Philosophy section, many of the above structures could likely be collapsed into fewer structures. In order to avoid making unnecessary modifications to FreeBSD, for the time being we will leave these structures mostly as is. We will however eliminate the fields from the structure that will never be used. These structure modifications are discussed below.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 087

We also show above a structure called iface. This is a structure that we define. It contains the arpcom structure, which in turn contains the ifnet structure. It also contains fields that enable us to blend our FreeBSD implementation with NT NDIS requirements. One such example is the NDIS binding handle used to call down to NDIS with requests (such as send).

THE FUNCTIONS.

FreeBSD initializes the above structures in two phases. First when a network interface is found, the ifnet, arpcom, and first ifaddr structures are initialized first by the network layer driver, and then via a call to the if_attach routine. The subsequent in_ifaddr structure(s) are initialized when a user dynamically configures the interface. This occurs in the in_ioctl and the in_ifinit routines. Since NT allows dynamic configuration of a network interface we will continue to perform the interface initialization in two phases, but we will consolidate these two phases as described below:

IFINIT.

The IfInit routine will be called from the ATKProtocolBindAdapter function. The IfInit function will initialize the Iface structure and associated arpcom and ifnet structures. It will then allocate and initialize an ifaddr structure in which to contain link-level information about the interface, and a sockaddr_dl structure to contain the interface name and MAC address. Finally it will add a pointer to the ifaddr structure into the ifnet_addrs array (using the if_index field of the ifnet structure) contained in the extended device object. IfInit will then call IfConfig for each IP address that it finds in the registry entry for the interface.

IFCONFIG.

IfConfig is called to configure an IP address for a given interface. It is passed a pointer to the ifnet structure for that interface along with all the information required to configure an IP address for that interface (such as IP address, netmask and broadcast info, etc). IfConfig will allocate an in_ifaddr structure to be used to configure the interface. It will chain it to the total chain of in_ifaddr structures contained in the extended device object, and will then configure the structure with the information given to it. After that it will add a static route for the newly configured network and then broadcast a gratuitous ARP request to notify others of our Mac/IP address and to detect duplicate IP addresses on the net.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 088

ARP.

We will port the FreeBSD ARP code to NT mostly as-is. For some reason, the FreeBSD ARP code is located in a file called if_ether.c. While the functionality of this file will remain the same, we will rename it to a more logical arp.c. The main structures used by ARP are the llinfo_arp structure and the rtentry structure (actually part of route). These structures will not require major modifications. The functions that will require modification are defined here.

IN_ARPINPUT.

This function is called to process an incoming ARP frame. An ARP frame can either be an ARP request or an ARP reply. ARP requests are broadcast, so we will see every ARP request on the network, while ARP replies are directed so we should only see ARP replies that are sent to us. This introduces the following possible cases for an incoming ARP frame:

1. ARP request trying to resolve our IP address – Under normal circumstances, ARP would reply to this ARP request with an ARP reply containing our MAC address. Since ARP requests will also be passed up to the Microsoft TCP/IP driver, we need not reply. Note however, that FreeBSD also creates or updates an ARP cache entry with the information derived from the ARP request. It does this in anticipation of the fact that any host that wishes to know our MAC address is likely to wish to talk to us soon. Since we will need to know his MAC address in order to talk back, we might as well add the ARP information now rather than issuing our own ARP request later.

2. ARP request trying to resolve someone else's IP address – Since ARP requests are broadcast, we see every one on the network. When we receive an ARP request of this type, we simply check to see if we have an entry for the host that sent the request in our ARP cache. If we do, we check to see if we still have the correct MAC address associated with that host. If it is incorrect, we update our ARP cache entry. Note that we do not create a new ARP cache entry in this case.

3. ARP reply – In this case we add the new ARP entry to our ARP cache. Having resolved the address, we check to see if there is any transmit requests pending for the resolve IP address, and if so, transmit them.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 089

Given the above three possibilities, the only major change to the in_arpinput code is that we will remove the code which generates an ARP reply for ARP requests that are meant for our interface.

ARPINTR.

This is the FreeBSD code that delivers an incoming ARP frame to in_arpinput. We will be calling in_arpinput directly from our ProtocolReceiveDPC routine (discussed in the NDIS section below) so this function is not needed.

ARPWHOHAS.

This is a single line function that serves only as a wrapper around arprequest. We will remove it and replace all calls to it with direct calls to arprequest.

ARPREQUEST.

This code simply allocates a mbuf, fills it in with an ARP header, and then passes it down to the ethernet output routine to be transmitted. For us, the code remains essentially the same except for the obvious changes related to how we allocate a network buffer, and how we send the filled in request.

ARP_IFINIT.

This is simply called when an interface is initialized to broadcast a gratuitous ARP request (described in the interface initialization section) and to set some ARP related fields in the ifaddr structure for the interface. We will simply move this functionality into the interface initialization code and remove this function.

ARPTIMER.

This is a timer-based function that is called every 5 minutes to walk through the ARP table looking for entries that have timed out. Although the time-out period for FreeBSD is 20 minutes, RFC 826 does not specify any timer requirements with regard to ARP so we can modify this value or delete the timer altogether to suit our needs. Either way the function won't require any major changes. All other functions in if_ether.c will not require any major changes.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 090

ROUTE.

On first thought, it might seem that we have no need for routing support since our ATCP driver will only receive IP datagrams whose destination IP address matches that of one of our own interfaces. Therefore, we will not "route" from one interface to another. Instead, the MICROSOFT TCP/IP driver will provide that service. We will, however, need to maintain an up-to-date routing table so that we know a) whether an outgoing connection belongs to one of our interfaces, b) to which interface it belongs, and c) what the first-hop IP address (gateway) is if the destination is not on the local network.

We discuss four aspects on the subject of routing in this section. They are as follows:

1. The mechanics of how routing information is stored.
2. The manner in which routes are added or deleted from the route table.
3. When and how route information is retrieved from the route table.
4. Notification of route table changes to interested parties.

THE ROUTE TABLE.

In FreeBSD, the route table is maintained using an algorithm known as PATRICIA (Practical Algorithm To Retrieve Information Coded in Alphanumeric). This is a complicated algorithm that is a bit costly to set up, but is very efficient to reference. Since the routing table should contain the same information for both NT and FreeBSD, and since the key used to search for an entry in the routing table will be the same for each (the destination IP address), we should be able to port the routing table software to NT without any major changes.

The software which implements the route table (via the PATRICIA algorithm) is located in the FreeBSD file, radix.c. This file will be ported directly to the ATCP driver with no significant changes required.

ADDING AND DELETING ROUTES.

Routes can be added or deleted in a number of different ways. The kernel adds or deletes routes when the state of an interface changes or when an ICMP redirect is received. User space programs such as the RIP daemon, or the route command also modify the route table.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 091

For kernel-based route changes, the changes can be made by a direct call to the routing software. The FreeBSD software that is responsible for the modification of route table entries is found in route.c. The primary routine for all route table changes is called rtrequest(). It takes as its arguments, the request type (ADD, RESOLVE, DELETE), the destination IP address for the route, the gateway for the route, the netmask for the route, the flags for the route, and a pointer to the route structure (struct rtentry) in which we will place the added or resolved route. Other routines in the route.c file include rtinit(), which is called during interface initialization time to add a static route to the network, rtredirect, which is called by ICMP when we receive a ICMP redirect, and an assortment of support routines used for the modification of route table entries. All of these routines found in route.c will be ported with no major modifications.

For user-space-based changes, we will have to be a bit more clever. In FreeBSD, route changes are sent down to the kernel from user-space applications via a special route socket. This code is found in the FreeBSD file, rtsock.c. Obviously this will not work for our ATCP driver. Instead the filter driver portion of our driver will intercept route changes destined for the Microsoft TCP driver and will apply those modifications to our own route table via the rtrequest routine described above. In order to do this, it will have to do some format translation to put the data into the format (sockaddr_in) expected by the rtrequest routine. Obviously, none of the code from rtsock.c will be ported to the ATCP driver. This same procedure will be used to intercept and process explicit ARP cache modifications.

CONSULTING THE ROUTE TABLE.

In FreeBSD, the route table is consulted in ip_output when an IP datagram is being sent. In order to avoid a complete route table search for every outgoing datagram, the route is stored into the in_pcb for the connection. For subsequent calls to ip_output, the route entry is then simply checked to ensure validity. While we will keep this basic operation as is, we will require a slight modification to allow us to coexist with the Microsoft TCP driver. When an active connection is being set up, our filter driver will have to determine whether the connection is going to be handled by one of the INIC interfaces. To do this, we will have to consult the route table from the filter driver portion of our driver. This is done via a call to the rtalloc1 function (found in route.c). If a valid route table entry is found, then we will take

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 092

control of the connection and set a pointer to the rtentry structure returned by rtalloc1 in our in_pcb structure.

WHAT TO DO WHEN A ROUTE CHANGES.

When a route table entry changes, there may be connections that have pointers to a stale route table entry. These connections will need to be notified of the new route. FreeBSD solves this by checking the validity of a route entry during every call to ip_output. If the entry is no longer valid, its reference to the stale route table entry is removed, and an attempt is made to allocate a new route to the destination. For our slow path, this will work fine. Unfortunately, since our IP processing is handled by the INIC for our fast path, this sanity check method will not be sufficient. Instead, we will need to perform a review of all of our fast path connections during every route table modification. If the route table change affects our connection, we will need to advise the INIC with a new first-hop address, or if the destination is no longer reachable, close the connection entirely.

ICMP.

Like the ARP code above, we will need to process certain types of incoming ICMP frames. Of the 10 possible ICMP message types, there are only three that we need to support. These include ICMP_REDIRECT, ICMP_UNREACH, and ICMP_SOURCEQUENCH. Any FreeBSD code to deal with other types of ICMP traffic will be removed. Instead, we will simply return NDIS_STATUS_NOT_ACCEPTED for all but the above ICMP frame types. This section describes how we will handle these ICMP frames.

ICMP_REDIRECT.

Under FreeBSD, an ICMP_REDIRECT causes two things to occur. First, it causes the route table to be updated with the route given in the redirect. Second, it results in a call back to TCP to cause TCP to flush the route entry attached to its associated in_pcb structures. By doing this, it forces ip_output to search for a new route. As mentioned in the Route section above, we will also require a call to a routine which will review all of the TCP fast-path connections, and update the route entries as needed (in this case because the route entry has been zeroed). The INIC will then be notified of the route changes.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 093

ICMP_UNREACH.

In both FreeBSD and Microsoft TCP, the ICMP_UNREACH results in no more than a simple statistic update. We will do the same.

ICMP_SOURCEQUENCH.

A source quench is sent to cause a TCP sender to close its congestion window to a single segment, thereby putting the sender into slow-start mode. We will keep the FreeBSD code as-is for slow-path connections. For fast path connections we will send a notification to the card that the congestion window for the given connection has been reduced. The INIC will then be responsible for the slow-start algorithm.

IP.

The FreeBSD IP code should require few modifications when porting to the ATCP driver. What few modifications will be required will be discussed in this section.

IP INITIALIZATION.

During initialization time, ip_init is called to initialize the array of protosw structures. These structures contain all the information needed by IP to be able to pass incoming data to the correct protocol above it. For example, when a UDP datagram arrives, IP locates the protosw entry corresponding to the UDP protocol type value (0x11) and calls the input routine specified in that protosw entry. We will keep the array of protosw structures intact, but since we are only handling the TCP and ICMP protocols above IP, we will strip the protosw array down substantially.

IP INPUT.

Following are the changes required for IP input (function ip_intr()).

NO IP FORWARDING.

Since we will only be handling datagrams for which we are the final destination, we should never be required to forward an IP datagram. All references to IP forwarding, and the ip_forward function itself, can be removed.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 094

IP OPTIONS.

The only options supported by FreeBSD at this time include record route, strict and loose source and record route, and timestamp. For the timestamp option, FreeBSD only logs the current time into the IP header so that before it is forwarded. Since we will not be forwarding IP datagrams, this seems to be of little use to us. While FreeBSD supports the remaining options, NT essentially does nothing useful with them. For the moment, we will not bother dealing with IP options. They will be added in later if needed.

IP REASSEMBLY.

There is a small problem with the FreeBSD IP reassembly code. The reassembly code reuses the IP header portion of the IP datagram to contain IP reassembly queue information. It can do this because it no longer requires the original IP header. This is an absolute no-no with the NDIS 4.0 method of handling network packets. The NT DDK explicitly states that we must not modify packets given to us by NDIS. This is not the only place in which the FreeBSD code modifies the contents of a network buffer. It also does this when performing endian conversions. At the moment we will leave this code as is and violate the DDK rules. We believe we can do this because we are going to ensure that no other transport driver looks at these frames. If this becomes a problem we will have to modify this code substantially by moving the IP reassembly fields into the mbuf header.

IP OUTPUT.

There are only two modifications required for IP output. The first is that since, for the moment, we are not dealing with IP options, there is no need for the code that inserts the IP options into the IP header. Second, we may discover that it is impossible for us to ever receive an output request that requires fragmentation. Since TCP performs Maximum Segment Size negotiation, we should theoretically never attempt to send a TCP segment larger than the MTU.

NDIS PROTOCOL DRIVER.

This section defines protocol driver portion of the ATCP driver. The protocol driver portion of the ATCP driver is defined by the set of routines registered with NDIS via a call to NdisRegisterProtocol. These routines are limited to those that are called (indirectly) by the

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 095

INIC miniport driver beneath us. For example, we register a ProtocolReceivePacket routine so that when the INIC driver calls NdisMIndicateReceivePacket it will result in a call from NDIS to our driver. Strictly speaking, the protocol driver portion of our driver does not include the method by which our driver calls down to the miniport (for example, the method by which we send network packets). Nevertheless, we will describe that method here for lack of a better place to put it. That said, we cover the following topics in this section of the document: 1) Initialization; 2) Receive; 3) Transmit; 4) Query/Set Information; 5) Status indications; 6) Reset; and 7) Halt.

INITIALIZATION.

The protocol driver initialization occurs in two phases. The first phase occurs when the ATCP DriverEntry routine calls ATKProtoSetup. The ATKProtoSetup routine performs the following:

1. Allocate resources – We attempt to allocate many of the required resources as soon as possible so that we are more likely to get the memory we want. This mostly applies to allocating and initializing our mbuf and mbuf header pools.

2. Register Protocol – We call NdisRegisterProtocol to register our set of protocol driver routines.

3. Locate and initialize bound NICs – We read the Linkage parameters of the registry to determine which NIC devices we are bound to. For each of these devices we allocate and initialize a IFACE structure (defined above). We then read the TCP parameters out of the registry for each bound device and set the corresponding fields in the IFACE structure.

After the underlying INIC devices have completed their initialization, NDIS will call our driver's ATKBindAdapter function for each underlying device. It will perform the following:

1. Open the device specified in the call the ATKBindAdapter.

2. Find the IFACE structure that was created in ATKProtoSetup for this device.

3. Query the miniport for adapter information. This includes such things as link speed and MAC address. Save relevant information in the IFACE structure.

4. Perform the interface initialization as specified in the section on Interface Initialization.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 096

RECEIVE.

Receive is handled by the protocol driver routine ATKReceivePacket. Before we describe this routine, it is important to consider each possible receive type and how it will be handled.

RECEIVE OVERVIEW.

Our INIC miniport driver will be bound to our transport driver as well as the generic Microsoft TCP driver (and possibly others). The ATCP driver will be bound exclusively to INIC devices, while the Microsoft TCP driver will be bound to INIC devices as well as other types of NICs. This is illustrated in Fig. 31. By binding the driver in this fashion, we can choose to direct incoming network data to our own ATCP transport driver, the Microsoft TCP driver, or both. We do this by playing with the ethernet "type" field as follows.

To NDIS and the transport drivers above it, our card is going to be registered as a normal ethernet card. When a transport driver receives a packet from our driver, it will expect the data to start with an ethernet header, and consequently, expects the protocol type field to be in byte offset 12. If Microsoft TCP finds that the protocol type field is not equal to either IP, or ARP, it will not accept the packet. So, to deliver an incoming packet to our driver, we must simply map the data such that byte 12 contains a non-recognized ethernet type field. Note that we must choose a value that is greater than 1500 bytes so that the transport drivers do not confuse it with an 802.3 frame. We must also choose a value that will not be accepted by other transport driver such as Appletalk or IPX. Similarly, if we want to direct the data to Microsoft TCP, we can then simply leave the ethernet type field set to IP (or ARP). Note that since we will also see these frames we can choose to accept or not-accept them as necessary. Incoming packets are delivered as follows:

A. Packets delivered to ATCP only (not accepted by MSTCP):

1. All TCP packets destined for one of our IP addresses. This includes both slow-path frames and fast-path frames. In the slow-path case, the TCP frames are given in there entirety (headers included). In the fast-path case, the ATKReceivePacket is given a header buffer that contains status information and data with no headers (except those above TCP). More on this later.

B. Packets delivered to Microsoft TCP only (not accepted by ATCP):

1. All non-TCP packets.

[redacted]

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 097

2. All packets that are not destined for one of our interfaces (packets that will be routed). Continuing the above example, if there is an IP address 144.48.252.4 associated with the 3com interface, and we receive a TCP connect with a destination IP address of 144.48.252.4, we will actually want to send that request up to the ATCP driver so that we create a fast-path connection for it. This means that we will need to know every IP address in the system and filter frames based on the destination IP address in a given TCP datagram. This can be done in the INIC miniport driver. Since it will be the ATCP driver that learns of dynamic IP address changes in the system, we will need a method to notify the INIC miniport of all the IP addresses in the system. More on this later.

C. Packets delivered to both:

1. All ARP frames.
2. All ICMP frames.

TWO TYPES OF RECEIVE PACKETS.

There are several circumstances in which the INIC will need to indicate extra information about a receive packet to the ATCP driver. One such example is a fast path receive in which the ATCP driver will need to be notified of how much data the card has buffered. To accomplish this, the first (and sometimes only) buffer in a received packet will actually be an INIC header buffer. The header buffer contains status information about the receive packet, and may or may not contain network data as well. The ATCP driver will recognize a header buffer by mapping it to an ethernet frame and inspecting the type field found in byte 12. We will indicate all TCP frames destined for us in this fashion, while frames that are destined for both our driver and the Microsoft TCP driver (ARP, ICMP) will be indicated without a header buffer. Fig. 32 shows an example of an incoming TCP packet. Fig. 33 shows an example of an incoming ARP frame.

NDIS 4 PROTOCOLRECEIVEPACKET OPERATION.

NDIS has been designed such that all packets indicated via NdisMIndicateReceivePacket by an underlying miniport are delivered to the ProtocolReceivePacket routine for all protocol drivers bound to it. These protocol drivers can choose to accept or not accept the data. They can either accept the data by copying the data out of the packet indicated to it, or alternatively they can keep the packet and return it later via

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 098

a call to NdisReturnPackets. By implementing it in this fashion, NDIS allows more than one protocol driver to accept a given packet. For this reason, when a packet is delivered to a protocol driver, the contents of the packet descriptor, buffer descriptors and data must all be treated as read-only. At the moment, we intend to violate this rule. We choose to violate this because much of the FreeBSD code modifies the packet headers as it examines them (mostly for endian conversion purposes). Rather than modify all of the FreeBSD code, we will instead ensure that no other transport driver accepts the data by making sure that the ethernet type field is unique to us (no one else will want it). Obviously this only works with data that is only delivered to our ATCP driver. For ARP and ICMP frames we will instead copy the data out of the packet into our own buffer and return the packet to NDIS directly. While this is less efficient than keeping the data and returning it later, ARP and ICMP traffic should be small enough, and infrequent enough, that it doesn't matter.

The DDK specifies that when a protocol driver chooses to keep a packet, it should return a value of 1 (or more) to NDIS in its ProtocolReceivePacket routine. The packet is then later returned to NDIS via the call to NdisReturnPackets. This can only happen after the ProtocolReceivePacket has returned control to NDIS. This requires that the call to NdisReturnPackets must occur in a different execution context. We can accomplish this by scheduling a DPC, scheduling a system thread, or scheduling a kernel thread of our own. For brevity in this section, we will assume it is a done through a DPC. In any case, we will require a queue of pending receive buffers on which to place and fetch receive packets.

After a receive packet is dequeued by the DPC it is then either passed to TCP directly for fast-path processing, or it is sent through the FreeBSD path for slow-path processing. Note that in the case of slow-path processing, we may be working on data that needs to be returned to NDIS (TCP data) or we may be working on our own copy of the data (ARP and ICMP). When we finish with the data we will need to figure out whether or not to return the data to NDIS or not. This will be done via fields in the mbuf header used to map the data. When the mfreem routine is called to free a chain of mbufs, the fields in the mbuf will be checked and, if required, the packet descriptor pointed to by the mbuf will be returned to NDIS.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 099

MBUF ↔ PACKET MAPPING.

As noted in the section on mbufs above, we will map incoming data to mbufs so that our FreeBSD port requires fewer modifications. Depending on the type of data received, this mapping will appear differently. Here are some examples:

In Fig. 34A, we show incoming data for a TCP fast-path connection. In this example, the TCP data is fully contained in the header buffer. The header buffer is mapped by the mbuf and sent upstream for fast-path TCP processing. In this case it is required that the header buffer be mapped and sent upstream because the fast-path TCP code will need information contained in the header buffer in order to perform the processing. When the mbuf in this example is freed, the mfreem routine will determine that the mbuf maps a packet that is owned by NDIS and will then free the mbuf header only and call NdisReturnPackets to free the data.

In Fig. 34B, we show incoming data for a TCP slow-path connection. In this example the mbuf points to the start of the TCP data directly instead of the header buffer. Since this buffer will be sent up for slow-path FreeBSD processing, we can not have the mbuf pointing to a header buffer (FreeBSD would get awfully confused). Again, when mfreem is called to free the mbuf, it will discover the mapped packet, free the mbuf header, and call NDIS to free the packet and return the underlying buffers. Note that even though we do not directly map the header buffer with the mbuf we do not lose it because of the link from the packet descriptor. Note also that we could alternatively have the INIC miniport driver only pass us the TCP data buffer when it receives a slow-path receive. This would work fine except that we have determined that even in the case of slow-path connections we are going to attempt to offer some assistance to the host TCP driver (most likely by checksum processing only). In this case there may be some special fields that we need to pass up to the ATCP driver from the INIC driver. Leaving the header buffer connected seems the most logical way to do this.

Finally, in Fig. 34C, we show a received ARP frame. Recall that for incoming ARP and ICMP frames we are going to copy the incoming data out of the packet and return it directly to NDIS. In this case the mbuf simply points to our data, with no corresponding packet descriptor. When we free this mbuf, mfreem will discover this and free not only the mbuf header, but the data as well.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 100

OTHER RECEIVE PACKETS.

We use this receive mechanism for other purposes besides the reception of network data. It is also used as a method of communication between the ATCP driver and the INIC. One such example is a TCP context flush from the INIC. When the INIC determines, for whatever reason, that it can no longer manage a TCP connection, it must flush that connection to the ATCP driver. It will do this by filling in a header buffer with appropriate status and delivering it to the INIC driver. The INIC driver will in turn deliver it to the protocol driver which will treat it essentially like a fast-path TCP connection by mapping the header buffer with an mbuf header and delivering it to TCP for fast-path processing. There are two advantages to communicating in this manner. First, it is already an established path, so no extra coding or testing is required. Second, since a context flush comes in, in the same manner as received frames, it will prevent us from getting a slow-path frame before the context has been flushed.

SUMMARY

Having covered all of the various types of receive data, following are the steps that are taken by the ATKProtocolReceivePacket routine.

1. Map incoming data to an ethernet frame and check the type field;
2. If the type field contains our custom INIC type then it should be TCP;
3. If the header buffer specifies a fast-path connection, allocate one or more mbufs headers to map the header and possibly data buffers. Set the packet descriptor field of the mbuf to point to the packet descriptor, set the mbuf flags appropriately, queue the mbuf, and return 1;
4. If the header buffer specifies a slow-path connection, allocate a single mbuf header to map the network data, set the mbuf fields to map the packet, queue the mbuf and return 1. Note that we design the INIC such that we will never get a TCP segment split across more than one buffer;
5. If the type field of the frame indicates ARP or ICMP;
6. Allocate a mbuf with a data buffer. Copy the contents of the packet into the mbuf. Queue the mbuf, and return 0 (not accepted); and
7. If the type field is not either the INIC type, ARP or ICMP, we don't want it. Return 0.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 101

The receive processing will continue when the mbufs are dequeued. At the moment this is done by a routine called ATKProtocolReceiveDPC. It will do the following:

1. Dequeue a mbuf from the queue; and
2. Inspect the mbuf flags. If the mbuf is meant for fast-path TCP, it will call the fast-path routine directly. Otherwise it will call the ethernet input routine for slow-path processing.

TRANSMIT.

In this section we discuss the ATCP transmit path.

NDIS 4 SEND OPERATION.

The NDIS 4 send operation works as follows. When a transport/protocol driver wishes to send one or more packets down to an NDIS 4 miniport driver, it calls NdisSendPackets with an array of packet descriptors to send. As soon as this routine is called, the transport/protocol driver relinquishes ownership of the packets until they are returned, one by one in any order, via a NDIS call to the ProtocolSendComplete routine. Since this routine is called asynchronously, our ATCP driver must save any required context into the packet descriptor header so that the appropriate resources can be freed. This is discussed further in the following sections.

TYPES OF "SENDS".

Like the Receive path described above, the transmit path is used not only to send network data, but is also used as a communication mechanism between the host and the INIC. Here are some examples of the types of sends performed by the ATCP driver.

FAST-PATH TCP SEND.

When the ATCP driver receives a transmit request with an associated MDL, it will package up the MDL physical addresses into a command buffer, map the command buffer with a buffer and packet descriptor, and call NdisSendPackets with the corresponding packet. The underlying INIC driver will issue the command buffer to the INIC. When the corresponding response buffer is given back to the host, the INIC miniport will call NdisMSendComplete which will result in a call to the ATCP ProtocolSendComplete

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 102

(ATKSendComplete) routine, at which point the resources associated with the send can be freed. We will allocate and use a mbuf to hold the command buffer. By doing this we can store the context necessary in order to clean up after the send completes. This context includes a pointer to the MDL and presumably some other connection context as well. The other advantage to using a mbuf to hold the command buffer is that it eliminates having another special set of code to allocate and return command buffer. We will store a pointer to the mbuf in the reserved section of the packet descriptor so we can locate it when the send is complete. Fig. 35 illustrates the relationship between the client's MDL, the command buffer, and the buffer and packet descriptors.

FAST-PATH TCP RECEIVE.

As described in the Fast-Path Input Data Flow section above, the receive process typically occurs in two phases. First the INIC fills in a host receive buffer with a relatively small amount of data, but notifies the host of a large amount of pending data (either through a large amount of buffered data on the card, or through a large amount of expected NetBios data). This small amount of data is delivered to the client through the TDI interface. The client will then respond with a MDL in which the data should be placed. Like the Fast-path TCP send process, the receive portion of the ATCP driver will then fill in a command buffer with the MDL information from the client, map the buffer with packet and buffer descriptors and send it to the INIC via a call to NdisSendPackets. Again, when the response buffer is returned to the INIC miniport, the ATKSendComplete routine will be called and the receive will complete. This relationship between the MDL, command buffer and buffer and packet descriptors are the same as shown in the Fast-path send section above.

SLOW-PATH (FREEBSD).

Slow-path sends pass through the FreeBSD stack until the ethernet header is prepended in ether_output and the packet is ready to be sent. At this point a command buffer will be filled with pointers to the ethernet frame, the command buffer will be mapped with a packet and buffer descriptor and NdisSendPackets will be called to hand the packet off to the miniport. Fig. 36 shows the relationship between the mbufs, command buffer, and buffer and packet descriptors. Since we will use a mbuf to map the command buffer, we can simply link

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 103

the data mbufs directly off of the command buffer mbuf. This will make the freeing of resources much simpler.

NON-DATA COMMAND BUFFER.

The transmit path is also used to send non-data commands to the card. As shown in Fig. 37, for example, the ATCP driver gives a context to the INIC by filling in a command buffer, mapping it with a packet and buffer descriptor, and calling NdisSendPackets.

ATKPROTOCOLSENDCOMPLETE.

Given the above different types of sends, the ATKProtocolSendComplete routine will perform various types of actions when it is called from NDIS. First it must examine the reserved area of the packet descriptor to determine what type of request has completed. In the case of a slow-path completion, it can simply free the mbufs, command buffer, and descriptors and return. In the case of a fast-path completion, it will need to notify the TCP fast path routines of the completion so TCP can in turn complete the client's IRP. Similarly, when a non-data command buffer completes, TCP will again be notified that the command sent to the INIC has completed.

TDI FILTER DRIVER.

In a first embodiment of the product, the INIC handles only simple-case data transfer operations on a TCP connection. (These of course constitute the large majority of CPU cycles consumed by TCP processing in a conventional driver.)

There are many other complexities of the TCP protocol which must still be handled by host driver software: connection setup and breakdown, out-of-order data, nonstandard flags, etc.

The NT OS contains a fully functional TCP/IP driver, and one solution would be to enhance this so that it is able to detect our INIC and take advantage of it by "handing off" data-path processing where appropriate.

Unfortunately, we do not have access to NT source, let alone permission to modify NT. Thus the solution above, while a goal, cannot be done immediately. We instead provide our own custom driver software on the host for those parts of TCP processing which are not handled by the INIC.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 104

This presents a challenge. The NT network driver framework does make provision for multiple types of protocol driver: but it does not easily allow for multiple instances of drivers handling the SAME protocol.

For example, there are no "hooks" into the Microsoft TCP/IP driver which would allow for routing of IP packets between our driver (handling our INICs) and the Microsoft driver (handling other NICs).

Our approach to this is to retain the Microsoft driver for all non-TCP network processing (even for traffic on our INICs), but to invisibly "steal" TCP traffic on our connections and handle it via our own (BSD-derived) driver. The Microsoft TCP/IP driver is unaware of TCP connections on interfaces we handle.

The network "bottom end" of this artifice is described earlier in the document. In this section we will discuss the "top end": the TDI interface to higher-level NT network client software.

We make use of an NT facility called a filter driver. NT allows a special type of driver ("filter driver") to attach itself "on top" of another driver in the system. The NT I/O manager then arranges that all requests directed to the attached driver are sent first to the filter driver; this arrangement is invisible to the rest of the system.

The filter driver may then either handle these requests itself, or pass them down to the underlying driver it is attached to. Provided the filter driver completely replicates the (externally visible) behavior of the underlying driver when it handles requests itself, the existence of the filter driver is invisible to higher-level software.

The filter driver attaches itself on top of the Microsoft TCP/IP driver; this gives us the basic mechanism whereby we can intercept requests for TCP operations and handle them in our driver instead of the Microsoft driver.

However, while the filter driver concept gives us a framework for what we want to achieve, there are some significant technical problems to be solved. The basic issue is that setting up a TCP connection involves a sequence of several requests from higher-level software, and it is not always possible to tell, for requests early in this sequence, whether the connection should be handled by our driver or by the Microsoft driver.

Thus for many requests, we store information about the request in case we need it later, but also allow the request to be passed down to the Microsoft TCP/IP driver in case the connection ultimately turns out to be one which that driver should handle.

[redacted]

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 105

Let us look at this in more detail, which will involve some examination of the TDI interface: the NT interface into the top end of NT network protocol drivers. Higher-level TDI client software which requires services from a protocol driver proceeds by creating various types of NT FILE_OBJECTs, and then making various DEVICE_IO_CONTROL requests on these FILE_OBJECTs.

There are two types of FILE_OBJECT of interest here. Local IP addresses that are represented by ADDRESS objects, and TCP connections that are represented by CONNECTION objects. The steps involved in setting up a TCP connection (from the "active" client side, for a CONNECTION object) are:

1) Create an ADDRESS object; 2) Create a CONNECTION object; 3) Issue a TDI_ASSOCIATE_ADDRESS io-control to associate the CONNECTION object with the ADDRESS object; and 4) Issue a TDI_CONNECT io-control on the CONNECTION object, specifying the remote address and port for the connection.

Initial thoughts were that handling this would be straightforward: we would tell, on the basis of the address given when creating the ADDRESS object, whether the connection is for one of our interfaces or not. After which, it would be easy to arrange for handling entirely by our code, or entirely by the Microsoft code: we would simply examine the ADDRESS object to see if it was "one of ours" or not.

There are two main difficulties, however. First, when the CONNECTION object is created, no address is specified: it acquires a local address only later when the TDI_ASSOCIATE_ADDRESS is done. Also, when a CONNECTION object is created, the caller supplies an opaque "context cookie" which will be needed for later communications with that caller. Storage of this cookie is the responsibility of the protocol driver: it is not directly derivable just by examination of the CONNECTION object itself. If we simply passed the "create" call down to the Microsoft TCP/IP driver, we would have no way of obtaining this cookie later if it turns out that we need to handle the connection. Therefore, for every CONNECTION object which is created we allocate a structure to keep track of information about it, and store this structure in a hash table keyed by the address of the CONNECTION object itself, so that we can locate it if we later need to process requests on this object. We refer to this as a "shadow" object: it replicates information about the object stored in the Microsoft driver. (We must, of course, also pass the create request down to the Microsoft driver too, to allow it to set up its own administrative information about the object.)

[redacted]

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 106

A second major difficulty arises with ADDRESS objects. These are often created with the TCP/IP "wildcard" address (all zeros); the actual local address is assigned only later during connection setup (by the protocol driver itself.) Of course, a "wildcard" address does not allow us to determine whether connections that will be associated with this ADDRESS object should be handled by our driver or by the Microsoft one. Also, as with CONNECTION objects, there is "opaque" data associated with ADDRESS objects that cannot be derived just from examination of the object itself. (In this case addresses of callback functions set on the object by TDI_SET_EVENT io-controls.)

Thus, as in the CONNECTION object case, we create a "shadow" object for each ADDRESS object which is created with a wildcard address. In this we store information (principally addresses of callback functions) which we will need if we are handling connections on CONNECTION objects associated with this ADDRESS object. We store similar information, of course, for any ADDRESS object which is explicitly for one of our interface addresses; in this case we don't need to also pass the create request down to the Microsoft driver.

With this concept of "shadow" objects in place, let us revisit the steps involved in setting up a connection, and look at the processing required in our driver.

First, the TDI client makes a call to create the ADDRESS object. Assuming that this is a "wildcard" address, we create a "shadow" object before passing the call down to the Microsoft driver.

The next step (omitted in the earlier list for brevity) is normally that the client makes a number of TDI_SET_EVENT io-control calls to associate various callback functions with the ADDRESS object. These are functions that should be called to notify the TDI client when certain events (such arrival of data or disconnection requests etc) occur. We store these callback function pointers in our "shadow" address object, before passing the call down to the Microsoft driver.

Next, the TDI client makes a call to create a CONNECTION object. Again, we create our "shadow" of this object.

Next, the client issues the TDI_ASSOCIATE_ADDRESS io-control to bind the CONNECTION object to the ADDRESS object. We note the association in our "shadow" objects, and also pass the call down to the Microsoft driver.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 107

Finally the TDI client issues a TDI_CONNECT io-control on the CONNECTION object, specifying the remote IP address (and port) for the desired connection. At this point, we examine our routing tables to determine if this connection should be handled by one of our interfaces, or by some other NIC. If it is ours, we mark the CONNECTION object as "one of ours" for future reference (using an opaque field which NT FILE_OBJECTS provide for driver use.) We then proceed with connection setup and handling in our driver, using information stored in our "shadow" objects. The Microsoft driver does not see the connection request or any subsequent traffic on the connection.

If the connection request is NOT for one of our interfaces, we pass it down to the Microsoft driver. Note carefully, however, that we can not simply discard our "shadow" objects at this point. The TDI interface allows re-use of CONNECTION objects: on termination of a connection, it is legal for the TDI client to dissociate the CONNECTION object from its current . Thus our "shadow" objects must be retained for the lifetime ADDRESS object, re-associate it with another, and use it for another connection of the NT FILE_OBJECTS: the subsequent connection could turn out to be via one of our interfaces.

TIMERS.

KEEPALIVE TIMER.

We don't want to implement keepalive timers on the INIC. It would in any case be a very poor use of resources to have an INIC context sitting idle for two hours.

IDLE TIMER.

We will keep an idle timer in the ATCP driver for connections that are managed by the INIC (resetting it whenever we see activity on the connection), and cause a flush of context back to the host if this timer expires. We may want to make the threshold substantially lower than 2 hours, to reclaim INIC context slots for useful work sooner. May also want to make that dependent on the number of contexts which have actually been handed out: don't need to reclaim them if we haven't handed out the max.

RECEIVE AND TRANSMIT MICROCODE DESIGN.

This section provides a general description of the design of the microcode that will execute on two of the sequencers of the Protocol Processor on the INIC. The overall

[redacted]

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 108

philosophy of the INIC is discussed in other sections. This section will discuss the INIC microcode in detail.

DESIGN OVERVIEW.

As specified in other sections, the INIC supplies a set of 3 custom processors that will provide considerable hardware-assist to the microcode running thereon. The paragraphs immediately following list the main hardware-assist features:

1) Header processing with specialized DMA engines to validate an input header and generate a context hash, move the header into fast memory and do header comparisons on a DRAM-based TCP control block;

2) DRAM fifos for free buffer queues (large & small), receive-frame queues, event queues etc.;

3) Header compare logic;

4) Checksum generation;

5) Multiple register contexts with register access controlled by simply setting a context register. The Protocol Processor will provide 512 SRAM-based registers to be shared among the 3 sequencers;

6) Automatic movement of input frames into DRAM buffers from the MAC Fifos;

7) Run receive processing on one sequencer and transmit processing on the other. This was chosen as opposed to letting both sequencers run receive and transmit. One of the main reasons for this is that the header-processing hardware can not be shared and interlocks would be needed to do this. Another reason is that interlocks would be needed on the resources used exclusively by receive and by transmit;

8) The INIC will support up to 256 TCP connections (TCB's). A TCB is associated with an input frame when the frame's source and destination IP addresses and source and destination ports match that of the TCB. For speed of access, the TCB's will be maintained in a hash table in NIC DRAM to save sequential searching. There will however, be an index in hash order in SRAM. Once a hash has been generated, the TCB will be cached in SRAM. There will be up to 8 cached TCBs in SRAM. These cache locations can be shared between both sequencers so that the sequencer with the heavier load will be able to use more cache buffers. There will also be 8 header buffers to be shared between the sequencers. Note that each header buffer is not statically linked to a specific TCB buffer. In fact the link is dynamic

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 109

on a per-frame basis. The need for this dynamic linking will be explained in later sections. Suffice to say here that if there is a free header buffer, then somewhere there is also a free TCB SRAM buffer;

9) There were 2 basic implementation options considered here. The first was single-stack and the second was a process model. The process model was chosen here because the custom processor design is providing zero-cost overhead for context switching through the use of a context base register, and because there will be more than enough process slots (or contexts) available for the peak load. It is also expected that all "local" variables will be held permanently in registers whilst an event is being processed;

10) The features that provide this are 256 of the 512 SRAM-based registers that will be used for the register contexts. This can be divided up into 16 contexts (or processes) of 16 registers each. Then 8 of these will be reserved for receive and 8 for transmit. A Little's Law analysis has shown that in order to support 512 byte frames at maximum arrival rate of 4 * 100 Mbits, requires more than 8 jobs to be in process in the NIC. However each job requires an SRAM buffer for a TCB context and at present, there are only 8 of these currently specified due to SRAM space limits. So more contexts (e.g. 32 * 8 regs each) do not seem worthwhile. Refer to the section entitled "LOAD CALCULATIONS" for more details of this analysis. A context switch simply involves reloading the context base register based on the context to be restarted, and jumping to the appropriate address for resumption;

11) To better support the process model chosen, the code will lock an active TCB into an SRAM buffer while either sequencer is operating on it. This implies there will be no swapping to and from DRAM of a TCB once it is in SRAM and an operation is started on it. More specifically, the TCB will not be swapped after requesting that a DMA be performed for it. Instead, the system will switch to another active "process". Then it will resume the former process at the point directly after where the DMA was requested. This constitutes a zero-cost switch as mentioned above;

12) Individual TCB state machines will be run from within a "process". There will be a state machine for the receive side and one for the transmit side. The current TCB states will be stored in the SRAM TCB index table entry;

13) The INIC will have 16 MB of DRAM. The current specification calls for dividing a large portion of this into 2K buffers and control allocation / deallocation of these buffers

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 110

through one of the DRAM fifos mentioned above. These fifos will also be used to control small host buffers, large host buffers, command buffers and command response buffers;

14) For events from one sequencer to the other (i.e. RCV ↔ XMT), the current specification calls for using simple SRAM CIO buffers, one for each direction;

15) Each sequencer handles its own timers independently of the others;

16) Contexts will be passed to the INIC through the Transmit command and response buffers. INIC-initiated TCB releases will be handled through the Receive small buffers. Host-initiated releases will use the Command buffers. There needs to be strict handling of the acquisition and release of contexts to avoid windows where for example, a frame is received on a context just after the context was passed to the INIC, but before the INIC has "accepted" it; and

17) T/TCP (Transaction TCP): the initial INIC will not handle T/TCP connections. This is because they are typically used for the HTTP protocol and the client for that protocol typically connects, sends a request and disconnects in one segment. The server sends the connect confirm, reply and disconnect in his first segment. Then the client confirms the disconnect. This is a total of 3 segments for the life of a context. Typical data lengths are on the order of 300 bytes from the client and 3K from the server. The INIC will provide as good an assist as seems necessary here by checksumming the frame and splitting headers and data. The latter is only likely when data is forwarded with a request such as when a filled-in form is sent by the client.

SRAM REQUIREMENTS.

SRAM requirements for the Receive and Transmit engines are shown in Fig. 38. Depending upon the available space, the number of TCB buffers may be increased to 16.

GENERAL PHILOSOPHY.

The basic plan is to have the host determine when a TCP connection is able to be handed to the INIC, setup the TCB and pass it to the card via a command in the Transmit queue. TCBs that the INIC owns can be handed back to the host via a request from the Receive or Transmit sequencers or from the host itself at any time.

When the INIC receives a frame, one of its immediate tasks is to determine if the frame is for a TCB that it controls. If not, the frame is passed to the host on a generic interface TCB.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 111

On transmit, the transmit request will specify a TCB hash number if the request is on a INIC-controlled TCB. Thus the initial state for the INIC will be transparent mode in which all received frames are directly passed through and all transmit requests will be simply thrown on the appropriate wire. This state is maintained until the host passes TCBs to the INIC to control. Note that frames received for which the INIC has no TCB (or it is with the host) will still have the TCP checksum verified if TCP/IP, and may split the TCPIP header off into a separate buffer.

REGISTER USAGE.

There will be 512 registers available. The first 256 will be used for process contexts. The remaining 256 will be split between the three sequencers as follows: 1) 257 – 320: 64 for RCV general processing / main loop; 2) 321 - 384: 64 for XMT general processing / main loop; and 3) 385 – 512: 128 for three sequencer use.

RECEIVE PROCESSING.

MAIN LOOP.

Fig. 39 is a summary of the main loop of Receive.

RECEIVE EVENTS.

The events that will be processed on a given context are:

1) accept a context;

2) release a context command (from the host via Transmit);

3) release a context request (from Transmit);

4) receive a valid frame; this will actually become 2 events based on the received frame - receive an ACK, receive a segment;

5) receive an "invalid" frame i.e. one that causes the TCB to be flushed to the host;

6) a valid ACK needs to be sent (delayed ACK timer expiry); and

7) There are expected to be the following sources of events: a) Receive input queue: it is expected that hardware will automatically DMA arriving frames into frame buffers and queue an event into a RCV-event queue; b) Timer event queue: expiration of a timer will queue an event into this queue; and c) Transmit sequencer queue: for requests from the transmit processor.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 112

For the sake of brevity the following only discusses receive-frame processing.

RECEIVE DETAILS – VALID CONTEXT.

The base for the receive processing done by the INIC on an existing context is the fast-path or "header prediction" code in the FreeBSD release. Thus the processing is divided into three parts: header validation and checksumming, TCP processing and subsequent SMB processing.

HEADER VALIDATION.

There is considerable hardware assist here. The first step in receive processing is to DMA the frame header into an SRAM header buffer. It is useful for header validation to be implemented in conjunction with this DMA by scanning the data as it flies by. The following tests need to be "passed":

1) MAC header: destination address is our MAC address (not MC or BC too), the Ethertype is IP; 2) IP header: header checksum is valid, header length = 5, IP length > header length, protocol = TCP, no fragmentation, destination IP is our IP address; and 3) TCP header: checksum is valid (incl. pseudo-header), header length = 5 or 8 (timestamp option), length is valid, dest port = SMB or FTP data, no FIN/SYN/URG/PSH/RST bits set, timestamp option is valid if present, segment is in sequence, the window size did not change, this is not a retransmission, it is a pure ACK or a pure receive segment, and most important, a valid context exists. The valid-context test is non-trivial in the amount of work involved to determine it. Also note that for pure ACKs, the window-size test will be relaxed. This is because initially the output PERSIST state is to be handled on the INIC.

Many but perhaps not all of these tests will be performed in hardware – depending upon the embodiment.

TCP PROCESSING.

Once a frame has passed the header validation tests, processing splits based on whether the frame is a pure ACK or a pure received segment.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 113

PURE RCV PACKET.

The design is to split off headers into a small header buffer and pass the aligned data in separate large buffers. Since a frame has been received, eventually some receiver process on the host will need to be informed. In the case of FTP, the frame is pure data and it is passed to the host immediately. This involves getting large buffers and DMAing the data into them, then setting the appropriate details in a small buffer that is used to notify the host. However for SMB, the INIC is performing reassembly of data when the frame consists of headers and data. So there may not yet be a complete SMB to pass to the host. In this case, a small buffer will be acquired and the header moved into it. If the received segment completes an SMB, then the procedures are pretty much as for FTP. If it does not, then the scheme is to at least move the received data (not the headers) to the host to free the INIC buffers and to save latency. The list of in-progress host buffers is maintained in the TCB and moved to the header buffer when the SMB is complete.

The final part of pure-receive processing is to fire off the delayed ACK timer for this segment.

PURE ACK.

Pure ACK processing implies this TCB is the sender, so there may be transmit buffers that can be returned to the host. If so, send an event to the Transmit processor (or do the processing here). If there is more output available, send an event to the transmit processor. Then appropriate actions need to be taken with the retransmission timer.

SMB PROCESSING.

Fig. 40 shows the format of the SMB header of an SMB frame. The LENGTH field of the NetBIOS header will be used to determine when a complete SMB has been received and the header buffer with appropriate details can be posted to the host. The interesting commands are the write commands: SMBwrite (0xB), SMBwriteBraw (0x1D), SMBwriteBmpx (0x1E), SMBwriteBs (0x1F), SMBwriteclose (0x2C), SMBwriteX (0x2F), SMBwriteunlock (0x14). These are interesting because they will have data to be aligned in host memory. The point to note about these commands is that they each have a different WCT field, so that the start offset of the data depends on the command type. SMB processing will thus need to be cognizant of these types.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 114

RECEIVE DETAILS – NO VALID CONTEXT.

The design here is to provide as much assist as possible. Frames will be checksummed and the TCPIP headers may be split off.

RECEIVE NOTES.

1. PRU_RCVD or the equivalent in Microsoft language: the host application has to tell the INIC when he has accepted the received data that has been queued. This is so that the INIC can update the receive window. It is an advantage for this mechanism to be efficient. This may be accomplished by piggybacking these on transmit requests (not necessarily for the same TCB).

2. Keepalive Timer: for a INIC-controlled TCB, the INIC will not maintain this timer. This leaves the host with the job of determining that the TCB is still active.

3. Timestamp option: it is useful to support this option in the fast path because the BSD implementation does. Also, it can be very helpful in getting a much better estimate of the round-trip time (RTT) which TCP needs to use.

4. Idle timer: the INIC will not maintain this timer (see Note 2 above).

5. Frame with no valid context: The INIC may split TCP/IP headers into a separate header buffer.

TRANSMIT PROCESSING.

MAIN LOOP.

Fig. 41 is a summary of the main loop of Transmit.

TRANSMIT EVENTS.

The events that will be processed on a given context and their sources are: 1) accept a context (from the Host); 2) release a context command (from the Host); 3) release a context command (from Receive); 4) valid send request and window > 0 (from host or RCV sequencer); 5) valid send request and window = 0 (from host or RCV sequencer); 6) send a window update (host has accepted data); 7) persist timer expiration (persist timer); 8) context-release event e.g. window shrank (XMT processing or retransmission timer); and 9) receive-release request ACK(from RCV sequencer).

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 115

TRANSMIT DETAILS – VALID CONTEXT.

The following is an overview of the transmit flow: The host posts a transmit request to the INIC by filling in a command buffer with appropriate data pointers etc and posting it to the INIC via the Command Buffer Address register. Note that there is one host command buffer queue, but there are four physical transmit lines. So each request needs to include an interface number as well as the context number. The INIC microcode will DMA the command in and place it in one of four internal command queues which the transmit sequencer will work on. This is so that transmit processing can round-robin service these four queues to keep all four interfaces busy, and not let a highly-active interface lock out the others (which would happen with a single queue). The transmit request may be a segment that is less than the MSS, or it may be as much as a full 64K SMB READ. Obviously the former request will go out as one segment, the latter as a number of MSS-sized segments. The transmitting TCB must hold on to the request until all data in it has been transmitted and acked. Appropriate pointers to do this will be kept in the TCB. A large buffer is acquired from the free buffer fifo, and the MAC and TCP/IP headers are created in it. It may be quicker/simpler to keep a basic frame header set up in the TCB and either DMA directly this into the frame each time. Then data is DMA'd from host memory into the frame to create an MSS-sized segment. This DMA also checksums the data. Then the checksum is adjusted for the pseudo-header and placed into the TCP header, and the frame is queued to the MAC transmit interface which may be controlled by the third sequencer. The final step is to update various window fields etc in the TCB. Eventually either the entire request will have been sent and acked, or a retransmission timer will expire in which case the context is flushed to the host. In either case, the INIC will place a command response in the Response queue containing the command buffer handle from the original transmit command and appropriate status.

The above discussion has dealt how an actual transmit occurs. However the real challenge in the transmit processor is to determine whether it is appropriate to transmit at the time a transmit request arrives. There are many reasons not to transmit: the receiver's window size is <= 0, the Persist timer has expired, the amount to send is less than a full segment and an ACK is expected / outstanding, the receiver's window is not half-open etc. Much of the transmit processing will be in determining these conditions.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 116

TRANSMIT DETAILS – NO VALID CONTEXT.

The main difference between this and a context-based transmit is that the queued request here will already have the appropriate MAC and TCP/IP (or whatever) headers in the frame to be output. Also the request is guaranteed not to be greater than MSS-sized in length. So the processing is fairly simple. A large buffer is acquired and the frame is DMAed into it, at which time the checksum is also calculated. If the frame is TCP/IP, the checksum will be appropriately adjusted if necessary (pseudo-header etc) and placed in the TCP header. The frame is then queued to the appropriate MAC transmit interface. Then the command is immediately responded to with appropriate status through the Response queue.

TRANSMIT NOTES.

1) Slow-start: the INIC will handle the slow-start algorithm that is now a part of the TCP standard. This obviates waiting until the connection is sending a full-rate before passing it to the INIC.

2) Window Probe vs Window Update - an explanation for posterity. A Window Probe is sent from the sending TCB to the receiving TCB, and it means the sender has the receiver in PERSIST state. Persist state is entered when the receiver advertises a zero window. It is thus the state of the transmitting TCB. In this state, he sends periodic window probes to the receiver in case an ACK from the receiver has been lost. The receiver will return his latest window size in the ACK. A Window Update is sent from the receiving TCB to the sending TCB, usually to tell him that the receiving window has altered. It is mostly triggered by the upper layer when it accepts some data. This probably means the sending TCB is viewing the receiving TCB as being in PERSIST state.

3) Persist state: it is designed to handle Persist state on the INIC. It seems unreasonable to throw a TCB back to the host just because its receiver advertised a zero window. This would normally be a transient situation, and would tend to happen mostly with clients that do not support slow-start. Alternatively, the code can easily be changed to throw the TCB back to the host as soon as a receiver advertises a zero window.

4) MSS-sized frames: the INIC code will expect all transmit requests for which it has no TCB to not be greater than the MSS. If any request is, it will be dropped and an appropriate response status posted.

[redacted]

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 117

5) Silly Window avoidance: as a receiver, the INIC will do the right thing here and not advertise small windows – this is easy. However it is necessary to also do things to avoid this as a sender, for the cases where a stupid client does advertise small windows. Without getting into too much detail here, the mechanism requires the INIC code to calculate the largest window advertisement ever advertised by the other end. It is an attempt to guess the size of the other end's receive buffer and assumes the other end never reduces the size of its receive buffer. See Stevens, "TCP/IP Illustrated", Vol. 1, pp. 325-326 (1994).

THE UTILITY PROCESSOR.

SUMMARY.

The following is a summary of the main functions of the utility sequencer of the microprocessor:

1) Look at the event queues: Event13Type & Event23Type (we assume there will be an event status bit for this - USE_EV13 and USE_EV23) in the events register; these are events from sequencers 1 and 2; they will mainly be XMIT requests from the XMT sequencer. Dequeue request and place the frame on the appropriate interface.

2) RCV-frame support: in the model, RCV is done through VinicReceive() which is registered by the lower-edge driver, and is called at dispatch-level. This routine calls VinicTransferDataComplete() to check if the xfer (possibly DMA) of the frame into host buffers is complete. The latter rtne is also called at dispatch level on a DMA-completion interrupt. It queues complete buffers to the RCV sequencer via the normal queue mechanism.

3) Other processes may also be employed here for supporting the RCV sequencer.

4) Service the following registers (this will probably involve micro-interrupts):

a) Header Buffer Address register:

Buffers are 256 bytes long on 256-byte boundaries.

31-8 - physical addr in host of a set of contiguous hddr buffers.

7-0 - number of hddr buffers passed.

Use contents to add to SmallHType queue.

b) Data Buffer Handle & Data Buffer Address registers:

Buffers are 4K long aligned on 4K boundaries.

Use contents to add to the FreeType queue.

c) Command Buffer Address register:

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 118

Buffers are multiple of 32 bytes up to 1K long (2**5 * 32).

31-5 - physical addr in host of cmd buffer.

4-0 - length of cmd in bytes/32 (i.e. multiples of 32 bytes).

Points to host cmd; get FreeSType buffer and move.

command into it; queue to Xmit0-Xmit3Type queues.

d) Response Buffer Address register:

Buffers are 32 bytes long on 32-byte boundaries.

31-8 - physical addr in host of a set of contiguous resp buffers.

7-0 - number of resp buffers passed.

Use contents to add to the ResponseType queue.

5) Low buffer threshold support: set approp bits in the ISR when the available-buffers count in the various queues filled by the host falls below a threshold.

FURTHER OPERATIONS OF THE UTILITY PROCESSOR.

The utility processor of the microprocessor housed on the INIC is responsible for setting up and implementing all configuration space and memory mapped operations, and also as described below, for managing the debug interface.

All data transfers, and other INIC initiated transfers will be done via DMA. Configuration space for both the network processor function and the utility processor function will define a single memory space for each. This memory space will define the basic communication structure for the host. In general, writing to one of these memory locations will perform a request for service from the INIC. This is detailed in the memory description for each function. This section defines much of the operation of the Host interface, but should be read in conjunction with the Host Interface Strategy for the Alacritech INIC to fully define the Host/INIC interface.

Two registers, DMA hardware and an interrupt function comprise the INIC interface to the Host through PCI. The interrupt function is implemented via a four bit register (PCI_INT) tied to the PCI interrupt lines. This register is directly accessed by the microprocessor.

THE MICROPROCESSOR uses two registers, the PCI_Data_Reg and the PCI_Address_Reg, to enable the Host to access Configuration Space and the memory space allocated to the INIC. These registers are not available to the Host, but are used by THE

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 119

MICROPROCESSOR to enable Host reads and writes. The function of these two registers is as follows.

1) PCI_Data_Reg: This register can be both read and written by THE MICROPROCESSOR. On write operations from the host, this register contains the data being sent from the host. On read operations, this register contains the data to be sent to the host.

2) PCI_Address_Reg: This is the control register for memory reads and writes from the host. The structure of the register is shown in Fig. 42. During a write operation from the Host the PCI_Data_Reg contains valid data after Data Valid is set in the PCI_Address_Reg. Both registers are locked until THE MICROPROCESSOR writes the PCI_Data_Reg, which resets Data Valid. All read operations will be direct from SRAM. Memory space based reads will return 00. Configuration space reads will be mapped as shown in Fig. 43.

CONFIGURATION SPACE.

The INIC is implemented as a multi-function device. The first device is the network controller, and the second device is the debug interface. An alternative production embodiment may implement only the network controller function. Both configuration space headers will be the same, except for the differences noted in the following description.

Vendor ID – This field will contain the Alacritech Vendor ID. One field will be used for both functions. The Alacritech Vendor ID is hex 139A.

Device ID – Chosen at Alacritech on a device specific basis. One field will be used for both functions.

Command – Initialized to 00. All bits defined below as not enabled (0) will remain 0. Those that are enabled will be set to 0 or 1 depending on the state of the system. Each function (network and debug) will have its own command field, as shown in Fig. 44.

Status – This is not initialized to zero. Each function will have its own field. The configuration is as shown in Fig. 45.

Revision ID – The revision field will be shared by both functions.

Class Code – This is 02 00 00 for the network controller, and for the debug interface. The field will be shared.

Cache Line Size – This is initialized to zero. Supported sizes are 16, 32, 64 and 128 bytes. This hardware register is replicated in SRAM and supported separately for each

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 120

function, but THE MICROPROCESSOR will implement the value set in Configuration Space 1 (the network processor).

Latency Timer – This is initialized to zero. The function is supported. This hardware register is replicated in SRAM. Each function is supported separately, but THE MICROPROCESSOR will implement the value set in Configuration Space 1 (the network processor).

Header Type – This is set to 80 for both functions, but will be supported separately.

BIST – Is implemented. In addition to responding to a request to run self test, if test after reset fails, a code will be set in the BIST register. This will be implemented separately for each function.

Base Address Register – A single base address register is implemented for each function. It is 64 bits in length, and the bottom four bits are configured as follows: Bit 0 – 0, indicates memory base address; Bit 1,2 – 00, locate base address anywhere in 32 bit memory space; and Bit 3 – 1, memory is prefetchable.

CardBus CIS Pointer – Not implemented—initialized to 0.

Subsystem Vendor ID – Not implemented—initialized to 0.

Subsystem ID – Not implemented—initialized to 0.

Expansion ROM Base Address – Not implemented—initialized to 0.

Interrupt Line – Implemented—initialized to 0. This is implemented separately for each function.

Interrupt Pin – This is set to 01, corresponding to INTA# for the network controller, and 02, corresponding to INTB# for the debug interface. This is implemented separately for each function.

Min_Gnt – This can be set at a value in the range of 10, to allow reasonably long bursts on the bus. This is implemented separately for each function.

Max_Lat – This can be set to 0 to indicate no particular requirement for frequency of access to PCI. This is implemented separately for each function.

MEMORY SPACE.

Because each of the following functions may or may not reside in a single location, and may or may not need to be in SRAM at all, the address for each is really only used as an identifier (label). There is, therefore, no control block anywhere in memory that represents

[redacted]

this memory space. When the host writes one of these registers, the utility processor will construct the data required and transfer it. Reads to this memory will generate 00 for data.

NETWORK PROCESSOR.

The following four byte registers, beginning at location h00 of the network processor's allocated memory, are defined.

00 – Interrupt Status Pointer -- Initialized by the host to point to a four byte area where status is stored.

04 – Interrupt Status – Returned status from host. Sent after one or more status conditions have been reset. Also an interlock for storing any new status. Once status has been stored at the Interrupt Status Pointer location, no new status will be ORed until the host writes the Interrupt Status Register. New status will be ored with any remaining uncleared status (as defined by the contents of the returned status) and stored again at the Interrupt Status Pointer location. Bits are as follows:

Bit 31 – ERR -- Error bits are set;

Bit 30 – RCV – Receive has occurred;

Bit 29 – XMT – Transmit command complete; and

Bit 25 – RMISS – Receive drop occurred due to no buffers.

08 – Interrupt Mask – Written by the host. Interrupts are masked for each of the bits in the interrupt status when the same bit in the mask register is set. When the Interrupt Mask register is written and as a result a status bit is unmasked, an interrupt is generated. Also, when the Interrupt Status Register is written, enabling new status to be stored, when it is stored if a bit is stored that is not masked by the Interrupt Mask, an interrupt is generated.

0C – Header Buffer Address – Written by host to pass a set of header buffers to the INIC.

10 – Data Buffer Handle – First register to be written by the Host to transfer a receive data buffer to the INIC. This data is Host reference data. It is not used by the INIC, it is returned with the data buffer. However, to insure integrity of the buffer, this register must be interlocked with the Data Buffer Address register. Once the Data Buffer

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 122

Address register has been written, neither register can be written until after the Data Buffer Handle register has been read by THE MICROPROCESSOR.

14 – Data Buffer Address – Pointer to the data buffer being sent to the INIC by the Host. Must be interlocked with the Data Buffer Handle register.

18 – Command Buffer Address XMT0 – Pointer to a set of command buffers sent by the Host. THE MICROPROCESSOR will DMA the buffers to local DRAM found on the FreeSType queue and queue the Command Buffer Address XMT0 with the local address replacing the host Address.

1C – Command Buffer Address SMT1.

20 – Command Buffer Address SMT2.

24 – Command Buffer Address SMT3.

28 – Response Buffer Address -- Pointer to a set of response buffers sent by the Host. These will be treated in the same fashion as the Command Buffer Address registers.

UTILITY PROCESSOR.

Ending status will be handled by the utility processor in the same fashion as it is handled by the network processor. At present two ending status conditions are defined B31 – command complete, and B30 – error. When end status is stored an interrupt is generated.

Two additional registers are defined, Command Pointer and Data Pointer. The Host is responsible for insuring that the Data Pointer is valid and points to sufficient memory before storing a command pointer. Storing a command pointer initiates command decode and execution by the debug processor. The Host must not modify either command or Data Pointer until ending status has been received, at which point a new command may be initiated.

Memory space is write only by the Host, reads will receive 00. The format is as follows:

00 – Interrupt Status Pointer -- Initialized by the host to point to a four byte area where status is stored.

04 – Interrupt Status – Returned status from host. Sent after one or more status conditions have been reset. Also an interlock for storing any new status. Once status has been stored at the Interrupt Status Pointer location, no new status will be stored until the host writes the Interrupt Status Register. New status will be ored with any remaining

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 123

uncleared status (as defined by the contents of the returned status) and stored again at the Interrupt Status Pointer location. Bits are as follows:

Bit 31 – CC – Command Complete;

Bit 30 – ERR – Error;

Bit29 – Transmit Processor Halted;

Bit28 – Receive Processor Halted; and

Bit27 – Utility Processor Halted.

08 – Interrupt Mask – Written by the host. Interrupts are masked for each of the bits in the interrupt status when the same bit in the mask register is set. When the Interrupt Mask register is written and as a result a status bit is unmasked, an interrupt is generated. Also, when the Interrupt Status Register is written, enabling new status to be stored, when it is stored if a bit is stored that is not masked by the Interrupt Mask, an interrupt is generated.

0C – Command Pointer – Points to command to be executed. Storing this pointer initiates command decode and execution.

10 – Data Pointer – Points to the data buffer. This is used for both read and write data, determined by the command function.

DEBUG INTERFACE.

In order to provide a mechanism to debug the microcode running on the microprocessor sequencers, a debug process has been defined which will run on the utility sequencer. This processor will interface with a control program on the host processor over PCI.

PCI INTERFACE.

This interface is defined in the combination of the Utility Processor and the Host Interface Strategy sections, above.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 124

COMMAND FORMAT.

The first byte of the command, the command byte, defines the structure of the remainder of the command.

COMMAND BYTE.

The first five bits of the command byte are the command itself. The next bit is used to specify an alternate processor, and the last two bits specify which processors are intended for the command.

PROCESSOR BITS.

00 – Any Processor;

01 – Transmit Processor;

10 – Receive Processor; and

11 – Utility Processor.

ALTERNATE PROCESSOR.

This bit defines which processor should handle debug processing if the utility processor is defined as the processor in debug.

0 – Transmit Processor; and

1 – Receive Processor.

SINGLE BYTE COMMANDS.

00 – Halt - This command asynchronously halts the processor.

08 – Run - This command starts the processor.

10 – Step - This command steps the processor.

EIGHT BYTE COMMANDS.

18 – Break

0	1	2 - 3	4 – 7
Command	Reserved	Count	Address

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 125

This command sets a stop at the specified address. A count of 1 causes the specified processor to halt the first time it executes the instruction. A count of 2 or more causes the processor to halt after that number of executions. The processor is halted just before executing the instruction. A count of 0 does not halt the processor, but causes a sync signal to be generated. If a second processor is set to the same break address, the count data from the first break request is used, and each time either processor executes the instruction the count is decremented.

20 – Reset Break

0	1 - 3	4 – 7
Command	Reserved	Address

This command resets a previously set break point at the specified address. Reset break fully resets that address. If multiple processors were set to that break point, all will be reset.

28 – Dump

0	1	2 - 3	4 – 7
Command	Descriptor	Count	Address

This command transfers to the host the contents of the descriptor. For descriptors larger than four bytes, a count, in four byte increments is specified. For descriptors utilizing an address the address field is specified.

DESCRIPTOR.

00 – Register - This descriptor uses both count and address fields. Both fields are four byte based (a count of 1 transfers four bytes).

01 – Sram - This descriptor uses both count and address fields. Count is in four byte blocks. Address is in bytes, but if it is not four byte aligned, it is forced to the lower four byte aligned address.

02 – DRAM - This descriptor uses both count and address fields. Count is in four byte blocks. Address is in bytes, but if it is not four byte aligned, it is forced to the lower four byte aligned address.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 126

03 – Cstore - This descriptor uses both count and address fields. Count is in four byte blocks. Address is in bytes, but if it is not four byte aligned, it is forced to the lower four byte aligned address.

Stand-alone descriptors: The following descriptors do not use either the count or address fields. They transfer the contents of the referenced register.

04 – CPU_STATUS;
05 – PC;
06 – ADDR_REGA;
07 – ADDR_REGB;
08 – RAM_BASE;
09 – FILE_BASE;
0A – INSTR_REG_L;
0B – INSTR_REG_H;
0C – MAC_DATA;
0D – DMA_EVENT;
0E – MISC_EVENT;
0F – Q_IN_RDY;
10 – Q_OUT_RDY;
11 – LOCK STATUS;
12 – STACK - This returns 12 bytes; and
13 – SENSE_REG.

This register contains four bytes of data. If error status is posted for a command, if the next command that is issued reads this register, a code describing the error in more detail may be obtained. If any command other than a dump of this register is issued after error status, sense information will be reset.

30 – Load

0	1	2 - 3	4 – 7
Command	Descriptor	Count	Address

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 127

This command transfers from the host the contents of the descriptor. For descriptors larger than four bytes, a count, in four byte increments is specified. For descriptors utilizing an address the address field is specified.

DESCRIPTOR.

- 00 – Register - This descriptor uses both count and address fields. Both fields are four byte based.
- 01 – Sram - This descriptor uses both count and address fields. Count is in four byte blocks. Address is in bytes, but if it is not four byte aligned, it is forced to the lower four byte aligned address.
- 02 – DRAM - This descriptor uses both count and address fields. Count is in four byte blocks. Address is in bytes, but if it is not four byte aligned, it is forced to the lower four byte aligned address.
- 03 – Cstore- This descriptor uses both count and address fields. Count is in four byte blocks. Address is in bytes, but if it is not four byte aligned, it is forced to the lower four byte aligned address. This applies to WCS only.

Stand-alone descriptors: The following descriptors do not use either the count or address fields. They transfer the contents of the referenced register.

- 04 – ADDR_REGA;
- 05 – ADDR_REGB;
- 06 – RAM_BASE;
- 07 – FILE_BASE;
- 08 – MAC_DATA;
- 09 – Q_IN_RDY;
- 0A – Q_OUT_RDY;
- 0B – DBG_ADDR; and
- 38 – MAP.

This command allows an instruction in ROM to be replaced by an instruction in WCS. The new instruction will be located in the Host buffer. It will be stored in the first eight bytes of the buffer, with the high bits unused. To reset a mapped out instruction, map it to location 00.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 128

0	1 – 3	4 – 7
Command	Address to Map To	Address to Map Out

HARDWARE SPECIFICATION.

FEATURES:

1) PERIPHERAL COMPONENT INTERCONNECT (PCI) INTERFACE.

a) Universal PCI interface supports both 5.0V and 3.3V signaling environments;

b) Supports both 32-bit and 64 bit PCI interface;

c) Supports PCI clock frequencies from 15MHz to 66MHz;

d) High performance bus mastering architecture;

e) Host memory based communications reduce register accesses;

f) Host memory based interrupt status word reduces register reads;

g) Plug and Play compatible;

h) PCI specification revision 2.1 compliant;

i) PCI bursts up to 512 bytes;

j) Supports cache line operations up to 128 bytes;

k) Both big-endian and little-endian byte alignments supported; and

l) Supports Expansion ROM.

2) NETWORK INTERFACE.

a) Four internal 802.3 and ethernet compliant Macs;

b) Media Independent Interface (MII) supports external PHYs;

c) 10BASE-T, 100BASE-TX/FX and 100BASE-T4 supported;

d) Full and half-duplex modes supported;

e) Automatic PHY status polling notifies system of status change;

f) Provides SNMP statistics counters;

g) Supports broadcast and multicast packets;

h) Provides promiscuous mode for network monitoring or multiple unicast address detection;

i) Supports "huge packets" up to 32KB;

j) Mac-layer loop-back test mode; and

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 129

k) Supports auto-negotiating Phys.

3) MEMORY INTERFACE.

a) External DRAM buffering of transmit and receive packets;

b) Buffering configurable as 4MB, 8MB, 16MB or 32MB;

c) 32-bit interface supports throughput of 224MB/s;

d) Supports external FLASH ROM up to 4 MB, for diskless boot applications; and

e) Supports external serial EEPROM for custom configuration and Mac addresses.

4) PROTOCOL PROCESSOR.

a) High speed, custom, 32-bit processor executes 66 million instructions per second;

b) Processes IP, TCP and NETBIOS protocols;

c) Supports up to 256 resident TCP/IP contexts; and

d) Writable control store (WCS) allows field updates for feature enhancements.

5) POWER.

a) 3.3V chip operation; and

b) PCI controlled 5.0V/3.3V I/O cell operation.

6) PACKAGING.

a) 272-pin plastic ball grid array;

b) 91 PCI signals;

c) 68 MII signals;

d) 58 external memory signals;

e) 1 clock signal;

f) 54 signals split between power and ground; and

g) 272 total pins.

GENERAL DESCRIPTION.

The microprocessor (see Fig. 46) is a 32-bit, full-duplex, four channel, 10/100-Megabit per second (Mbps), Intelligent Network Interface Controller (INIC), designed to provide high-speed protocol processing for server applications. It combines the functions of a standard

[redacted]

network interface controller and a protocol processor within a single chip. Although designed specifically for server applications, the microprocessor can be used by PCs, workstations and routers or anywhere that TCP/IP protocols are being utilized.

When combined with four 802.3/MII compliant Phys and Synchronous DRAM (SDRAM), the INIC comprises four complete ethernet nodes. It contains four 802.3/ethernet compliant Macs, a PCI Bus Interface Unit (BIU), a memory controller, transmit fifos, receive fifos and a custom TCP/IP/NETBIOS protocol processor. The INIC supports 10Base-T , 100Base-TX, 100Base-FX and 100Base-T4 via the MII interface attachment of appropriate Phys.

The INIC Macs provide statistical information that may be used for SNMP. The Macs operate in promiscuous mode allowing the INIC to function as a network monitor, receive broadcast and multicast packets and implement multiple Mac addresses for each node.

Any 802.3/MII compliant PHY can be utilized, allowing the INIC to support 10BASE-T, 10BASE-T2, 100BASE-TX, 100Base-FX and 100BASE-T4 as well as future interface standards. PHY identification and initialization is accomplished through host driver initialization routines. PHY status registers can be polled continuously by the INIC and detected PHY status changes reported to the host driver. The Mac can be configured to support a maximum frame size of 1518 bytes or 32768 bytes.

The 64-bit, multiplexed BIU provides a direct interface to the PCI bus for both slave and master functions. The INIC is capable of operating in either a 64-bit or 32-bit PCI environment, while supporting 64-bit addressing in either configuration. PCI bus frequencies up to 66MHz are supported yielding instantaneous bus transfer rates of 533MB/s. Both 5.0V and 3.3V signaling environments can be utilized by the INIC. Configurable cache-line size up to 256B will accommodate future architectures, and Expansion ROM/Flash support allows for diskless system booting. Non-PC applications are supported via programmable big and little endian modes. Host based communication has been utilized to provide the best system performance possible.

The INIC supports Plug-N-Play auto-configuration through the PCI configuration space. External pull-up and pull-down resistors, on the memory I/O pins, allow selection of various features during chip reset. Support of an external eeprom allows for local storage of configuration information such as Mac addresses.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 131

External SDRAM provides frame buffering, which is configurable as 4MB, 8MB, 16MB or 32MB using the appropriate SIMMs. Use of -10 speed grades yields an external buffer bandwidth of 224MB/s. The buffer provides temporary storage of both incoming and outgoing frames. The protocol processor accesses the frames within the buffer in order to implement TCP/IP and NETBIOS. Incoming frames are processed, assembled then transferred to host memory under the control of the protocol processor. For transmit, data is moved from host memory to buffers where various headers are created before being transmitted out via the Mac.

1) CORES/CELLS.

a) LSI Logic Ethernet-110 Core, 100Base and 10Base Mac with MII interface;

b) LSI Logic single port SRAM, triple port SRAM and ROM available;

c) LSI Logic PCI 66MHz, 5V compatible I/O cell; and

d) LSI Logic PLL.

2) DIE SIZE / PIN COUNT.

LSI Logic G10 process. Fig. 47 shows the area on the die of each module.

3) DATAPATH BANDWIDTH (See Fig. 48).

4) CPU BANDWIDTH (See Fig. 49).

5) PERFORMANCE FEATURES.

a) 512 registers improve performance through reduced scratch ram accesses and reduced instructions;

b) Register windowing eliminates context-switching overhead;

c) Separate instruction and data paths eliminate memory contention;

d) Totally resident control store eliminates stalling during instruction fetch;

e) Multiple logical processors eliminate context switching and improve real-time response;

f) Pipelined architecture increases operating frequency;

g) Shared register and scratch ram improve inter-processor communication;

h) Fly-by state-Machine assists address compare and checksum calculation;

i) TCP/IP-context caching reduces latency;

j) Hardware implemented queues reduce CPU overhead and latency;

k) Horizontal microcode greatly improves instruction efficiency;

l) Automatic frame DMA and status between Mac and DRAM buffer; and

[redacted]

m) Deterministic architecture coupled with context switching eliminates processor stalls.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 133

PROCESSOR.

The processor is a convenient means to provide a programmable state-machine which is capable of processing incoming frames, processing host commands, directing network traffic and directing PCI bus traffic. Three processors are implemented using shared hardware in a three-level pipelined architecture which launches and completes a single instruction for every clock cycle. The instructions are executed in three distinct phases corresponding to each of the pipeline stages where each phase is responsible for a different function.

The first instruction phase writes the instruction results of the last instruction to the destination operand, modifies the program counter (Pc), selects the address source for the instruction to fetch, then fetches the instruction from the control store. The fetched instruction is then stored in the instruction register at the end of the clock cycle.

The processor instructions reside in the on-chip control-store, which is implemented as a mixture of ROM and SRAM. The ROM contains 1K instructions starting at address 0x0000 and aliases each 0x0400 locations throughout the first 0x8000 of instruction space. The SRAM (WCS) will hold up to 0x2000 instructions starting at address 0x8000 and aliasing each 0x2000 locations throughout the last 0x8000 of instruction space. The ROM and SRAM are both 49-bits wide accounting for bits [48:0] of the instruction microword. A separate mapping ram provides bits [55:49] of the microword (MapAddr) to allow replacement of faulty ROM based instructions. The mapping ram has a configuration of 128x7 which is insufficient to allow a separate map address for each of the 1K ROM locations. To allow re-mapping of the entire 1K ROM space, the map ram address lines are connected to the address bits Fetch[9:3]. The result is that the ROM is re-mapped in blocks of 8 contiguous locations.

The second instruction phase decodes the instruction which was stored in the instruction register. It is at this point that the map address is checked for a non-zero value which will cause the decoder to force a Jmp instruction to the map address. If a non-zero value is detected then the decoder selects the source operands for the Alu operation based on the values of the OpdASel, OpdBSel and AluOp fields. These operands are then stored in the decode register at the end of the clock cycle. Operands may originate from File, SRAM, or flip-flop based registers. The second instruction phase is also where the results of the previous instruction are written to the SRAM.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 134

The third instruction phase is when the actual Alu operation is performed, the test condition is selected and the Stack push and pop are implemented. Results of the Alu operation are stored in the results register at the end of the clock cycle.

Fig. 50 is a block diagram of the CPU. Fig. 50 shows the hardware functions associated with each of the instruction phases. Note that various functions have been distributed across the three phases of the instruction execution in order to minimize the combinatorial delays within any given phase.

INSTRUCTION SET.

The micro-instructions are divided into six types according to the program control directive. The micro-instruction is further divided into sub-fields for which the definitions are dependent upon the instruction type. The six instruction types are listed in Fig. 51.

All instructions (see Fig. 51) include the Alu operation (AluOp), operand "A" select (OpdASel), operand "B" select (OpdBSel) and Literal fields. Other field usage depends upon the instruction type.

The "jump condition code" (Jcc) instruction causes the program counter to be altered if the condition selected by the "test select" (TstSel) field is asserted. The new program counter (Pc) value is loaded from either the Literal field or the AluOut as described in the following section and the Literal field may be used as a source for the Alu or the ram address if the new Pc value is sourced by the Alu.

The "jump" (Jmp) instruction causes the program counter to be altered unconditionally. The new program counter (Pc) value is loaded from either the Literal field or the AluOut as described in the following section. The format allows instruction bits 23:16 to be used to perform a flag operation and the Literal field may be used as a source for the Alu or the ram address if the new Pc value is sourced by the Alu.

The "jump subroutine" (Jsr) instruction causes the program counter to be altered unconditionally. The new program counter (Pc) value is loaded from either the Literal field or the AluOut as described in the following section. The old program counter value is stored on the top location of the Pc-Stack which is implemented as a LIFO memory. The format allows instruction bits 23:16 to be used to perform a flag operation and the Literal field may be used as a source for the Alu or the ram address if the new Pc value is sourced by the Alu.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 135

The "Nxt" (Nxt) instruction causes the program counter to increment. The format allows instruction bits 23:16 to be used to perform a flag operation and the Literal field may be used as a source for the Alu or the ram address.

The "return from subroutine" (Rts) instruction is a special form of the Nxt instruction in which the "flag operation" (FlgSel) field is set to a value of 0hff. The current Pc value is replaced with the last value stored in the stack. The Literal field may be used as a source for the Alu or the ram address.

The Map instruction is provided to allow replacement of instructions which have been stored in ROM and is implemented any time the "map enable" (MapEn) bit has been set and the content of the "map address" (MapAddr) field is non-zero. The instruction decoder forces a jump instruction with the Alu operation and destination fields set to pass the MapAddr field to the program control block.

The program control is determined by a combination of PgmCtrl, DstOpd, FlgSel and TstSel. The behavior of the program control is defined with the "C-like" description in Fig. 52. Figs. 53-61 show ALU operations, selected operands, selected tests, and flag operations.

SRAM CONTROL SEQUENCER (SramCtrl).

SRAM is the nexus for data movement within the INIC. A hierarchy of sequencers, working in concert, accomplish the movement of data between DRAM, SRAM, CPU, ethernet and the Pci bus. Slave sequencers, provided with stimulus from master sequencers, request data movement operations by way of the SRAM, Pci bus, DRAM and Flash. The slave sequencers prioritize, service and acknowledge the requests.

The data flow block diagram of Fig. 62 shows all of the master and slave sequencers of the INIC product. Request information such as r/w, address, size, endian and alignment are represented by each request line. Acknowledge information to master sequencers include only the size of the transfer being acknowledged.

The block diagram of Fig. 63 illustrates how data movement is accomplished for a Pci slave write to DRAM. Note that the Psi (Pci slave in) module functions as both a master sequencer. Psi sends a write request to the SramCtrl module. Psi requests Xwr to move data from SRAM to DRAM. Xwr subsequently sends a read request to the SramCtrl module then writes the data to the DRAM via the Xctrl module. As each piece of data is moved from the SRAM to Xwr, Xwr sends an acknowledge to the Psi module.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 136

The SRAM control sequencer services requests to store to, or retrieve data from an SRAM organized as 1024 locations by 128 bits (16KB). The sequencer operates at a frequency of 133MHz, allowing both a CPU access and a DMA access to occur during a standard 66MHz CPU cycle. One 133MHz cycle is reserved for CPU accesses during each 66MHz cycle while the remaining 133MHz cycle is reserved for DMA accesses on a prioritized basis.

The block diagram of Fig. 64 shows the major functions of the SRAM control sequencer. A slave sequencer begins by asserting a request along with r/w, ram address, endian, data path size, data path alignment and request size. SramCtrl prioritizes the requests. The request parameters are then selected by a multiplexer which feeds the parameters to the SRAM via a register. The requestor provides the SRAM address which when coupled with the other parameters controls the input and output alignment. SRAM outputs are fed to the output aligner via a register. Requests are acknowledged in parallel with the returned data.

Fig. 65 is a timing diagram depicting two ram accesses during a single 66MHz clock cycle.

EXTERNAL MEMORY CONTROL (Xctrl).

Xctrl (See Fig. 66) provides the facility whereby Xwr, Xrd, Dcfg and Eectrl access external Flash and DRAM. Xctrl includes an arbiter, i/o registers, data multiplexers, address multiplexers and control multiplexers. Ownership of the external memory interace is requested by each block and granted to each of the requesters by the arbiter function. Once ownership has been granted the multiplexers select the address, data and control signals from owner, allowing access to external memory.

EXTERNAL MEMORY READ SEQUENCER (Xrd).

The Xrd sequencer acts only as a slave sequencer. Servicing requests issued by master sequencers, the Xrd sequencer moves data from external SDRAM or flash to the SRAM, via the Xctrl module, in blocks of 32 bytes or less. The nature of the SDRAM requires fixed burst sizes for each of it's internal banks with ras precharge intervals between each access. By selecting a burst size of 32 bytes for SDRAM reads and interleaving bank accesses on a 16 byte boundary, we can ensure that the ras precharge interval for the first bank is satisfied before burst completion for the second bank, allowing us to re-instruct the first bank and continue with uninterrupted DRAM access. SDRAMs require a consistent burst size be

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 137

utilized each and every time the SDRAM is accessed. For this reason, if an SDRAM access does not begin or end on a 32 byte boundary, SDRAM bandwidth will be reduced due to less than 32 bytes of data being transferred during the burst cycle.

Fig. 67 depicts the major functional blocks of the Xrd external memory read sequencer. The first step in servicing a request to move data from SDRAM to SRAM is the prioritization of the master sequencer requests. Next the Xrd sequencer takes a snapshot of the DRAM read address and applies configuration information to determine the correct bank, row and column address to apply. Once sufficient data has been read, the Xrd sequencer issues a write request to the SramCtrl sequencer which in turn sends an acknowledge to the Xrd sequencer. The Xrd sequencer passes the acknowledge along to the level two master with a size code indicating how much data was written during the SRAM cycle allowing the update of pointers and counters. The DRAM read and SRAM write cycles repeat until the original burst request has been completed at which point the Xrd sequencer prioritizes any remaining requests in preparation for the next burst cycle.

Contiguous DRAM burst cycles are not guaranteed to the Xrd sequencer as an algorithm is implemented which ensures highest priority to refresh cycles followed by flash accesses, DRAM writes then DRAM reads.

Fig. 68 is a timing diagram illustrating how data is read from SDRAM. The DRAM has been configured for a burst of four with a latency of two clock cycles. Bank A is first selected/activated followed by a read command two clock cycles later. The bank select/activate for bank B is next issued as read data begins returning two clocks after the read command was issued to bank A. Two clock cycles before we need to receive data from bank B we issue the read command. Once all 16 bytes have been received from bank A we begin receiving data from bank B.

EXTERNAL MEMORY WRITE SEQUENCER (Xwr).

The Xwr sequencer is a slave sequencer. Servicing requests issued by master sequencers, the Xwr sequencer moves data from SRAM to the external SDRAM or flash, via the Xctrl module, in blocks of 32 bytes or less while accumulating a checksum of the data moved. The nature of the SDRAM requires fixed burst sizes for each of it's internal banks with ras precharge intervals between each access. By selecting a burst size of 32 bytes for SDRAM writes and interleaving bank accesses on a 16 byte boundary, we can ensure that the ras

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 138

prechage interval for the first bank is satisfied before burst completion for the second bank, allowing us to re-instruct the first bank and continue with uninterrupted DRAM access. SDRAMs require a consistent burst size be utilized each and every time the SDRAM is accessed. For this reason, if an SDRAM access does not begin or end on a 32 byte boundary, SDRAM bandwidth will be reduced due to less than 32 bytes of data being transferred during the burst cycle.

Fig. 69 depicts the major functional blocks of the Xwr sequencer. The first step in servicing a request to move data from SRAM to SDRAM is the prioritization of the level two master requests. Next the Xwr sequencer takes a Snapshot of the DRAM write address and applies configuration information to determine the correct DRAM, bank, row and column address to apply. The Xwr sequencer immediately issues a read command to the SRAM to which the SRAM responds with both data and an acknowledge. The Xwr sequencer passes the acknowledge to the level two master along with a size code indicating how much data was read during the SRAM cycle allowing the update of pointers and counters. Once sufficient data has been read from SRAM, the Xwr sequencer issues a write command to the DRAM starting the burst cycle and computing a checksum as the data flys by. The SRAM read cycle repeats until the original burst request has been completed at which point the Xwr sequencer prioritizes any remaining requests in preparation for the next burst cycle.

Contiguous DRAM burst cycles are not guaranteed to the Xwr sequencer as an algorithm is implemented which ensures highest priority to refresh cycles followed by flash accesses then DRAM writes.

Fig. 70 is a timing diagram illustrating how data is written to SDRAM. The DRAM has been configured for a burst of four with a latency of two clock cycles. Bank A is first selected/activated followed by a write command two clock cycles later. The bank select/activate for bank B is next issued in preparation for issuing the second write command. As soon as the first 16 byte burst to bank A completes we issue the write command for bank B and begin supplying data.

PCI MASTER-OUT SEQUENCER (Pmo).

The Pmo sequencer (See Fig. 71) acts only as a slave sequencer. Servicing requests issued by master sequencers, the Pmo sequencer moves data from an SRAM based fifo to a Pci target, via the PciMstrIO module, in bursts of up to 256 bytes. The nature of the PCI bus

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 139

dictates the use of the write line command to ensure optimal system performance. The write line command requires that the Pmo sequencer be capable of transferring a whole multiple (1X, 2X, 3X, ...) of cache lines of which the size is set through the Pci configuration registers. To accomplish this end, Pmo will automatically perform partial bursts until it has aligned the transfers on a cache line boundary at which time it will begin usage of the write line command. The SRAM fifo depth, of 256 bytes, has been chosen in order to allow Pmo to accommodate cache line sizes up to 128 bytes. Provided the cache line size is less than 128 bytes, Pmo will perform multiple, contiguous cache line bursts until it has exhausted the supply of data.

Pmo receives requests from two separate sources; the DRAM to Pci (D2p) module and the SRAM to Pci (S2p) module. An operation first begins with prioritization of the requests where the S2p module is given highest priority. Next, the Pmo module takes a Snapshot of the SRAM fifo address and uses this to generate read requests for the SramCtrl sequencer. The Pmo module then proceeds to arbitrate for ownership of the Pci bus via the PciMstrIO module. Once the Pmo holding registers have sufficient data and Pci bus mastership has been granted, the Pmo module begins transferring data to the Pci target. For each successful transfer, Pmo sends an acknowledge and encoded size to the master sequencer, allow it to update it's internal pointers, counters and status. Once the Pci burst transaction has terminated, Pmo parks on the Pci bus unless another initiator has requested ownership. Pmo again prioritizes the incoming requests and repeats the process.

PCI MASTER-OUT SEQUENCER (Pmi).

The Pmi sequencer (See Fig. 72) acts only as a slave sequencer. Servicing requests issued by master sequencers, the Pmi sequencer moves data from a Pci target to an SRAM based fifo, via the PciMstrIO module, in bursts of up to 256 bytes. The nature of the PCI bus dictates the use of the read multiple command to ensure optimal system performance. The read multiple command requires that the Pmi sequencer be capable of transferring a cache line or more of data. To accomplish this end, Pmi will automatically perform partial cache line bursts until it has aligned the transfers on a cache line boundary at which time it will begin usage of the read multiple command. The SRAM fifo depth, of 256 bytes, has been chosen in order to allow Pmi to accommodate cache line sizes up to 128 bytes. Provided the cache line size is less than 128 bytes, Pmi will perform multiple, contiguous cache line bursts until it has filled the fifo.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 140

Pmi receive requests from two separate sources; the Pci to DRAM (P2d) module and the Pci to SRAM (P2s) module. An operation first begins with prioritization of the requests where the P2s module is given highest priority. The Pmi module then proceeds to arbitrate for ownership of the Pci bus via the PciMstrIO module. Once the Pci bus mastership has been granted and the Pmi holding registers have sufficient data, the Pmi module begins transferring data to the SRAM fifo. For each successful transfer, Pmi sends an acknowledge and encoded size to the master sequencer, allowing it to update it's internal pointers, counters and status. Once the Pci burst transaction has terminated, Pmi parks on the Pci bus unless another initiator has requested ownership. Pmi again prioritizes the incoming requests and repeats the process.

DRAM TO PCI SEQUENCER (D2P).

The D2p sequencer (See Fig. 73) acts is a master sequencer. Servicing channel requests issued by the CPU, the D2p sequencer manages movement of data from DRAM to the Pci bus by issuing requests to both the Xrd sequencer and the Pmo sequencer. Data transfer is accomplished using an SRAM based fifo through which data is staged.

D2p can receive requests from any of the processor's thirty-two DMA channels. Once a command request has been detected, D2p fetches a DMA descriptor from an SRAM location dedicated to the requesting channel which includes the DRAM address, Pci address, Pci endian and request size. D2p then issues a request to the D2s sequencer causing the SRAM based fifo to fill with DRAM data. Once the fifo contains sufficient data for a Pci transaction, D2s issues a request to Pmo which in turn moves data from the fifo to a Pci target. The process repeats until the entire request has been satisfied at which time D2p writes ending status in to the SRAM DMA descriptor area and sets the channel done bit associated with that channel. D2p then monitors the DMA channels for additional requests. Fig. 74 is an illustration showing the major blocks involved in the movement of data from DRAM to Pci target.

PCI TO DRAM SEQUENCER (P2d).

The P2d sequencer (See Fig. 75) acts as both a slave sequencer and a master sequencer. Servicing channel requests issued by the CPU, the P2d sequencer manages movement of data from Pci bus to DRAM by issuing requests to both the Xwr sequencer and the Pmi sequencer. Data transfer is accomplished using an SRAM based fifo through which data is staged.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 141

P2d can receive requests from any of the processor's thirty-two DMA channels. Once a command request has been detected, P2d, operating as a slave sequencer, fetches a DMA descriptor from an SRAM location dedicated to the requesting channel which includes the DRAM address, Pci address, Pci endian and request size. P2d then issues a request to Pmo which in turn moves data from the Pci target to the SRAM fifo. Next, P2d issues a request to the Xwr sequencer causing the SRAM based fifo contents to be written to the DRAM. The process repeats until the entire request has been satisfied at which time P2d writes ending status in to the SRAM DMA descriptor area and sets the channel done bit associated with that channel. P2d then monitors the DMA channels for additional requests. Fig. 76 is an illustration showing the major blocks involved in the movement of data from a Pci target to DRAM.

SRAM TO PCI SEQUENCER (S2p).

The S2p sequencer (See Fig. 77) acts as both a slave sequencer and a master sequencer. Servicing channel requests issued by the CPU, the S2p sequencer manages movement of data from SRAM to the Pci bus by issuing requests to the Pmo sequencer

S2p can receive requests from any of the processor's thirty-two DMA channels. Once a command request has been detected, S2p, operating as a slave sequencer, fetches a DMA descriptor from an SRAM location dedicated to the requesting channel which includes the SRAM address, Pci address, Pci endian and request size. S2p then issues a request to Pmo which in turn moves data from the SRAM to a Pci target. The process repeats until the entire request has been satisfied at which time S2p writes ending status in to the SRAM DMA descriptor area and sets the channel done bit associated with that channel. S2p then monitors the DMA channels for additional requests. Fig. 78 is an illustration showing the major blocks involved in the movement of data from SRAM to Pci target.

PCI TO SRAM SEQUENCER (P2s).

The P2s sequencer (See Fig. 79) acts as both a slave sequencer and a master sequencer. Servicing channel requests issued by the CPU, the P2s sequencer manages movement of data from Pci bus to SRAM by issuing requests to the Pmi sequencer.

P2s can receive requests from any of the processor's thirty-two DMA channels. Once a command request has been detected, P2s, operating as a slave sequencer, fetches a DMA descriptor from an SRAM location dedicated to the requesting channel which includes the

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 142

SRAM address, Pci address, Pci endian and request size. P2s then issues a request to Pmo which in turn moves data from the Pci target to the SRAM. The process repeats until the entire request has been satisfied at which time P2s writes ending status in to the DMA descriptor area of SRAM and sets the channel done bit associated with that channel. P2s then monitors the DMA channels for additional requests. Fig. 80 is an illustration showing the major blocks involved in the movement of data from a Pci target to DRAM.

DRAM TO SRAM SEQUENCER (D2s).

The D2s sequencer (See Fig. 81) acts as both a slave sequencer and a master sequencer. Servicing channel requests issued by the CPU, the D2s sequencer manages movement of data from DRAM to SRAM by issuing requests to the Xrd sequencer.

D2s can receive requests from any of the processor's thirty-two DMA channels. Once a command request has been detected, D2s, operating as a slave sequencer, fetches a DMA descriptor from an SRAM location dedicated to the requesting channel which includes the DRAM address, SRAM address and request size. D2s then issues a request to the Xrd sequencer causing the transfer of data to the SRAM. The process repeats until the entire request has been satisfied at which time D2s writes ending status in to the SRAM DMA descriptor area and sets the channel done bit associated with that channel. D2s then monitors the DMA channels for additional requests. Fig. 82 is an illustration showing the major blocks involved in the movement of data from DRAM to SRAM.

SRAM TO DRAM SEQUENCER (S2d).

The S2d sequencer (See Fig. 83) acts as both a slave sequencer and a master sequencer. Servicing channel requests issued by the CPU, the S2d sequencer manages movement of data from SRAM to DRAM by issuing requests to the Xwr sequencer.

S2d can receive requests from any of the processor's thirty-two DMA channels. Once a command request has been detected, S2d, operating as a slave sequencer, fetches a DMA descriptor from an SRAM location dedicated to the requesting channel which includes the DRAM address, SRAM address, checksum reset and request size. S2d then issues a request to the Xwr sequencer causing the transfer of data to the DRAM. The process repeats until the entire request has been satisfied at which time S2d writes ending status in to the SRAM DMA descriptor area and sets the channel done bit associated with that channel. S2d then monitors

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 143

the DMA channels for additional requests. Fig. 84 is an illustration showing the major blocks involved in the movement of data from SRAM to DRAM.

PCI SLAVE INPUT SEQUENCER (Psi).

The Psi sequencer (See Fig. 85) acts as both a slave sequencer and a master sequencer. Servicing requests issued by a Pci master, the Psi sequencer manages movement of data from Pci bus to SRAM and Pci bus to DRAM via SRAM by issuing requests to the SramCtrl and Xwr sequencers.

Psi manages write requests to configuration space, expansion rom, DRAM, SRAM and memory mapped registers. Psi separates these Pci bus operations in to two categories with different action taken for each. DRAM accesses result in Psi generating write request to an SRAM buffer followed with a write request to the Xwr sequencer. Subsequent write or read DRAM operations are retry terminated until the buffer has been emptied. An event notification is set for the processor allowing message passing to occur through DRAM space.

All other Pci write transactions result in Psi posting the write information including Pci address, Pci byte marks and Pci data to a reserved location in SRAM, then setting an event flag which the event processor monitors. Subsequent writes or reads of configuration, expansion rom, SRAM or registers are terminated with retry until the processor clears the event flag. This allows the INIC pipelining levels to a minimum for the posted write and give the processor ample time to modify data for subsequent Pci read operations.

Fig. 85 depicts the sequence of events when Psi is the target of a Pci write operation. Note that events 4 through 7 occur only when the write operation targets the DRAM.

PCI SLAVE OUTPUT SEQUENCER (Pso).

The Pso sequencer (See Fig. 86) acts as both a slave sequencer and a master sequencer. Servicing requests issued by a Pci master, the Pso sequencer manages movement of data to Pci bus from SRAM and to Pci bus from DRAM via SRAM by issuing requests to the SramCtrl and Xrd sequencers.

Pso manages read requests to configuration space, expansion rom, DRAM, SRAM and memory mapped registers. Pso separates these Pci bus operations in to two categories with different action taken for each. DRAM accesses result in Pso generating read request to the Xrd sequencer followed with a read request to SRAM buffer. Subsequent write or read DRAM

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 144

operations are retry terminated until the buffer has been emptied.

All other Pci read transactions result in Pso posting the read request information including Pci address and Pci byte marks to a reserved location in SRAM, then setting an event flag which the event processor monitors. Subsequent writes or reads of configuration, expansion rom, SRAM or registers are terminated with retry until the processor clears the event flag. This allows the INIC to use a microcoded response mechanism to return data for the request. The processor decodes the request information, formulates or fetches the requested data and stores it in SRAM then clears the event flag allowing Pso to fetch the data and return it on the Pci bus.

Fig. 78 depicts the sequence of events when Pso is the target of a Pci read operation.

FRAME RECEIVE SEQUENCER (RcvX).

The receive sequencer (See Fig. 87) (RcvSeq) analyzes and manages incoming packets, stores the result in DRAM buffers, then notifies the processor through the receive queue (RcvQ) mechanism. The process begins when a buffer descriptor is available at the output of the FreeQ. RcvSeq issues a request to the Qmg which responds by supplying the buffer descriptor to RcvSeq. RcvSeq then waits for a receive packet. The Mac, network, transport and session information is analyzed as each byte is received and stored in the assembly register (AssyReg). When four bytes of information is available, RcvSeq requests a write of the data to the SRAM. When sufficient data has been stored in the SRAM based receive fifo, a DRAM write request is issued to Xwr. The process continues until the entire packet has been received at which point RcvSeq stores the results of the packet analysis in the beginning of the DRAM buffer. Once the buffer and status have both been stored, RcvSeq issues a write-queue request to Qmg. Qmg responds by storing a buffer descriptor and a status vector provided by RcvSeq. The process then repeats. If RcvSeq detects the arrival of a packet before a free buffer is available, it ignores the packet and sets the FrameLost status bit for the next received packet.

Fig. 88 depicts the sequence of events for successful reception of a packet followed by a definition of the receive buffer and the buffer descriptor as stored on the RcvQ. Fig. 89 shows the Receive Buffer Descriptor. Figs. 90-92 show the Receive Buffer Format.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 145

FRAME TRANSMIT SEQUENCER (XmtX).

The transmit sequencer (See Fig. 93) (XmtSeq) analyzes and manages outgoing packets, using buffer descriptors retrieved from the transmit queue (XmtQ) then storing the descriptor for the freed buffer in the free buffer queue (FreeQ). The process begins when a buffer descriptor is available at the output of the XmtQ. XmtSeq issues a request to the Qmg which responds by supplying the buffer descriptor to XmtSeq. XmtSeq then issues a read request to the Xrd sequencer. Next, XmtSeq issues a read request to SramCtrl then instructs the Mac to begin frame transmission. Once the frame transmission has completed, XmtSeq stores the buffer descriptor on the FreeQ thereby recycling the buffer.

Fig. 94 depicts the sequence of events for successful transmission of a packet followed by a definition of the receive buffer and the buffer descriptor as stored on the XmtQ. Fig. 95 shows the Transmit Buffer Descriptor. Fig. 96 shows the Transmit Buffer Format. Fig. 97 shows the Transmit Status Vector.

QUEUE MANAGER (Qmg).

The INIC includes special hardware assist for the implementation of message and pointer queues. The hardware assist is called the queue manager (See Fig. 98) (Qmg) and manages the movement of queue entries between CPU and SRAM, between DMA sequencers and SRAM as well as between SRAM and DRAM. Queues comprise three distinct entities; the queue head (QHd), the queue tail (QTl) and the queue body (QBdy). QHd resides in 64 bytes of scratch ram and provides the area to which entries will be written (pushed). QTl resides in 64 bytes of scratch ram and contains queue locations from which entries will be read (popped). QBdy resides in DRAM and contains locations for expansion of the queue in order to minimize the SRAM space requirements. The QBdy size depends upon the queue being accessed and the initialization parameters presented during queue initialization.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 146

Qmg accepts operations from both CPU and DMA sources (See Fig. 99). Executing these operations at a frequency of 133MHz, Qmg reserves even cycles for DMA requests and reserves odd cycles for CPU requests. Valid CPU operations include initialize queue (InitQ), write queue (WrQ) and read queue (RdQ). Valid DMA requests include read body (RdBdy) and write body (WrBdy). Qmg working in unison with Q2d and D2q generate requests to the Xwr and Xrd sequencers to control the movement of data between the QHd, QTl and QBdy.

Fig. 98 shows the major functions of Qmg. The arbiter selects the next operation to be performed. The dual-ported SRAM holds the queue variables HdWrAddr, HdRdAddr, TlWrAddr, TlRdAddr, BdyWrAddr, BdyRdAddr and QSz. Qmg accepts an operation request, fetches the queue variables from the queue ram (Qram), modifies the variables based on the current state and the requested operation then updates the variables and issues a read or write request to the SRAM controller. The SRAM controller services the requests by writing the tail or reading the head and returning an acknowledge.

DMA OPERATIONS.

DMA operations are accomplished through a combination of thirtytwo DMA channels (DmaCh) and seven DMA sequencers (DmaSeq). Each DMA channel provides a mechanism whereby a CPU can issue a command to any of the seven DMA sequencers. Where as the DMA channels are multi-purpose, the DMA sequencers they command are single purpose as shown in Fig. 100.

The processors manage DMA in the following way. The processor writes a DMA descriptor to an SRAM location reserved for the DMA channel. The format of the DMA descriptor is dependent upon the targeted DMA sequencer. The processor then writes the DMA sequencer number to the channel command register.

Each of the DMA sequencers polls all thirtytwo DMA channels in search of commands to execute. Once a command request has been detected, the DMA sequencer fetches a DMA descriptor from a fixed location in SRAM. The SRAM location is fixed and is determined by the DMA channel number. The DMA sequencer loads the DMA descriptor in to it's own registers, executes the command, then overwrites the DMA descriptor with ending status. Once the command has halted, due to completion or error, and the ending status has been written, the DMA sequencer sets the done bit for the current DMA channel.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 147

The done bit appears in a DMA event register which the CPU can examine. The CPU fetches ending status from SRAM, then clears the done bit by writing zeroes to the channel command (ChCmd) register. The channel is now ready to accept another command.

The format of the channel command register is as shown in Fig. 101. The format of the P2d or P2s descriptor is as shown in Fig. 102. The format of the S2p or D2p descriptor is as shown in Fig. 103. The format of the S2d, D2d or D2s descriptor is as shown in Fig. 104. The format of the ending status of all channels is as shown in Fig. 105. The format of the ChEvnt register is as shown in Fig. 106. Fig. 107 is a block diagram of MAC CONTROL (Macctrl).

LOAD CALCULATIONS.

The following load calculations are based on the following basic formulae:

N = X * R (Little's Law) where:

N = number of jobs in the system (either in progress or in a queue),

X = system throughput,

R = response time (which includes time waiting in queues).

U = X * S (from Little's Law) where:

S = service time,

U = utilization.

R = S / (1-U) for exponential service times (which is the worst-case assumption).

A 256-byte frame at 100Mb/sec takes 20 usec per frame.

4 * 100 Mbit ethernets receiving at full frame rate is:

51200 (4 * 12800) frames/sec @ 1024 bytes/frame,

102000 frames/sec @ 512 bytes/frame,

204000 frames/sec @ 256 bytes/frame.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 148

The following calculations assume 250 instructions/frame, 45nsec clock. Thus

S = 250 * 45 nsecs = 11.2 usecs.

Av Frame Size	Thruput (X)	Utilization (U)	Response (R)	Nbr in system (N)
1024	51200	0.57	26 usecs	1.3
512	102000	> 1	--	--
256	204000	> 1	--	--

Lets look at it for varying instructions per frame assuming 512 bytes per frame average.

Instns Per Frame	Service Time (S)	Thruput (X)	Utilization (U)	Response (R)	Nbr in system (N)
250	11.2 usec	102000	> 1	--	--
250	11.2	85000 (*)	0.95	224 usecs	19
250	11.2	80000 (**)	0.89	101	8
225	10	102000	1.0	--	--
225	10	95000 (*)	0.95	200	19
225	10	89000 (**)	0.89	90	8
200	9	102000	0.9	90	9
150	6.7	102000	0.68	20	2

(*) shows what frame rate can be supported to get a utilization of less than 1.

(**) shows what frame rate can be supported with 8 SRAM CCB buffers and at least 8 process contexts.

If 100 instructions / frame is used, S = 100 * 45 nsecs = 4.5 usecs, and we can support 256 byte frames:

100	4.5	204000	0.91	50	10

Note that these calculations assume that response times increase exponentially as utilization increases. This is the worst-case assumption, and probably may not be true for our

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 149

system. The figures show that to support a theoretical full 4 * 100 Mbit receive load with an average frame size of 512 bytes, there will need to be 19 active "jobs" in the system, assuming 250 instructions per frame. Due to SRAM limitations, the current design specifies 8 SRAM buffers for active TCBs, and not to swap a TCB out of SRAM once it is active. So under these limitations, the INIC will not be able to keep up with the full frame rate. Note that the initial implementation is trying to use only 8KB of SRAM, although 16KB may be available, in which case 19 TCB SRAM buffers could be used. This is a cost trade-off. The real point here is the effect of instructions/frame on the throughput that can be maintained. If the instructions/frame drops to 200, then the INIC is capable of handling the full theoretical load (102000 frames/second) with only 9 active TCBs. If it drops to 100 instructions per frame, then the INIC can handle full bandwidth at 256 byte frames (204000 frames/second) with 10 active CCBs. The bottom line is that all hardware-assist that reduces the instructions/frame is really worthwhile. If header-assist hardware can save us 50 instructions per frame then it goes straight to the throughput bottom line.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 150

CLAIMS

1. A method for network communication, the method comprising:

receiving a plurality of packets from the network, each of the packets including a media access control layer header, a network layer header and a transport layer header;

processing the packets by a first mechanism, so that for each packet the network layer header and the transport layer header are validated without an interrupt dividing the processing of the network layer header and the transport layer header;

sorting the packets, dependent upon the processing, into first and second types of packets, so that the packets of the first type each contain data;

sending the data from each packet of the first type to a destination without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination.

2. The method of claim 1, wherein processing the packets by a first mechanism further comprises:

processing the media access control layer header for each packet without an interrupt dividing the processing of the media access control layer header and the network layer header.

3. The method of claim 1, further comprising:

processing an upper layer header of at least one of the packets by a second mechanism, thereby determining the destination, wherein the upper layer header corresponds to a protocol layer above the transport layer.

4. The method of claim 1, further comprising:

processing an upper layer header of at least one of the packets of the second type by a second mechanism, thereby determining the destination.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 151

5. The method of claim 1, further comprising:

processing a transport layer header of another packet by a second mechanism, prior to receiving the plurality of packets from the network, thereby establishing a Transport Control Protocol (TCP) connection for the packets of the first type.

6. The method of claim 1, wherein sorting the packets includes classifying each of the packets of the first type as having an Internet Protocol (IP) header and a Transport Control Protocol (TCP).

7. The method of claim 1, further comprising:

transmitting a second plurality of packets to the network, each of the second plurality of packets containing a media access control layer header, a network layer header and a transport layer header, including processing the second plurality of packets by the first mechanism, so that for each packet the media access control layer header, the network layer header and the transport layer header are processed without an interrupt dividing the processing of the media access control layer header, the network layer header and the transport layer header.

8. The method of claim 1, wherein the first mechanism is a sequencer running microcode.

9. A method for communicating information over a network, the method comprising:

obtaining data from a source allocated by a first processor;

dividing the data into multiple segments;

prepending a packet header to each of the segments by a second processor, thereby forming a packet corresponding to each segment, each packet header containing a media access control layer header, a network layer header and a transport layer header, wherein the prepending of each packet header occurs without an interrupt dividing the prepending of the network layer header and the transport layer header; and

transmitting the packets to the network.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 152

10. The method of claim 9, wherein prepending a packet header to each of the segments by a second processor further comprises:

prepending the media access control layer header for each packet without an interrupt dividing the prepending of the media access control layer header and the network layer header.

11. The method of claim 9, wherein each packet header contains an Internet Protocol (IP) header and a Transport Control Protocol (TCP) header.
the media access control layer header,

12. The method of claim 9, further comprising establishing a Transport Control Protocol (TCP) connection by the first processor and using the connection to prepend the packet header to each of the segments by the second processor.

13. The method of claim 9, further comprising creating a template header and forming each packet header based upon the template header.

14. The method of claim 9, wherein obtaining data from the source in memory allocated by the first processor is performed by a Direct Memory Access (DMA) unit controlled by the second processor.

15. The method of claim 9, further comprising prepending an upper layer header to the data, prior to dividing the data into multiple segments.

16. The method of claim 9, further comprising:

receiving another packet from the network, the other packet containing a receive header including information corresponding to a network layer and a transport layer; and

selecting whether to process the other packet by the first processor or by the second processor.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 153

17. A method for communicating information over a network, the method comprising:

providing multiple segments of data;

prepending an outbound packet header to each of the segments, thereby forming an outbound packet corresponding to each segment, the outbound packet header containing an outbound media access control layer header, an outbound network layer header and an outbound transport layer header, wherein the prepending of each outbound packet header occurs without an interrupt dividing the prepending of the outbound media access control layer header, the outbound network layer header and the outbound transport layer header;

transmitting the outbound packets to the network;

receiving multiple inbound packets from the network, each of the inbound packets including an inbound media access control layer header, an inbound network layer header and an inbound transport layer header;

processing the inbound packets, so that for each packet the inbound network layer header and the inbound transport layer header are validated without an interrupt dividing the processing of the inbound network layer header and the inbound transport layer header.

18. The method of claim 17, wherein the processing the inbound packets is performed simultaneously with the prepending the outbound packet header to each of the segments.

19. The method of claim 17, further comprising creating a template header and using the template header to form each outbound packet header.

20. The method of claim 17, wherein providing multiple segments of data includes dividing a block of data into the segments.

21. The method of claim 20, further comprising prepending an upper layer header to the block of data, prior to dividing the block of data into multiple segments.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 154

22. The method of claim 17, further comprising:

sending data from each inbound packet to a destination without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination.

23. The method of claim 17, further comprising:

processing an upper layer header of at least one of the packets by a second mechanism, thereby determining the destination, wherein the upper layer header corresponds to a protocol layer above the transport layer.

24. The method of claim 17, further comprising:

processing a transport layer header of another inbound packet, prior to receiving the plurality of packets from the network, thereby establishing a Transport Control Protocol (TCP) connection for the inbound packets.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 155

ALA-006E

FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

Laurence B. Boucher
Stephen E. J. Blightman
Peter K. Craft
David A. Higgen
Clive M. Philbrick
Daryl D. Starr

ABSTRACT OF THE DISCLOSURE

A system for protocol processing in a computer network has an intelligent network interface card (INIC) or communication processing device (CPD) associated with a host computer. The INIC provides a fast-path that avoids protocol processing for most large multi-packet messages, greatly accelerating data communication. The INIC also assists the host for those message packets that are chosen for processing by host software layers. A communication control block for a message is defined that allows DMA controllers of the INIC to move data, free of headers, directly to or from a destination or source in the host. The context is stored in the INIC as a communication control block (CCB) that can be passed back to the host for message processing by the host. The INIC contains specialized hardware circuits that are much faster at their specific tasks than a general purpose CPU. A preferred embodiment includes a trio of pipelined processors with separate processors devoted to transmit, receive and management processing, with full duplex communication for four fast Ethernet nodes.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 156

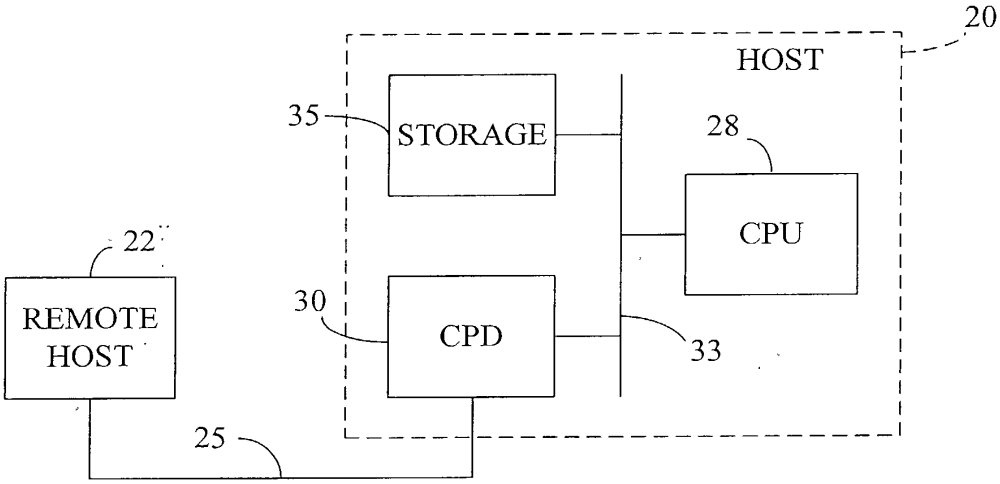

20
HOST
35
STORAGE
28
CPU
22
REMOTE HOST
30
CPD
33
25

FIG. 1

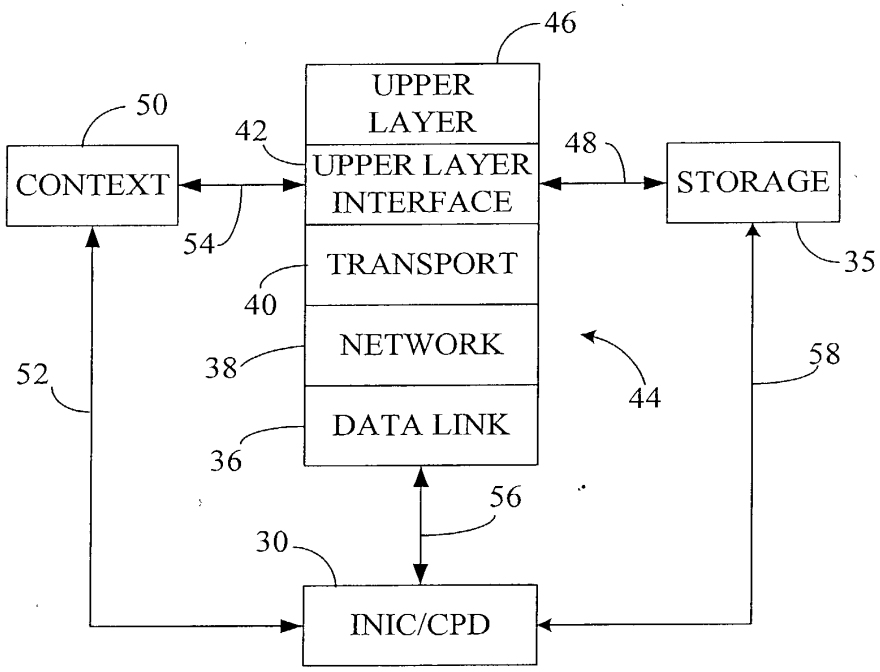

46
UPPER LAYER
50
42
CONTEXT
UPPER LAYER INTERFACE
48
STORAGE
54
TRANSPORT
35
40
NETWORK
58
52
38
44
DATA LINK
36
56
30
INIC/CPD

FIG. 2

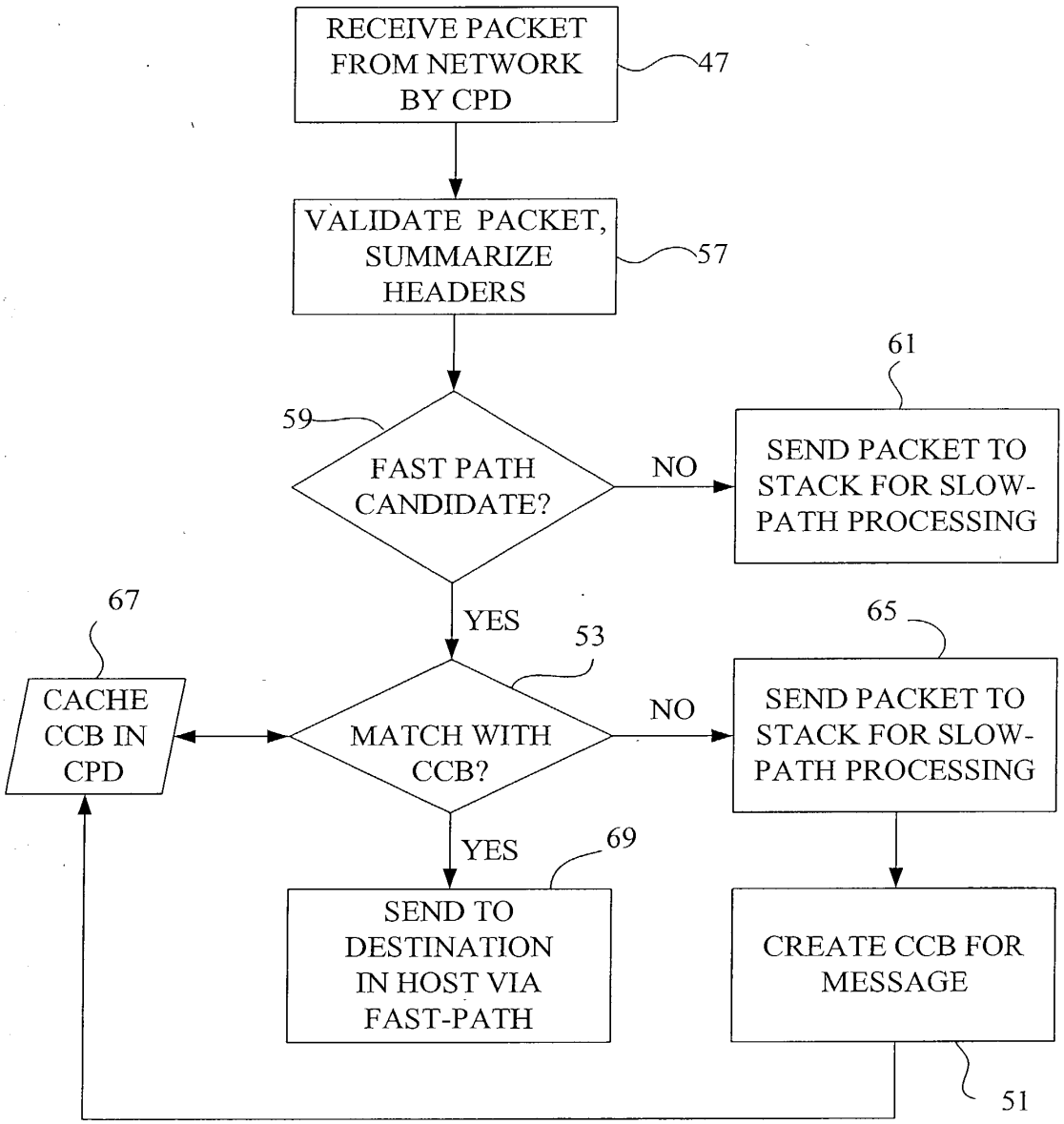

RECEIVE PACKET FROM NETWORK BY CPD
47
VALIDATE PACKET, SUMMARIZE HEADERS
57
59
FAST PATH CANDIDATE?
NO
61
SEND PACKET TO STACK FOR SLOW-PATH PROCESSING
YES
67
CACHE CCB IN CPD
53
MATCH WITH CCB?
NO
65
SEND PACKET TO STACK FOR SLOW-PATH PROCESSING
YES
69
SEND TO DESTINATION IN HOST VIA FAST-PATH
CREATE CCB FOR MESSAGE
51

FIG. 3

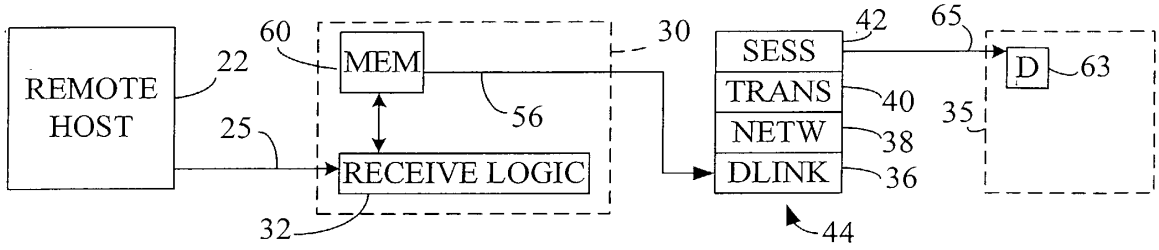


FIG. 4A

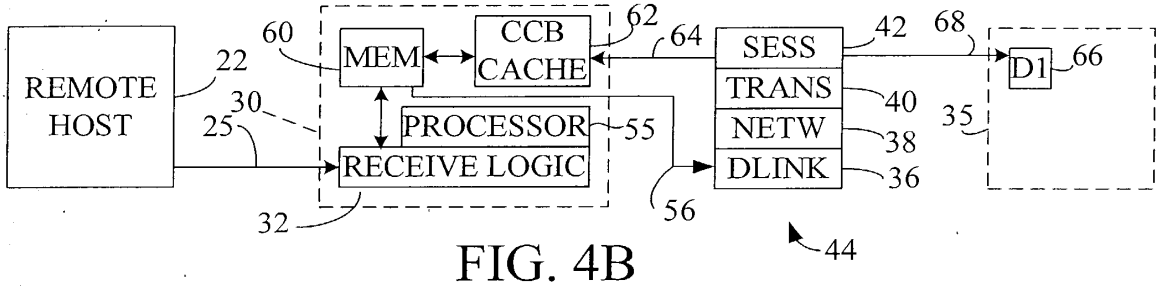


FIG. 4B

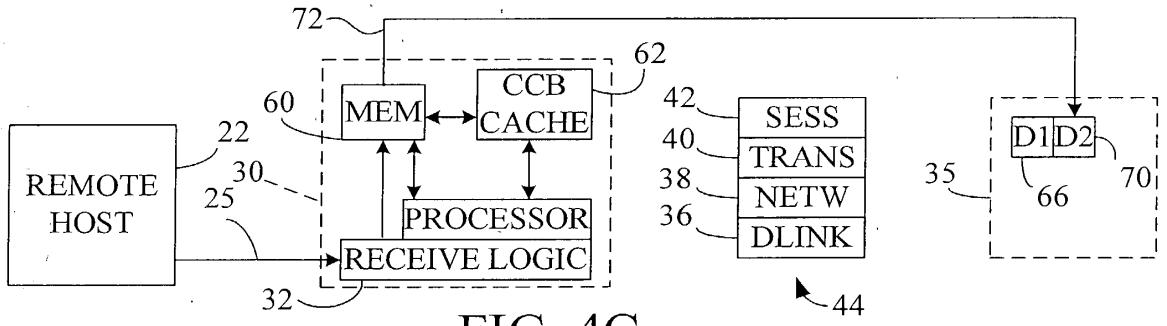


FIG. 4C

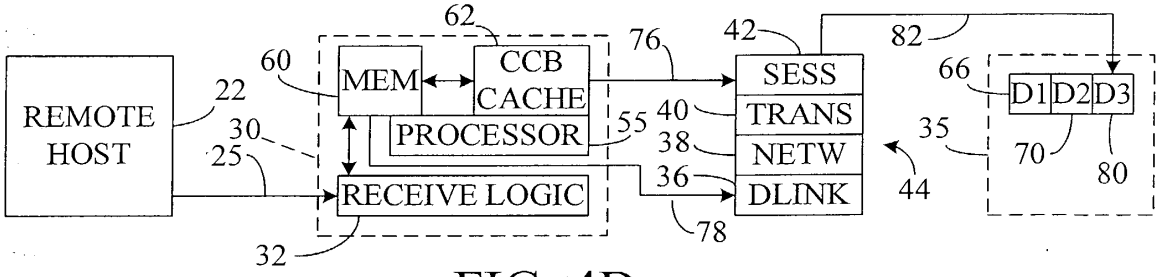


FIG. 4D

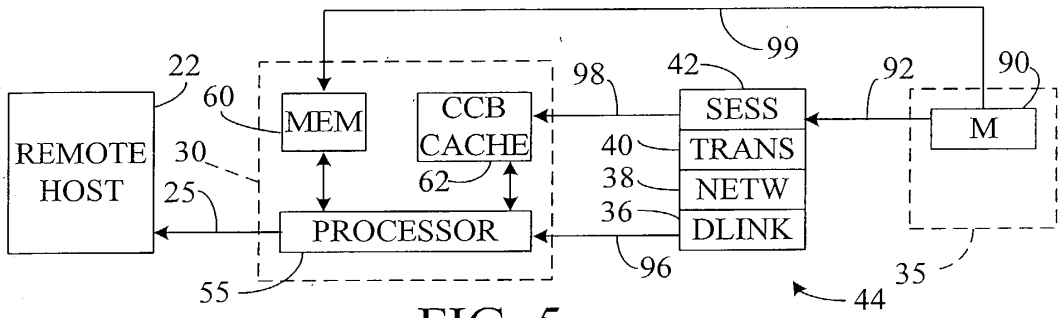

22
60
MEM
CCB
CACHE
98
42
SESS
92
90
M
99
REMOTE
HOST
30
25
62
40
TRANS
38
NETW
36
DLINK
PROCESSOR
55
96
35
44

FIG. 5

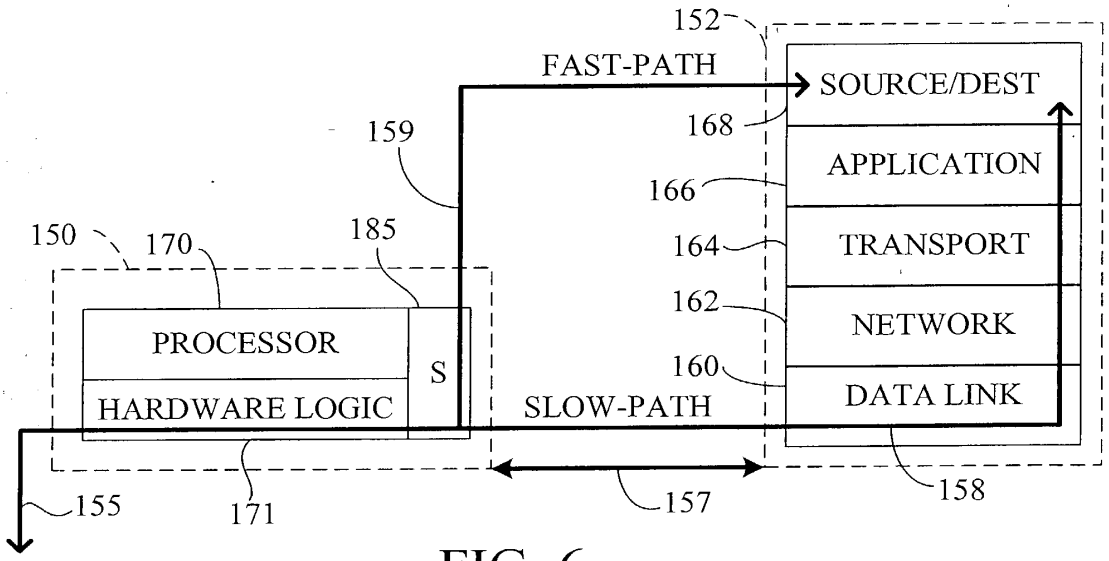

152
FAST-PATH
SOURCE/DEST
168
159
APPLICATION
166
150
170
185
164
TRANSPORT
162
NETWORK
PROCESSOR
S
160
DATA LINK
HARDWARE LOGIC
SLOW-PATH
155
171
157
158

FIG. 6

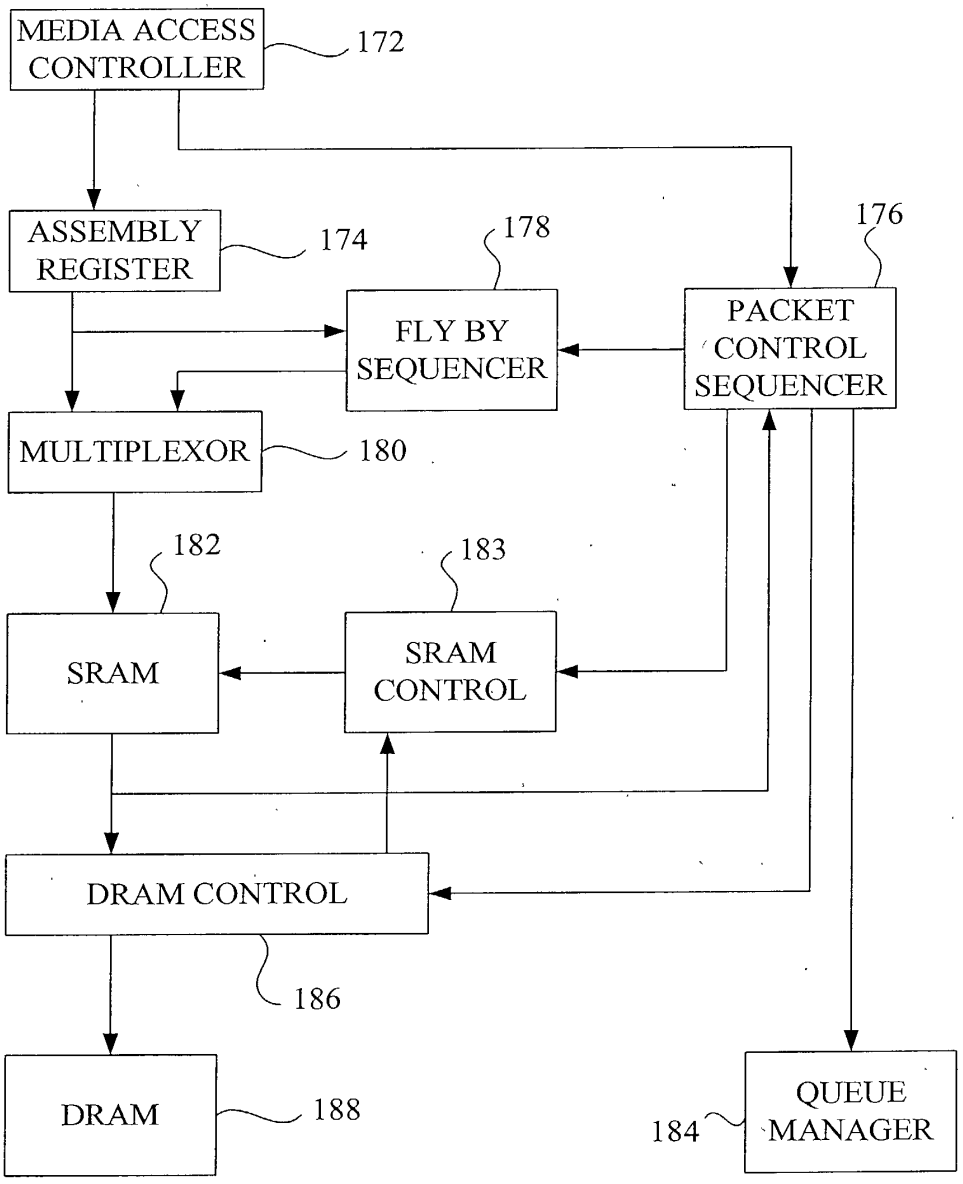

MEDIA ACCESS CONTROLLER
172
ASSEMBLY REGISTER
174
178
176
FLY BY SEQUENCER
PACKET CONTROL SEQUENCER
MULTIPLEXOR
180
182
183
SRAM
SRAM CONTROL
DRAM CONTROL
186
DRAM
188
184
QUEUE MANAGER

FIG. 7

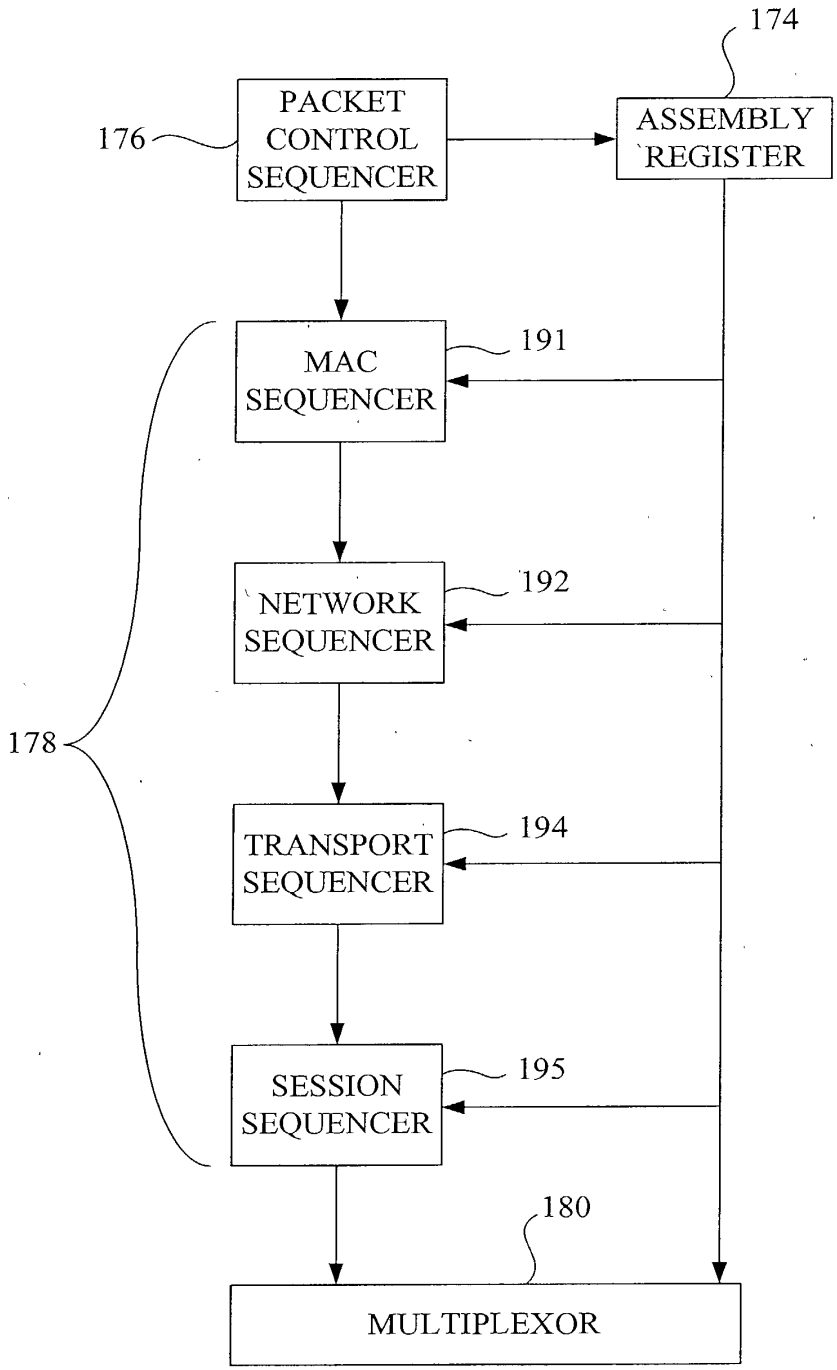

174
PACKET CONTROL SEQUENCER
176
ASSEMBLY REGISTER
MAC SEQUENCER
191
NETWORK SEQUENCER
192
178
TRANSPORT SEQUENCER
194
SESSION SEQUENCER
195
180
MULTIPLEXOR

FIG. 8

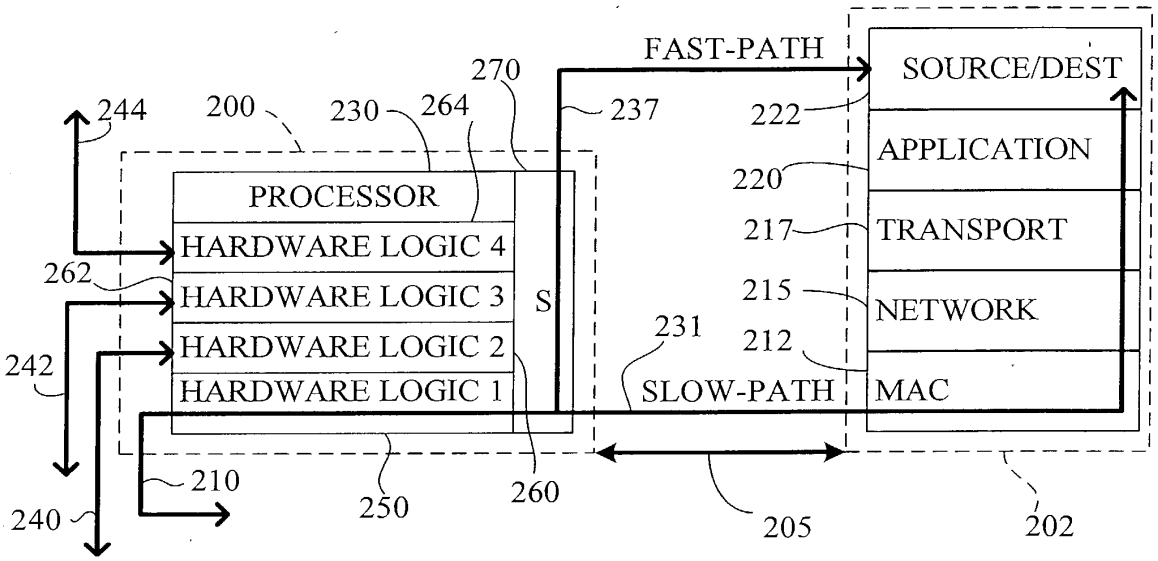

FAST-PATH
SOURCE/DEST
270
244
200
230
264
237
222
APPLICATION
220
PROCESSOR
217
TRANSPORT
HARDWARE LOGIC 4
262
HARDWARE LOGIC 3
S
215
NETWORK
231
242
HARDWARE LOGIC 2
212
HARDWARE LOGIC 1
SLOW-PATH
MAC
210
250
260
205
202
240

FIG. 9

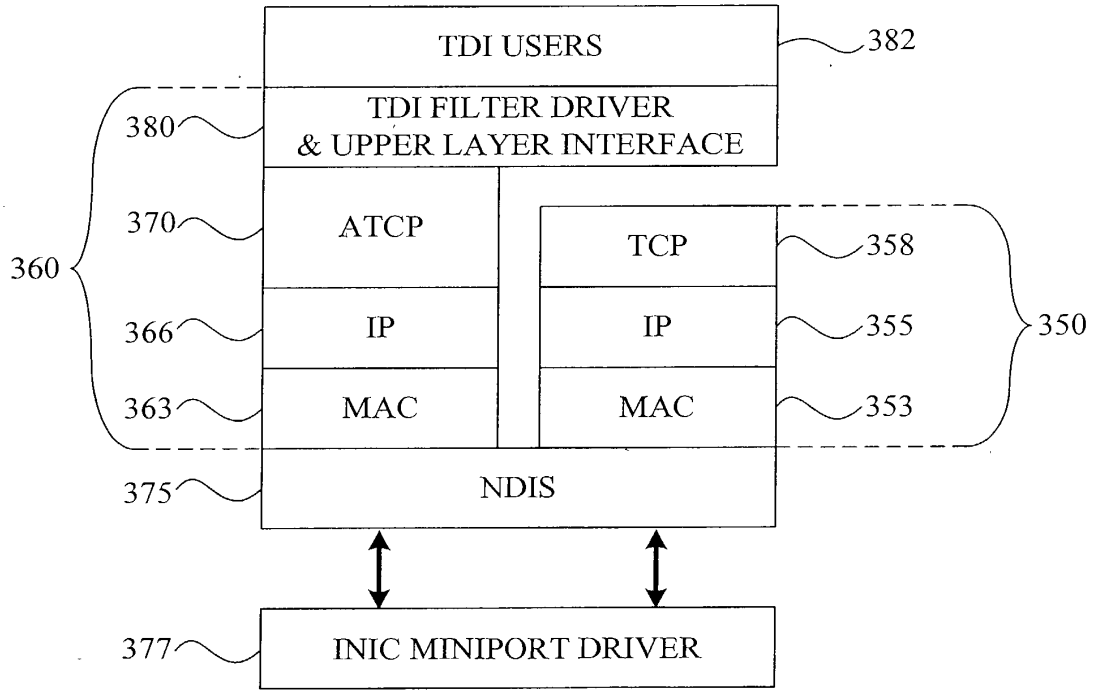

TDI USERS
382
TDI FILTER DRIVER
& UPPER LAYER INTERFACE
380
370
ATCP
TCP
358
360
366
IP
IP
355
350
363
MAC
MAC
353
375
NDIS
377
INIC MINIPORT DRIVER

FIG. 11

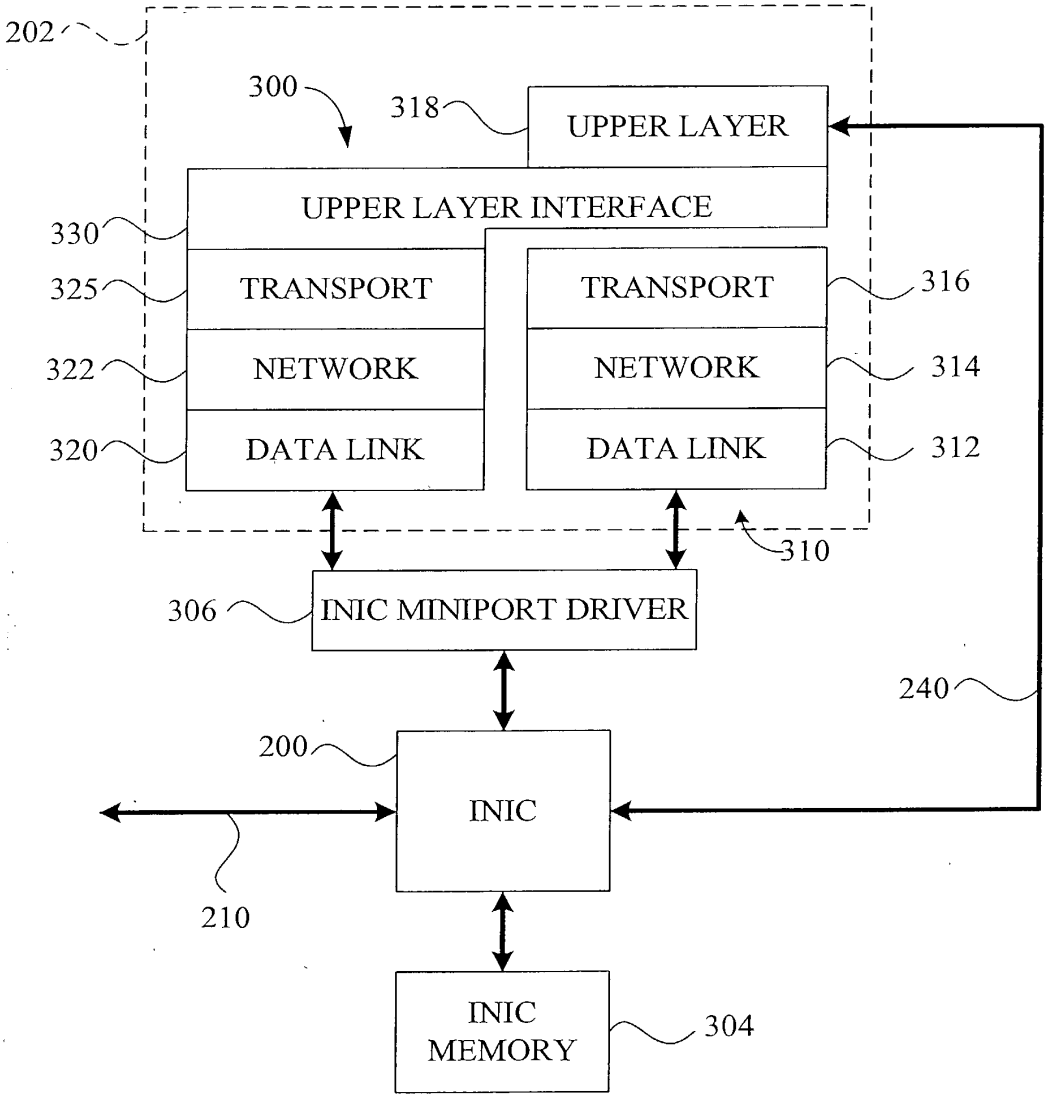

202
300
318
UPPER LAYER
UPPER LAYER INTERFACE
330
325
TRANSPORT
TRANSPORT
316
322
NETWORK
NETWORK
314
320
DATA LINK
DATA LINK
312
310
306
INIC MINIPORT DRIVER
240
200
INIC
210
INIC
MEMORY
304

FIG. 10

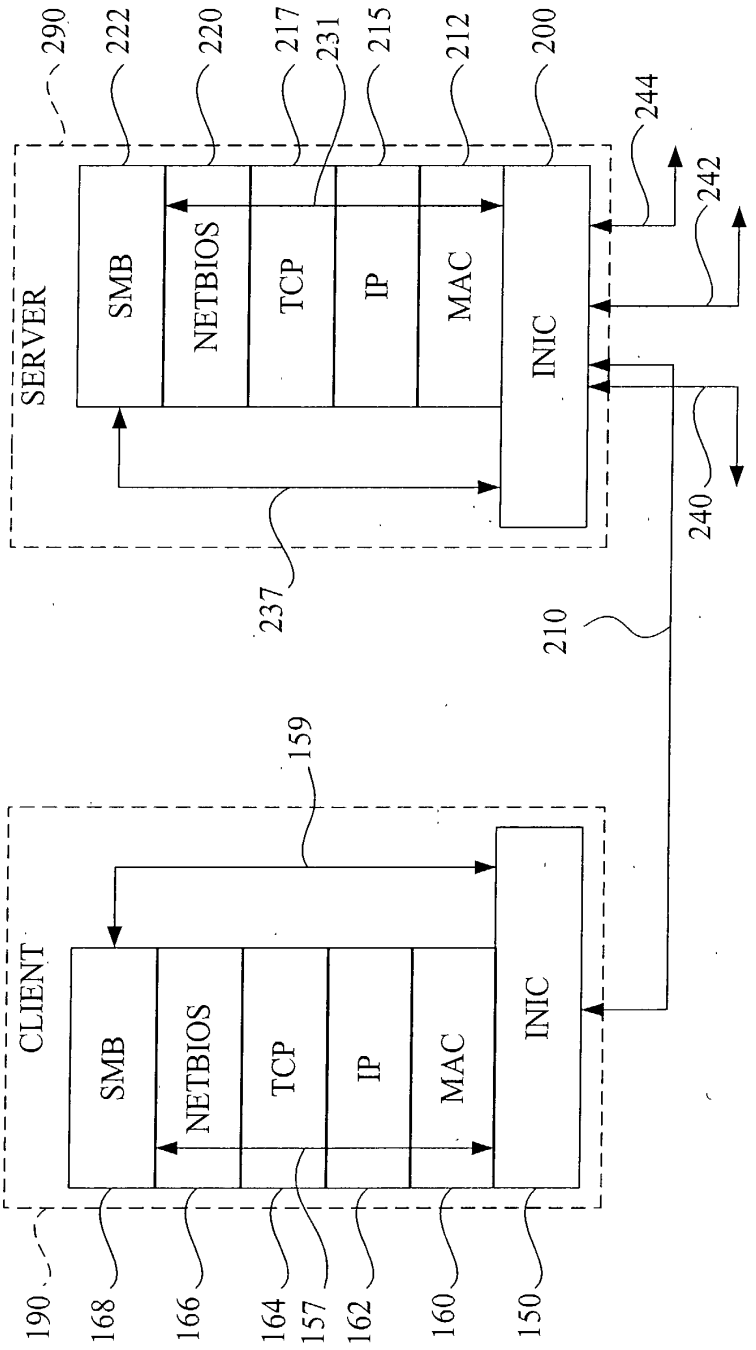

290
222
220
217
231
215
212
200
244
242
SERVER
SMB
NETBIOS
TCP
IP
MAC
INIC
240
237
210
159
CLIENT
SMB
NETBIOS
TCP
IP
MAC
INIC
190
168
166
164
157
162
160
150

FIG. 12

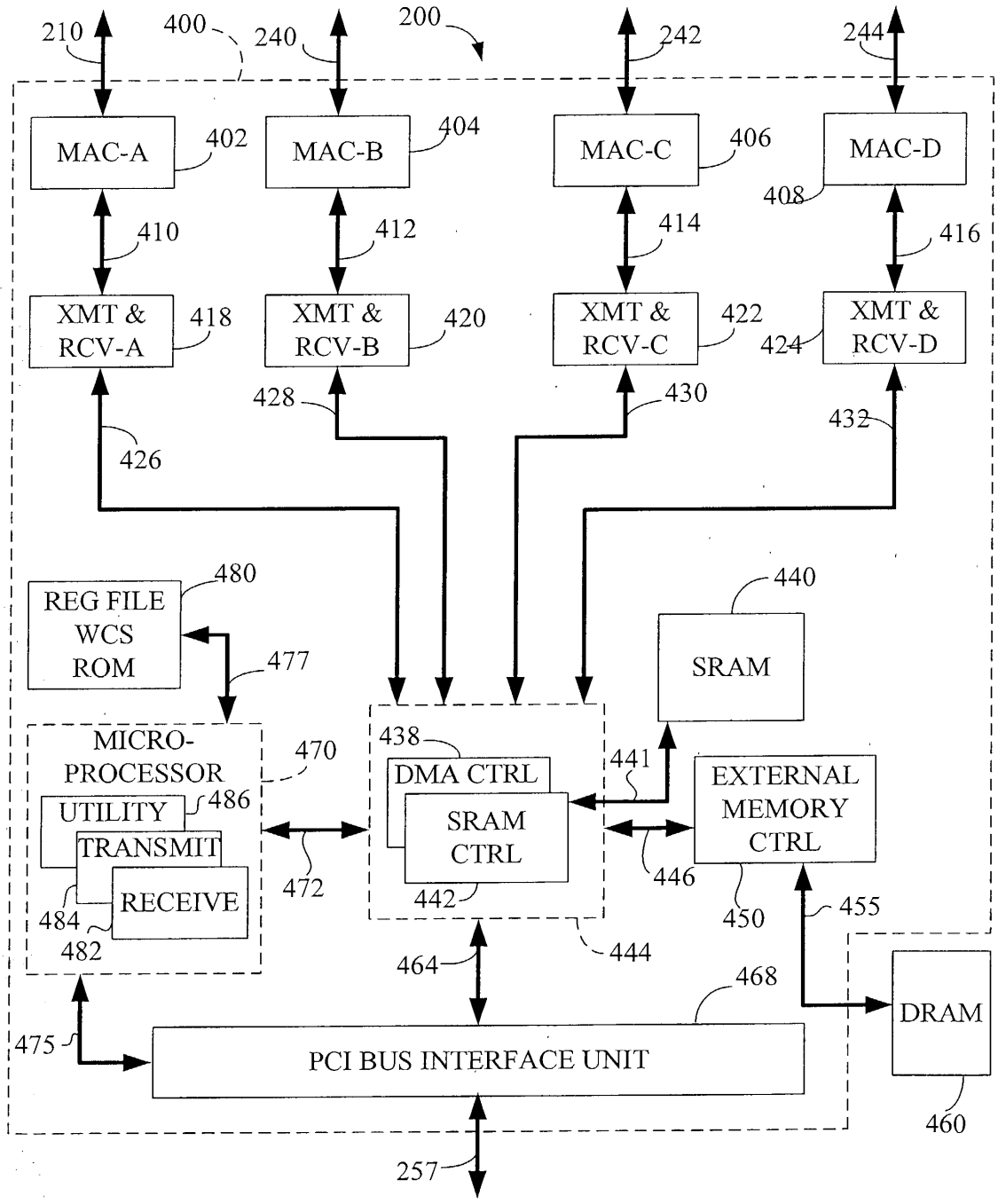

210
400
240
200
242
244
MAC-A
402
MAC-B
404
MAC-C
406
MAC-D
408
410
412
414
416
XMT & RCV-A
418
XMT & RCV-B
420
XMT & RCV-C
422
XMT & RCV-D
424
428
430
432
426
480
REG FILE WCS ROM
477
440
SRAM
MICRO-PROCESSOR
470
438
DMA CTRL
441
UTILITY
486
TRANSMIT
SRAM CTRL
EXTERNAL MEMORY CTRL
RECEIVE
484
482
472
442
446
450
455
444
464
468
475
DRAM
PCI BUS INTERFACE UNIT
460
257

FIG. 13

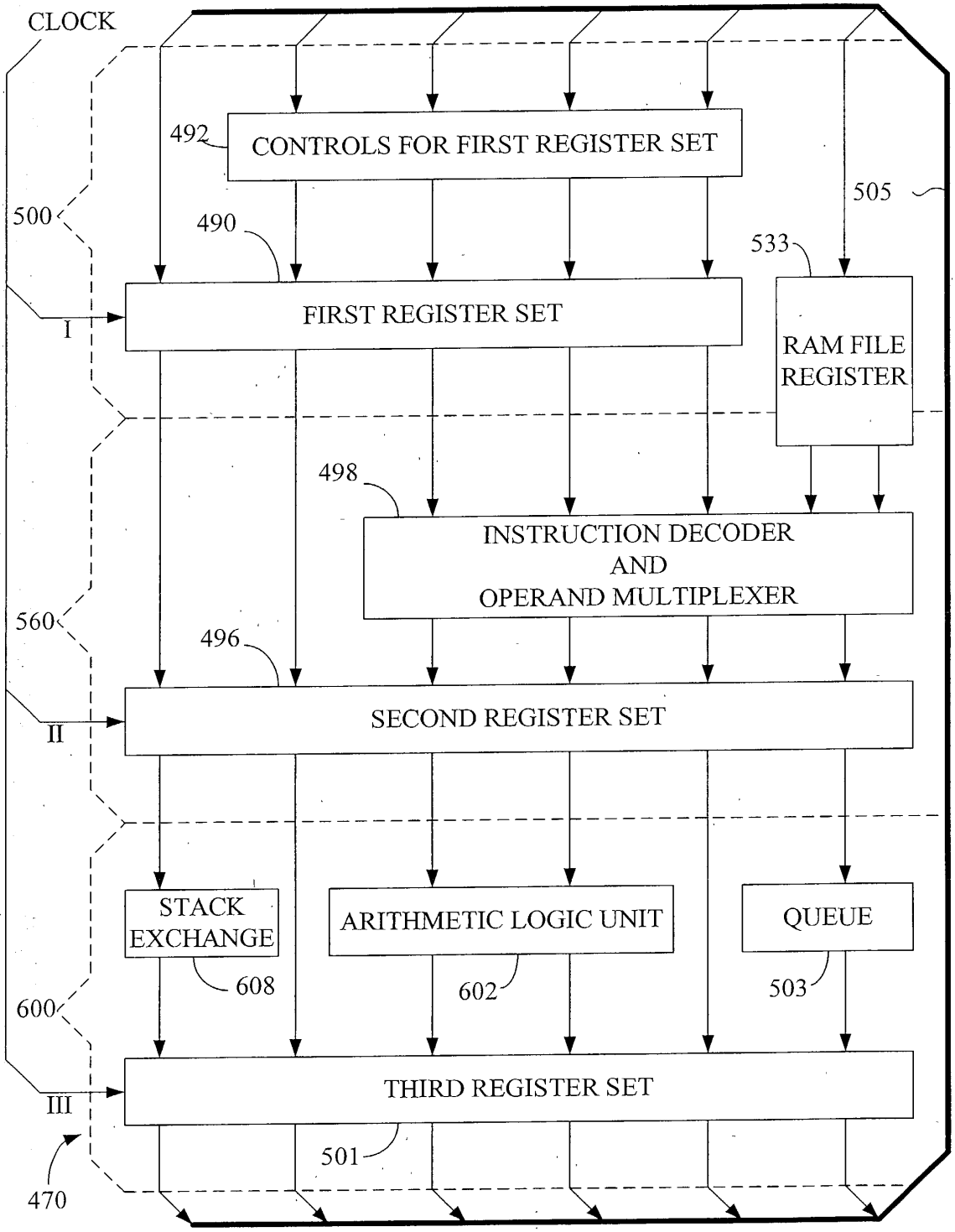

CLOCK
492
CONTROLS FOR FIRST REGISTER SET
505
500
490
533
FIRST REGISTER SET
RAM FILE REGISTER
I
498
INSTRUCTION DECODER AND OPERAND MULTIPLEXER
560
496
SECOND REGISTER SET
II
STACK EXCHANGE
608
ARITHMETIC LOGIC UNIT
602
QUEUE
503
600
THIRD REGISTER SET
III
501
470

FIG. 14

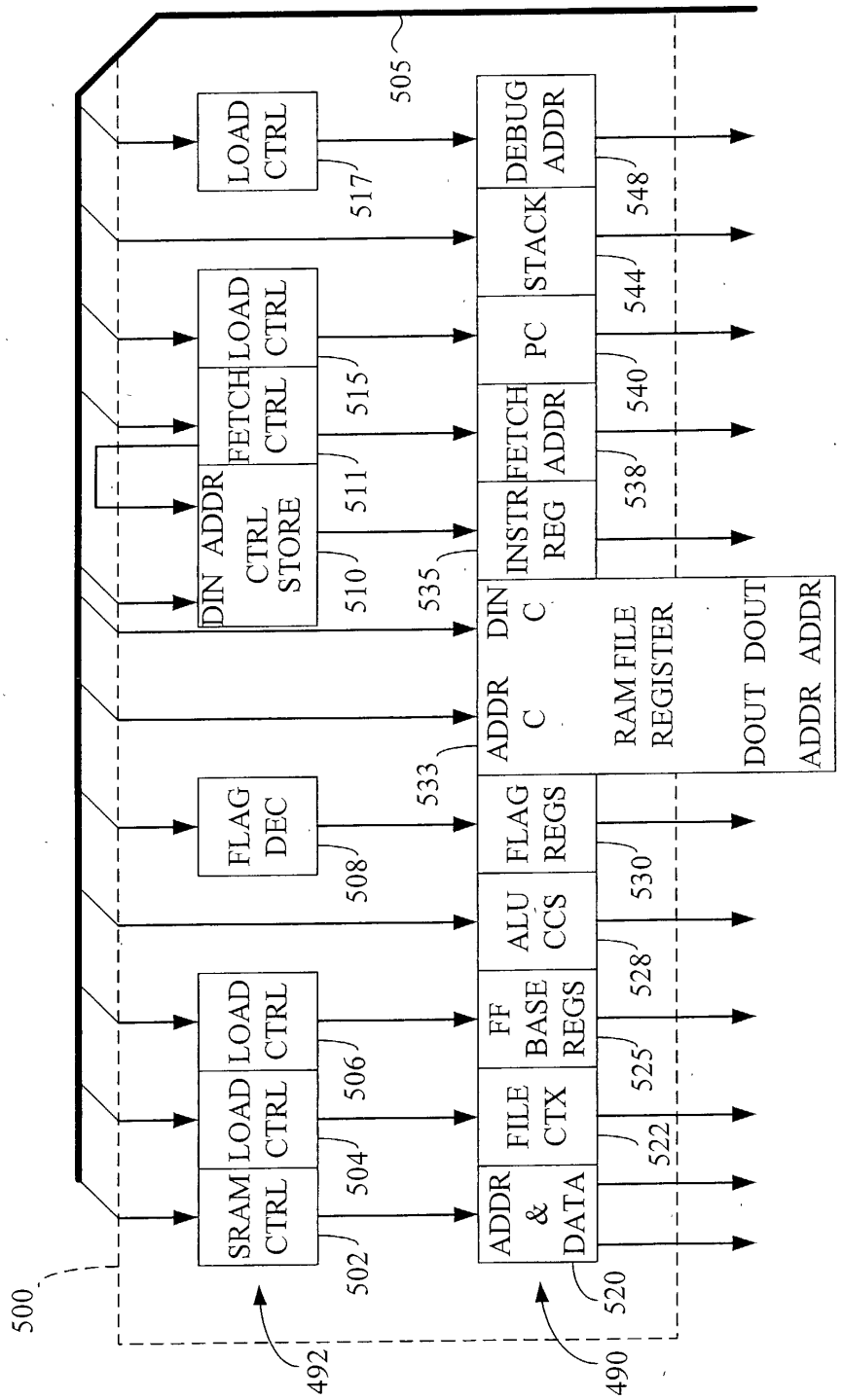

500
505
492
490
SRAM CTRL
LOAD CTRL
LOAD CTRL
LOAD CTRL
FLAG DEC
DIN ADDR CTRL STORE
FETCH CTRL
LOAD CTRL
LOAD CTRL
ADDR & DATA
FILE CTX
FF BASE REGS
ALU CCS
FLAG REGS
ADDR C
DIN C
INSTR REG
FETCH ADDR
PC
STACK
DEBUG ADDR
RAM FILE REGISTER
DOUT ADDR
DOUT ADDR
502
504
506
508
510
511
515
517
520
522
525
528
530
533
535
538
540
544
548

FIG. 15A

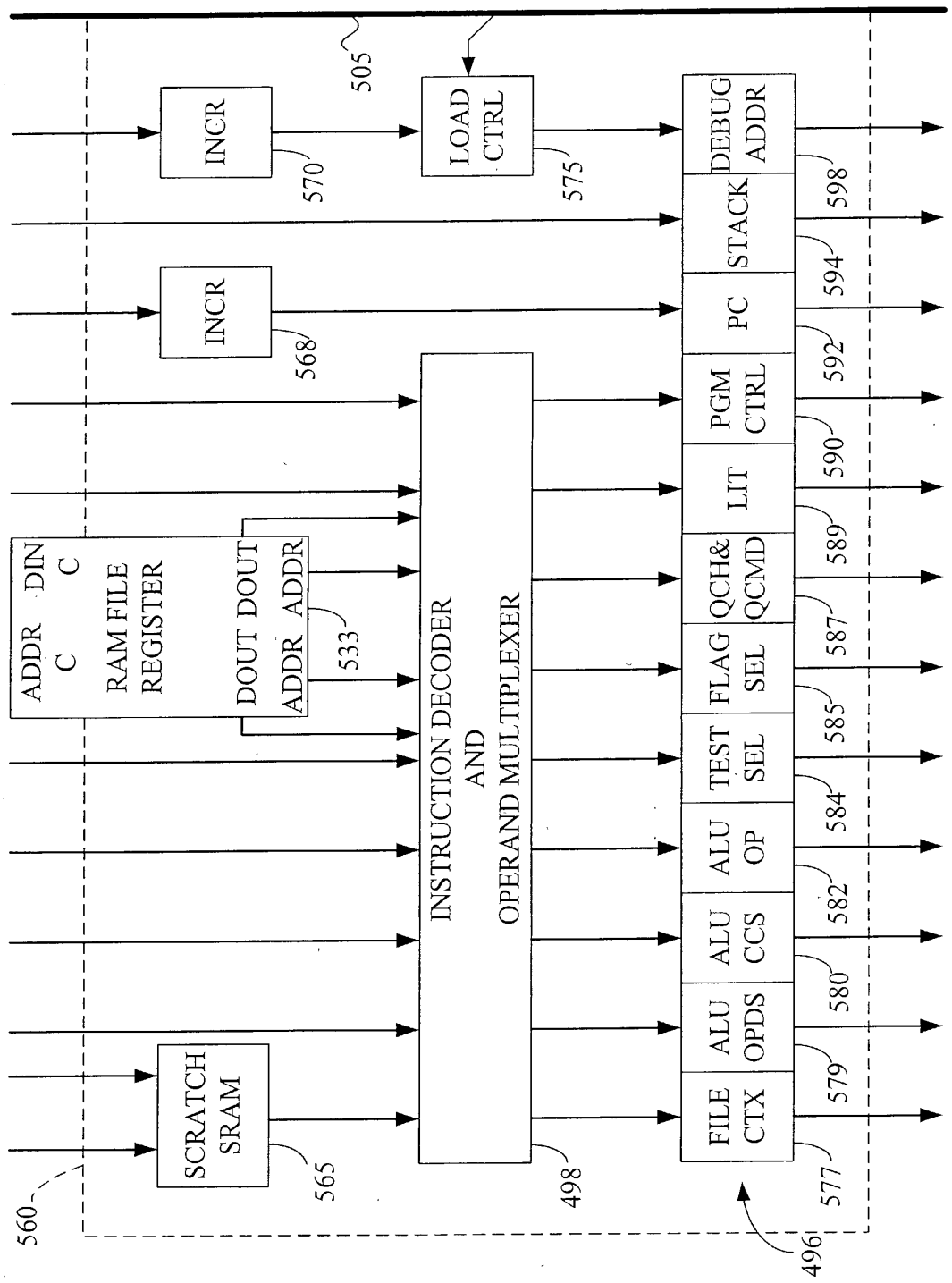

505
INCR
570
LOAD CTRL
575
DEBUG ADDR
598
INCR
568
STACK
594
PC
592
PGM CTRL
590
LIT
589
QCH& QCMD
587
FLAG SEL
585
TEST SEL
584
ALU OP
582
ALU CCS
580
ALU OPDS
579
FILE CTX
577
ADDR DIN C C RAM FILE REGISTER
DOUT DOUT ADDR ADDR
533
INSTRUCTION DECODER AND OPERAND MULTIPLEXER
498
SCRATCH SRAM
565
560
496

FIG. 15B

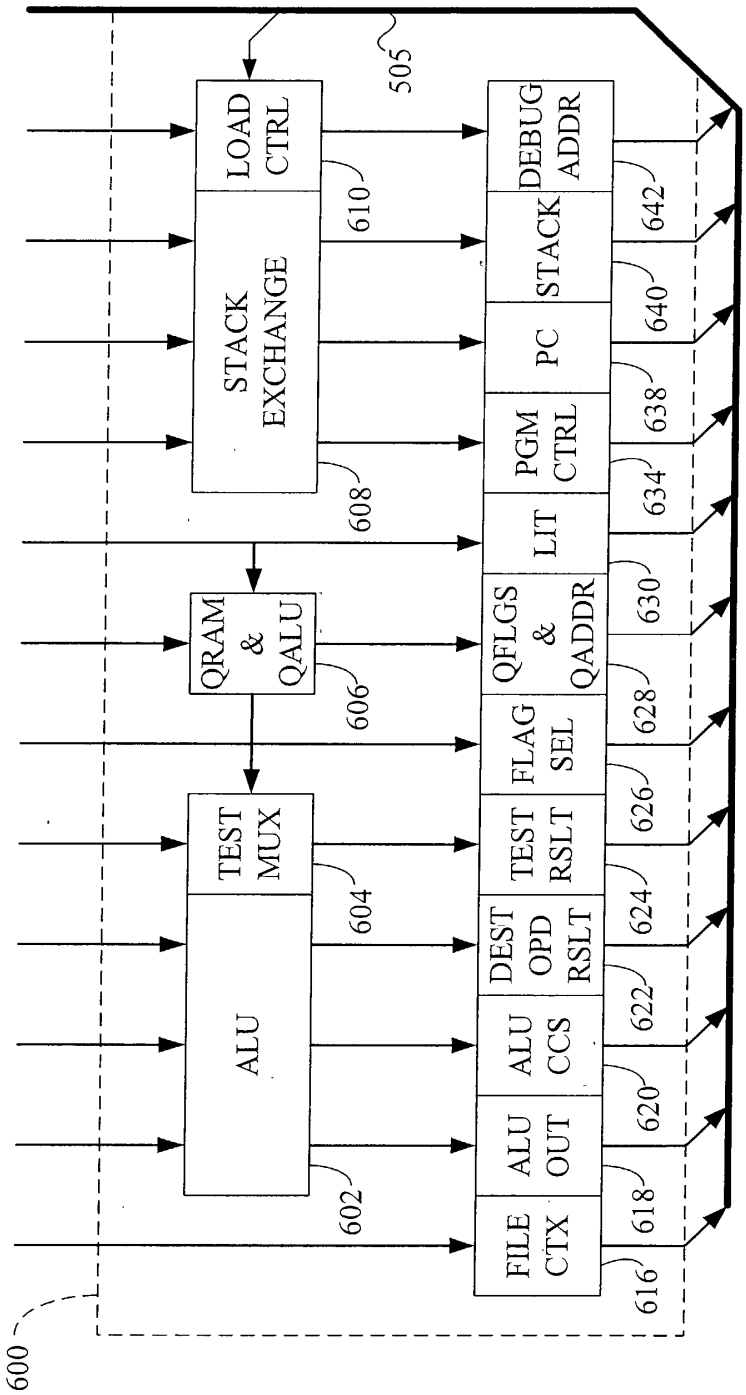

505
LOAD CTRL
STACK EXCHANGE
QRAM & QALU
TEST MUX
ALU
FILE CTX
ALU OUT
ALU CCS
DEST OPD RSLT
TEST RSLT
FLAG SEL
QFLGS & QADDR
LIT
PGM CTRL
PC
STACK
DEBUG ADDR
600
602
604
606
608
610
616
618
620
622
624
626
628
630
634
638
640
642

FIG. 15C

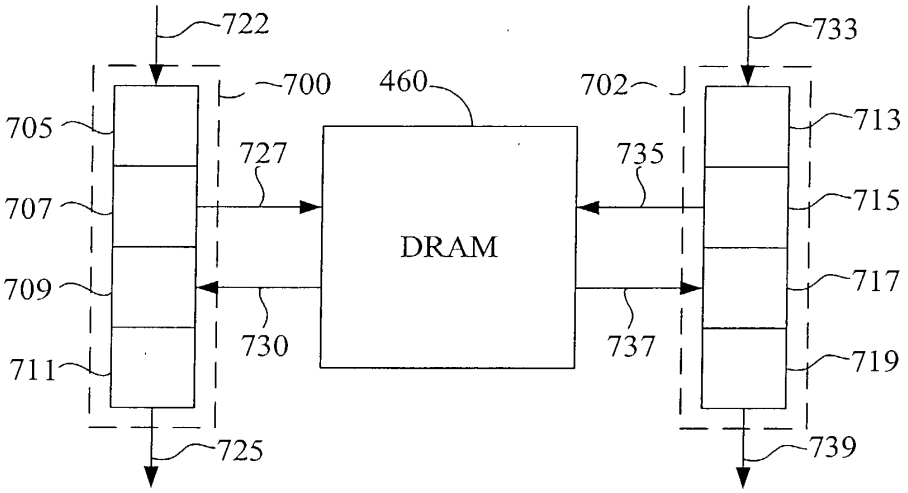

722
700
460
702
733
705
727
735
713
707
DRAM
715
709
717
730
737
711
719
725
739

FIG. 16

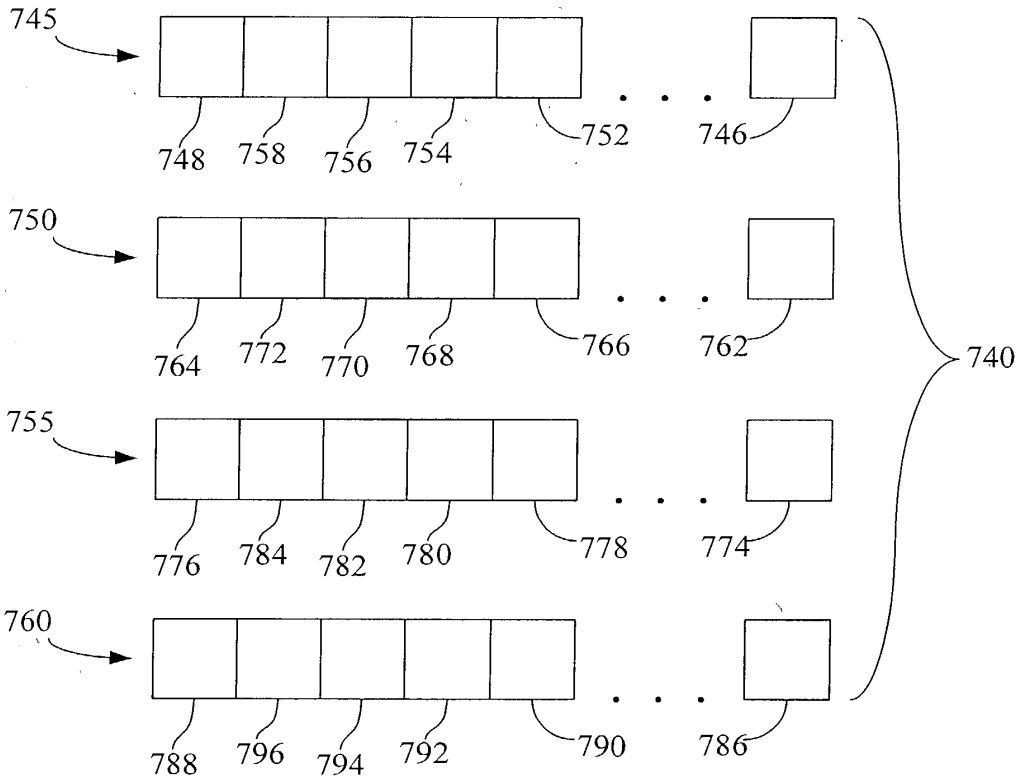

745
748
758
756
754
752
746
750
764
772
770
768
766
762
740
755
776
784
782
780
778
774
760
788
796
794
792
790
786

FIG. 17

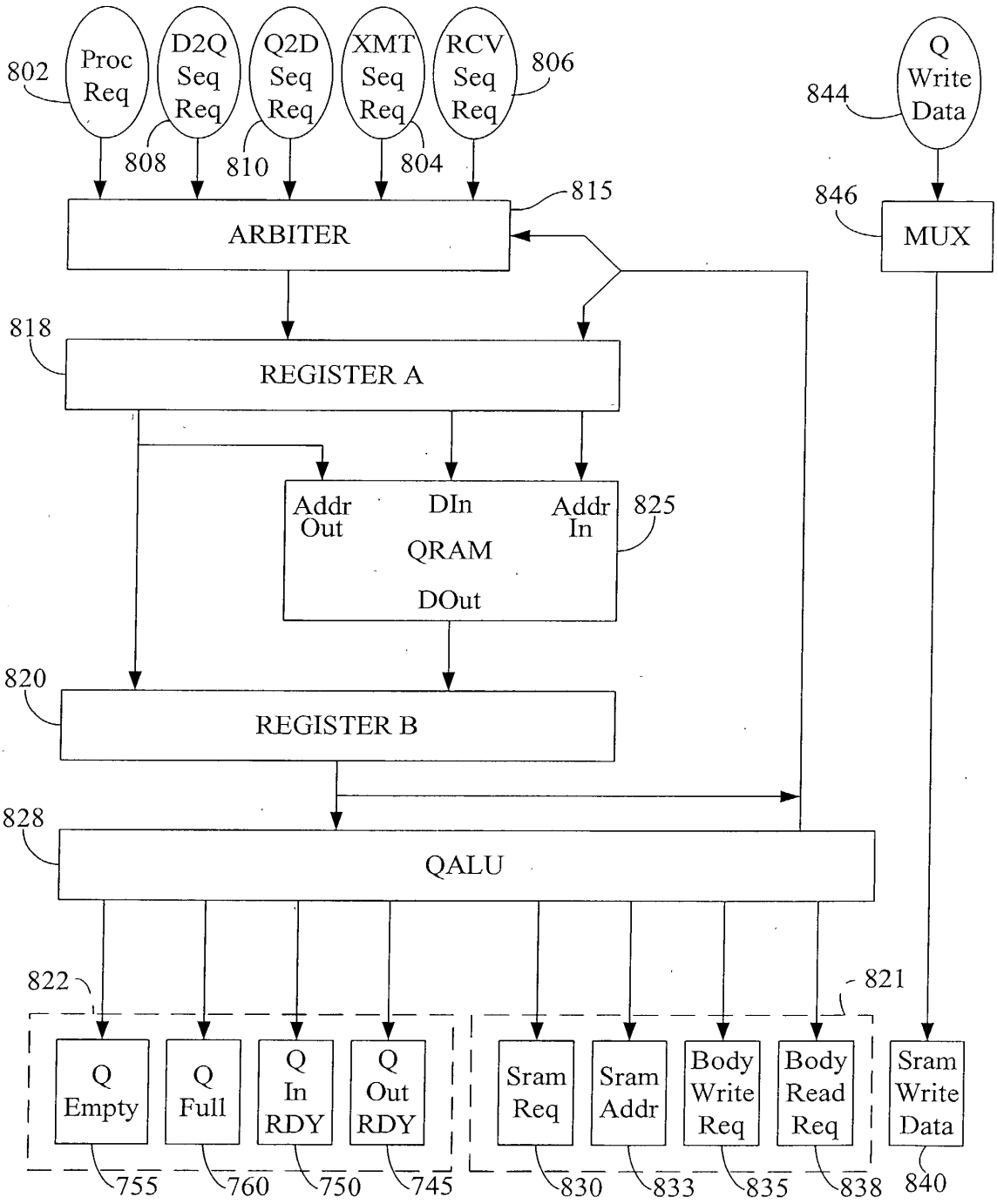

802
Proc Req
D2Q Seq Req
Q2D Seq Req
XMT Seq Req
RCV Seq Req
806
808
810
804
844
Q Write Data
846
MUX
815
ARBITER
818
REGISTER A
Addr Out
DIn
Addr In
825
QRAM
DOut
820
REGISTER B
828
QALU
822
821
Q Empty
Q Full
Q In RDY
Q Out RDY
Sram Req
Sram Addr
Body Write Req
Body Read Req
Sram Write Data
755
760
750
745
830
833
835
838
840

FIG. 18

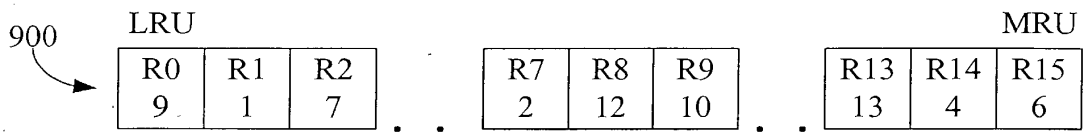


FIG. 19A

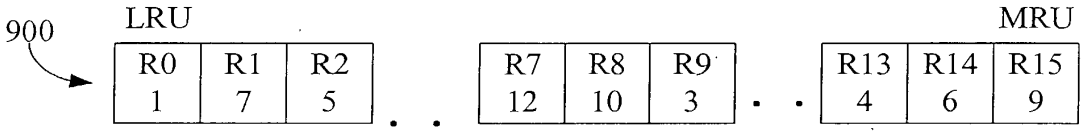


FIG. 19B

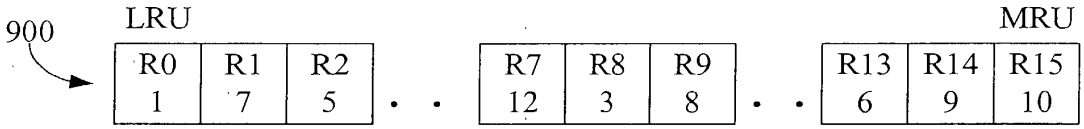


FIG. 19C

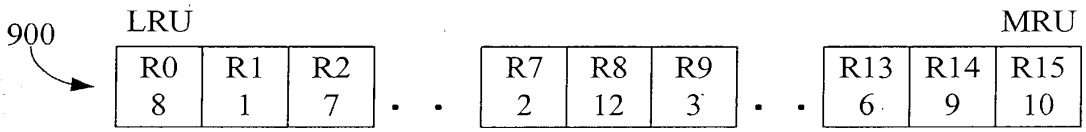


FIG. 19D

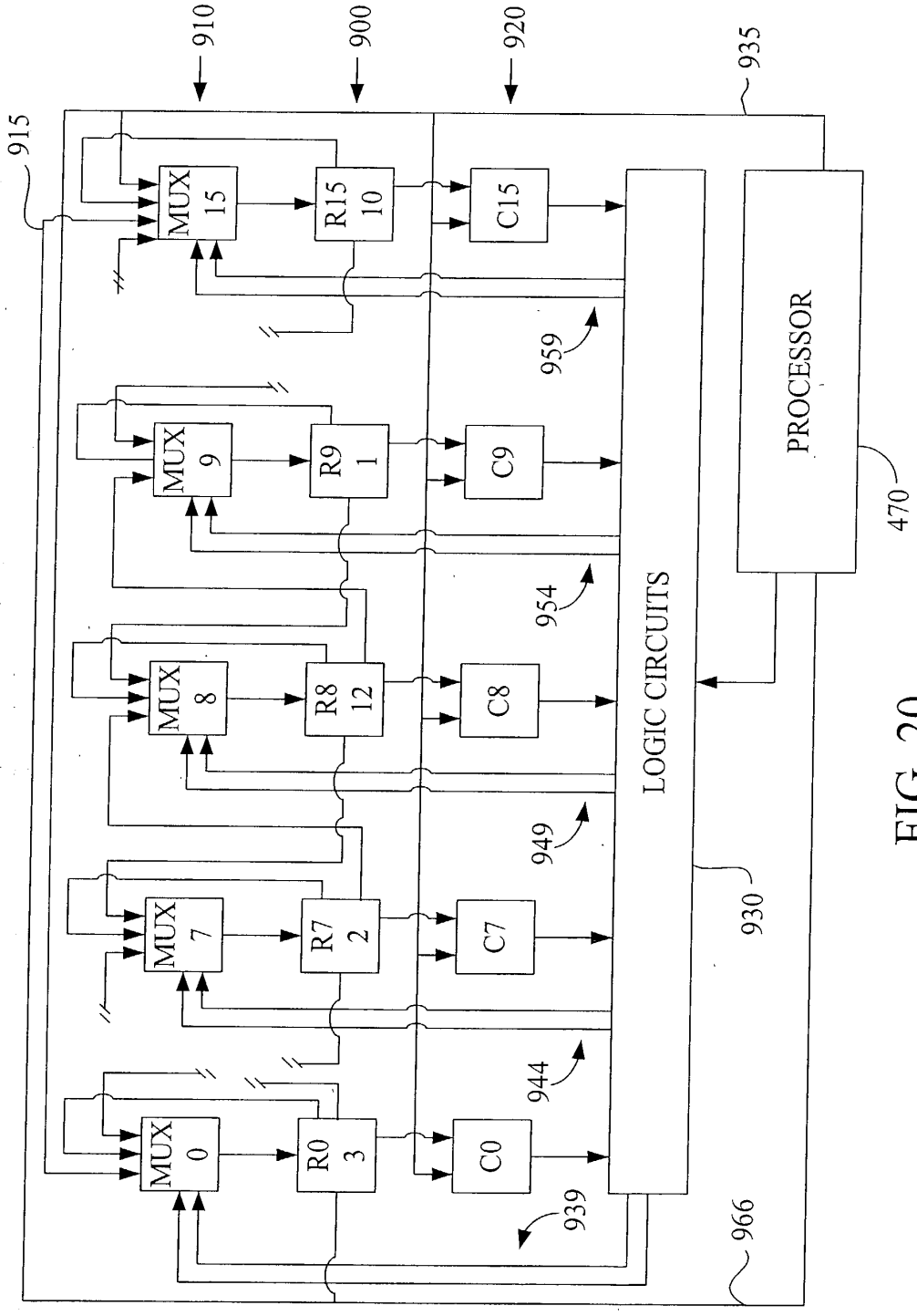

910
900
920
935
915
MUX 15
R15 10
C15
959
MUX 9
R9 1
C9
954
MUX 8
R8 12
C8
949
MUX 7
R7 2
C7
944
MUX 0
R0 3
C0
939
966
930
LOGIC CIRCUITS
PROCESSOR
470

FIG. 20

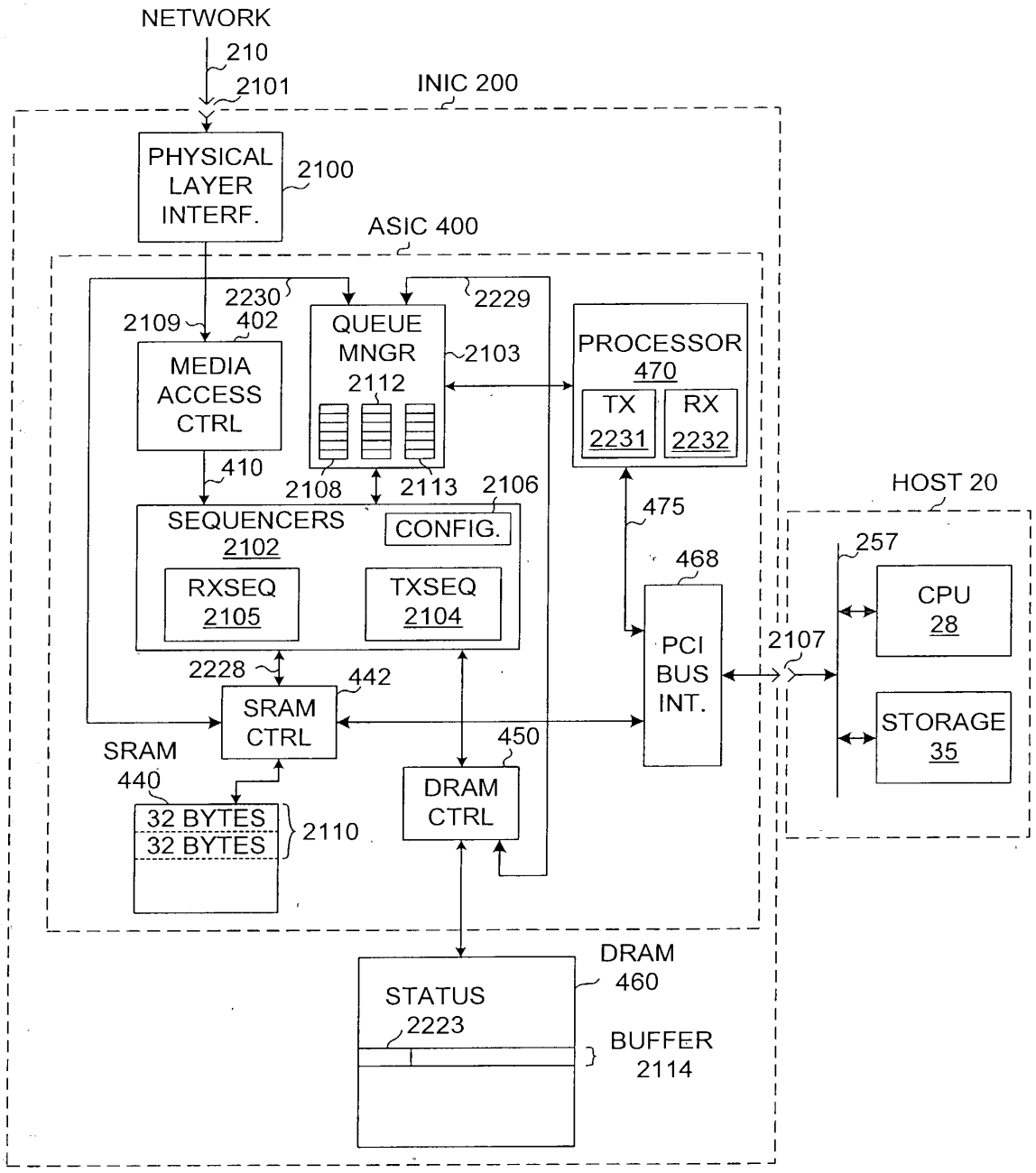

NETWORK
210
2101
INIC 200
PHYSICAL LAYER INTERF.
2100
ASIC 400
2230
2229
2109
402
QUEUE MNGR 2112
2103
PROCESSOR 470
TX 2231
RX 2232
MEDIA ACCESS CTRL
410
2108
2113
2106
SEQUENCERS 2102
CONFIG.
RXSEQ 2105
TXSEQ 2104
475
468
HOST 20
257
CPU 28
PCI BUS INT.
2107
2228
442
SRAM CTRL
STORAGE 35
SRAM 440
450
DRAM CTRL
32 BYTES
32 BYTES
2110
DRAM 460
STATUS 2223
BUFFER 2114

FIG. 21

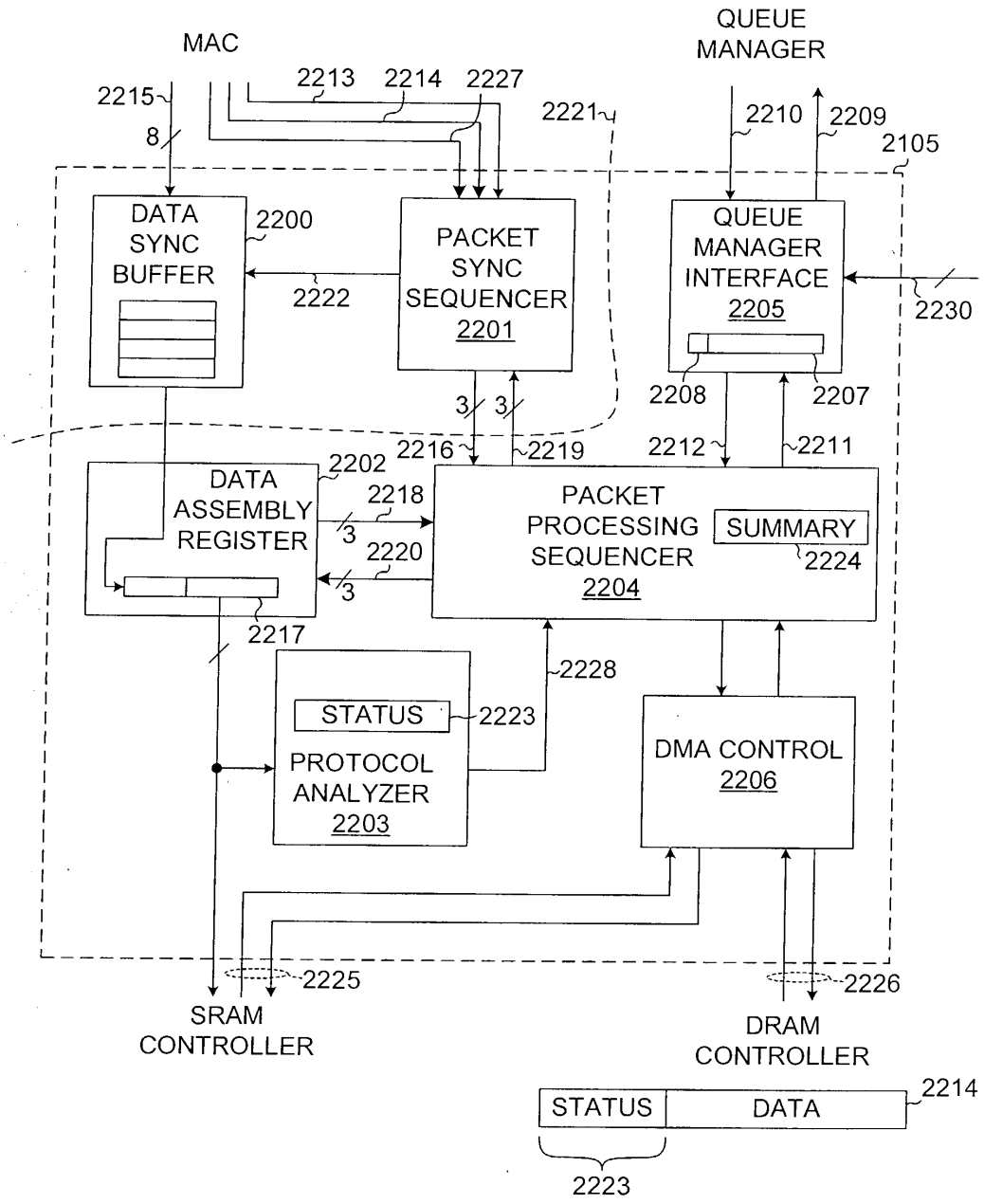

MAC
QUEUE MANAGER
2215
8
2213
2214
2227
2221
2210
2209
2105
DATA SYNC BUFFER
2200
2222
PACKET SYNC SEQUENCER 2201
QUEUE MANAGER INTERFACE 2205
2230
2208
2207
3
3
2216
2219
2212
2211
DATA ASSEMBLY REGISTER
2202
2218
3
2220
3
PACKET PROCESSING SEQUENCER 2204
SUMMARY
2224
2217
2228
STATUS
2223
PROTOCOL ANALYZER 2203
DMA CONTROL 2206
2225
2226
SRAM CONTROLLER
DRAM CONTROLLER
STATUS
DATA
2214
2223

FIG. 22

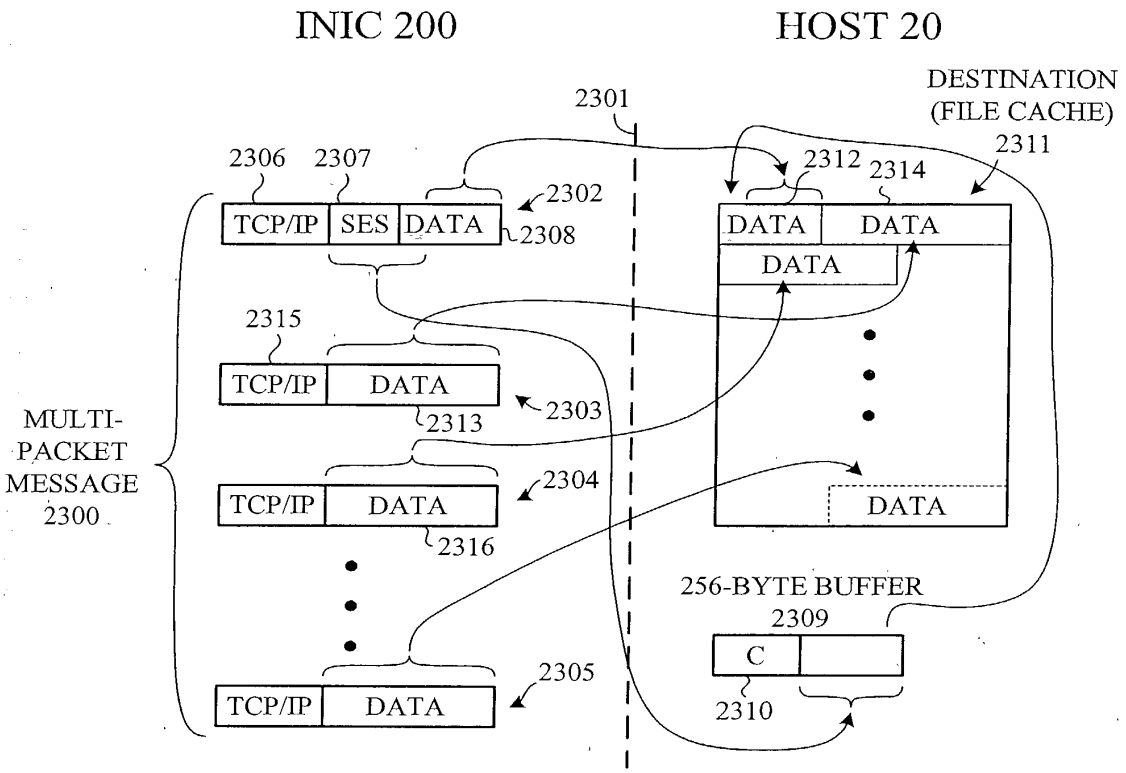

INIC 200
HOST 20
DESTINATION
(FILE CACHE)
2301
2306
2307
2312
2314
2311
2302
TCP/IP
SES
DATA
2308
DATA
DATA
DATA
2315
TCP/IP
DATA
2303
2313
MULTI-
PACKET
MESSAGE
2300
TCP/IP
DATA
2304
2316
DATA
256-BYTE BUFFER
2309
C
2305
TCP/IP
DATA
2310

FIG. 23

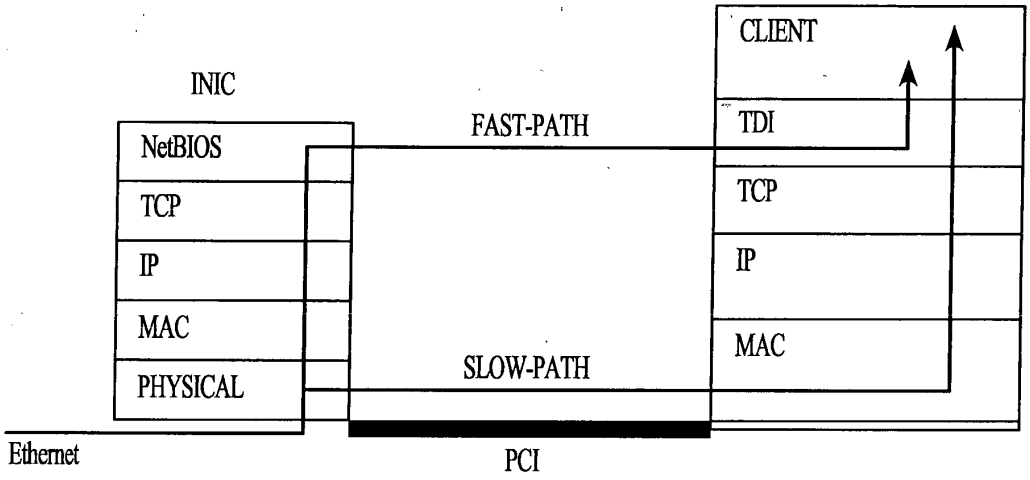

INIC
NetBIOS
TCP
IP
MAC
PHYSICAL
Ethernet
FAST-PATH
SLOW-PATH
PCI
CLIENT
TDI
TCP
IP
MAC

FIG. 24

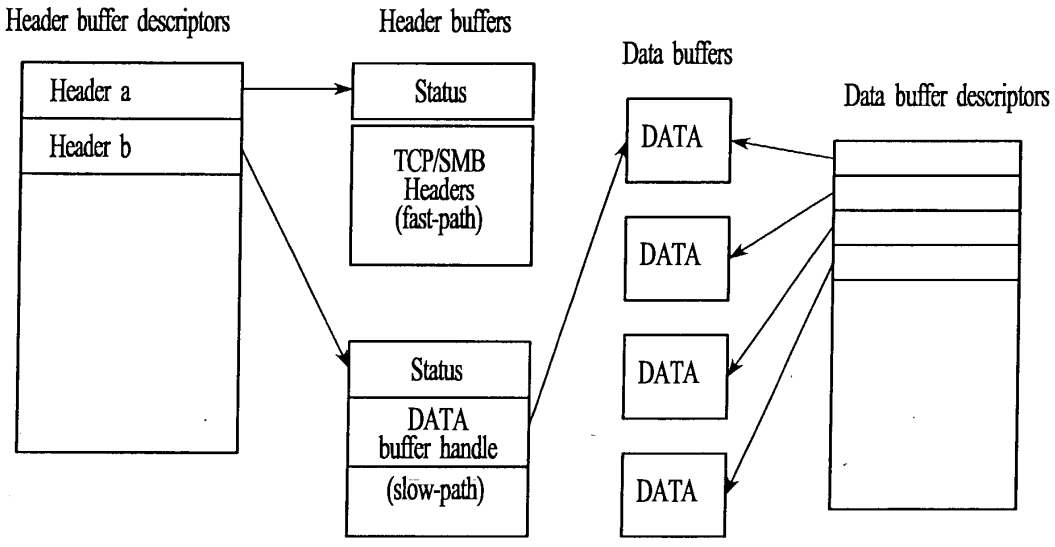

Header buffer descriptors
Header a
Header b
Header buffers
Status
TCP/SMB
Headers
(fast-path)
Status
DATA
buffer handle
(slow-path)
Data buffers
DATA
DATA
DATA
DATA
Data buffer descriptors

FIG. 25

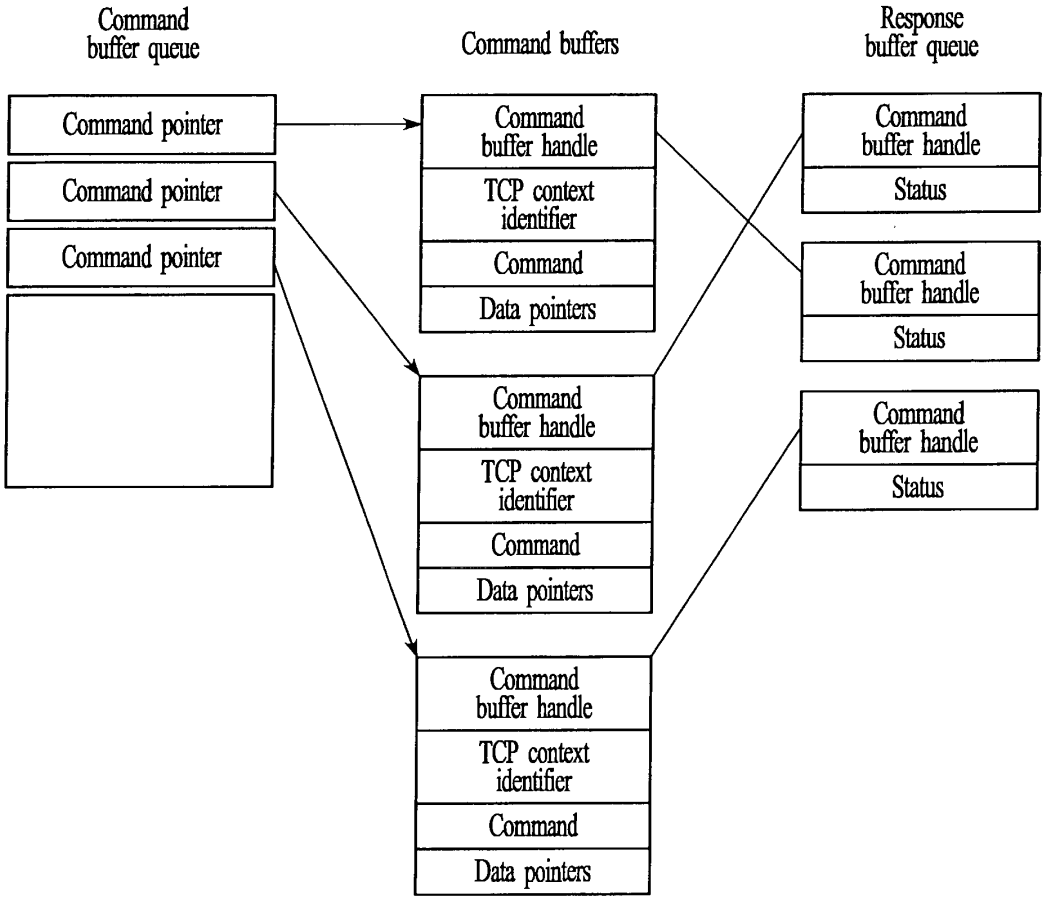

Command buffer queue
Command buffers
Response buffer queue
Command pointer
Command pointer
Command pointer
Command buffer handle
TCP context identifier
Command
Data pointers
Command buffer handle
TCP context identifier
Command
Data pointers
Command buffer handle
TCP context identifier
Command
Data pointers
Command buffer handle
Status
Command buffer handle
Status
Command buffer handle
Status

FIG. 26

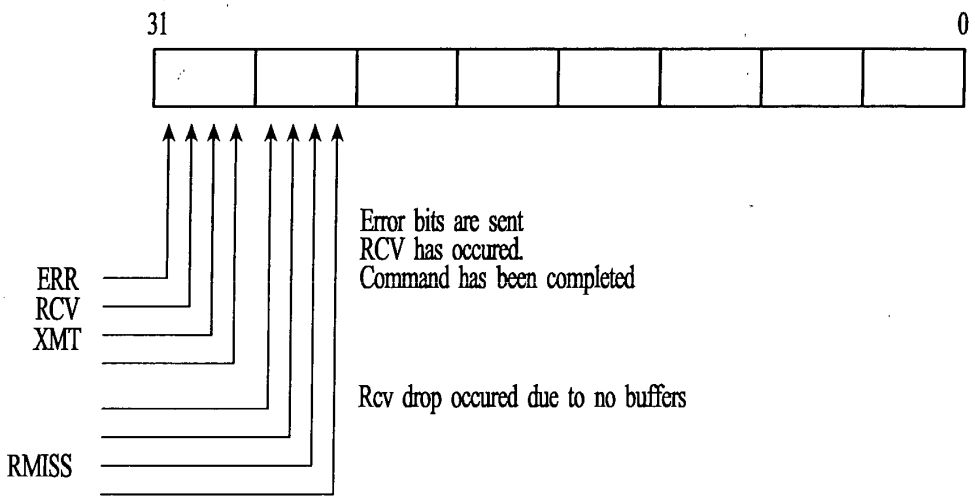


FIG. 27

ISR	0x0	Interrupt Status
IMR	0x4	Interrupt Mask
HBAR	0x8	Header Buffer Address
DBHR	0xC	Data Buffer Handle
DBAR	0x10	Data Buffer Address
CBAR0	0x14	Command Buffer Address XMT0
CBAR1	0x18	Command Buffer Address XMT1
CBAR2	0x1C	Command Buffer Address XMT2
CBAR3	0x20	Command Buffer Address XMT3
CBAR4	0x24	Command Buffer Address RCV
RBAR	0x28	Response Buffer Address

FIG. 28

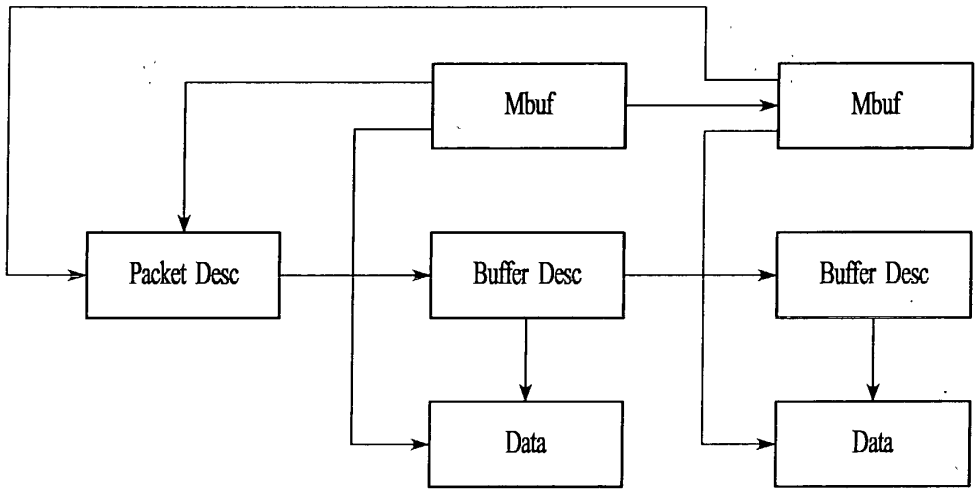

Mbuf
Mbuf
Packet Desc
Buffer Desc
Buffer Desc
Data
Data

FIG. 29

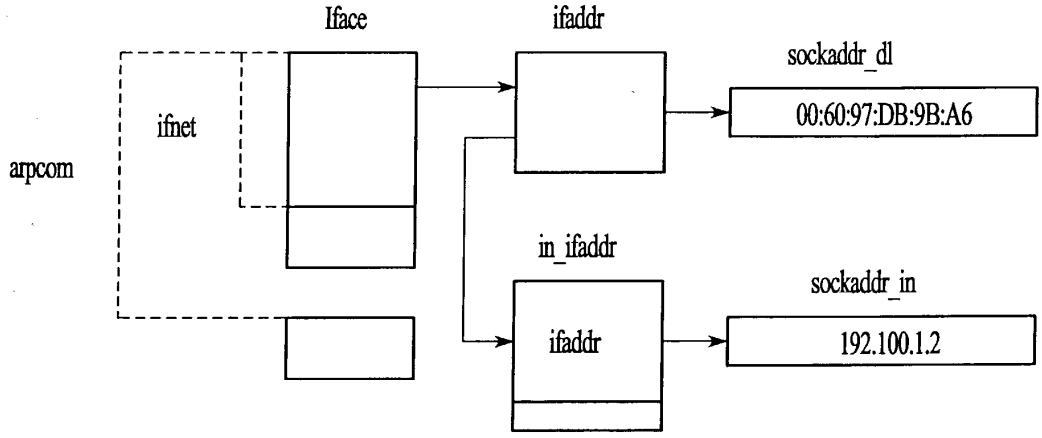

Iface
ifaddr
sockaddr_dl
ifnet
00:60:97:DB:9B:A6
arpcom
in_ifaddr
sockaddr_in
ifaddr
192.100.1.2

FIG. 30

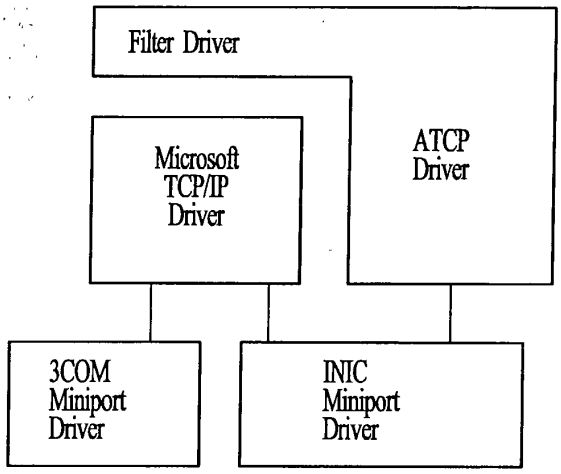
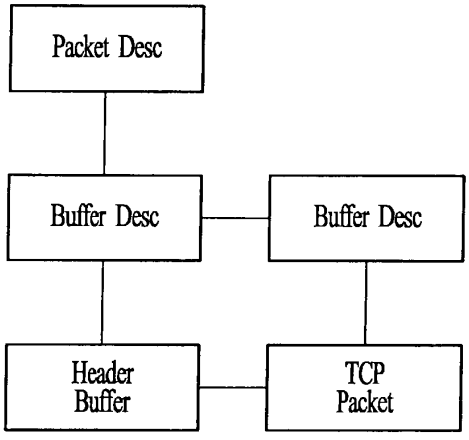
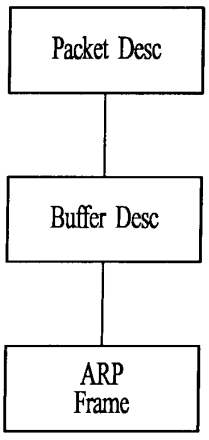


FIG. 31



Example of incoming TCP pkt

FIG. 32



Example of incoming ARP Frame

FIG. 33

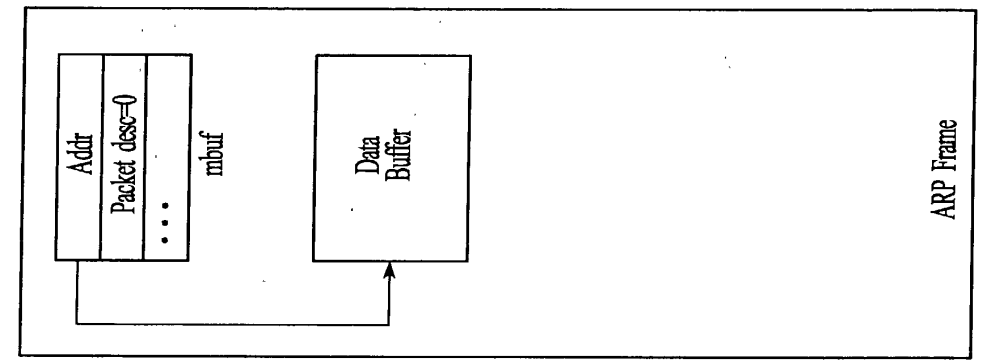


FIG. 34C

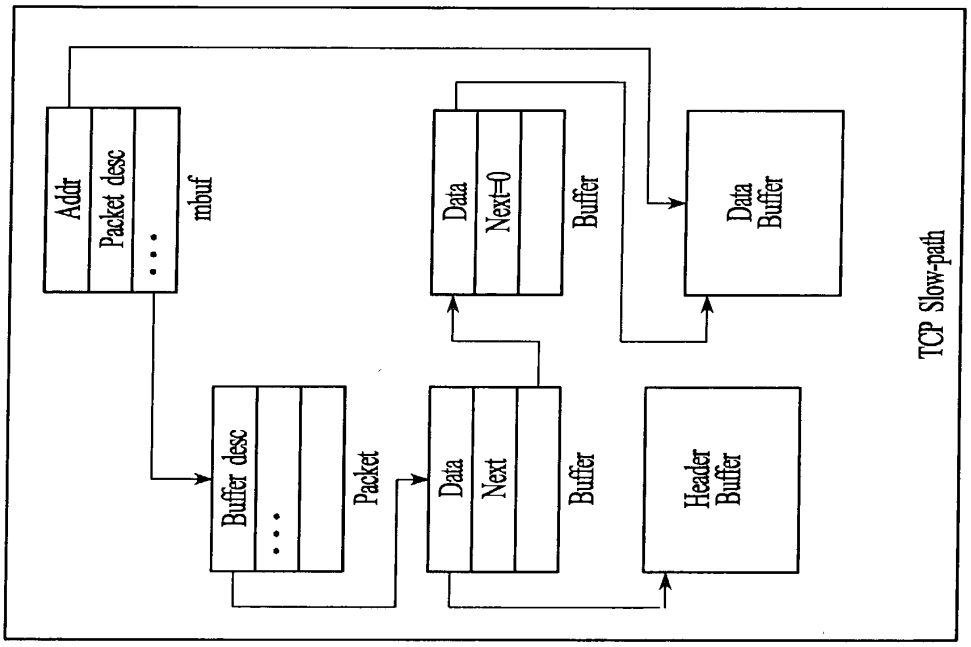


FIG. 34B

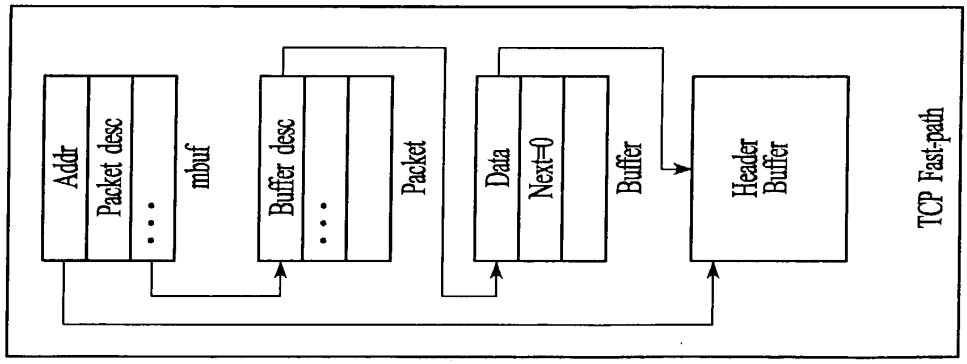


FIG. 34A

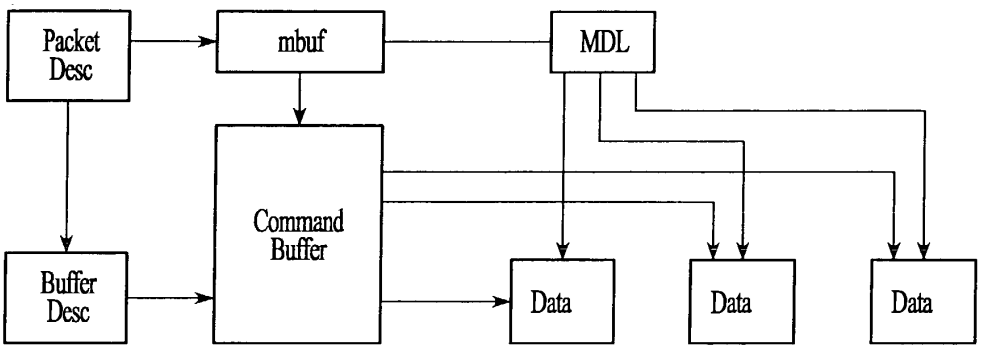

Packet Desc
mbuf
MDL
Command Buffer
Buffer Desc
Data
Data
Data

FIG. 35

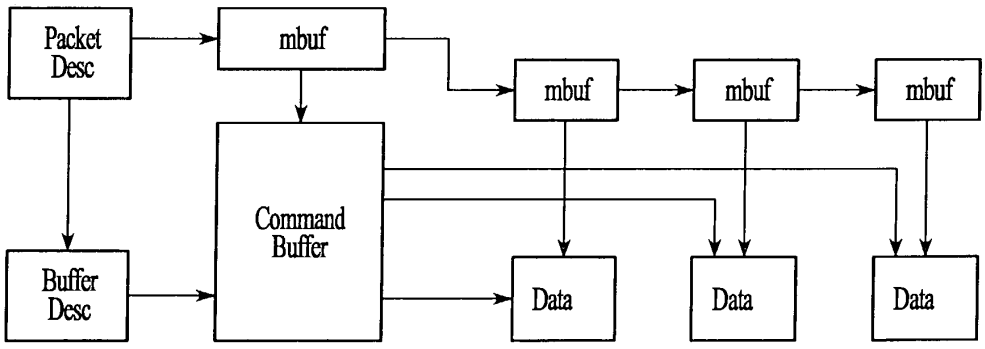

Packet Desc
mbuf
mbuf
mbuf
mbuf
Command Buffer
Buffer Desc
Data
Data
Data

FIG. 36

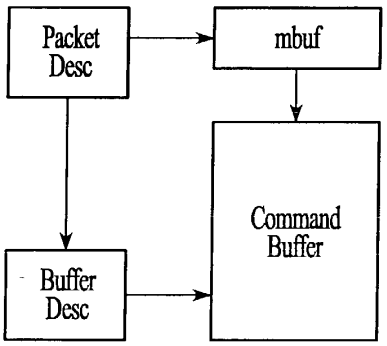


FIG. 37

SRAM requirements for the Receive and Transmit engines:

TCB buffers	256 bytes	* 16	4096
Header buffers	128 bytes	* 16	2048
TCB hash index	16 bytes	* 256	4096
Timers			128
DRAM Fifo queues	128 bytes	* 16	2048
			~12K bytes

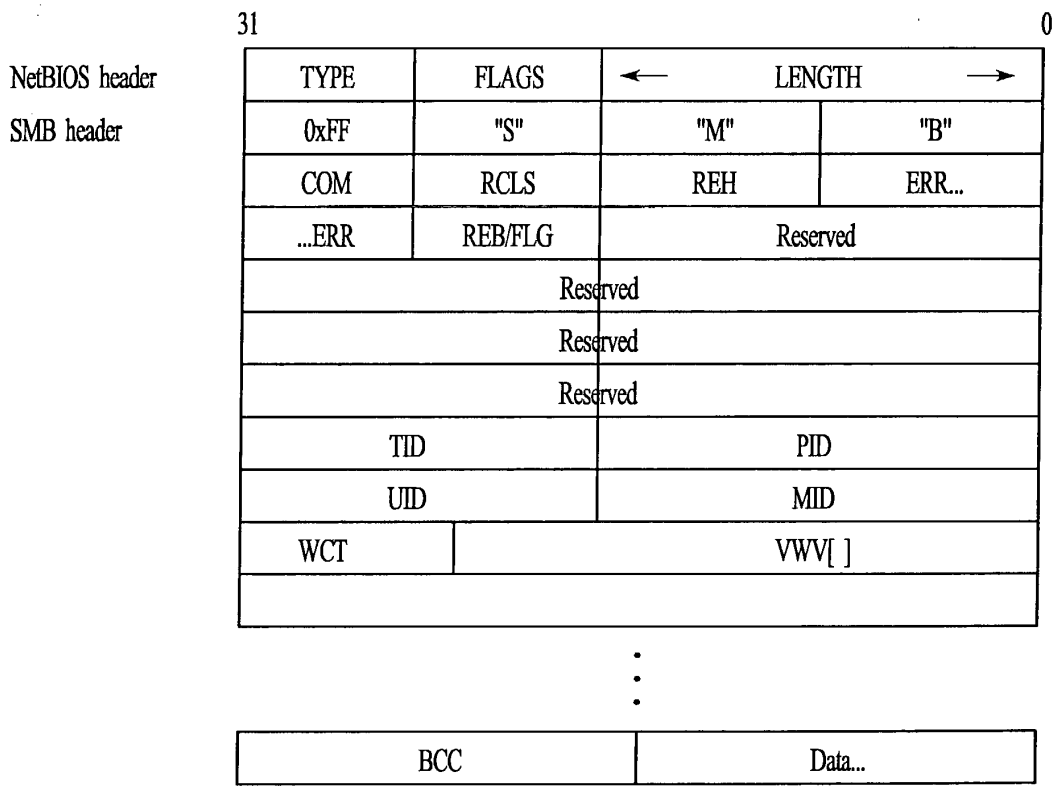
FIG. 38

Summary of the main loop of Receive:

```
forever {
        while there are any Receive events {
                if (a new event) {
                        if (no new context available)
                                ignore the event;
                }
                call appropriate event handler to service the event;
                this may make a waiting process runnable or set up
                a new process to be run (get free context, hddr buffer,
                TCB buffer, set the context up).
        }
        while any process contexts are runable {
                run them by jumping to the start/resume address;
                if (process complete)
                        free the context;
        }
}
```

FIG. 39

Format of the SMB header of an SMB frame:



Notes (interesting fields):

LENGTH	17 bit Length of SMB message (0 - 128K)
COM	SMB command
WCT	Count (16 bit) of parameter words in VWV [ ]
VWV	Variable number of parameter words
BCC	Bytes of data following

FIG. 40

Summary of the main loop of Transmit:

```
forever {
        while there are any Transmit events {
                if (a new event) {
                        if (no new context available)
                                ignore the event;
                }
                call appropriate event handler to service the event;
                this may make a waiting process runnable or set up
                a new process to be run (get free context, hddr buffer,
                TCB buffer, set the context up).
        }
        while any process contexts are runable {
                run them by jumping to the start/resume address;
                if (process complete)
                        free the context;
        }
}
```

FIG. 41

Bit 31 - 24 Byte enable 7 - 0. Only the low order four bits are valid for 32 bit addressing mode.
Bit 23 - 0 Memory access
1 Configuration access
Bit 22 - 0 Read (to Host)
1 Write (to Host)
Bit 21 - 1 Data Valid
Bit 20 - 16 Reserved
Bit 15 - 0 Address

FIG. 42

Configuration Space 1	SRAM Address Offset
00	00
04	04
08	08
0C	0C
10	10
3C	14

Configuration Space 2	
00	00
04	18
08	08
0C	1C
10	20
3C	24

All other reads to configuration space will return 00.

FIG. 43

Bit 0 - 0 I/O accesses are not enabled
Bit 1 - 1 Memory accesses are enabled
Bit 2 - 1 Bus master is enabled
Bit 3 - 0 Special Cycle is not enabled
Bit 4 - 1 Memory Write and Invalidate is enabled
Bit 5 - 0 VGA palette snooping is not enabled
Bit 6 - 1 Parity checking is enabled
Bit 7 - 0 Address data stepping is not enabled
Bit 8 - SERR# is enabled
Bit 9 - 0 Fast back to back is not enabled

FIG. 44

Bit 5 - 1 66 MHz capable is enabled. This bit will be set if the INIC Detects the system running at 66 MHz on reset
Bit 6 - 0 User Definable Features is not enabled
Bit 7 - 1 Fast Back-to-Back slave transfers enabled
Bit 8 - 1 Parity Error enabled - This bit is initialized to 0
Bit 9,10 - 00 - Fast device select will be set if we are at 33 MHz
01 - Medium device select will be set if we are at 66 MHz
Bit 11 - 1 Target Abort is implemented. Initialized to 0.
Bit 12 - 1 Target Abort is implemented. Initialized to 0.
Bit 13 - 1 Master Abort is implemented. Initialized to 0.
Bit 14 - 1 SERR# is implemented. Initialized to 0.
Bit 15 - 1 Parity error is implemented. Initialized to 0.

FIG. 45

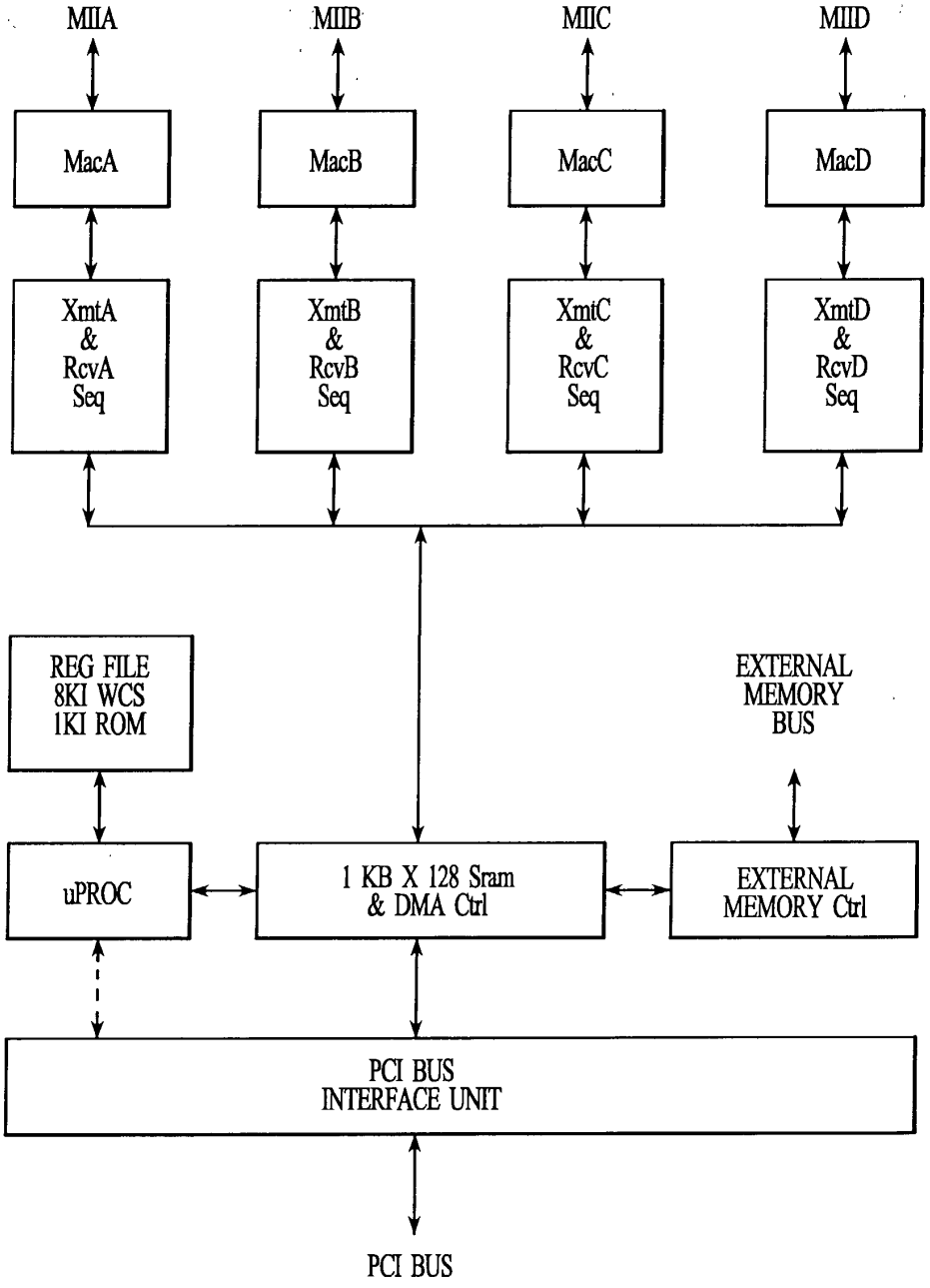

MIIA
MIIB
MIIC
MIID
MacA
MacB
MacC
MacD
XmtA & RcvA Seq
XmtB & RcvB Seq
XmtC & RcvC Seq
XmtD & RcvD Seq
REG FILE 8KI WCS 1KI ROM
EXTERNAL MEMORY BUS
uPROC
1 KB X 128 Sram & DMA Ctrl
EXTERNAL MEMORY Ctrl
PCI BUS INTERFACE UNIT
PCI BUS

FIG. 46

MODULE	DESCR	SPEED	AREA
Scratch RAM,	1Kx128 sport,	4.37 ns nom.,	06.77 mm^2
WCS,	8Kx49 sport,	6.40 ns nom.,	18.29 mm^2
MAP,	128x7 sport,	3.50 ns nom.,	00.24 mm^2
ROM,	1Kx49 32col,	5.00 ns nom.,	00.45 mm^2
REGs,	512x32 tport,	6.10 ns nom.,	03.49 mm^2
Macs,	.75 mm^2 x 4 =		03.30 mm^2
PLL,	.5 mm^2 =		00.55 mm^2
MISC LOGIC,	117,260 gates / (5035 gates / $mm^{2)}$ =		23.29 mm^2
TOTAL CORE			56.22 mm^2

$(\text{Core side})^2$		=	56.22 mm^2
Core side		=	07.50 mm
Die side	= core side + 1.0 mm (I/O cells)	=	08.50 mm
Die area	= 8.5 mm x 8.5 mm	=	72.25 mm^2
Pads needed	= 220 signals x 1.25 (vss, vdd)	=	275 pins
LSI PBGA		=	272 pins

FIG. 47

(10MB/s/100Base) x 2 (full duplex) x 4 connections	=	80 MB/s
Average frame size	=	512 B
Frame rate = 80MB/s / 512B	=	156,250 frames / s
Cpu overhead / frame = (256B context read) + (64B header read) + (128B context write) + (128B misc.)	=	512B / frame
Total bandwidth = (512B in) + (512B out) + (512B Cpu)	=	1536B / frame
Dram Bandwidth required = (1536B/frame) x (156,250 frames/s)	=	240MB/s
Dram Bandwidth @ 60MHz = (32 bytes / 167ns)	=	202MB/s
Dram Bandwidth @ 66MHz = (32 bytes / 150ns)	=	224MB/s
PCI Bandwidth required	=	80MB/s
PCI Bandwidth available @ 30 MHz, 32b, average	=	46MB/s
PCI Bandwidth available @ 33 MHz, 32b, average	=	50MB/s
PCI Bandwidth available @ 60 MHz, 32b, average	=	92MB/s
PCI Bandwidth available @ 66 MHz, 32b, average	=	100MB/s
PCI Bandwidth available @ 30 MHz, 64b, average	=	92MB/s
PCI Bandwidth available @ 33 MHz, 64b, average	=	100MB/s
PCI Bandwidth available @ 60 MHz, 64b, average	=	184MB/s
PCI Bandwidth available @ 66 MHz, 64b, average	=	200MB/s

FIG. 48

Receive frame interval = 512B / 40MB/s	=	12.8us
Instructions / frame @ 60MHz = (12.8us/frame) / (50ns/instruction) instructions/frame	=	256
Instructions / frame @ 66MHz = (12.8us/frame) / (45ns/instruction) instructions/frame	=	284
Required instructions / frame	=	250 instructions/frame

FIG. 49

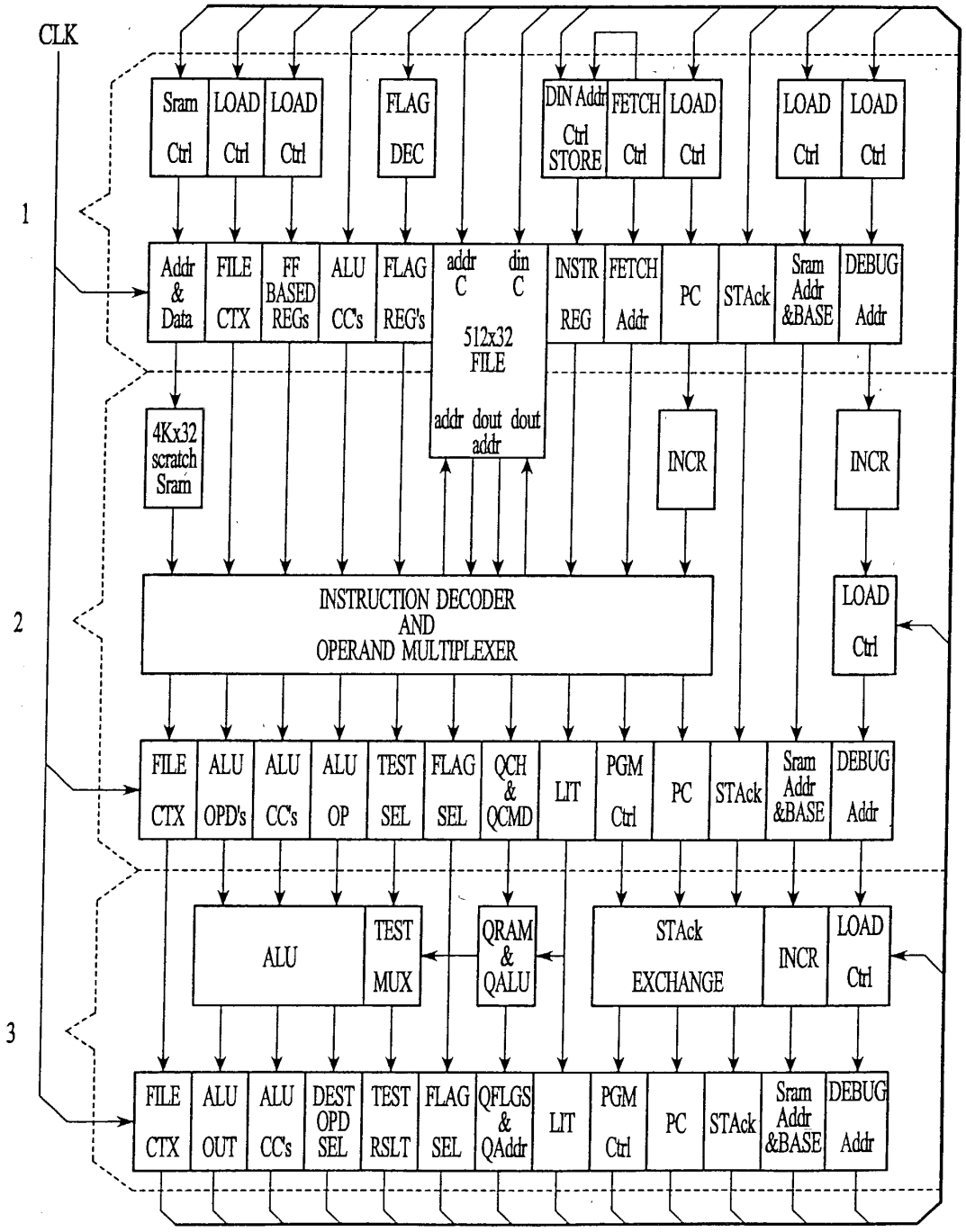

CLK
1
2
3
Sram Ctrl
LOAD Ctrl
LOAD Ctrl
FLAG DEC
DIN Addr Ctrl STORE
FETCH Ctrl
LOAD Ctrl
LOAD Ctrl
LOAD Ctrl
Addr & Data
FILE CTX
FF BASED REGs
ALU CC's
FLAG REG's
addr C
din C
512x32 FILE
addr dout dout addr
INSTR REG
FETCH Addr
PC
STAck
Sram Addr &BASE
DEBUG Addr
4Kx32 scratch Sram
INCR
INCR
INSTRUCTION DECODER AND OPERAND MULTIPLEXER
LOAD Ctrl
FILE CTX
ALU OPD's
ALU CC's
ALU OP
TEST SEL
FLAG SEL
QCH & QCMD
LIT
PGM Ctrl
PC
STAck
Sram Addr &BASE
DEBUG Addr
ALU
TEST MUX
QRAM & QALU
STAck EXCHANGE
INCR
LOAD Ctrl
FILE CTX
ALU OUT
ALU CC's
DEST OPD SEL
TEST RSLT
FLAG SEL
QFLGS & QAddr
LIT
PGM Ctrl
PC
STAck
Sram Addr &BASE
DEBUG Addr

FIG. 50

INSTRUCTION-WORD FORMAT

TYPE	[55:49]	[48:47]	[46:42]	[41:33]	[32:24]	[23:16]	[15:00]
Jcc	0b0000000	0b00,	**AluOp,**	**OpdASel,**	**OpdBSel,**	**TstSel,**	**Literal**
Jmp	0b0000000	0b01,	**AluOp,**	**OpdASel,**	**OpdBSel,**	**FlgSel,**	**Literal**
Jsr	0b0000000	0b10,	**AluOp,**	**OpdASel,**	**OpdBSel,**	**FlgSel,**	**Literal**
Rts	0b0000000	0b11,	**AluOp,**	**OpdASel,**	**OpdBSel,**	0hff,	**Literal**
Nxt	0b0000000	0b11,	**AluOp,**	**OpdASel,**	**OpdBSel,**	**FlgSel,**	**Literal**
Map	**MapAddr**	0BXX,	0BXXXXX,	0BXXXXXXXXX,	0BXXXXXXXXX,	0HXX,	0HXXXX

FIG. 51

SEQUENCER BEHAVIOR

```
if (MapEn & (MapAddr != 0b0000000)){                    //re-map instr
        Stackc = Stackc;
        StackB = StackB;
        StackA = StackA;
    InstrAddr = 0h8000 | Pc[2:0] | (MapAddr << 3);
        Pc = InstrAddr + (Execute & ~DbgMd);
     Fetch = DbgMd ? DbgAddr:InstrAddr;
   DbgAddr = DbgAddr + (Execute & DbgMd);}

else if (PgmCtrl == Jcc){                               //conditional jump
        Stackc = Stackc;
        StackB = StackB;
        StackA = StackA;
    InstrAddr = ~Tst@TstSel ? Pc:(AluDst==Pc) ? AluOut:Literal;
        Pc = InstrAddr + (Execute & ~DbgMd)
     Fetch = DbgMd ? DbgAddr:InstrAddr;
   DbgAddr = DbgAddr + (Execute & DbgMd);}

else if (PgmCtrl == Jmp){                               //jump
        Stackc = Stackc;
        StackB = StackB;
        StackA = StackA;
    InstrAddr = (AluDst == Pc) ? AluOut:Literal;
        Pc = InstrAddr + (Execute & ~DbgMd)
     Fetch = DbgMd ? DbgAddr:InstrAddr;
   DbgAddr = DbgAddr + (Execute & DbgMd);}

else if (PgmCtrl == Jsr){                               //jump subroutine
        Stackc = StackB;
        StackB = StackA;
        StackA = Pc;
    InstrAddr = (AluDst == Pc) ? AluOut:Literal;
        Pc = InstrAddr + (Execute & ~DbgMd)
     Fetch = DbgMd ? DbgAddr:InstrAddr;
   DbgAddr = DbgAddr + (Execute & DbgMd);}

else if (FlgSel == Rts){                                //return subroutine
    InstrAddr = StackA;
      StackA = StackB;
      StackB = Stackc;
      Stackc = ErrVec;
        Pc = InstrAddr + (Execute & ~DbgMd)
     Fetch = DbgMd ? DbgAddr:InstrAddr;
   DbgAddr = DbgAddr + (Execute & DbgMd);}

else    {
    InstrAddr = Pc;                                     //continue
      StackA = StackA;
      StackB = StackB;
      Stackc = Stackc;
        Pc = InstrAddr + (Execute & ~DbgMd)
     Fetch = DbgMd ? DbgAddr:InstrAddr;
   DbgAddr = DbgAddr + (Execute & DbgMd);}
```

FIG. 52

ALU OPERATIONS

AluOp	OPERATION	
0b00000	**A** = (**A** & ~(1 << **B**)); **C** = 0; **V** = (**B** >= 32) ? 1:0;	//bit clear
0b00001	**A** = (**A** & **B**); **C** = 0; **V** = 0;	//logical and
0b00010	**A** = (**Literal** & **B**); **C** = 0; **V** = 0;	//logical and
0b00011	**A** = (~**Literal** & **B**); **C** = 0; **V** = 0;	//logical and not
0b00100	**A** = (**A** | (1 << **B**)); **C** = 0; **V** = (**B** >= 32) ? 1:0;	//bit set
0b00101	**A** = (**A** | **B**); **C** = 0; **V** = 0;	//logical or
0b00110	**A** = (**Literal** | **B**); **C** = 0; **V** = 0;	//logical or
0b00111	**A** = (~**Literal** | **B**); **C** = 0; **V** = 0;	//logical or not
0b01000	for (i=31; i>=0; i--) if **B**[i] continue; **A**=i; **C** = 0; **V** = (**B**) ? 0:1;	//priority enc
0b01001	**A** = (**A** ^ **B**); **C** = 0; **V** = 0;	//logical xor
0b01010	**A** = ({**Literal**} ^ **B**); **C** = 0; **V** = 0;	//logical xor
0b01011	**A** = ({~**Literal**} ^ **B**); **C** = 0; **V** = 0;	//logical xor not
0b01100	**A** = **B**; **C** = 0; **V** = 0;	//move
0b01101	**A** = **B**[31:24] ^ **B**[23:16] ^ **B**[15:08] ^ **B**[07:00]; **C** = 0; **V** = 0;	//hash
0b01110	**A** = {**B**[23:16],**B**[31:24],**B**[07:00],**B**[15:08]}; **C** = 0; **V** = 0;	//swap bytes
0b01111	**A** = {**B**[15:00], **B**[31:16]}; **C** = 0; **V** = 0;	//swap doublets

FIG. 53

AluOp	FUNCTION	
0b10000	**A** = (**A** + **B**); **C** = (**A** + **B**)[32]; **V** = 0;	//add **B**
0b10001	**A** = (**A** + **B** + **C**); **C** = (**A** + **B** + **C**)[32]; **V** = 0;	//add **B, carry**
0b10010	**A** = (**Literal** + **B**); **C** = (**Literal** + **B**)[32]; **V** = 0;	//add constant
0b10011	**A** = (-**Literal** + **B**); **C** = (-**Literal** + **B**)[32]; **V** = 0;	//sub constant
0b10100	**A** = (**A** - **B**); **C** = (**A** - **B**)[32]; **V** = 0;	//sub **B**
0b10101	**A** = (**A** - **B** - ~**C**); **C** = (**A** - **B** - ~**C**)[32]; **V** = 0;	//sub **B, borrow**
0b10110	**A** = (-**A** + **B**); **C** = (-**A** + **B**)[32]; **V** = 0;	//sub **A**
0b10111	**A** = (-**A** + **B** - ~**C**); **C** = (-**A** + **B** - ~**C**)[32]; **V** = 0;	//sub **A, borrow**
0b11000	**A** = (**A** << **B**); **C** = **A**[31]; **V** = (**B** >= 32) ? 0:1;	//shift left **A**
0b11001	**A** = (**B** << **Literal**); **C** = **B**[31]; **V** = (**Literal** >= 32) ? 0:1;	//shift left **B**
0b11010	**A** = (**B** << 1); **C** = **B**[31]; **V** = 0;	//shift left **B**
0b11011	**n** = (**A** - **B**); **C** = (**A** - **B**)[32]; **V** = 0;	//compare
0b11100	**A** = (**A** >> **B**); **C** = **A**[0]; **V** = (**B** >= 32) ? 1:0;	//shift right **A**
0b11101	**A** = (**B** >> **Literal**); **C** = **A**[0]; **V** = (**Literal** >= 32) ? 1:0;	//shift right **B**
0b11110	**A** = (**B** >> 1); **C** = **A**[0]; **V** = 0;	//shift right **B**
0b11111	**n** = (**B** - **A**); **C** = (**B** - **A**)[32]; **V** = 0;	//compare

FIG. 54

OpdSel	SELECTED OPERANDs	
0b0000aaaaa	**File**	**File@(OpdSel**[4:0] | **FileBase**); Allows paged access to any part of the register file.
0b0001aaaaa	**CpuReg**	**File@**{2'b11, **CpuId, OpdSel**[4:0]}; Allows direct access to Cpu specific registers.
0b001XXXXXX	**reserved**	Reserved for future expansion.
0b0100000XX	**CpuStatus**	0b0000000000000BHD00000000000000CC This is a read-only register providing information about the Cpu executing (**OpdSel**[1:0]) cycles after the current cycle. **"CC"** represents a value indicating the Cpu. Currently, only **CpuId** values of 0, 1 and 2 are returned. **"H"** represents the current state of **Hlt**, **"D"** indicates **DbgMd** and **"B"** indicates **BigMd**. Writing this register has no effect.
0b0100001XX	**reserved**	Reserved for future expansion.
0b0100010XX	**Pc**	0x0000AAAA Writing to this address causes the program control logic to use **AluOut** as the new **Pc** value in the event of a **Jmp**, **Jcc** or **Jsr** instruction for the Cpu executing during the current cycle. If the current instruction is **Nxt**, **Map**, or **Rts**, the register write has no effect. Reading this register returns the value in **Pc** for the Cpu executing (**OpdSel**[1:0]) cycles after the current cycle.
0b0100011XX	**DbgAddr**	0xD000AAAA Writing to this register alters the contents of the debug address register (**DbgAddr**) for the Cpu executing (**OpdSel**[1:0]) cycles after the current cycle. **DbgAddr** provides the fetch address for the control-store when **DbgMd** has been selected and the Cpu is executing. **DbgAddr** is also used as the control-store address when performing a **WrWcs@DbgAddr** or **RdWcs@DbgAddr** operation. "**D**" represents bit 31 of the register. It is a general purpose flag that is used for event indication during simulation. Reading this register returns a value of 0x00000000.
0b01001XXXX	**reserved**	Reserved for future expansion.

0b010100000 **RamAddr** {0b1CCC, 0x000, 0b1, AAAA}
RamAddr = **AluOut**[15] ? **AluOut** : (**AluOut** | **RamBase**);
PrevCC = **AluOut**[31] ? CCC : **AluCC**;

A read/write register. When reading this register, the Alu condition codes from the previous instruction are returned together with **RamAddr**.

bit	name	description
31		Always 1.
30	**PrevC**	Previous Alu Carry.
29	**PrevV**	Previous Alu Overflow.
28	**PrevZ**	Previous Alu Zero.
27:16		Always 0.
15		Always 1.
14:0	**RamAddr**	Contents of last Sram address used.

When writing this register, if alu_out[31] is set, the previous condition codes will be overwritten with bits 30:28 of **AluOut**. If **AluOut**[15] is set, bits 14:0 will be written to the **RamAddr**. If **AluOut** [15] is not set, bits 14:0 will be ored with the contents of the **RamBase** and written to the **RamAddr**

FIG. 55

OpdSel	SELECTED OPERANDs
0b010100001	AddrRegA 0x0000AAAA **AddrRegA** = **AluOut**; A read/write operand which loads **AddrRegA** used to provide the address for read and write operations. When **AddrRegA**[15] is set, the contents will be presented directly to the ram. When **AddrRegA**[15] is reset, the contents will first be ored with the contents of the **RamBase** register before presentation to the ram. Writing to this register takes priority over Literal loads using **FlgOp**. Reading this register returns the current value of the register.
0b010100010	**AddrRegB** 0x0000AAAA **AddrRegB** = **AluOut**; A read/write operand which loads **AddrRegB** used to provide the address for read and write operations. When **AddrRegB**[15] is set, the contents will be presented directly to the ram. When **AddrRegB**[15] is reset, the contents will first be ored with the contents of the **RamBase** register before presentation to the ram. Writing to this register takes priority over Literal loads using **FlgOp**. Reading this register returns the current value of the register.
0b010100011	**AddrRegAb** 0x0000AAAA **AddrRegA** = **AluOut**; **AddrRegB** = **AluOut**; A destination only operand which loads **AddrRegB** and **AddrRegA** used to provide the address for read and write operations Writing to this register takes priority over Literal loads using **FlgOp**. Reading this register returns the value 0x00000000.
0b010100100	**RamBase** 0x0000AAAA **RamBase** = **AluOut**; A read/write register which provides the base address for ram read and write cycles. When **RamAddr**[15] is set, the contents will not be used. When **RamAddr**[15] is reset, the contents will first be ored with the contents of the **RamBase** register before presentation to the ram. Reading this register returns the value for the current Cpu.
0b010100101	**FileBase** 0b00000000000000000000000AAAAAAAAA **FileBase** = **AluOut**; **FileAddr** = **OpdSel[8]** ? **OpdSel**:(**OpdSel** + **FileBase**); A read/write register which provides the base address for **file** read and write cycles. When **OpdSel[8]** is set, the contents will not be used and **OpdSel** will be presented directly to the address lines of the file. When **OpdSel**[8] is reset, the contents will first be ored with the contents of the **FileBase** register before presentation to the file. Reading this register returns the value for the current Cpu.
0b010100110	**InstrRegL** 0xIIIIIIII This is a read-only register which returns the contents of **InstrReg**[31:0]. Writing to this register has no effect.
0b010100111	**InstrRegH** 0x00IIIIII This is a read-only register which returns the contents of **InstrReg**[55:32]. Writing to this register has no effect.

FIG. 56

OpdSel	SELECTED OPERANDs	
0b010101000	**Minus1**	0xffffffff This is a read-only register which supplies a value 0xffffffff.. Writing to this register has no effect.
0b010101001	**FreeTime**	A free-running timer with a resolution of 1.00 microseconds and a maximum count of 71 minutes. This timer is cleared during reset.
0b010101010	**LiteralL**	**Instr**[15:0] A read-only register. Writing to this register has no effect
0b010101011	**LiteralH**	**Instr**[15:0]<<16; A read-only register. Writing to this register has no effect

0b010101100 **MacData** - Writing to this address loads the **AluOut** data into the **MacData** register for use during Mac operations. The Mac operation, resulting from writing to the **MacOp** register, determines the definition of the **MacData** register contents as follows.

MacOp	MacData definition
Mstop	0bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX **MacData** is not used for the **StopM** operation.
WrMcfg	**hrstl**, rsvd, rsvd, **crcen, fulld, hrstl, hugen, nopre, paden, prtyl, xdl10, ipgr1[6:0],** **ipgr2[6:0], ipgt[6:0].** Loads the **MacCfg** register with the contents of the **MacData** register. Refer to LSI Logic's *Ethernet-110 Core Technical Manual* for detailed definitions of these bits.
WrMrng	0bXXXXXXXXXXXXXXXXXXXXXXSSSSSSSSSS Loads **seed**[10:0] into the Mac's random number generator.
RdPhy	0bXXXXRRRRXXXXPPPPXXXXXXXXXXXXXXXX Reads register[R] of phy[P].
WrPhy	0bXXXXRRRRXXXXPPPPDDDDDDDDDDDDDDDD Writes register[R] of phy[P] with **MacData**[15:0].

Reading this register returns **prsd[15:0]** of **Mac0** which contains phy status data returned to the Mac at the completion of a **RdPhy** command. This data is invalid while **MacBsy** is asserted as a result of a **RdPhy** command. Refer to the appropriate phy technical manual for a definition of the phy register contents.

FIG. 57

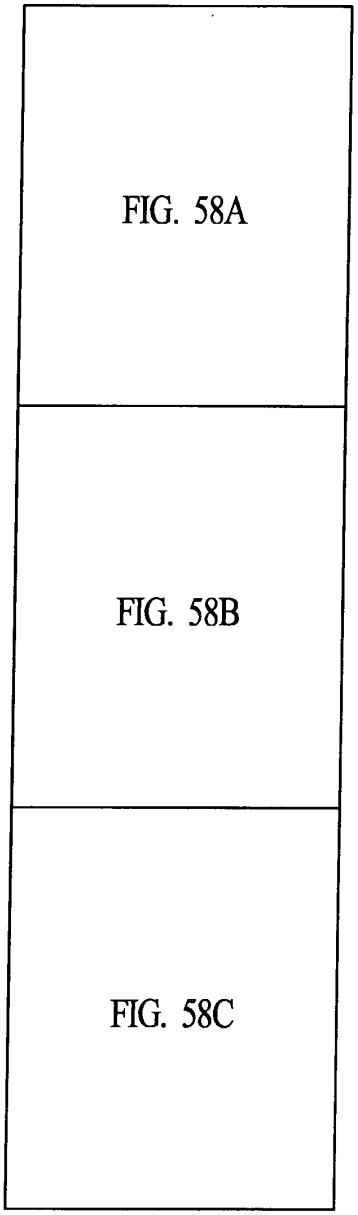


FIG. 58

OpdSel	SELECTED OPERANDs

0b010101101 **MacOp** - A write only register. Writing to this address loads the **MacSel** register and staRts execution of the specified operation as follows.

AluOut	description
0xXXXXX0XM	**Mstop** - Halts execution of a **MacOp** for **Mac**[M]. The user must wait for **MacBsy** to be deasserted before issuing another command or changing the contents of **MacData**.
0xXXXXX1XM	**WrMcfg** - Writes the contents of **MacData** to the **MacCfg** register of **Mac**[M]. The user must wait for **MacBsy** to be deasserted before issuing another command or changing the contents of **MacData**.
0xXXXXX2XM	**WrMrng** - Writes the contents of **MacData** to the **seed** register of **Mac**[M]. The user must wait for **MacBsy** to be deasserted before issuing another command or changing the contents of **MacData**.
0xXXXXX3XM	**RdPhy** - Reads the contents of reg[R] for phy[P] on the MII management bus of Mac[M]. The contents may be read from **MacData** after **MacBsy** has been de-asserted.
0xXXXXX4XM	**WrPhy** - Writes the contents of **MacData**[15:0] to e reg[R] of phy[P] on the MII management bus of **Mac**[M]. The user must wait for **MacBsy** to be deasserted before issuing another command or changing the contents of **MacData**.
0xXXXXX8XM	**WrAddrAL** - Writes the contents of **MacData**[15:0] to **MacAddrA[15:0]** for **Mac**[M].
0xXXXXX9XM	**WrAddrAH** - Writes the contents of **MacData**[11:0] to **MacAddrA[47:16]** for **Mac**[M].
0xXXXXXaXM	**WrAddrBL** - Writes the contents of **MacData**[15:0] to **MacAddrB[15:0]** for **Mac**[M].
0xXXXXXbXM	**WrAddrBH** - Writes the contents of **MacData**[11:0] to **MacAddrB[47:16]** for **Mac**[M].

b010101110 **ChCmd** A write-only register.

bit	name	description
31:11	reserved	Data written to these bits is ignored.
10:8	**command**	0 - Stops execution of the current operation and clears the corresponding event flag. 1 - Transfer data from ExtMem to ExtMem. 2 - Transfer data from Pci to ExtMem. 3 - Transfer data from ExtMem to Pci. 4 - Transfer data from Sram to ExtMem. 5 - Transfer data from ExtMem to Sram. 6 - Transfer data from Pci to Sram. 7 - Transfer data from Sram to Pci.
07:05	reserved	Data written to these bits is ignored.
04:00	**ChId**	Provides the channel number for the channel command.

FIG. 58A

0b010101110 **ChEvnt** A read-only register.

bit	name	description
31:00	**ChDn**	Each bit represents the done flag for the respective dma channel. These bits are set by a dma sequencer upon completion of the channel command. Cleared when the processor writes 0 to the corresponding **ChCmd** register.

0b010101111 **GenEvnt** A read-only register.

bit	name	description
31	**PciRdEvnt**	Indicates that a PCI initiator is attempting to read a mproc. register.
30	**PciWrEvnt**	Indicates that a PCI initiator has posted a write to a mproc. register.
29	**TimeEvnt**	An event which occurs once every 2.00 milliseconds.
28:00	**reserved**	Reserved for future use.

0b010110000 **QCtrl** A write-only register used to select and manipulate a **Q**.

bit	name	description
31:11	reserved	Data written to these bits are ignored.
10:8	**QSz**	Used only during **InitQ** operations to specify the size of the **QBdy** in Dram. 7 – Queue depth is 32K entries (128KB). 6 – Queue depth is 16K entries (64KB). 5 – Queue depth is 8K entries (32KB). 4 – Queue depth is 4K entries (16KB). 3 – Queue depth is 2K entries (8KB). 2 – Queue depth is 1K entries (4KB). 1 – Queue depth is 512 entries (2KB). 0 – Queue depth is 256 entries (1KB).
7:5	**QOp**	Specifies the queue operation to perform. 7 – **DblQ** Disables all queues. 6 – **EnQ** Enables all queues. 5 – **RdBdy** Increments the **QBdyRdPtr** and increments the **QTlWrPtr**. 4 – **WrBdy** Decrements the **QBdyWrPtr** and increments the **QHdRdPtr**. 3 – **RdQ** Returns a queue entry in register **QData**.. 2 – **rsvd** Reserved. Not to be used. 1 – **InitQ** Set the queue status to empty and initializes **QSz**. 0 – **SelQ** Selects the **QId** to be utilized during writes to **QData**.

FIG. 58B

4:0	**QId**	Specifies the queue on which to perform all operations except **DblQ** or **EnQ**.
0b010110001	**QData**	A read/write register. Writing this register will result in the data being pushed on to the selected queue. Reading this register fetches queue data popped off during the previous **RdQ** operation.
0b010110010	**reserved**	Reserved for future expansion.
0b010110011	**XcvCtrl**	A write-only register used to enable and disable Mac transmit and receive sub-channels.

bit	name	description
31:09	reserved	Data written to these bits are ignored.
8	**enable**	When set, indicates to the Mac transmit or receive sequencer that the subchannel contains a transmit or receive descriptor.
07:05	reserved	Data written to these bits is ignored.
04	**RcvCh**	Selects a Mac receive subchannel when set. Selects a Mac transmit subchannel when cleared.
03	reserved	Data written to this bit are ignored.
02	**SubCh**	Selects subchannel B when set or A when reset.
01:00	**MacId**	Provides the Mac number for the subchannel enable bit.

0b010110100	**Lru**	0x0000000A A read/write operand indicating which of the 16 entries is least recently used. When Reading This register the least recently used entry is returned, after which it is automatically made the most recently used entry. This register should only be read in conjunction with a 'Move' operation of the ALU, else the results are unpredictable. Writing to this register forces the addressed entry to become the least recently used entry.
0b010110101	**Mru**	0x0000000A A write only operand forcing the addressed entry to become the most recently used entry.
0b010111000	**QInRdy**	A read-only register comprising **QHd** not full flags for each of the 32 queues.
0b010111001	**QOutRdy**	A read-only register comprising **QTl** not empty flags for each of the 32 queues.
0b010111010	**QEmpty**	A read-only register comprising **QEmpty** flags for each of the 32 queues.
0b010111011	**QFull**	A read-only register comprising **QFull** flags for each of the 32 queues.
0b0101111XX	**reserved**	Reserved for future expansion.
0b0110XXXXX	**Constants**	{0b000, **OpdSel**[4:0]}
0b01110XXXX	**reserved**	Reserved for future expansion.

FIG. 58C

OpdSel **SELECTED OPERANDs**

0b01111XXXX **Sram OPERATIONS**

OpdSel[3]	PostAddrOp
0	**nop**
1	**RamAddr = RamAddr + (OpdSel[1:0]);**

OpdSel[2]	transpose_Ctrl
0	don't transpose
1	transpose bytes

OpdSel[1:0]	RamOpdSz
0	quadlet
1	triplet
2	doublet
3	byte

RAM READ ATTRIBUTES				SOURCE_OPERAND			
endian mode	trans-pose	byte offs	Sram data	sz=Q	sz=T	sz=D	sz=B
little	0	0	abcd	abcd	0bcd	00cd	000d
little	0	1	abcX	trap	0abc	00bc	000c
little	0	2	abXX	trap	trap	00ab	000b
little	0	3	aXXX	trap	trap	trap	000a
little	1	0	abcd	dcba	0dcb	00dc	000d
little	1	1	abcX	trap	0cba	00cb	000c
little	1	2	abXX	trap	trap	00ba	000b
little	1	3	aXXX	trap	trap	trap	000a
BIG	0	0	abcd	abcd	0abc	00ab	000a
BIG	0	1	Xbcd	trap	0bcd	00bc	000b
BIG	0	2	XXcd	trap	trap	00cd	000c
BIG	0	3	XXXd	trap	trap	trap	000d
BIG	1	0	abcd	dcba	0cba	00ba	000a
BIG	1	1	Xbcd	trap	0dcb	00cb	000b
BIG	1	2	XXcd	trap	trap	00dc	000c
BIG	1	3	XXXd	trap	trap	trap	000d

RAM WRITE ATTRIBUTES			SOURCE_OPERAND				
endian mode	trans-pose	Opd size	Alu out	OF=0	0F=1	OF=2	OF=3
little	0	Q	abcd	abcd	trap	trap	trap
little	0	T	Xbcd	-bcd	bcd-	trap	trap
little	0	D	XXcd	--cd	-cd-	cd--	trap
little	0	B	XXXd	---d	--d-	-d--	d---
little	1	Q	abcd	dcba	trap	trap	trap
little	1	T	Xbcd	-dcb	dcb-	trap	trap
little	1	D	XXcd	--dc	-dc-	dc--	trap
little	1	B	XXXd	---d	--d-	-d--	d---
big	0	Q	abcd	abcd	trap	trap	trap
big	0	T	Xbcd	bcd-	-bcd	trap	trap
big	0	D	XXcd	cd--	-cd-	--cd	trap
big	0	B	XXXd	d---	-d--	--d-	---d
big	1	Q	abcd	dcba	trap	trap	trap
big	1	T	Xbcd	dcb-	-dcb	trap	trap
big	1	D	XXcd	dc--	-dc-	--dc	trap
big	1	B	XXXd	d---	-d--	--d-	---d

0b1aaaaaaaa **File** **File@OpdSel**[8:0];
Allows direct, non-paged, access to the top half of the register file.

FIG. 59

TstSel	SELECTED TEST	
0bX00XXXXX	**Tst** = **TstSel**[7] ^ **AluOut**[**TstSel**[4:0]]	//Alu bit
0bX0100000	**Tst** = **TstSel**[7] ^ **C**	//carry
0bX0100001	**Tst** = **TstSel**[7] ^ **V**	//error
0bX0100010	**Tst** = **TstSel**[7] ^ **Z**	//zero
0bX0100011	**Tst** = **TstSel**[7] ^ (**Z** | ~**C**)	//less or equal
0bX0100100	**Tst** = **TstSel**[7] ^ **PrevC**	//previous carry
0bX0100101	**Tst** = **TstSel**[7] ^ **PrevV**	//previous error
0bX0100110	**Tst** = **TstSel**[7] ^ **PrevZ**	//previous zero
0bX0100111	**Tst** = **TstSel**[7] ^ (**PrevZ** & **Z**)	//64b zero
0bX0101000	**Tst** = **TstSel**[7] ^ **QOpDn**	//queue op okay
0bX0101001	**Tst** = reserved	
0bX010101X	**Tst** = reserved	
0bX01011XX	**Tst** = reserved	
0bX0110XXX	**Tst** = **TstSel**[7] ^ **Lock**[**TstSel**[2:0]] **Lock**(**TstSel**[2:0]) = 1;	//tests the current value of //the Lock then set it.
0bX0111XXX	**Tst** = **TstSel**[7] ^ **Lock**[**TstSel**[2:0]]	//tests the value of Lock.
0bX01XXXXX	**Tst** = reserved	
0bX1XXXXXX	**Tst** = reserved	

FIG. 60

[redacted]

FlgSel	FLAG OPERATION	
0b00000000	No operation.	
0b00000001	**SelfRst**	Forces a self reset for the entire chip excluding the PCI configuration registers
0b00000010	**SelBigEnd**	Selects big-endian mode for ram accesses for the current Cpu.
0b00000011	**SelLitEnd**	Selects little-endian mode for ram accesses for the current Cpu.
0b00000100	**DblMap**	Disable instruction re-mapping for the current Cpu.
0b00000101	**EnbMap**	Enable instruction re-mapping for the current Cpu.
0b0000011X	reserved	
0b0000IXXX	reserved	
0b00010XXX	**ClrLck**	**Lock[FlgSel**[2:0]] = 0; Clears the semaphore register bit for the current Cpu only.
0b00011XXX	reserved	
0b0010XXXX	**AddrOp**	

FlgSel[3 2]	AddrSelect	
0	**RamAddr = Literal**[15]	? **Literal** (**Literal | RamBase**);
1	**RamAddr = AddrRegA**[15]	? **AddrRegA** · (**AddrRegA | RamBase**),
2	**RamAddr = AddrRegB**[15]	? **AddrRegB** (**AddrRegB | RamBase**);
3	if (**OpdA == RamAddr**) **RamAddr = AluOut**[15]	? **AluOut** . (**AluOut | RamBase**);
	else if (**OpdA == ram**) **RamAddr = AddrRegB**[15]	? **AddrRegB** · (**AddrRegB | RamBase**);
	else **RamAddr = AddrRegA**[15]	? **AddrRegA** . (**AddrRegA | RamBase**);

FlgSel[1 0]	addr reg load	
0	**nop**	
1	**AddrRegA = Literal**,	
2	**AddrRegB = Literal**,	
3	**AddrRegA = Literal**;	**AddrRegB = Literal**,

note: When specifying the same register for both the load and select fields, the current value of the register, before it is loaded with the new value, will be used for the ram address.

0b0011XXXX	reserved	
0b01000000	**WrWcsL@Dbg**	Causes the bits [31:0] of the control-store at address **DbgAddr** to be written with the current **AluOut** data.
0b01000001	**WrWcsH@Dbg**	Causes the bits [63:32] of the control-store at address **DbgAddr** to be written with the current **AluOut** data then increments **DbgAddr**.
0b01000010	**RdWcsL@Dbg**	Causes the bits [31:0] of the control-store at address **DbgAddr** to be moved to file address 0x1ff.
0b01000011	**RdWcsH@Dbg**	Causes the bits [63:32] of the control-store at address **DbgAddr** to be moved to file address 0x1ff then increments **DbgAddr**.
0b01000100	reserved	
0b010001XX	**Step**	Allows the Cpu (**FlgSel**[1:0]) cycles after the current cycle to execute a single instruction. There is no effect if the Cpu is not halted. An offset of 0 is not allowed.
0b010010XX	**PcMd**	Selects the **Pc** as the address source for the control-store during instruction fetches for the Cpu (**FlgSel**[1:0]) cycles after the current cycle.
0b010011XX	**DbgMd**	Selects the **DbgAddr** address register as the address source for the control-store during instruction fetches for the Cpu (**FlgSel**[1:0]) cycles after the current cycle.
0b010100XX	**Hlt**	Halts the Cpu (**FlgSel**[1:0]) cycles after the current cycle.
0b010101XX	**Run**	Clears Halt for the Cpu (**FlgSel**[1:0]) cycles after the current cycle.
0b01011XXX	reserved	
0b011XXXXX	reserved	
0b1XXXXXXX	reserved	

FIG. 61

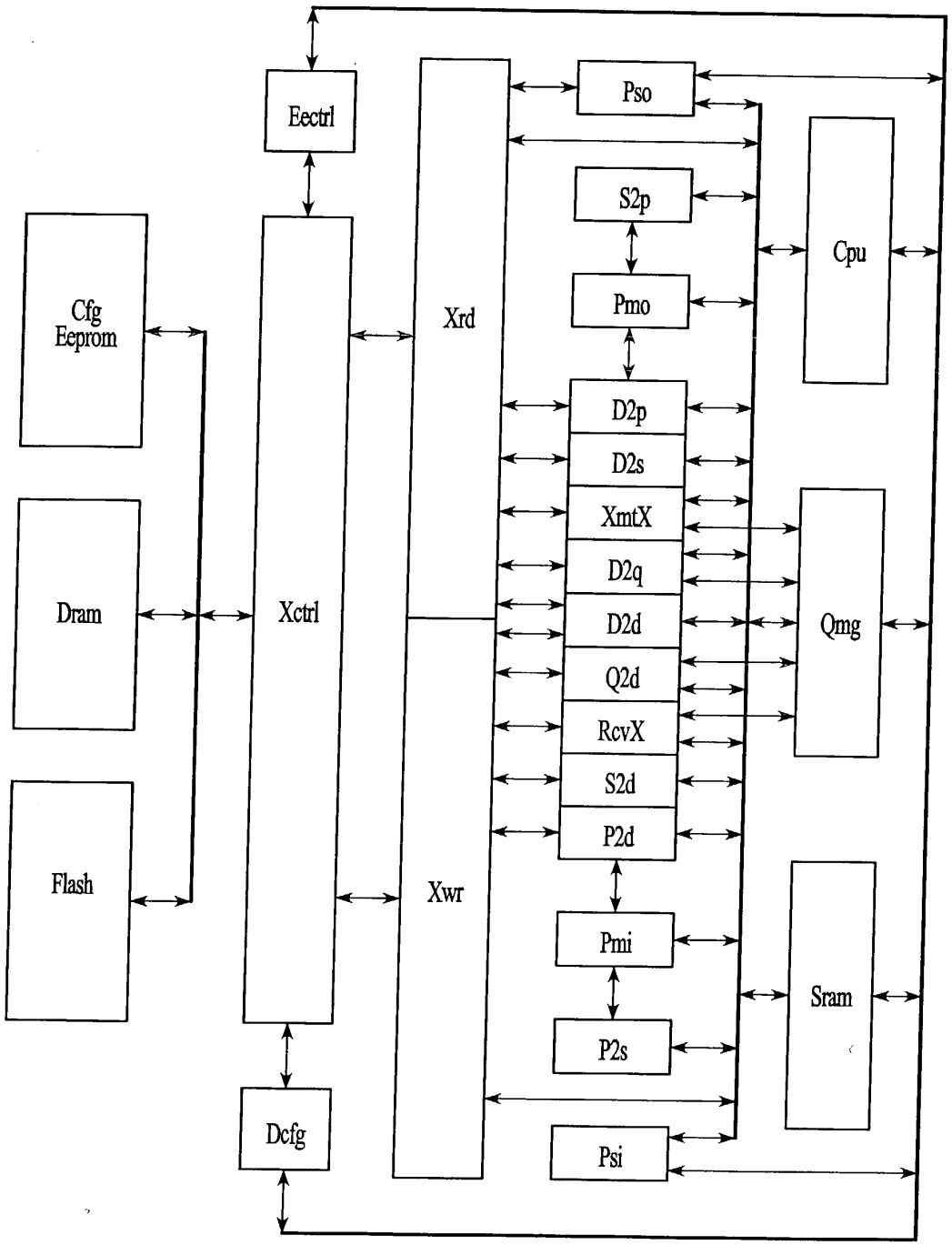

Eectrl
Pso
S2p
Cpu
Cfg
Eeprom
Pmo
Xrd
D2p
D2s
XmtX
D2q
Dram
Xctrl
D2d
Qmg
Q2d
RcvX
S2d
Flash
P2d
Xwr
Pmi
Sram
P2s
Dcfg
Psi

FIG. 62

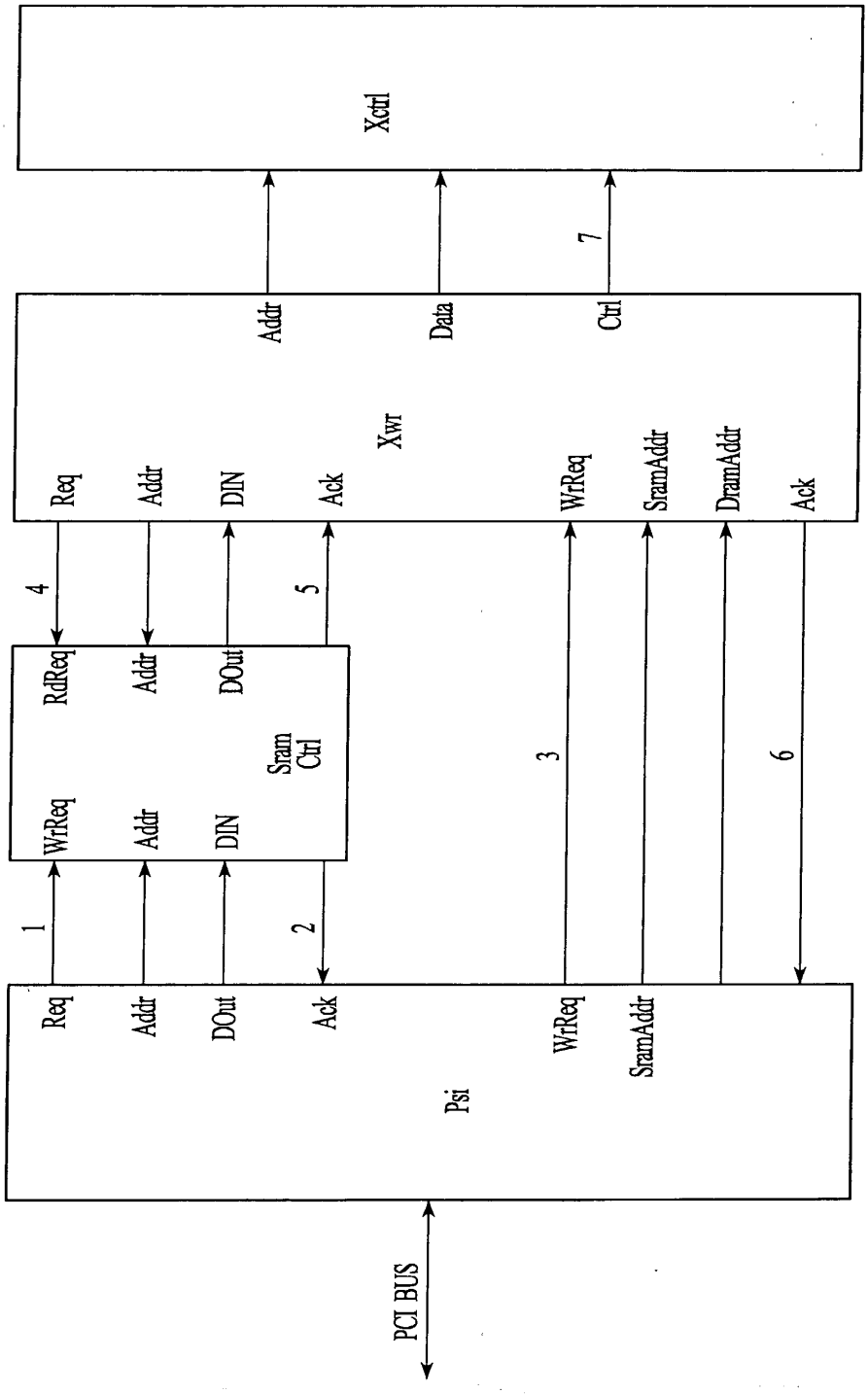

Xctrl
Addr
Data
Ctrl
7
Xwr
Req
Addr
DIN
Ack
WrReq
SramAddr
DramAddr
Ack
4
5
RdReq
Addr
DOut
Sram Ctrl
WrReq
Addr
DIN
3
6
1
2
Req
Addr
DOut
Ack
WrReq
SramAddr
Psi
PCI BUS

FIG. 63

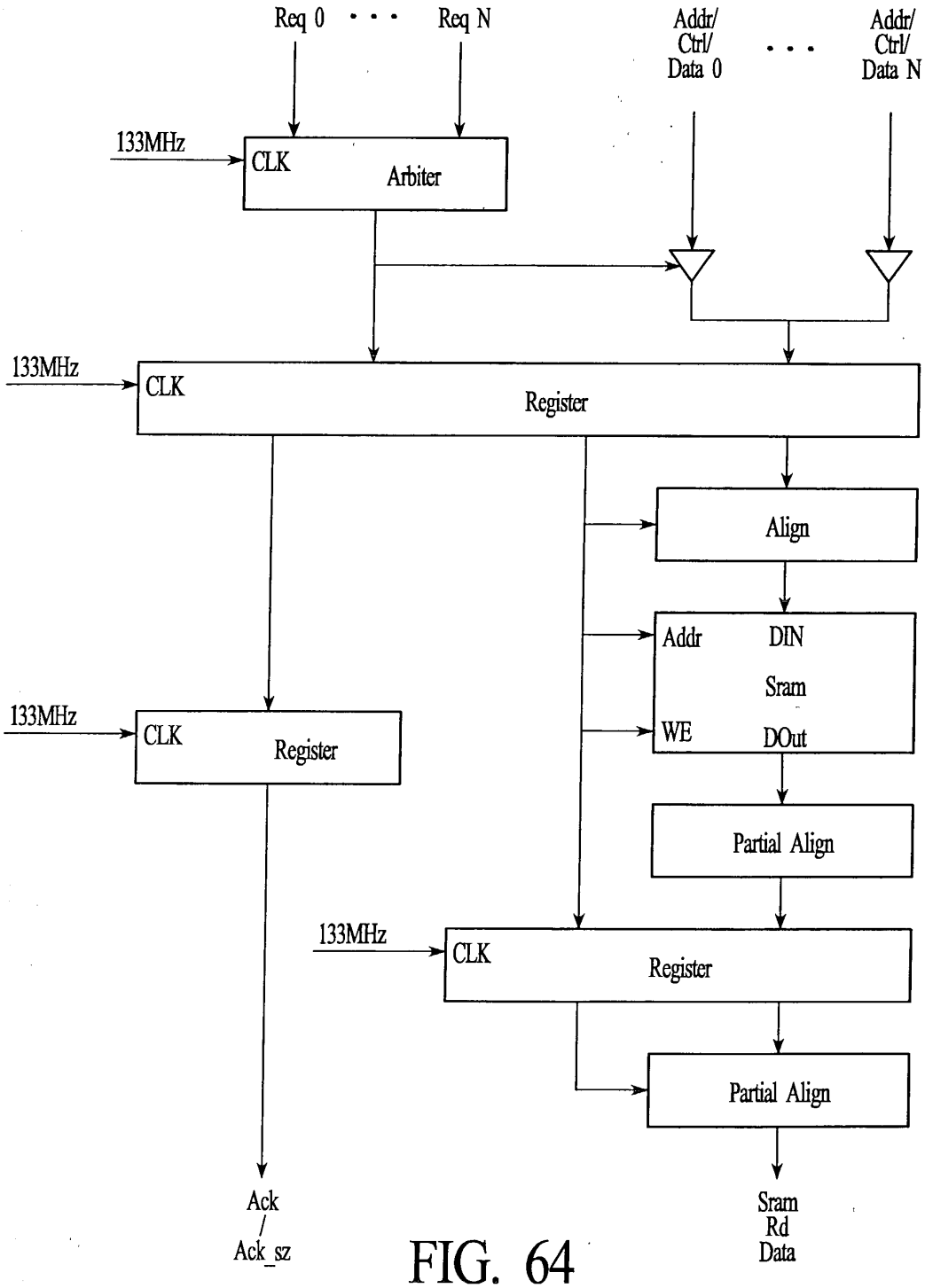

Req 0
Req N
Addr/
Ctrl/
Data 0
Addr/
Ctrl/
Data N
133MHz
CLK
Arbiter
133MHz
CLK
Register
Align
Addr
DIN
Sram
WE
DOut
133MHz
CLK
Register
Partial Align
133MHz
CLK
Register
Partial Align
Ack
/
Ack_sz
Sram
Rd
Data

FIG. 64

Cpu CLOCK
Dma CLOCK
DMA Req
valid
valid
DMA Gnt
Sram CLOCK
1st CpuParams SELECTED
GRANT TO 1st DMA SEQUENCER
1st DMAParams SELECTED
2nd CpuParams SELECTED
GRANT TO 2nd DMA SEQUENCER
2nd DMAParams SELECTED
Ack AND Ack SIZE FOR Cpu
Ack AND Ack SIZE FOR 1st DMA Req
Ack AND Ack SIZE FOR Cpu
APPLY 1st Cpu Addr
ALIGN 1st Cpu INPUT Data
READ OR WRITE 1st Cpu Data
APPLY 1st DMA Addr
ALIGN 1st DMA INPUT Data
READ OR WRITE 1st DMA Data
APPLY 2nd Cpu Addr
ALIGN 2nd Cpu INPUT Data
READ OR WRITE 2nd Cpu Data
APPLY 2nd DMA Addr
ALIGN 2nd DMA INPUT Data
READ OR WRITE 2nd DMA Data
ALIGN AND RETURN 1st Cpu Data
ALIGN AND RETURN 1st DMA Data
ALIGN AND RETURN 2nd Cpu Data

FIG. 65

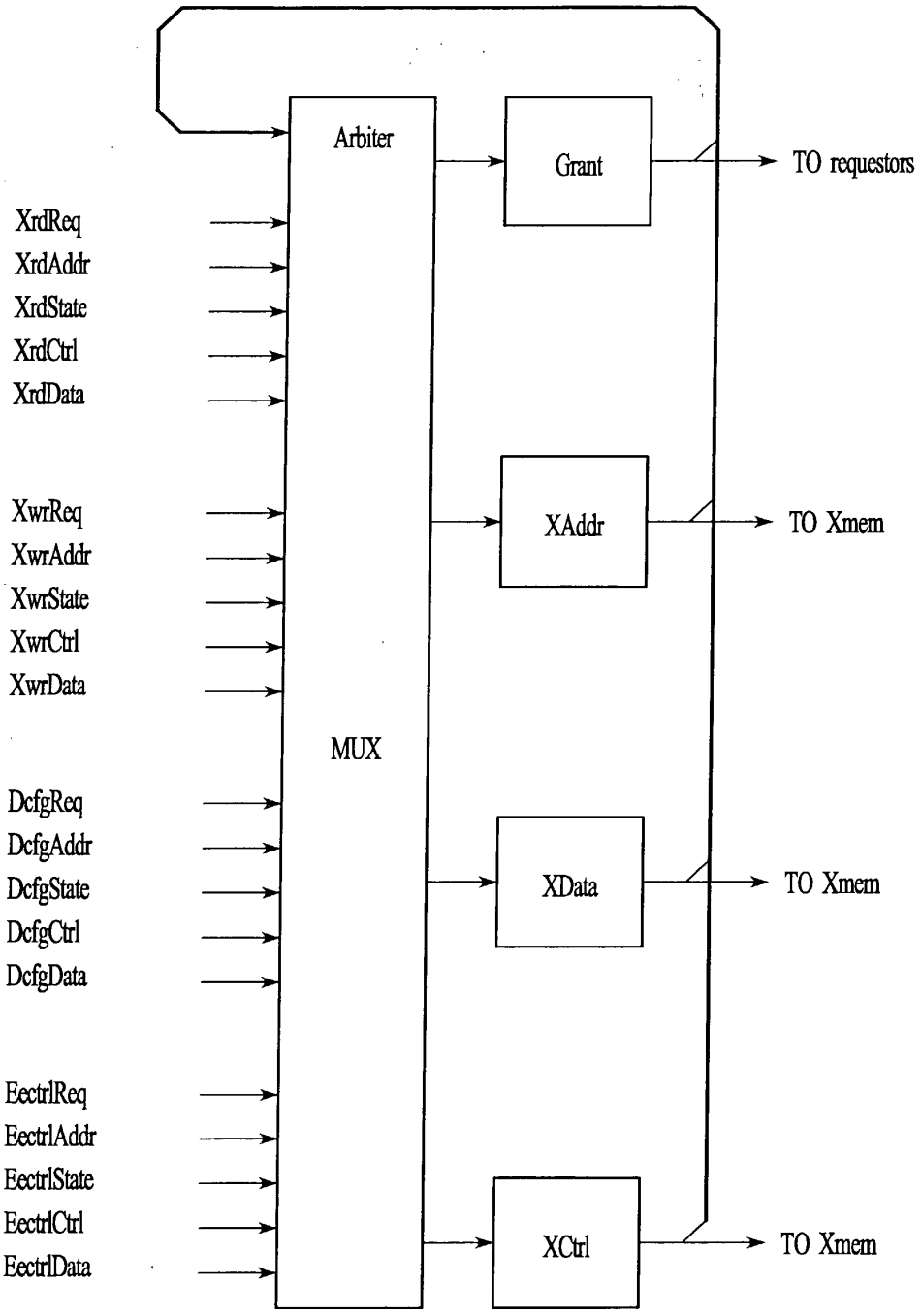

Arbiter
Grant
TO requestors
XrdReq
XrdAddr
XrdState
XrdCtrl
XrdData
XwrReq
XwrAddr
XwrState
XwrCtrl
XwrData
XAddr
TO Xmem
MUX
DcfgReq
DcfgAddr
DcfgState
DcfgCtrl
DcfgData
XData
TO Xmem
EectrlReq
EectrlAddr
EectrlState
EectrlCtrl
EectrlData
XCtrl
TO Xmem

FIG. 66

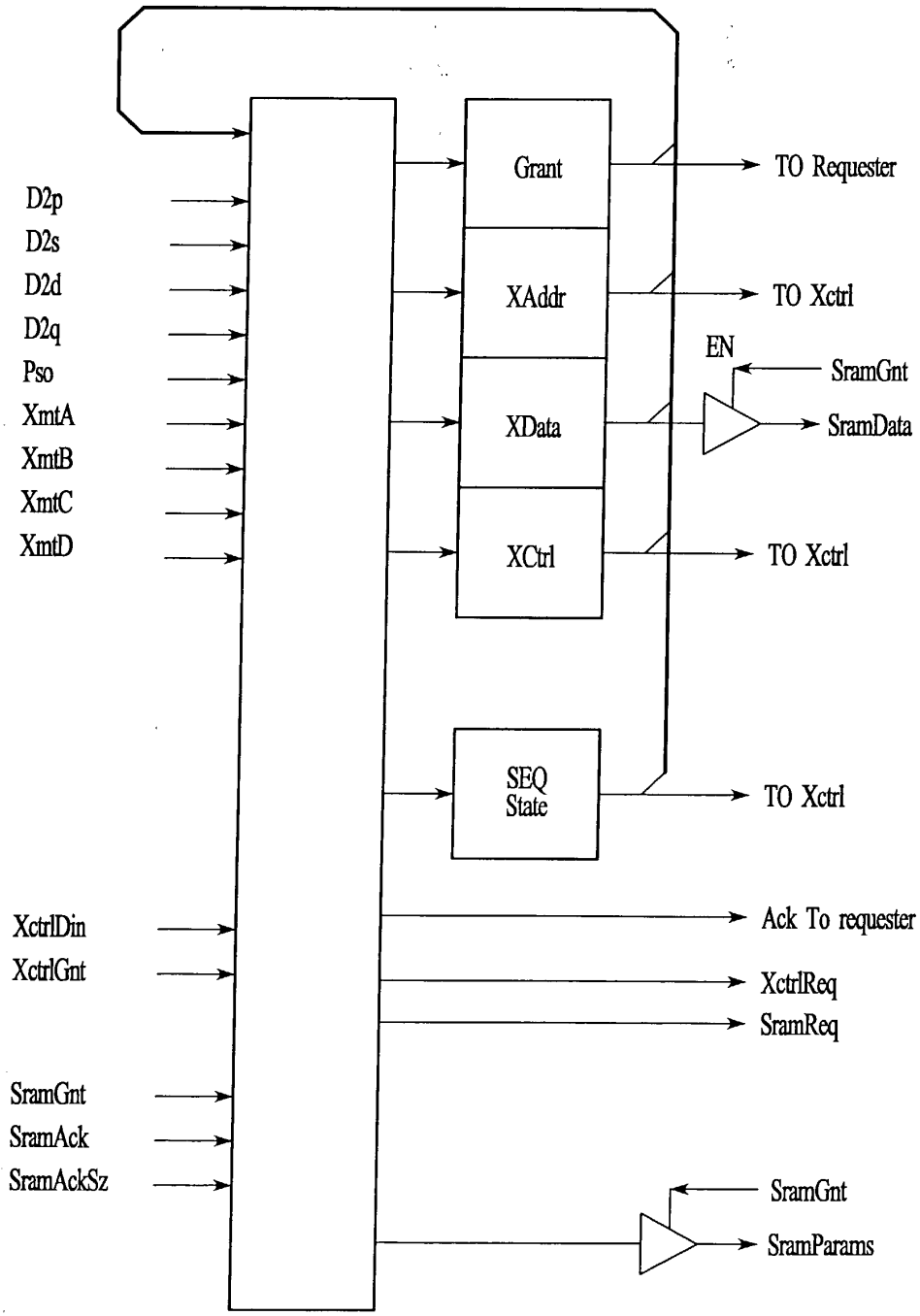

Grant
TO Requester
D2p
D2s
D2d
D2q
Pso
XmtA
XmtB
XmtC
XmtD
XAddr
TO Xctrl
EN
SramGnt
XData
SramData
XCtrl
TO Xctrl
SEQ State
TO Xctrl
XctrlDin
XctrlGnt
Ack To requester
XctrlReq
SramReq
SramGnt
SramAck
SramAckSz
SramGnt
SramParams

FIG. 67

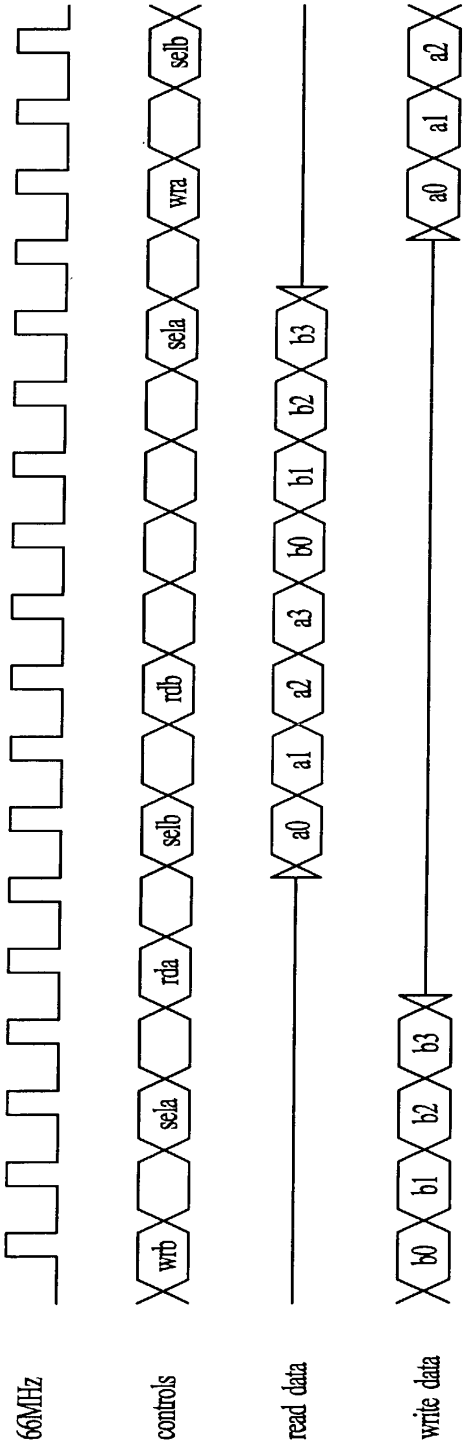

66MHz
controls
read data
write data
wrb
sela
rda
selb
rdb
selb
sela
wra
selb
a0
a1
a2
a3
b0
b1
b2
b3
b0
b1
b2
b3
a0
a1
a2

FIG. 68

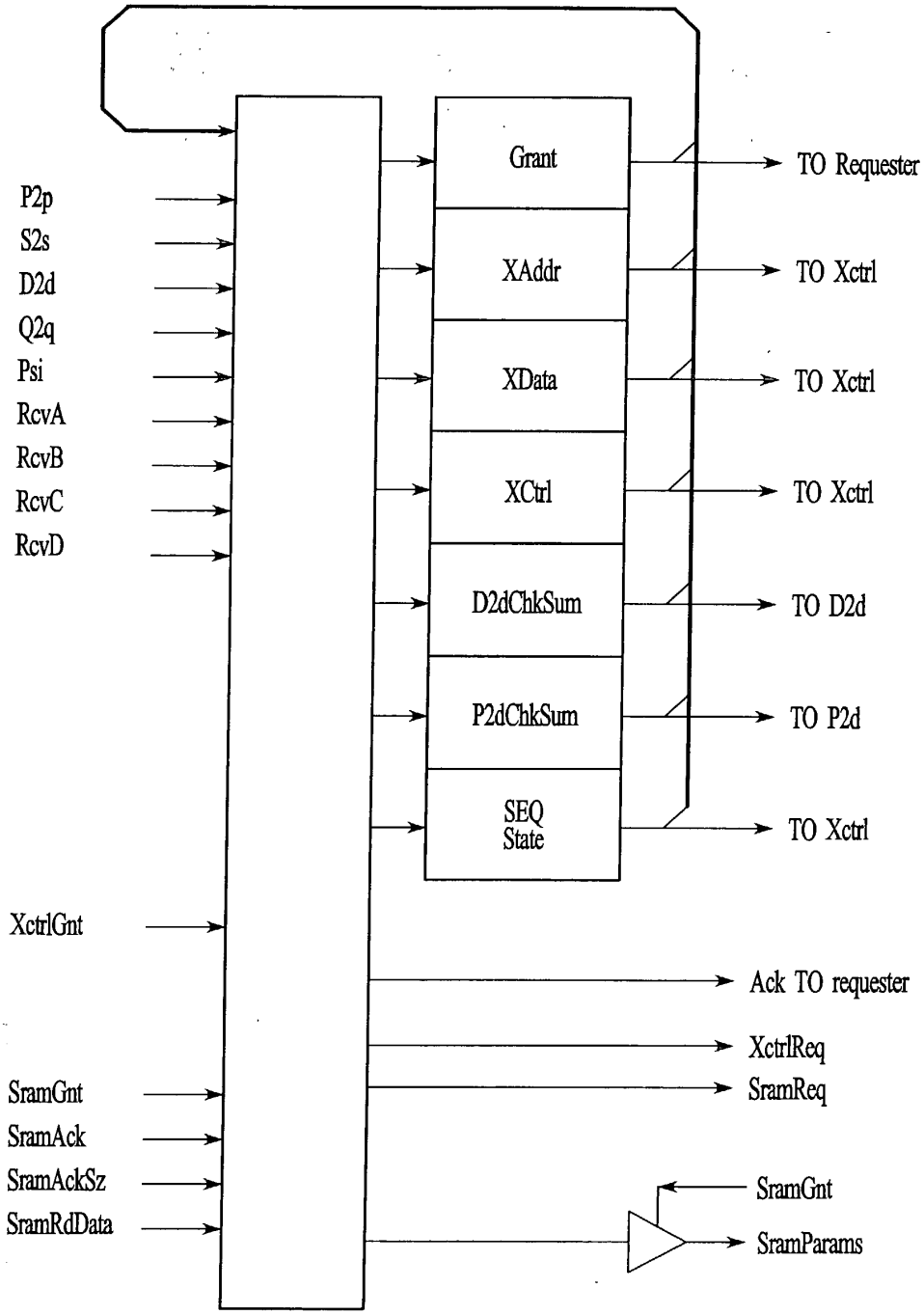

Grant
XAddr
XData
XCtrl
D2dChkSum
P2dChkSum
SEQ State
TO Requester
TO Xctrl
TO Xctrl
TO Xctrl
TO D2d
TO P2d
TO Xctrl
P2p
S2s
D2d
Q2q
Psi
RcvA
RcvB
RcvC
RcvD
XctrlGnt
SramGnt
SramAck
SramAckSz
SramRdData
Ack TO requester
XctrlReq
SramReq
SramGnt
SramParams

FIG. 69

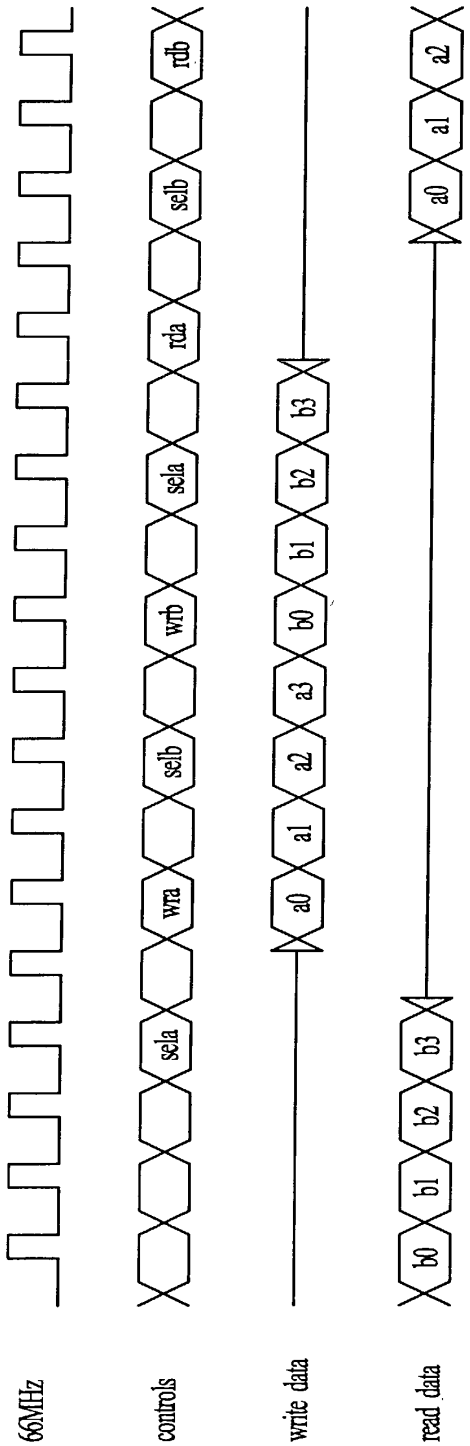

66MHz
controls
write data
read data
sela
wra
selb
wrb
sela
rda
selb
rdb
a0
a1
a2
a3
b0
b1
b2
b3
b0
b1
b2
b3
a0
a1
a2

FIG. 70

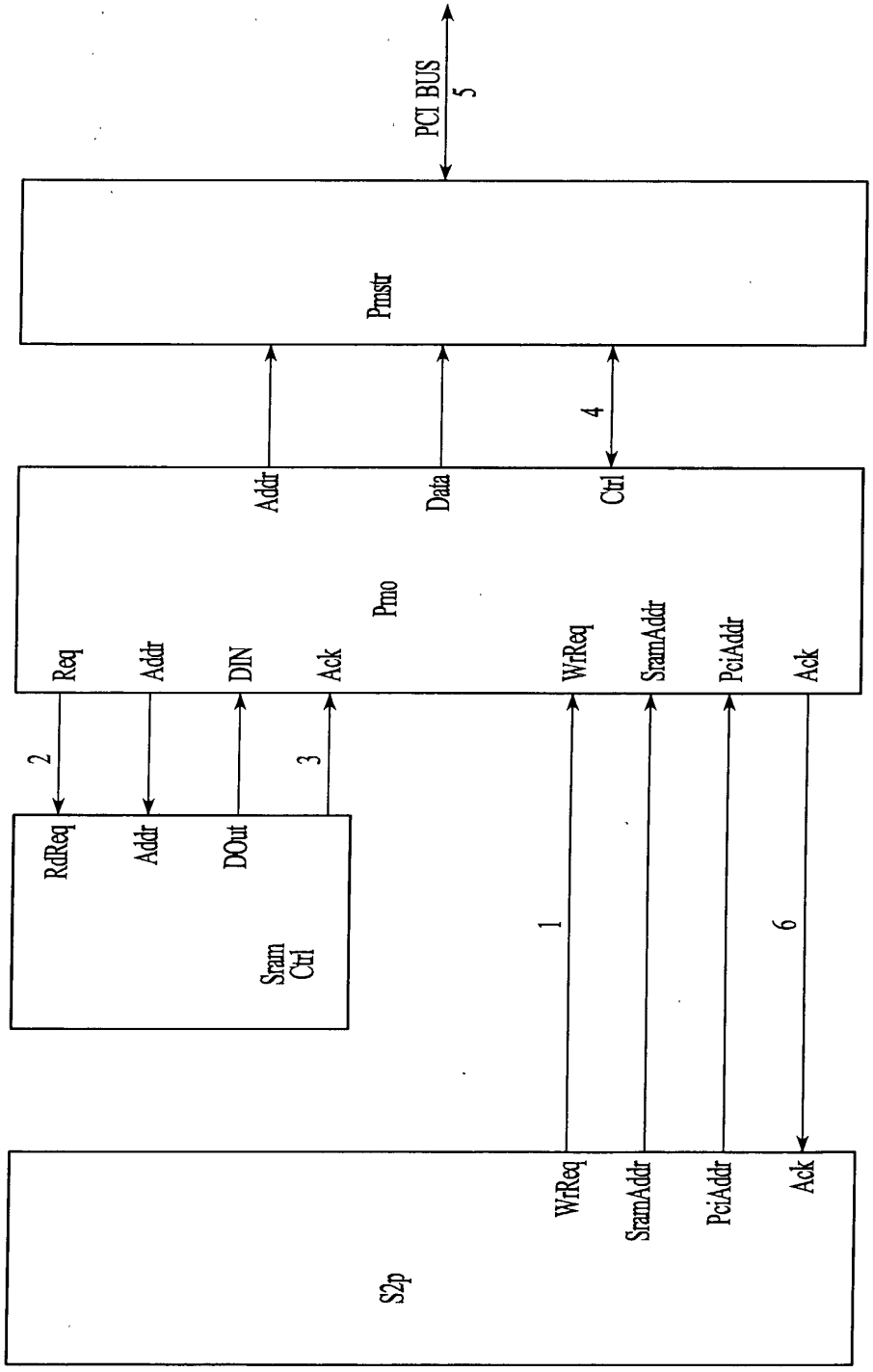

PCI BUS
5
Pmstr
4
Addr
Data
Ctrl
Pmo
Req
Addr
DIN
Ack
WrReq
SramAddr
PciAddr
Ack
2
3
RdReq
Addr
DOut
Sram
Ctrl
1
6
S2p
WrReq
SramAddr
PciAddr
Ack

FIG. 71

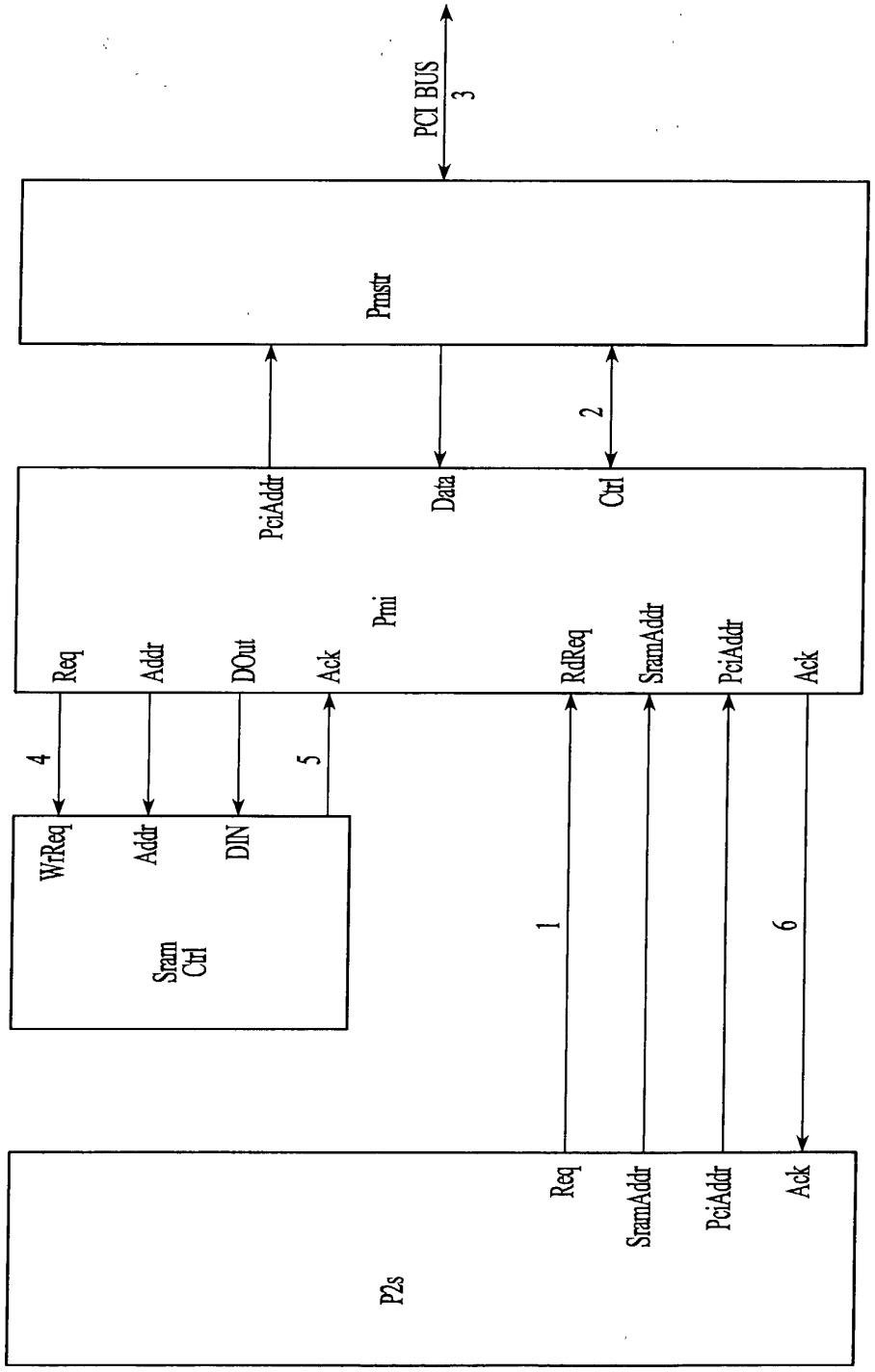

PCI BUS
3
Pmstr
2
PciAddr
Data
Ctrl
Pmi
Req
Addr
DOut
Ack
RdReq
SramAddr
PciAddr
Ack
4
5
WrReq
Addr
DIN
Sram
Ctrl
1
6
Req
SramAddr
PciAddr
Ack
P2s

FIG. 72

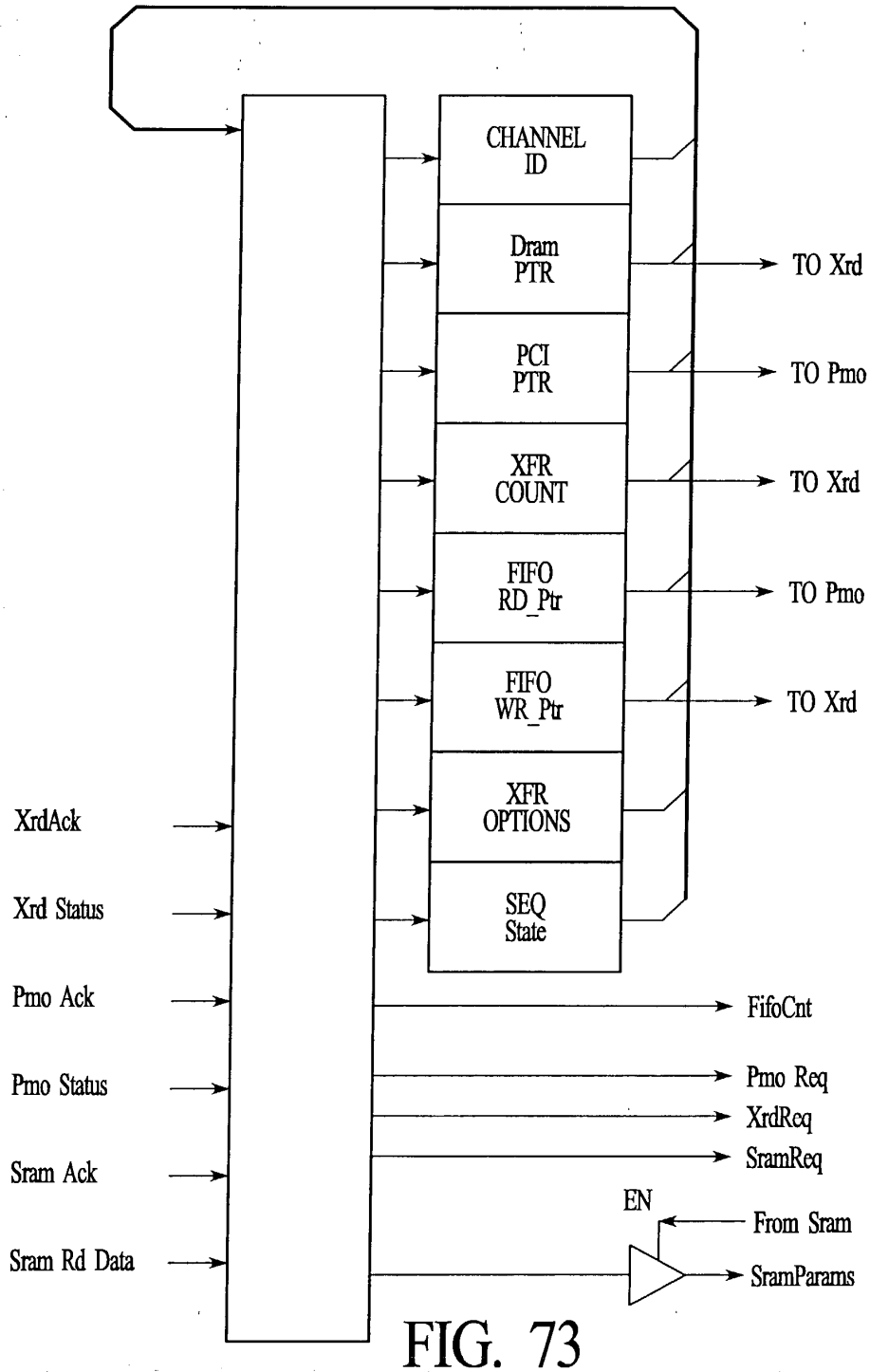

CHANNEL ID
Dram PTR
PCI PTR
XFR COUNT
FIFO RD_Ptr
FIFO WR_Ptr
XFR OPTIONS
SEQ State
TO Xrd
TO Pmo
TO Xrd
TO Pmo
TO Xrd
XrdAck
Xrd Status
Pmo Ack
Pmo Status
Sram Ack
Sram Rd Data
FifoCnt
Pmo Req
XrdReq
SramReq
EN
From Sram
SramParams

FIG. 73

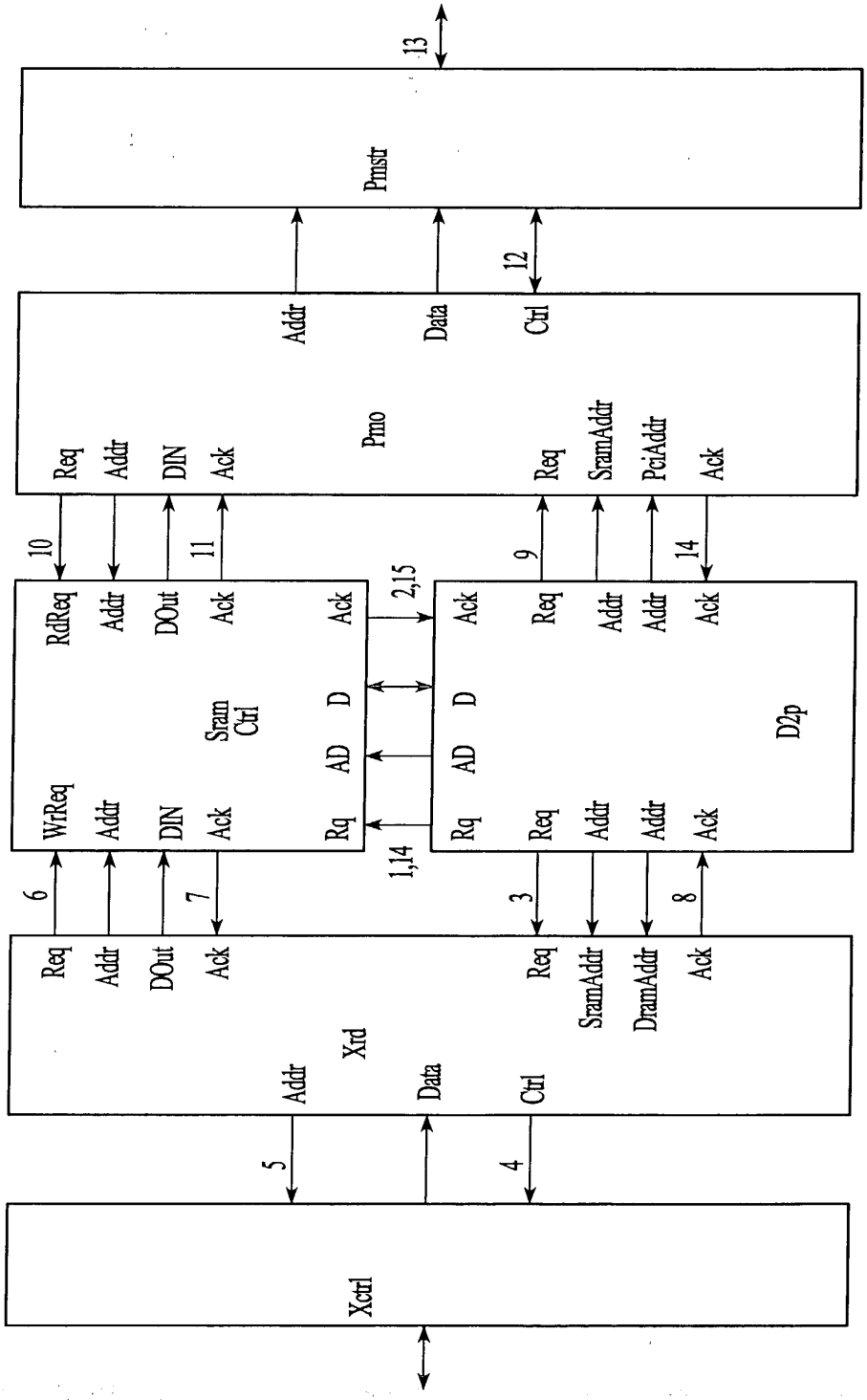

Pmstr
13
Addr
Data
Ctrl
12
Pmo
Req
Addr
DIN
Ack
Req
SramAddr
PciAddr
Ack
10
11
9
14
RdReq
Addr
DOut
Ack
Ack
2,15
Ack
Req
Addr
Addr
Ack
Sram Ctrl
D
AD
Rq
D2p
WrReq
Addr
DIN
Ack
Rq
Req
Addr
Addr
Ack
1,14
6
7
3
8
Req
Addr
DOut
Ack
Req
SramAddr
DramAddr
Ack
Xrd
Addr
Data
Ctrl
5
4
Xctrl

FIG. 74

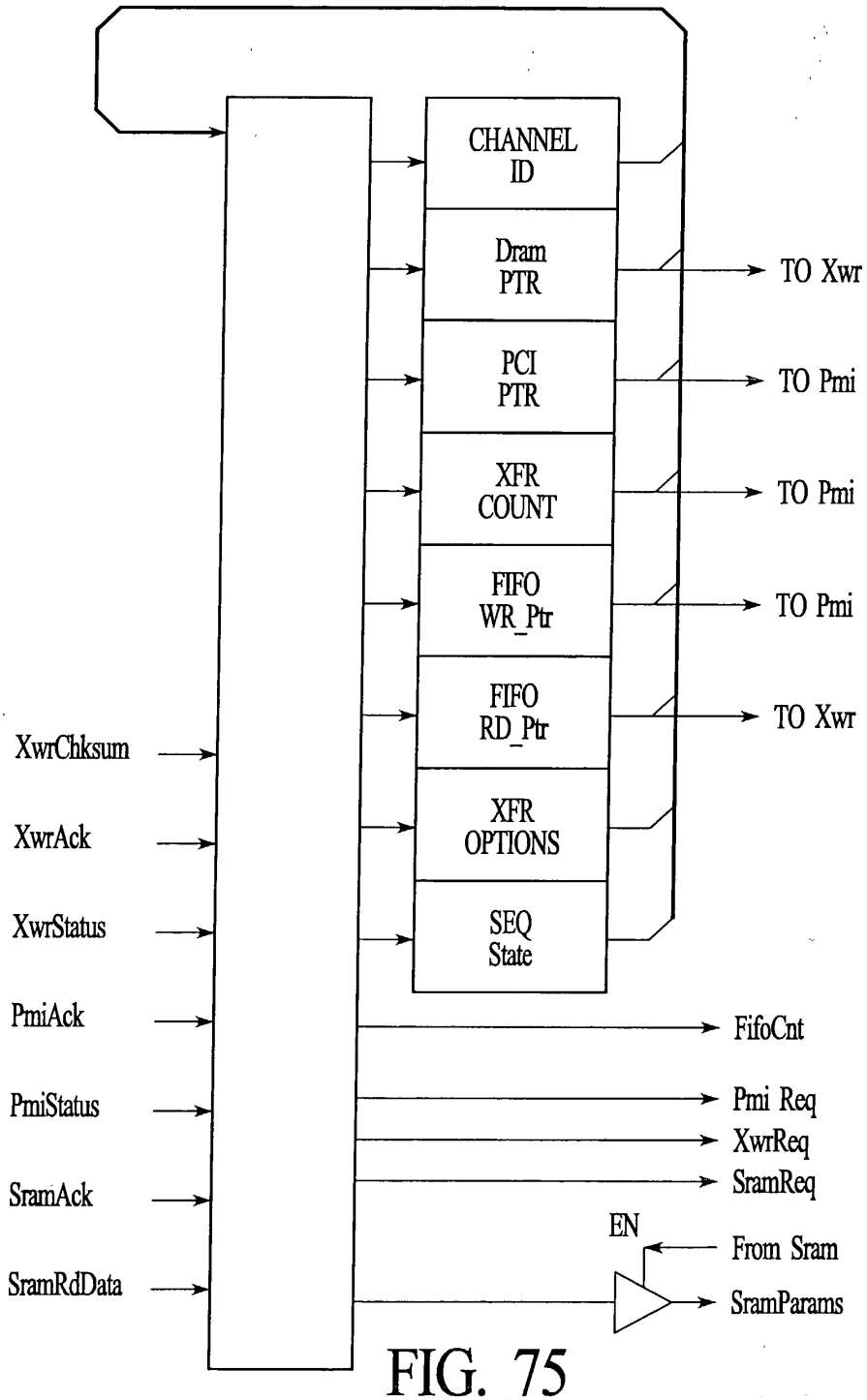

CHANNEL ID
Dram PTR
PCI PTR
XFR COUNT
FIFO WR_Ptr
FIFO RD_Ptr
XFR OPTIONS
SEQ State
TO Xwr
TO Pmi
TO Pmi
TO Pmi
TO Xwr
XwrChksum
XwrAck
XwrStatus
PmiAck
PmiStatus
SramAck
SramRdData
FifoCnt
Pmi Req
XwrReq
SramReq
EN
From Sram
SramParams

FIG. 75

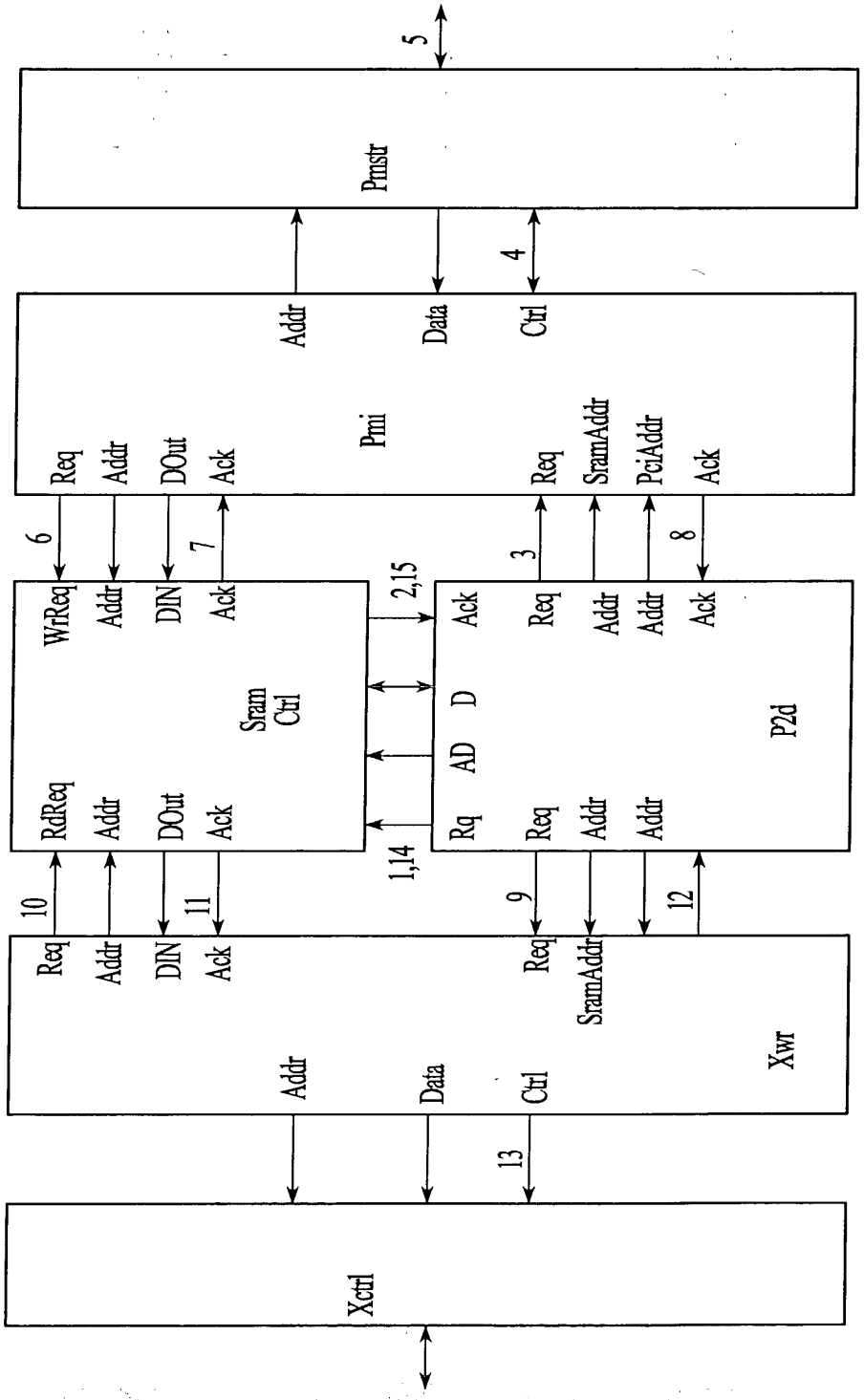

Pmstr
5
4
Addr
Data
Ctrl
Pmi
Req
Addr
DOut
Ack
SramAddr
PciAddr
6
7
3
8
WrReq
DIN
2,15
Sram Ctrl
D
AD
Rq
P2d
RdReq
1,14
10
11
9
12
SramAddr
Xwr
13
Xctrl

FIG. 76

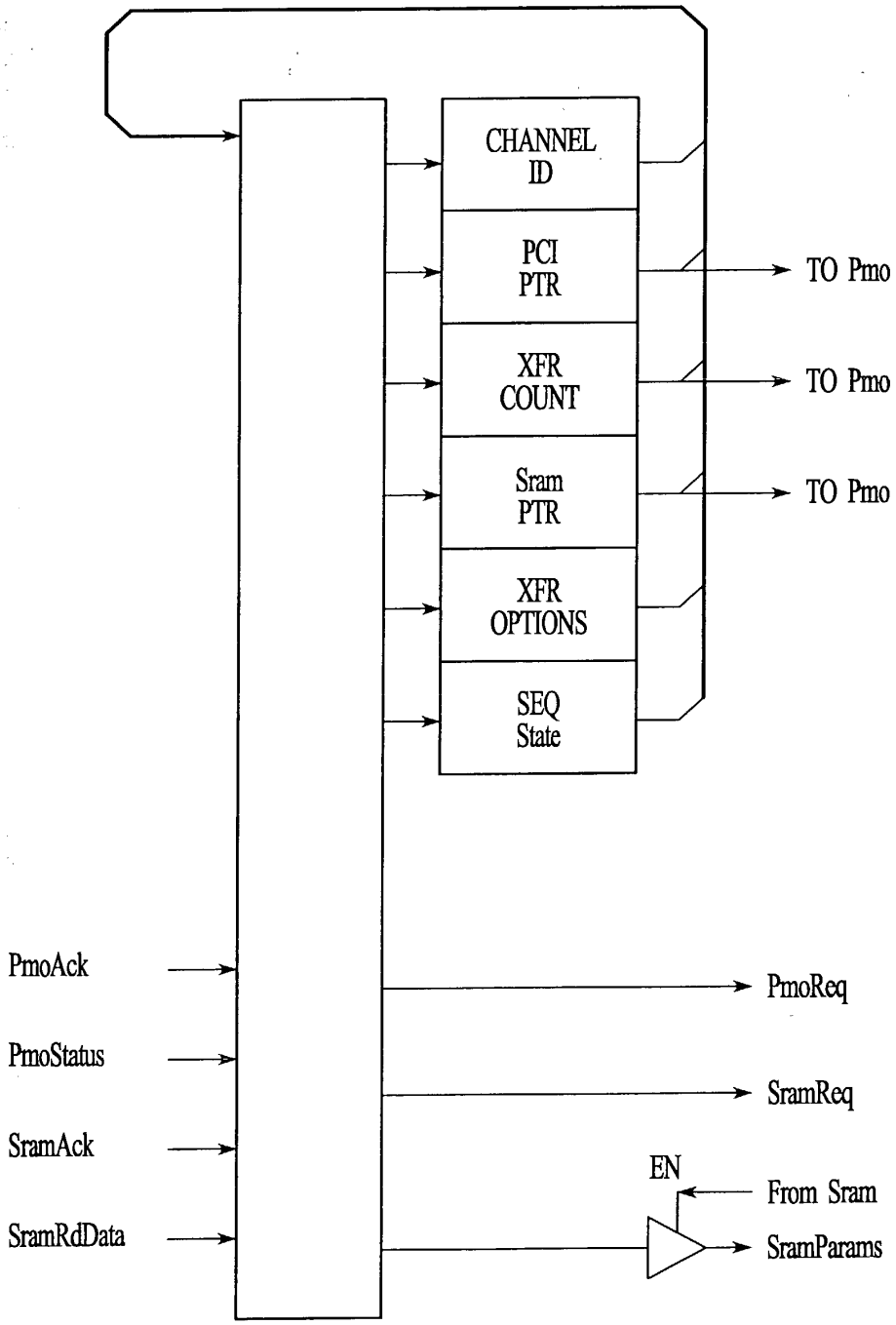

CHANNEL ID
PCI PTR
XFR COUNT
Sram PTR
XFR OPTIONS
SEQ State
TO Pmo
TO Pmo
TO Pmo
PmoAck
PmoStatus
SramAck
SramRdData
PmoReq
SramReq
EN
From Sram
SramParams

FIG. 77

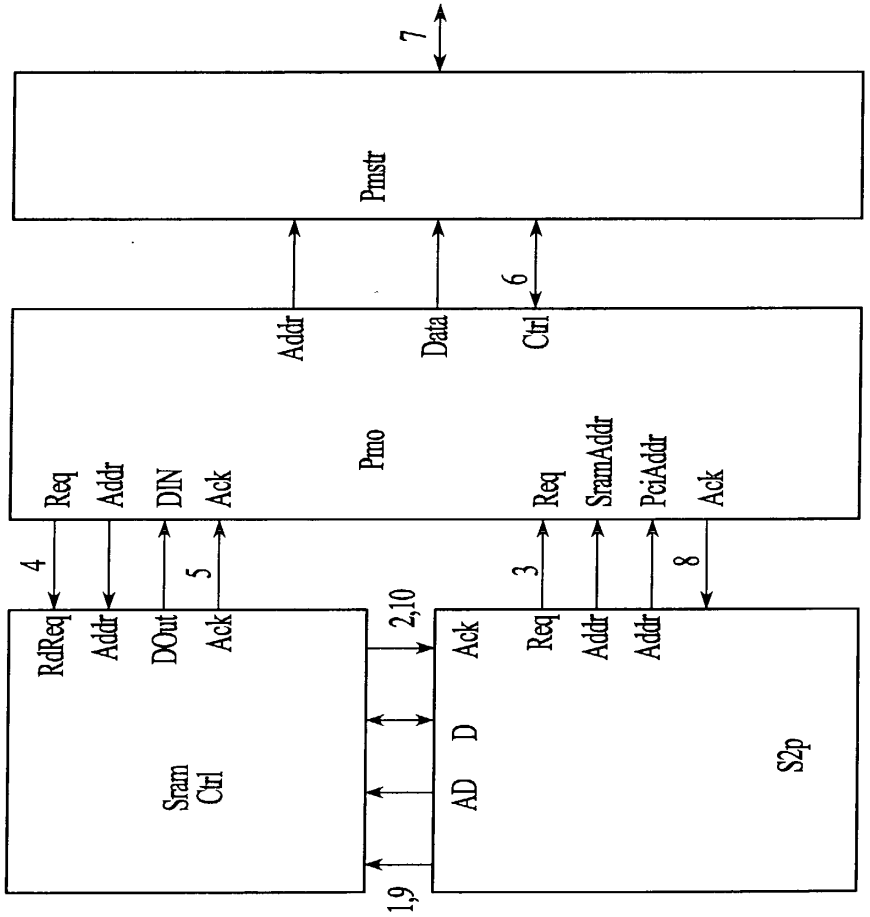

7
Pmstr
Addr
Data
6
Ctrl
Pmo
Req
Addr
DIN
Ack
Req
SramAddr
PciAddr
Ack
4
5
3
8
RdReq
Addr
DOut
Ack
2,10
Ack
Req
Addr
Addr
Sram
Ctrl
AD
D
S2p
1,9

FIG. 78

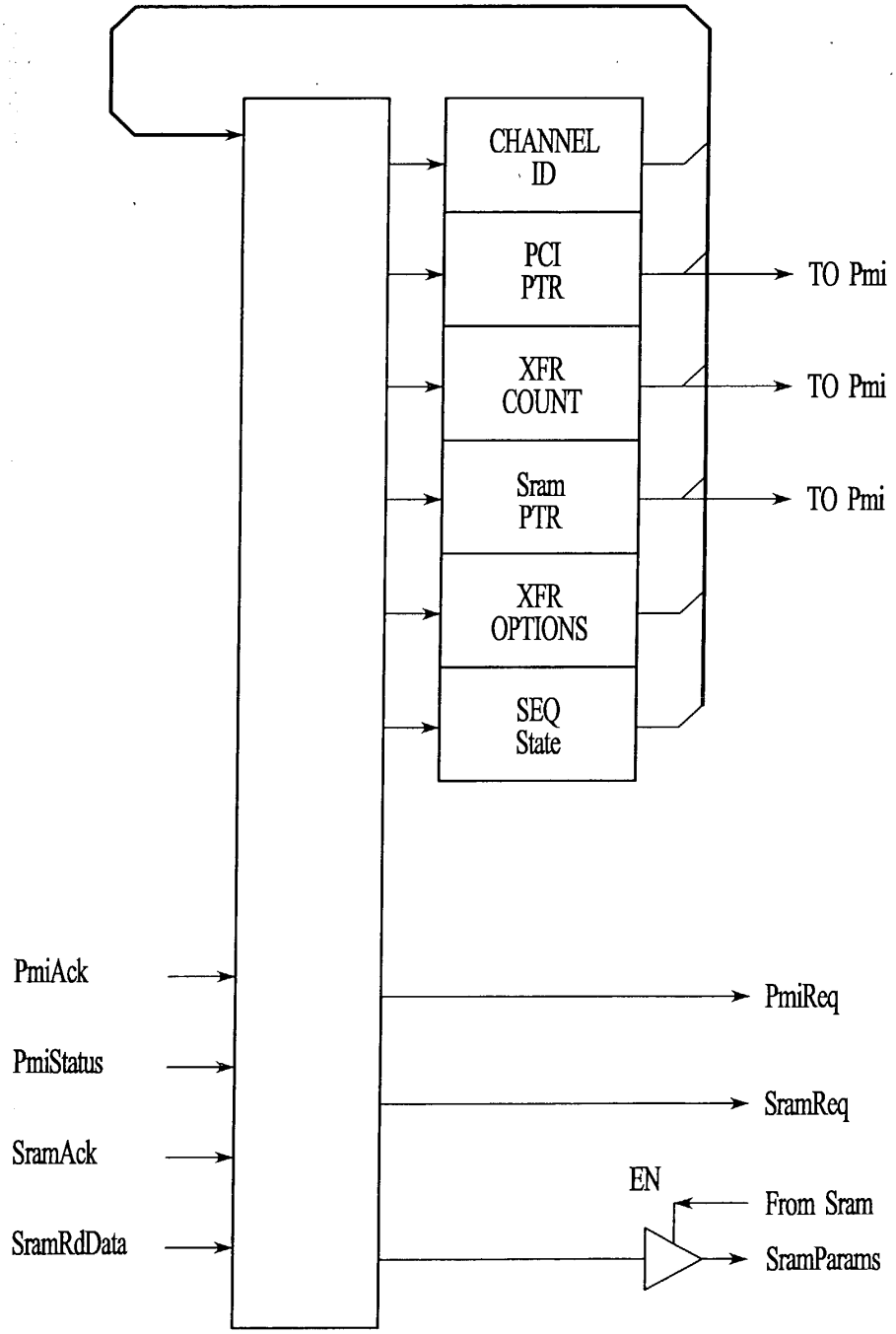

CHANNEL ID
PCI PTR
XFR COUNT
Sram PTR
XFR OPTIONS
SEQ State
TO Pmi
TO Pmi
TO Pmi
PmiAck
PmiStatus
SramAck
SramRdData
PmiReq
SramReq
EN
From Sram
SramParams

FIG. 79

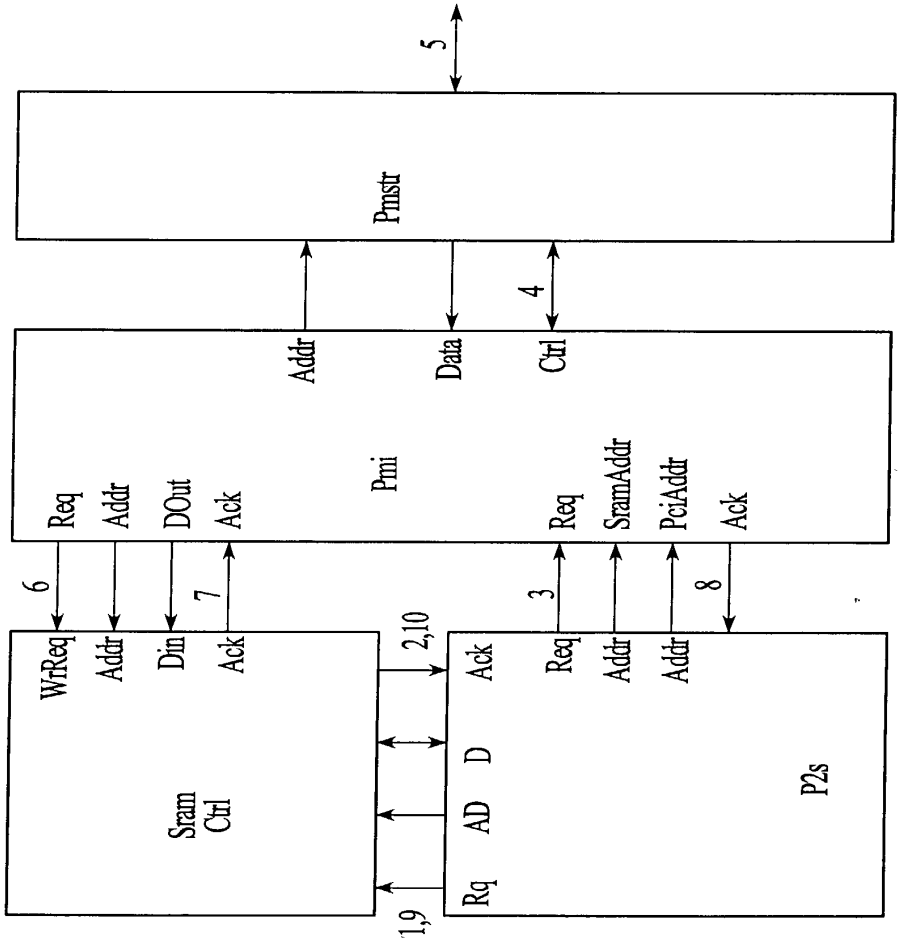

5
Pmstr
4
Addr
Data
Ctrl
Pmi
Req
Addr
DOut
Ack
Req
SramAddr
PciAddr
Ack
6
7
3
8
WrReq
Addr
Din
Ack
2,10
Ack
Req
Addr
Addr
Sram Ctrl
D
AD
Rq
1,9
P2s

FIG. 80

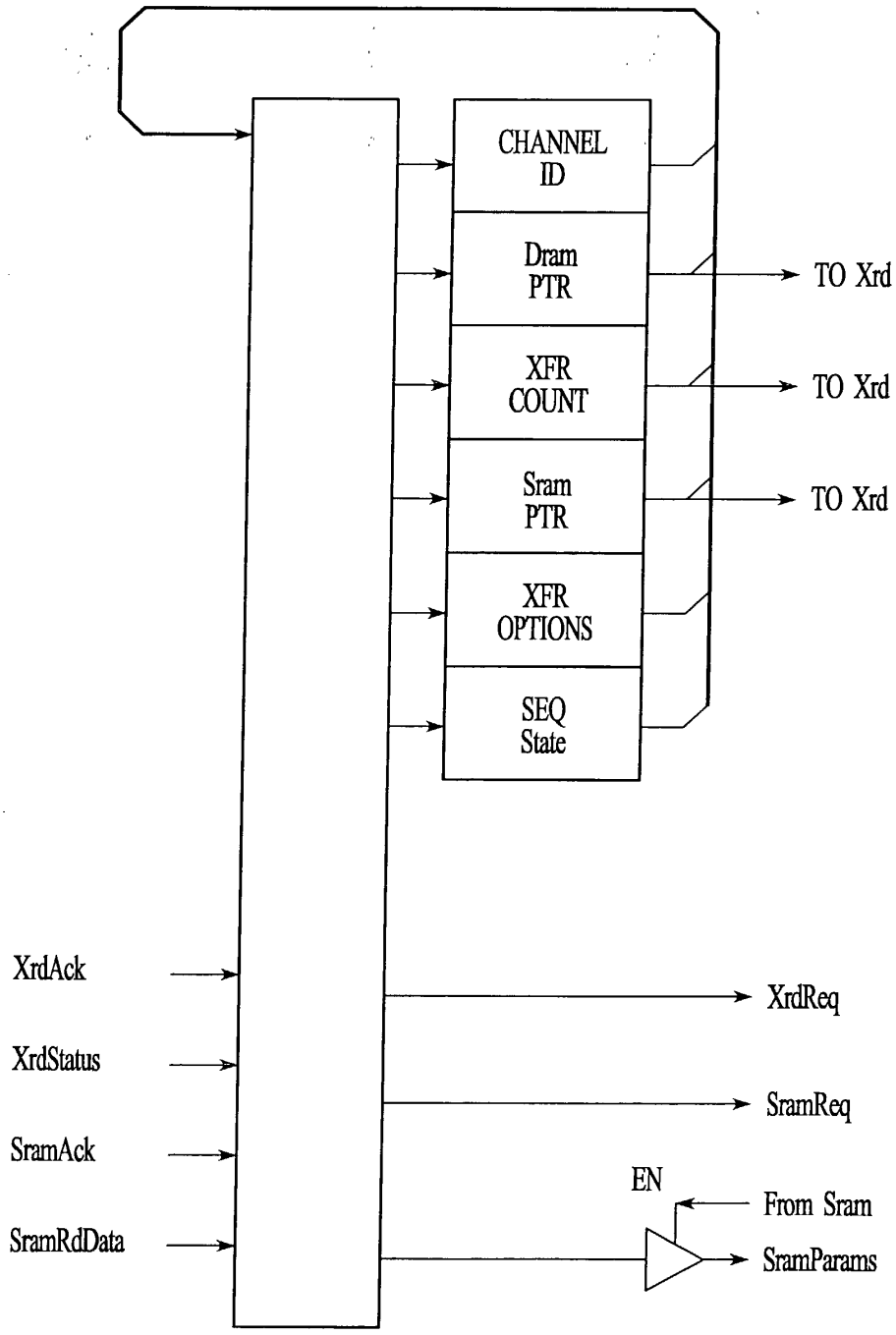

CHANNEL ID
Dram PTR
XFR COUNT
Sram PTR
XFR OPTIONS
SEQ State
TO Xrd
TO Xrd
TO Xrd
XrdAck
XrdStatus
SramAck
SramRdData
XrdReq
SramReq
EN
From Sram
SramParams

FIG. 81

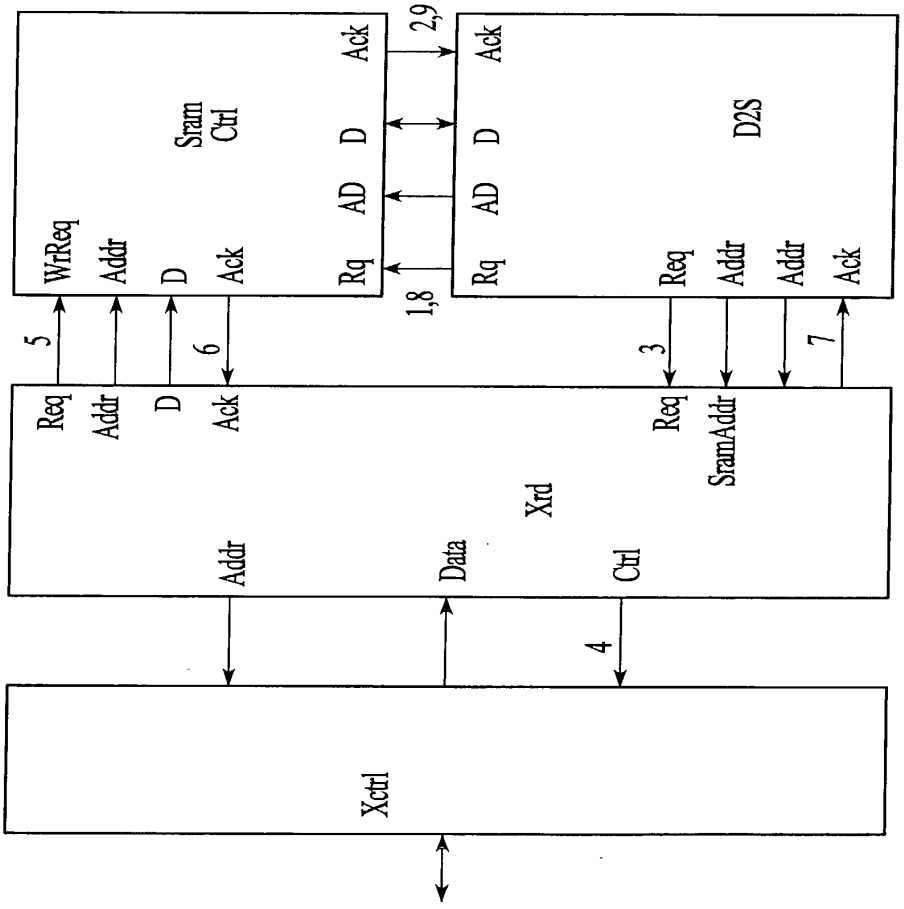

Sram Ctrl
WrReq
Addr
D
Ack
Rq
AD
D
Ack
2,9
1,8
D2S
Req
Addr
Addr
Ack
Rq
AD
D
Ack
5
6
3
7
Req
Addr
D
Ack
Req
SramAddr
Xrd
Addr
Data
Ctrl
4
Xctrl

FIG. 82

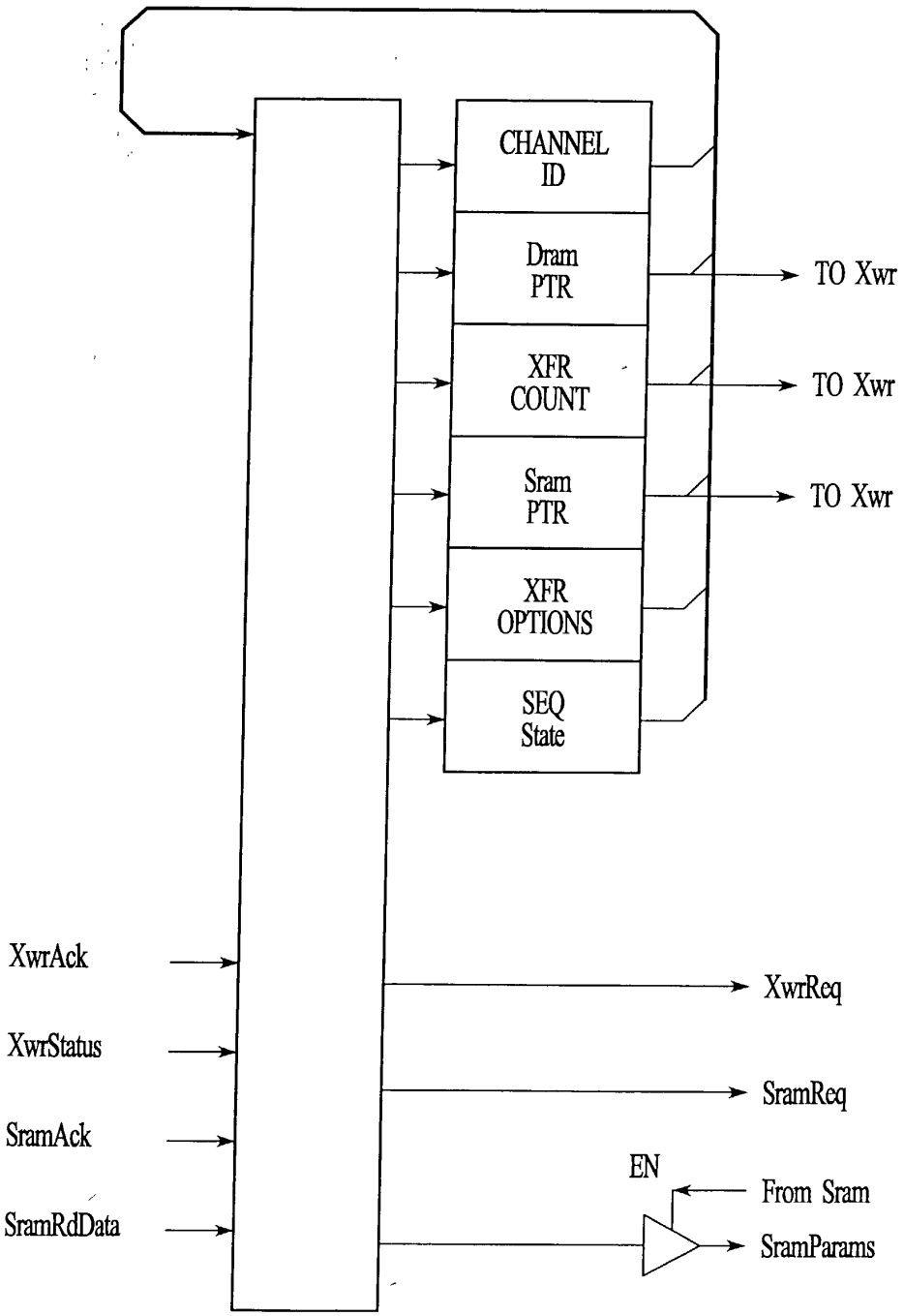

CHANNEL ID
Dram PTR
XFR COUNT
Sram PTR
XFR OPTIONS
SEQ State
TO Xwr
TO Xwr
TO Xwr
XwrAck
XwrStatus
SramAck
SramRdData
XwrReq
SramReq
EN
From Sram
SramParams

FIG. 83

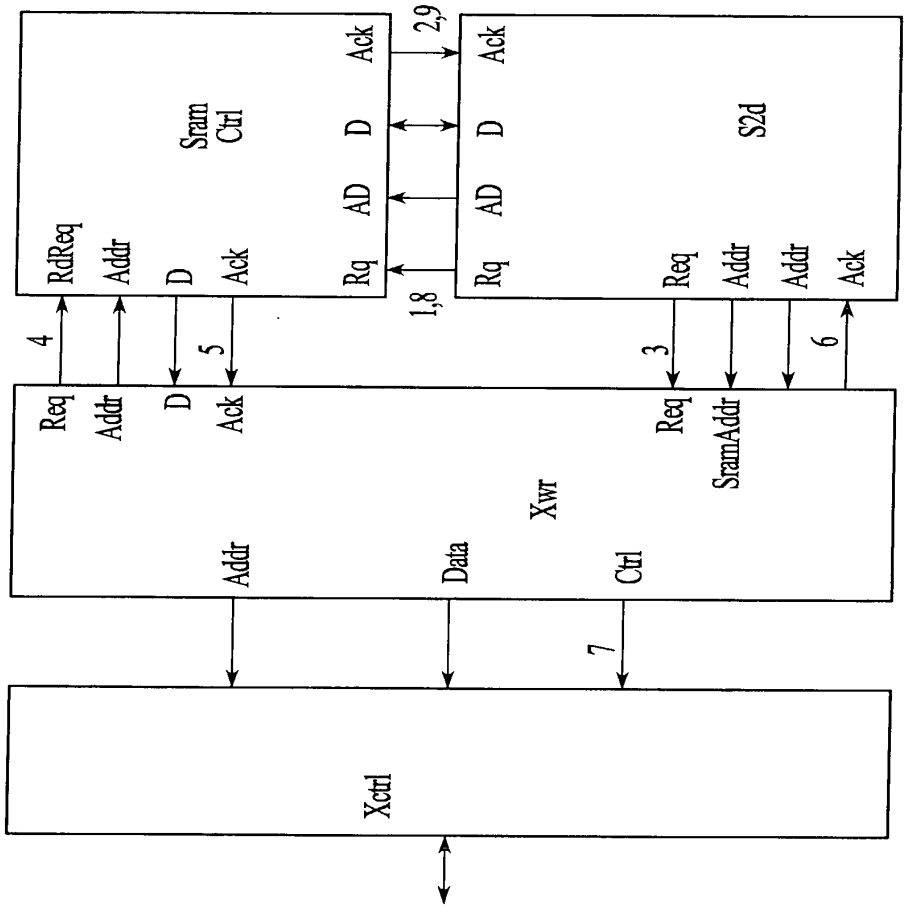

Sram Ctrl
RdReq
Addr
D
Ack
Rq
AD
D
Ack
2,9
1,8
S2d
Ack
D
AD
Rq
Req
Addr
Addr
Ack
4
5
3
6
Req
Addr
D
Ack
Req
SramAddr
Xwr
Addr
Data
Ctrl
7
Xctrl

FIG. 84

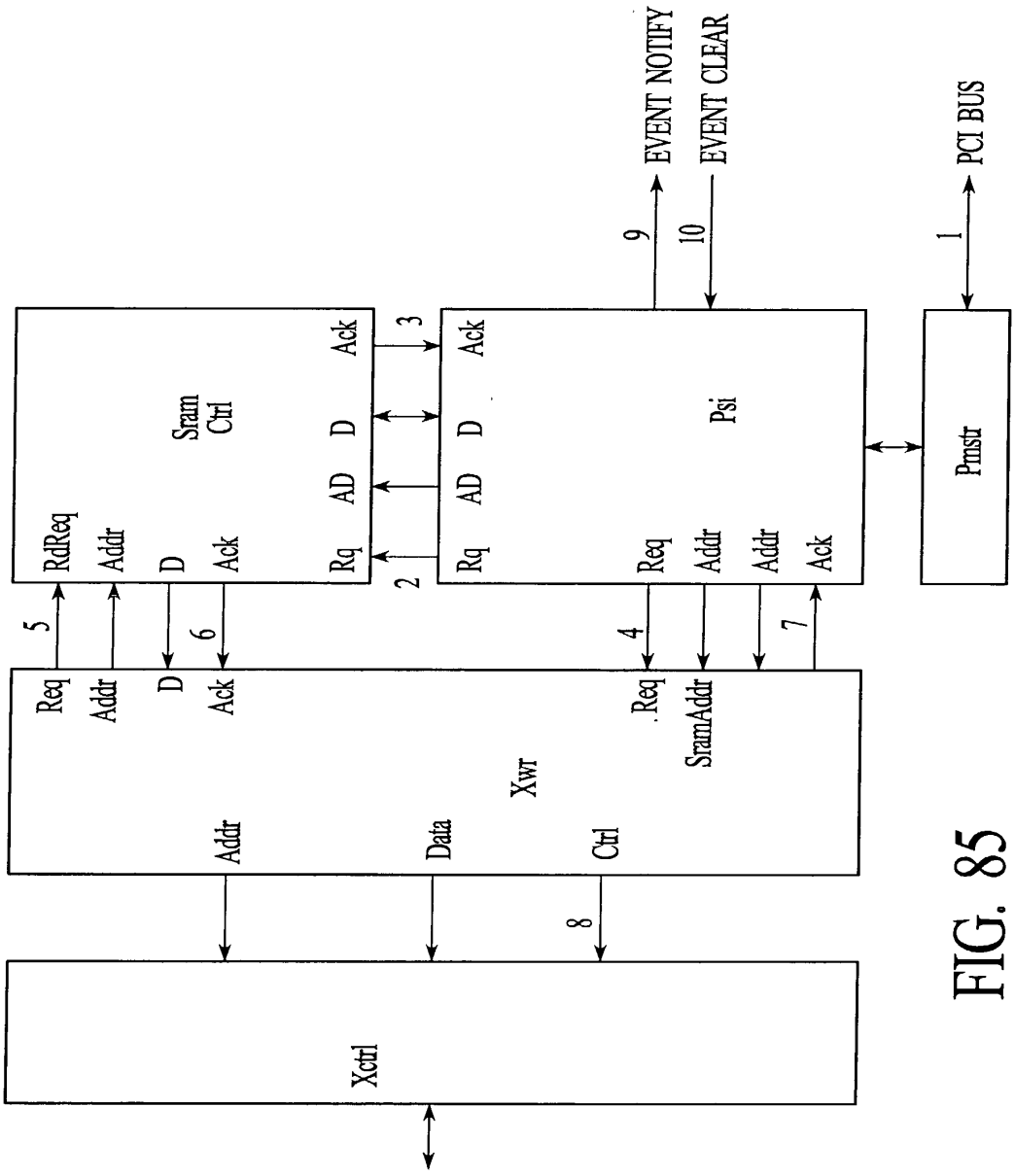

EVENT NOTIFY
EVENT CLEAR
PCI BUS
9
10
1
Sram Ctrl
RdReq
Addr
D
Ack
Rq
AD
D
Ack
3
2
Psi
Req
Addr
Addr
Ack
Pmstr
5
6
4
7
Req
Addr
D
Ack
Req
SramAddr
Xwr
Addr
Data
Ctrl
8
Xctrl

FIG. 85

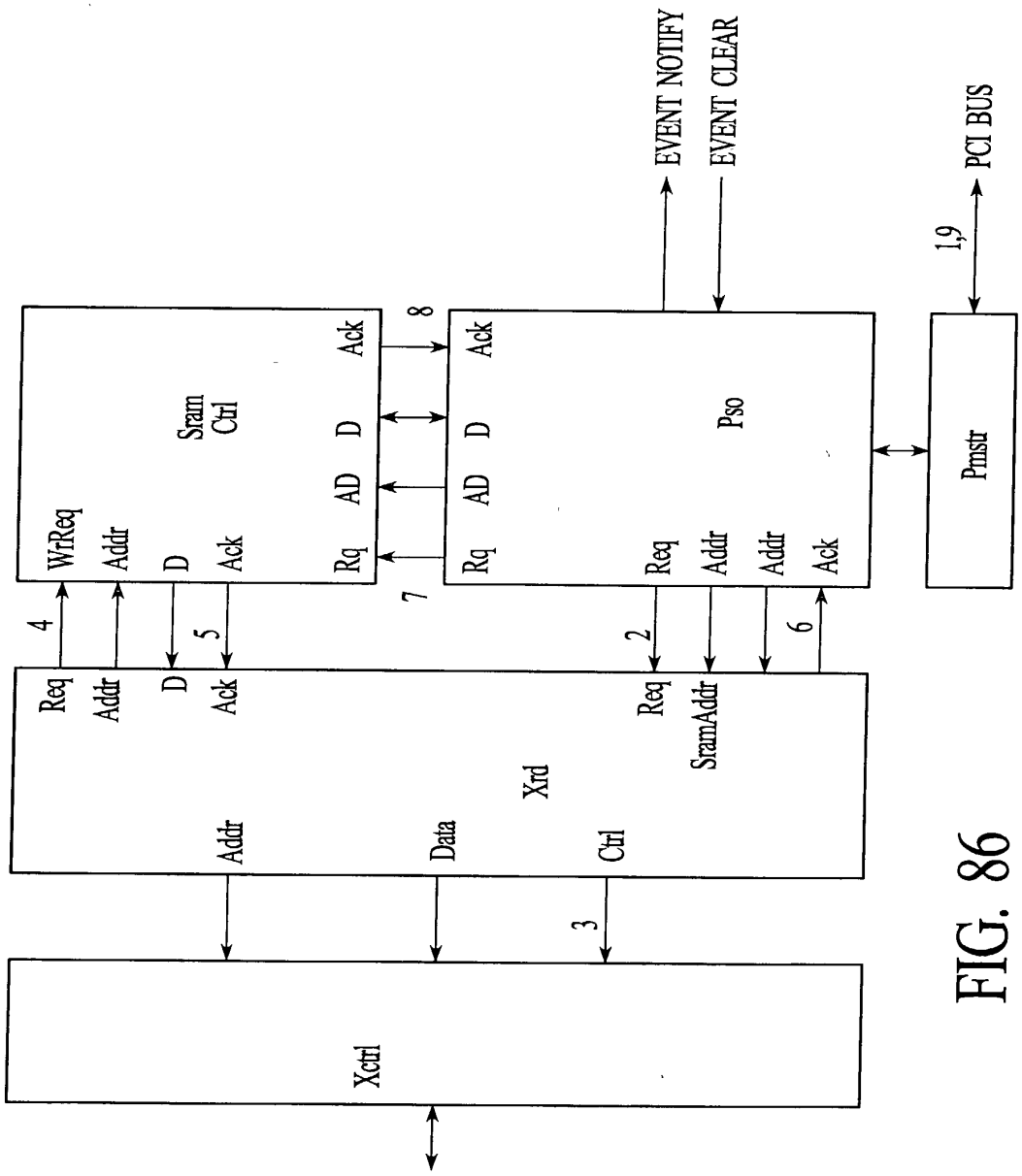

EVENT NOTIFY
EVENT CLEAR
PCI BUS
1,9
Sram Ctrl
WrReq
Addr
D
Ack
Rq
AD
D
Ack
8
7
Pso
Rq
AD
D
Ack
Req
Addr
Addr
Ack
Pmstr
4
5
2
6
Req
Addr
D
Ack
Req
SramAddr
Xrd
Addr
Data
Ctrl
3
Xctrl

FIG. 86

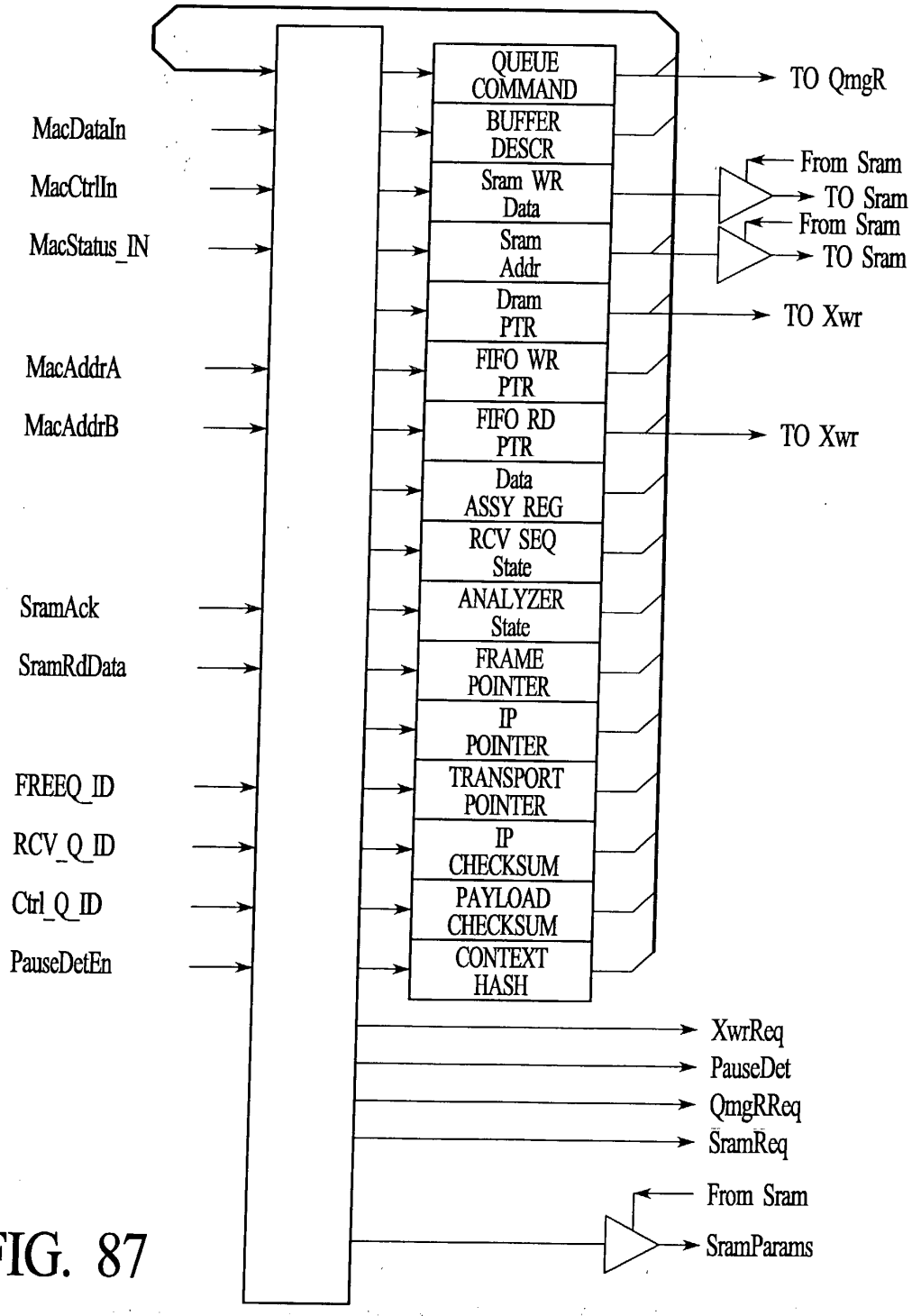

QUEUE COMMAND
BUFFER DESCR
Sram WR Data
Sram Addr
Dram PTR
FIFO WR PTR
FIFO RD PTR
Data ASSY REG
RCV SEQ State
ANALYZER State
FRAME POINTER
IP POINTER
TRANSPORT POINTER
IP CHECKSUM
PAYLOAD CHECKSUM
CONTEXT HASH
MacDataIn
MacCtrlIn
MacStatus_IN
MacAddrA
MacAddrB
SramAck
SramRdData
FREEQ_ID
RCV_Q_ID
Ctrl_Q_ID
PauseDetEn
TO QmgR
From Sram
TO Sram
From Sram
TO Sram
TO Xwr
TO Xwr
XwrReq
PauseDet
QmgRReq
SramReq
From Sram
SramParams

FIG. 87

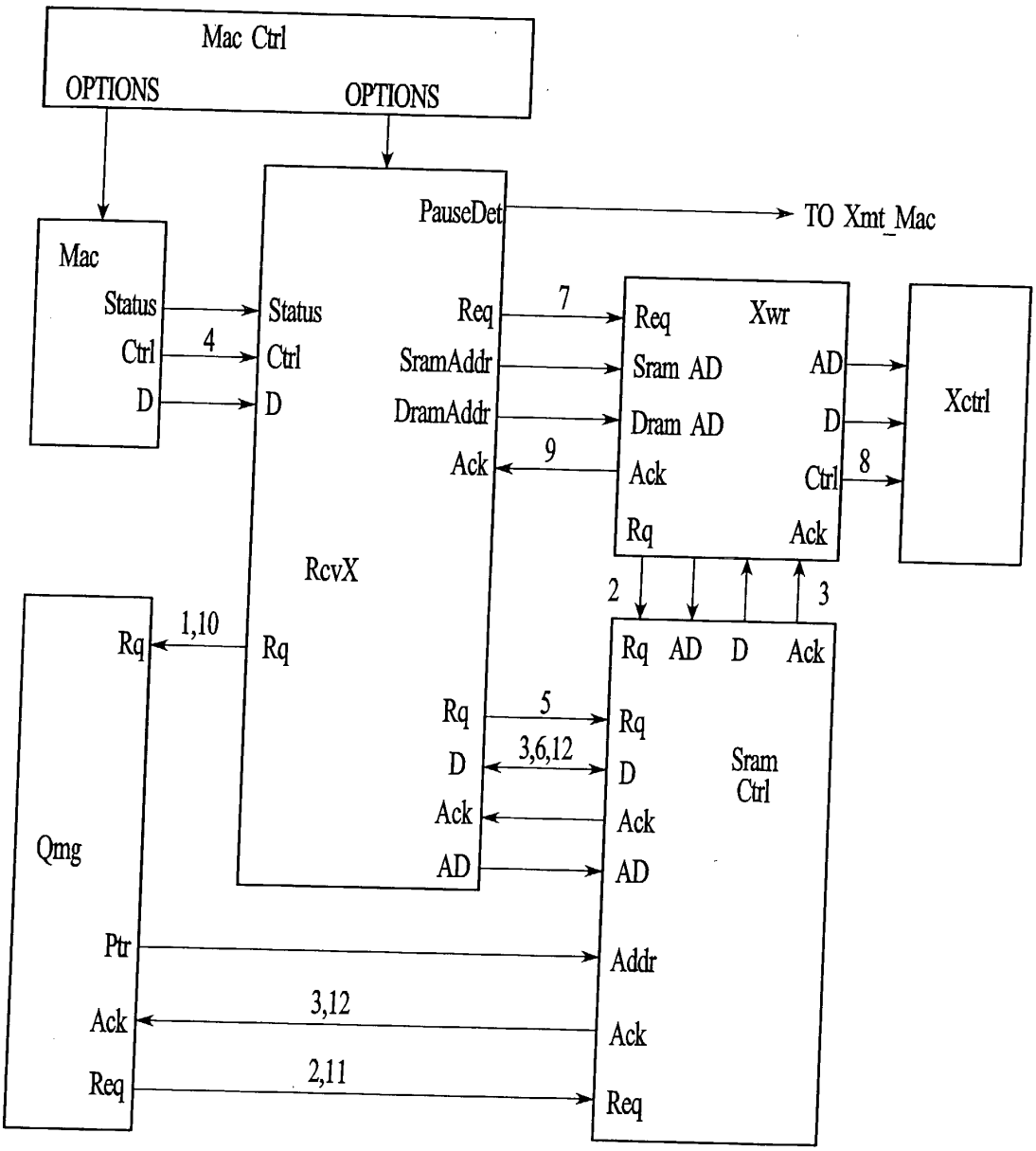

Mac Ctrl
OPTIONS
OPTIONS
Mac
Status
Ctrl
D
4
Status
Ctrl
D
RcvX
PauseDet
TO Xmt_Mac
Req
SramAddr
DramAddr
Ack
7
9
Req
Sram AD
Dram AD
Ack
Rq
Xwr
AD
D
Ctrl
Ack
8
Xctrl
2
3
Rq
1,10
Rq
Rq
AD
D
Ack
Rq
D
Ack
AD
5
3,6,12
Rq
D
Ack
AD
Sram
Ctrl
Qmg
Ptr
Ack
Req
3,12
2,11
Addr
Ack
Req

FIG. 88

RECEIVE BUFFER DESCRIPTOR

bit	name	description
31:30	**reserved**	
29:28	**size**	A copy of the bits in the **FreeBufDscr**.
27:00	**address**	Represents the last address +1 to which frame data was transferred. The address wraps around at the boundary dictated by the **S** bits. This can be used to determine the size of the frame received.

FIG. 89

TIME STAMP **OFFSET 0x0008:0x000B**

bit	name	description
31:00	**RcvTime**	The contents of **FreeClk** at the completion of the frame receive operation.

FIG. 90

CHECKSUM **OFFSET 0x000C:0x000F**

bit	name	description
31:16	**IpChksum**	Reflects the value of the IP header checksum at frame completion or IP header completion. If an IP datagram was not detected, the checksum provides a total for the entire data portion of the received frame. The data area is defined as those bytes received after the type field of an ethernet frame, the LLC header of an 802.3 frame or the SNAP header of an 802.3-SNAP frame.
15:00	**TcpChksum**	Reflects the value of the transport checksum at IP completion or frame completion. If IP was detected but session was unknown, the checksum will not include the psuedo-header. If IP was not detected, the checksum will be 0x0000.

RESERVED **OFFSET 0x0010:0x0011**

FRAME Data **OFFSET 0x0012:END OF BUFFER**

FIG. 91

RECEIVE BUFFER FORMAT

FRAME Status A **OFFSET 0x0000:0x0003**

bit	name	description
31	**attention**	Indicates one or more of the following: **CompositeErr, !IpDn, !MacADet & !MacBDet, IpMcst, IpBcst, !ethernet & !802.3Snap, !Ip4, !Tcp .**
30	**CompositeErr**	Set when any of the error bits of **ErrStatus** are set or if frame processing stops while receiving a Tcp or Udp header.
29	**CtrlFrame**	A control frame was received at our unicast or special MltCst address.
28	**IpDn**	Frame processing Hlted due to exhaustion of the IP4 length counter.
27	**802.3Dn**	Frame processing Hlted due to exhaustion of the 802.3 length counter.
26	**MacADet**	Frame's destination address matched the contents of **MacAddrA**.
25	**MacBDet**	Frame's destination address matched the contents of **MacAddrB.**
24	**MacMcst**	The Mac detected a MltCst address.
23	**MacBcst**	The Mac detected a BrdCst address.
22	**IpMcst**	The frame processor detected an IP MltCst address.
21	**IpBcst**	The frame processor detected an IP BrdCst address.
20	**Frag**	The frame processor detected a Frag IP datagram.
19	**IpOffst**	The frame processor detected a non-zero IP datagram offset.
18	**IpFlgs**	The frame processor detected flags within the IP datagram.
17	**IpOpts**	The frame processor detected a header length greater than 20 for the IP datagram.
16	**TcpFlgs**	The frame processor detected an abnormal header flag for the TCP segment.
15	**TcpOpts**	The frame processor detected a header length greater than 20 for the TCP segment.
14	**TcpUrg**	The frame processor detected a non-zero urgent pointer for the TCP segment.
13	**CarrierEvnt**	Refer to *E110 Technical Manual.*
12	**LongEvnt**	Refer to *E110 Technical Manual.*
11	**FrameLost**	Set when an incoming frame could not be processed as a result of an outstanding frame completion event not yet serviced by the utility processor.
10	**reserved**	
10	**NoAck**	The frame processor detected a
09:08	**FrameTyp**	00 - Reserved. 01- ethernet. 10 - 802.3. 11 - 802.3 Snap.
07:06	**NwkTyp**	00 - Unknown. 01- Ip4. 10 - Ip6 11 - ip other.
05:04	**TrnsptTyp**	00 - Unknown. 01- reserved. 10 - Tcp 11 - Udp
03	**NetBios**	A NetBios frame was detected.
02	**reserved**	
01:00	**channel**	The Mac on which this frame was received.

FRAME Status B **OFFSET 0x0004:0x0007**

bit	name	description
31	**802.3Shrt**	End of frame was encountered before the 802.3 length count was exhausted.
30	**BufOvr**	The frame length exceded the buffer space available.
29	**BadPkt**	Refer to *E110 Technical Manual.*
28	**InvldPrmbl**	Refer to *E110 Technical Manual.*
27	**CrcErr**	Refer to *E110 Technical Manual.*
26	**DrblNbbl**	Refer to *E110 Technical Manual.*
25	**CodeErr**	Refer to *E110 Technical Manual.*
24	**IpHdrShrt**	The IP4 header length field contained a value less than 0x5.
23	**IpIncmplt**	The frame terminated before the IP length counter was exhausted.
22	**IpSumErr**	The IP header checksum was not 0xffff at the completion of the IP header read.
21	**TcpSumErr**	The session checksum was not 0xffff at the termination of session processing.
20	**TcpHdrShrt**	The TCP header length field contained a value less than 0x5.
19:16	**PrcssCd**	The state of the frame processor at the time the frame processing terminated. 0b0000 Processing Mac header. 0b0001 Processing 802.3 LLC header. 0b0010 Processing 802.3 SNAP header. 0b0011 Processing unknown network data. 0b0100 Processing IP header. 0b0101 Processing IP data (unknown transport). 0b0110 Processing transport header (IP data). 0b0111 Processing transport data (IP data). 0b1000 Processing IP processing complete. 0b1001 Reserved. 0b101x Reserved. 0b11xx Reserved.
15:08	**MacHsh**	The Mac destination-address hash. Refer to *E110 Technical Manual.*
07:00	**CtxHsh**	The 8-bit context-hash generated by exclusive-oring all bytes of the IP source address, IP destination-address, transport source port and the transport destination port.

FIG. 92

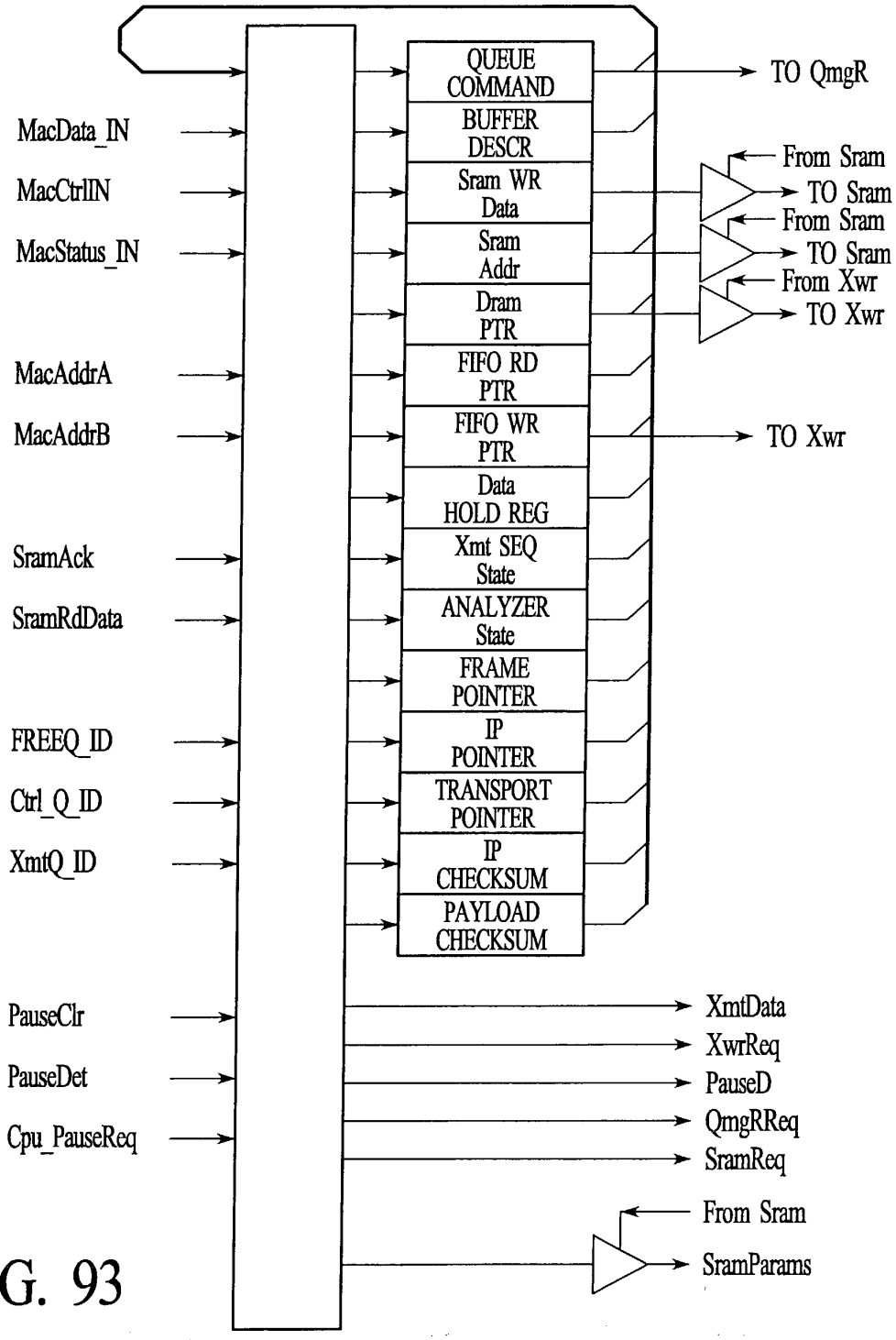

QUEUE COMMAND
BUFFER DESCR
Sram WR Data
Sram Addr
Dram PTR
FIFO RD PTR
FIFO WR PTR
Data HOLD REG
Xmt SEQ State
ANALYZER State
FRAME POINTER
IP POINTER
TRANSPORT POINTER
IP CHECKSUM
PAYLOAD CHECKSUM
TO QmgR
From Sram
TO Sram
From Sram
TO Sram
From Xwr
TO Xwr
TO Xwr
MacData_IN
MacCtrlIN
MacStatus_IN
MacAddrA
MacAddrB
SramAck
SramRdData
FREEQ_ID
Ctrl_Q_ID
XmtQ_ID
PauseClr
PauseDet
Cpu_PauseReq
XmtData
XwrReq
PauseD
QmgRReq
SramReq
From Sram
SramParams

FIG. 93

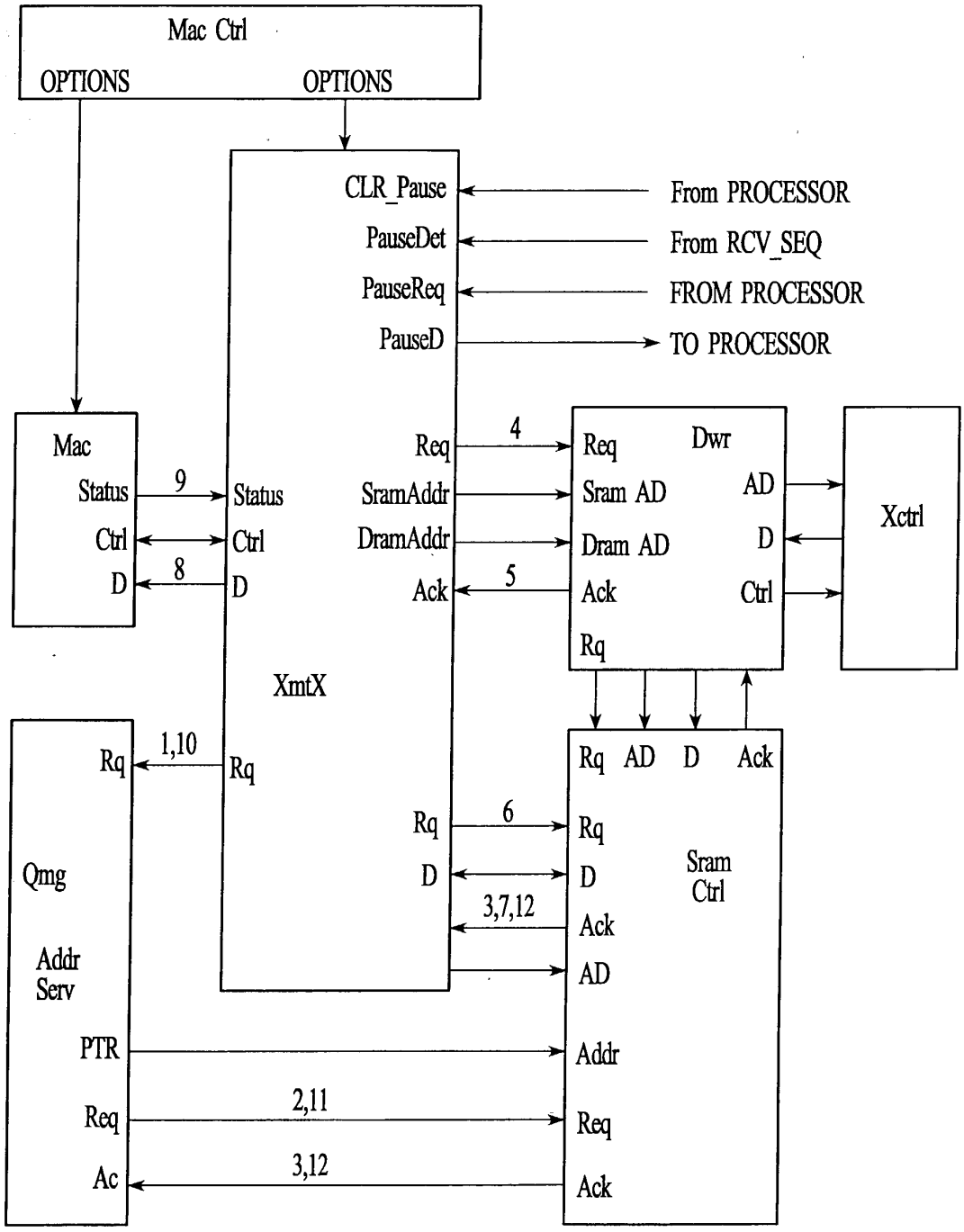

Mac Ctrl
OPTIONS
OPTIONS
CLR_Pause
From PROCESSOR
PauseDet
From RCV_SEQ
PauseReq
FROM PROCESSOR
PauseD
TO PROCESSOR
Mac
Status
Ctrl
D
9
8
Status
Ctrl
D
Req
SramAddr
DramAddr
Ack
4
5
Req
Sram AD
Dram AD
Ack
Rq
Dwr
AD
D
Ctrl
Xctrl
XmtX
Rq
1,10
Rq
Qmg
Addr
Serv
PTR
Req
Ac
Rq
D
6
3,7,12
AD
Rq AD D Ack
Rq
D
Ack
AD
Sram
Ctrl
Addr
Req
Ack
2,11
3,12

FIG. 94

TRANSMIT BUFFER DESCRIPTOR

bit	name	description
31	**ChksumEn**	When set, **XmtSeq** will insert a calculated checksum. When reset, **XmtSeq** will not alter the outgoing data stream.
30	**reserved**	
29:28	**size**	Represents the size of the buffer by indicating at what boundary the buffer should start and terminate. This is used in combination with **EndAddr** to determine the starting address of the buffer : S = 0 256B boundary. **A[7:0]** ignored. S = 1 2KB boundary. **A[10:0]** ignored. S = 2 4KB boundary. **A[11:0]** ignored. S = 3 32KB boundary. **A[14:0]** ignored.
27:00	**EndAddr**	The address of the last byte to transmit plus one.

FIG. 95

TRANSMIT BUFFER FORMAT

CHECKSUM PRIMER OFFSET 0x0000:0x0003

bit	name	description
31:00	**Primer**	A value to be added during checksum accumulation. For IPV4, this should include the psuedo-header values, protocol and Tcp-length.

RESERVED **OFFSET 0x0004:0x0005**

FRAME Data **OFFSET 0x0006:END OF BUFFER**

FIG. 96

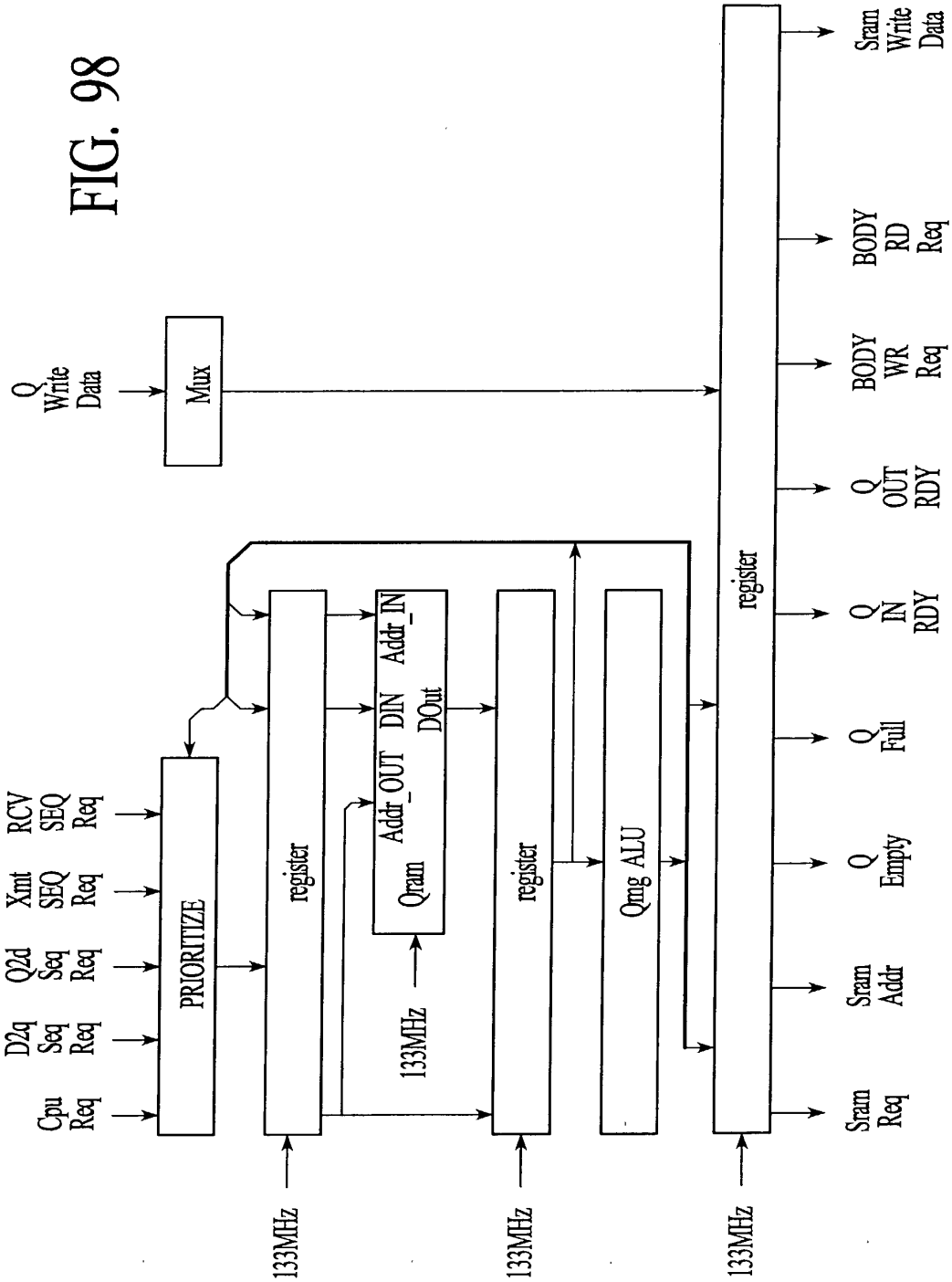
TRANSMIT Status VECTOR

bit	name	description
31	**LnkErr**	Indicates that a link status error occured before or during transmit.
30:15	**reserved**	
14	**ExcessDeferral**	Refer to *E110 Technical Manual.*
13	**LateAbort**	Refer to *E110 Technical Manual.*
12	**ExcessColl**	Refer to *E110 Technical Manual.*
11	**UnderRun**	Refer to *E110 Technical Manual.*
10	**ExcessLgth**	Refer to *E110 Technical Manual.*
09	**Okay**	Refer to *E110 Technical Manual.*
08	**deferred**	Refer to *E110 Technical Manual.*
07	**BrdCst**	Refer to *E110 Technical Manual.*
06	**MltCst**	Refer to *E110 Technical Manual.*
05	**CrcErr**	Refer to *E110 Technical Manual.*
04	**LateColl**	Refer to *E110 Technical Manual.*
03:00	**CollCnt**	Refer to *E110 Technical Manual.*

FIG. 97

[redacted]

FIG. 98


Cpu Req
D2q Seq Req
Q2d Seq Req
Xmt SEQ Req
RCV SEQ Req
PRIORITIZE
register
133MHz
Addr_OUT
Qram
DIN
Addr_IN
DOut
133MHz
register
133MHz
Qmg ALU
register
133MHz
Sram Req
Sram Addr
Q Empty
Q Full
Q IN RDY
Q OUT RDY
BODY WR Req
BODY RD Req
Sram Write Data
Q Write Data
Mux

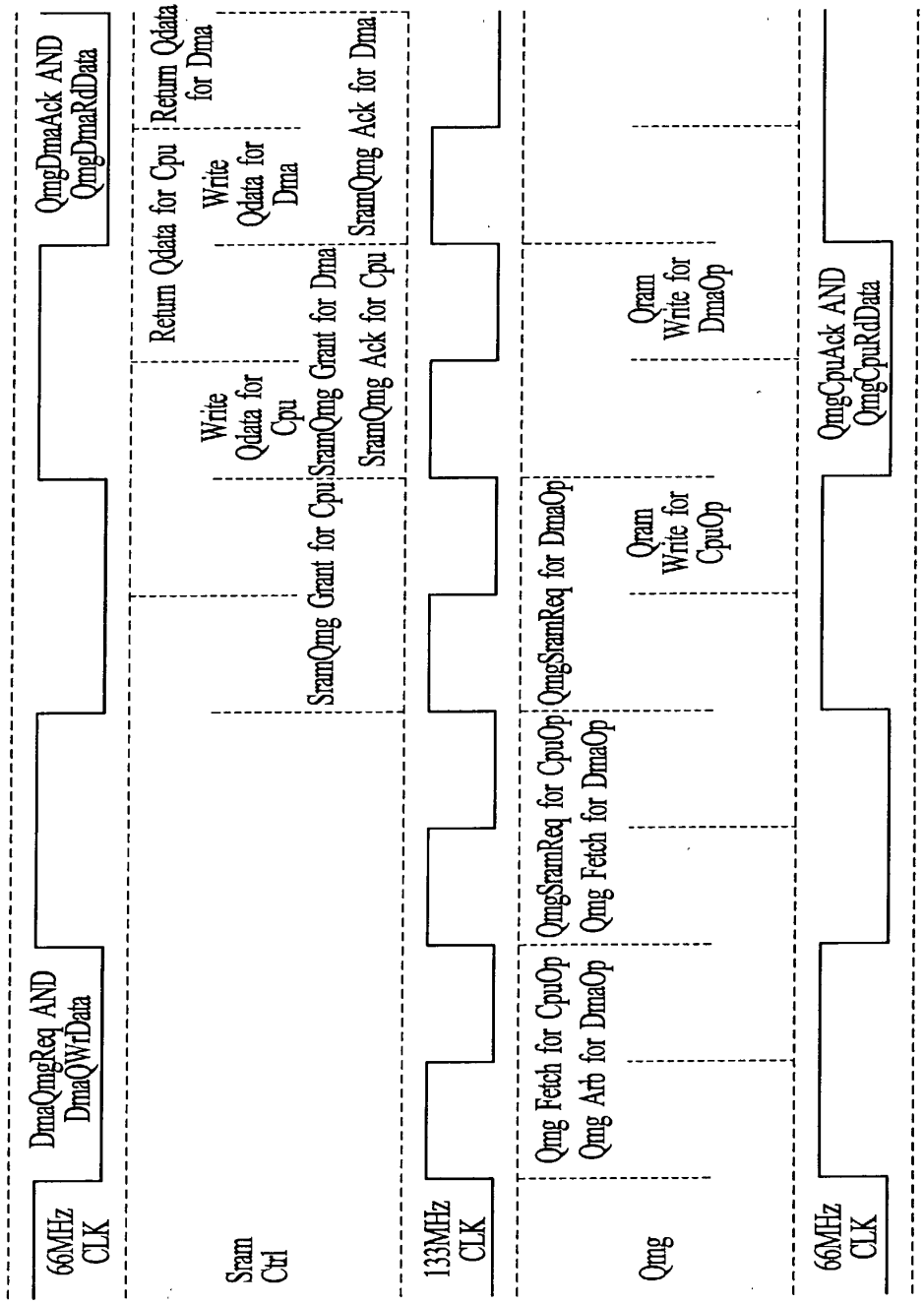

66MHz CLK
DmaQmgReq AND DmaQWrData
QmgDmaAck AND QmgDmaRdData
Sram Ctrl
SramQmg Grant for Cpu
Write Qdata for Cpu
SramQmg Ack for Cpu
Return Qdata for Cpu
SramQmg Grant for Dma
Write Qdata for Dma
SramQmg Ack for Dma
Return Qdata for Dma
133MHz CLK
Qmg
Qmg Fetch for CpuOp
Qmg Arb for DmaOp
QmgSramReq for CpuOp
Qmg Fetch for DmaOp
QmgSramReq for DmaOp
Qram Write for CpuOp
Qram Write for DmaOp
66MHz CLK
QmgCpuAck AND QmgCpuRdData

FIG. 99

DMA OPERATIONS

dma seq #	name	description
0	none	This is a no operation address.
1	**D2dSeq**	Moves data from ExtMem to ExtMem.
2	**D2sSeq**	Moves data from ExtMem bus to sram.
3	**D2pSeq**	Moves data from ExtMem to Pci bus.
4	**S2dSeq**	Moves data from sram to ExtMem.
5	**S2pSeq**	Moves data from sram to Pci bus.
6	**P2dSeq**	Moves data from Pci bus to ExtMem.
7	**P2sSeq**	Moves data from Pci bus to sram.

FIG. 100

bit	name	description
31:11	reserved	Data written to these bits is ignored.
10:8	**ChCmd**	0 - Stops execution of the current operation and clears the corresponding event flag. 1 - Transfer data from ExtMem to ExtMem. 2 - Transfer data from ExtMem bus to sram. 3 - Transfer data from ExtMem to Pci bus. 4 - Transfer data from sram to ExtMem. 5 - Transfer data from sram to Pci bus. 6 - Transfer data from Pci bus to ExtMem. 7 - Transfer data from Pci bus to Sram.
07:05	reserved	Data written to these bits is ignored.
04:00	**ChId**	Provides the channel number for the channel command.

FIG. 101

bit	name	description
127:96	**PciAddrH**	Bits [63:32] of the Pci address.
95:64	**PciAddrL**	Bits [31:00] of the Pci address.
59:32	**MemAddr**	Bits [27:00] of the ExtMem address or bits [15:00] of the Sram address.
31	**PciEndian**	When set, selects big endian mode for Pci transfers.
30	**WideDbl**	When set, disables Pci 64-bit mode.
22	**DstFlash**	Selects Flash for the external memory destination of **P2d**.
15:00	**XfrSz**	Bits [15:00] of the requested dma size expressed in bytes.

FIG. 102

bit	name	description
123:96	**MemAddr**	Bits [27:00] of the ExtMem address or bits [15:00] of the Sram address.
95:64	**PciAddrH**	Bits [63:32] of the Pci address.
63:32	**PciAddrL**	Bits [31:00] of the Pci address.
30	**SrcFlash**	Selects Flash for the external memory source of **D2p**.
23	**PciEndian**	When set, selects big endian mode for Pci transfers.
22	**WideDbl**	When set, disables Pci 64-bit mode.
15:00	**XfrSz**	Bits [15:00] of the requested dma size expressed in bytes.

FIG. 103

bit	name	description
127:124	**reserved**	Reserved for future use.
123:96	**SrcAddr**	Bits [27:00] of the ExtMem address or bits [15:00] of the Sram address.
95:60	**reserved**	Reserved for future use.
59:32	**DstAddr**	Bits [27:00] of the ExtMem address or bits [15:00] of the Sram address.
30	**FlashSel**	Selects Flash for the external memory source of **D2d** or **D2s**.
22	**FlashSel**	Selects Flash for the external memory destination of **S2p** or **D2d**.
15:00	**XfrSz**	Bits [15:00] of the requested dma size expressed in bytes.

FIG. 104

bit	name	description
127:64	**reserved**	Not used.
63:32	**ChkSum**	Represents the 1's compliment sum of all halfwords transferred during a **P2d** or **D2d** operation only.
31:24	**reserved**	Reserved for future use.
23:20	**SrcStatus**	TBD.
19:16	**DstStatus**	TBD.
15:00	**XfrSz**	Bits [15:00] of the residual dma size expressed in bytes. This value will be zero if the dma operation was successful

FIG. 105

bit	name	description
31:00	**ChDn**	Each bit represents the done flag for the respective dma channel. These bits are set by a dma sequencer upon completion of the channel command. Cleared when the processor writes 0 to the corresponding **ChCmd** register **ChCmdOp** field.

FIG. 106

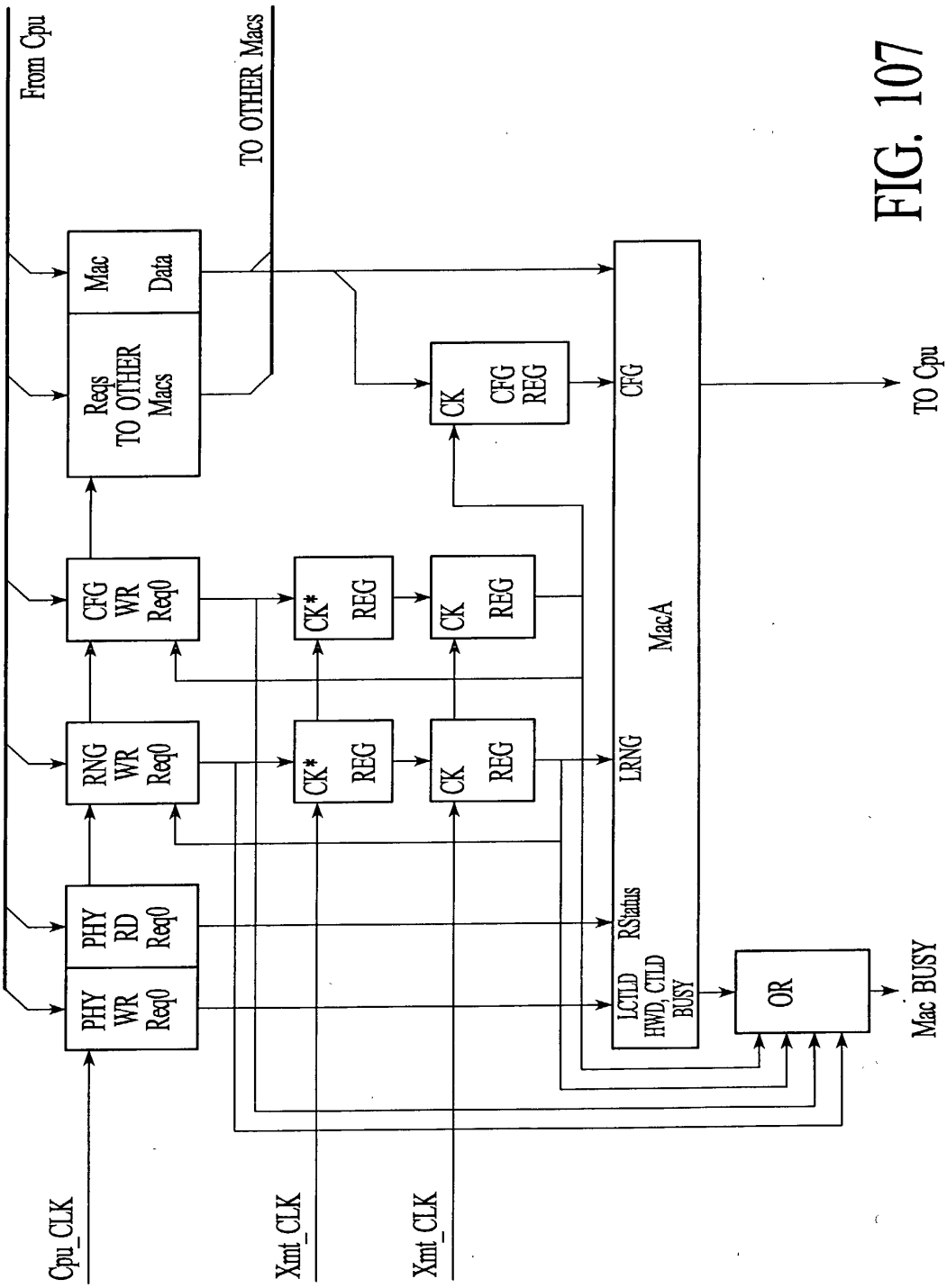

From Cpu
TO OTHER Macs
Mac Data
Reqs TO OTHER Macs
CFG WR Req0
RNG WR Req0
PHY RD Req0
PHY WR Req0
CK* REG
CK REG
CK CFG REG
CFG
LRNG
MacA
LCTLD HWD, CTLD BUSY
RStatus
OR
TO Cpu
Mac BUSY
Cpu_CLK
Xmt_CLK
Xmt_CLK

FIG. 107

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence, post-office address, and citizenship are as stated below next to my name. I believe I am the original, first and sole inventor (if only one name is listed below), or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought by way of the application entitled:

"FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION"

which (check) X is attached hereto.
____ and is amended by the Preliminary Amendment attached hereto.
____ was filed on _____, as Application Serial No. _____.
____ and was amended on _____ (if applicable).

I hereby state that I have reviewed and understood the contents of the above-identified specification, including the claims, as amended by any amendment referred to above. I acknowledge the duty to disclose all information which is material to patentability as defined in 37 CFR 1.56.

Foreign Application(s) and/or Claim of Foreign Priority

I hereby claim foreign priority benefits under Title 35, United States Code Section 119(a)-(d), of any foreign application(s) for patent or inventor's certificate, or any PCT international application(s) designating at least one country other than the United States of America listed below, and have also identified below any foreign application(s) for patent or inventor's certificate or an PCT international application(s) designating at least one country other than the United States of America filed by me on the same subject matter having a filing date before that of the application(s) on which priority is claimed:

APPLICATION NUMBER	COUNTRY	DAY/MONTH/YEAR FILED	PRIORITY CLAIMED UNDER 35 U.S.C. 119
N/A			YES NO
			YES NO

Provisional Application

I hereby claim the benefit under Title 35, United States Code Section 119(e) of any United States provisional application(s) listed below:

APPLICATION SERIAL NUMBER	FILING DATE
60/061,809	October 14, 1997
60/098,296	August 27, 1998

U.S. Priority Claim

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) or PCT international application(s) designating the United States of America listed on the following page and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application(s) in the manner provided by the first paragraph of Title 35, United States Code Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which became available between the filing date of the prior application(s) and the national or PCT international filing date of this application:

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 246

APPLICATION SERIAL NUMBER	FILING DATE	STATUS (patented/pending/abandoned)
09/067,544	April 27, 1998	Patented
09/141,713	August 28, 1998	Pending
09/384,792	August 27, 1999	Pending
09/416,925	October 13, 1999	Pending
09/439,603	November 12, 1999	Patented
09/464,283	December 15, 1999	Pending
09/514,425	February 28, 2000	Pending
09/675,484	September 29, 2000	Pending
09/675,700	September 29, 2000	Pending
09/789,366	February 20, 2001	Pending
09/801,488	March 7, 2001	Pending
09/802,426	March 9, 2001	Pending
09/802,550	March 9, 2001	Pending
09/802,551	March 9, 2001	Pending
09/855,979	March 9, 2001	Pending
09/970,124	March 9, 2001	Pending

Power of Attorney
As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) listed below to prosecute this application and transact all business in the Patent and Trademark Office connected therewith.

Mark A. Lauer, Reg. No. 36,578 **T. Lester Wallace, Reg. No. 34,748**

Send Correspondence to:	**Direct Telephone Calls To:**
Mark Lauer **7041 Koll Center Parkway, Suite 280** **Pleasanton, California 94566**	**Mark Lauer** **Tel: (925) 484-9295** **Fax: (925) 484-9291**

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereof.

Full Name of Inventor: Laurence B. Boucher **Citizenship:** United States of America

Residence: 20605 Montalvo Heights Drive
Saratoga, CA 95070

Post Office Address: Same as above

_____ _____
Inventor's Signature **Date**

[redacted]

Full Name of Inventor: Stephen E. J. Blightman **Citizenship:** United Kingdom

Residence: 3733 Arlen Court
San Jose, CA 95132.

Post Office Address: Same as above

Inventor's Signature **Date**

Full Name of Inventor: Peter K. Craft **Citizenship:** United States of America

Residence: 156 Henry Street
San Francisco, CA 94114

Post Office Address: Same as above

Inventor's Signature **Date**

Full Name of Inventor: David A. Higgen **Citizenship:** United Kingdom

Residence: 17880 Los Alamos Drive
Saratoga, CA 95070

Post Office Address: Same as above

David A Higgen 21 Feb 2002

Inventor's Signature **Date**

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 248

Full Name of Inventor: Clive M. Philbrick

Citizenship: Australia

Residence: 1170 Roycott Way
San Jose, CA 95125

Post Office Address: Same as above

Inventor's Signature **Date**

Full Name of Inventor: Daryl D. Starr

Citizenship: United States of America

Residence: 446 Folsom Court
Milpitas, CA 95035

Post Office Address: Same as above

Inventor's Signature Date

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 249

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence, post-office address, and citizenship are as stated below next to my name. I believe I am the original, first and sole inventor (if only one name is listed below), or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought by way of the application entitled:

"FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION"

which (check) _X_ is attached hereto.
____ and is amended by the Preliminary Amendment attached hereto.
____ was filed on _____, as Application Serial No. _____.
____ and was amended on _____ (if applicable).

I hereby state that I have reviewed and understood the contents of the above-identified specification, including the claims, as amended by any amendment referred to above. I acknowledge the duty to disclose all information which is material to patentability as defined in 37 CFR 1.56.

Foreign Application(s) and/or Claim of Foreign Priority

I hereby claim foreign priority benefits under Title 35, United States Code Section 119(a)-(d), of any foreign application(s) for patent or inventor's certificate, or any PCT international application(s) designating at least one country other than the United States of America listed below, and have also identified below any foreign application(s) for patent or inventor's certificate or an PCT international application(s) designating at least one country other than the United States of America filed by me on the same subject matter having a filing date before that of the application(s) on which priority is claimed:

APPLICATION NUMBER	COUNTRY	DAY/MONTH/YEAR FILED	PRIORITY CLAIMED UNDER 35 U.S.C. 119
N/A			YES ___ NO ___
			YES ___ NO ___

Provisional Application

I hereby claim the benefit under Title 35, United States Code Section 119(e) of any United States provisional application(s) listed below:

APPLICATION SERIAL NUMBER	FILING DATE
60/061,809	October 14, 1997
60/098,296	August 27, 1998

U.S. Priority Claim

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) or PCT international application(s) designating the United States of America listed on the following page and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application(s) in the manner provided by the first paragraph of Title 35, United States Code Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which became available between the filing date of the prior application(s) and the national or PCT international filing date of this application:

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 250

APPLICATION SERIAL NUMBER	FILING DATE	STATUS (patented/pending/abandoned)
09/067,544	April 27, 1998	Patented
09/141,713	August 28, 1998	Pending
09/384,792	August 27, 1999	Pending
09/416,925	October 13, 1999	Pending
09/439,603	November 12, 1999	Patented
09/464,283	December 15, 1999	Pending
09/514,425	February 28, 2000	Pending
09/675,484	September 29, 2000	Pending
09/675,700	September 29, 2000	Pending
09/789,366	February 20, 2001	Pending
09/801,488	March 7, 2001	Pending
09/802,426	March 9, 2001	Pending
09/802,550	March 9, 2001	Pending
09/802,551	March 9, 2001	Pending
09/855,979	March 9, 2001	Pending
09/970,124	March 9, 2001	Pending

Power of Attorney
As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) listed below to prosecute this application and transact all business in the Patent and Trademark Office connected therewith.

Mark A. Lauer, Reg. No. 36,578 **T. Lester Wallace, Reg. No. 34,748**

Send Correspondence to:	**Direct Telephone Calls To:**
Mark Lauer **7041 Koll Center Parkway, Suite 280** **Pleasanton, California 94566**	**Mark Lauer** **Tel: (925) 484-9295** **Fax: (925) 484-9291**

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereof.

Full Name of Inventor: Laurence B. Boucher **Citizenship:** United States of America

Residence: 20605 Montalvo Heights Drive
Saratoga, CA 95070

Post Office Address: Same as above

Laurence B. Boucher
Inventor's Signature

2/27/02
Date

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 251

Full Name of Inventor: Stephen E. J. Blightman

Citizenship: United Kingdom

Residence: 3733 Arlen Court
San Jose, CA 95132

Post Office Address: Same as above

Stephen EJ Blightman
Inventor's Signature

2/27/02
Date

Full Name of Inventor: Peter K. Craft

Citizenship: United States of America

Residence: 156 Henry Street
San Francisco, CA 94114

Post Office Address: Same as above

Inventor's Signature

3/26/02
Date

Full Name of Inventor: David A. Higgen

Citizenship: United Kingdom

Residence: 17880 Los Alamos Drive
Saratoga, CA 95070

Post Office Address: Same as above

Inventor's Signature

Date

[redacted]

Full Name of Inventor: Clive M. Philbrick

Citizenship: Australia

Residence: 1170 Roycott Way
San Jose, CA 95125

Post Office Address: Same as above

C M Philbrick
Inventor's Signature

2/26/02
Date

Full Name of Inventor: Daryl D. Starr

Citizenship: United States of America

Residence: 446 Folsom Court
Milpitas, CA 95035

Post Office Address: Same as above

Daryl D. Starr
Inventor's Signature

2/25/02
Date

[redacted]

PATENT APPLICATION SERIAL NO. _____

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

10/03/2002 DTESSEM1 00000026 10260878

01 FC:101 740.00 OP
02 FC:103 72.00 OP

PTO-1556
(5/87)

*U.S. Government Printing Office: 2002 — 489-267/69033

PATENT APPLICATION FEE DETERMINATION RECORD
Effective October 1, 2001

Application or Docket Number: 10260878

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS	24	
FOR	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	24 minus 20=	* 4
INDEPENDENT CLAIMS	2 minus 3 =	* 0
MULTIPLE DEPENDENT CLAIM PRESENT		☐

* If the difference in column 1 is less than zero, enter "0" in column 2

SMALL ENTITY TYPE ☐		OR	OTHER THAN SMALL ENTITY	
RATE	FEE		RATE	FEE
BASIC FEE	370.00	OR	BASIC FEE	740.00
X$ 9=		OR	X$18=	72
X42=		OR	X84=	
+140=		OR	+280=	
TOTAL		OR	TOTAL	812

CLAIMS AS AMENDED - PART II

AMENDMENT A

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X42=		OR	X84=	
+140=		OR	+280=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X42=		OR	X84=	
+140=		OR	+280=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X42=		OR	X84=	
+140=		OR	+280=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
***If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 255

MULTIPLE DEPENDENT CLAIM FEE CALCULATION SHEET
(FOR USE WITH FORM PTO-875)

SERIAL NO.	FILING DATE
APPLICANT(S)	

CLAIMS

	AS FILED		AFTER 1st AMENDMENT		AFTER 2nd AMENDMENT	
	IND.	DEP.	IND.	DEP.	IND.	DEP.
1	1					
2						
3						
4						
5						
6						
7						
8						
9	1					
10						
11						
12						
13						
14						
15						
16						
17	1					
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						
29						
30						
31						
32						
33						
34						
35						
36						
37						
38						
39						
40						
41						
42						
43						
44						
45						
46						
47						
48						
49						
50						
TOTAL IND.	3					
TOTAL DEP.	21					
TOTAL CLAIMS	24					

	*		*		*	
	IND.	DEP.	IND.	DEP.	IND.	DEP.
51						
52						
53						
54						
55						
56						
57						
58						
59						
60						
61						
62						
63						
64						
65						
66						
67						
68						
69						
70						
71						
72						
73						
74						
75						
76						
77						
78						
79						
80						
81						
82						
83						
84						
85						
86						
87						
88						
89						
90						
91						
92						
93						
94						
95						
96						
97						
98						
99						
100						
TOTAL IND.						
TOTAL DEP.						
TOTAL CLAIMS						

PTO-1360 (3-78)

*MAY BE USED FOR ADDITIONAL CLAIMS OR AMENDMENTS U.S. DEPARTMENT of COMMERCE

ARTIFACT SHEET

Enter artifact number below. Artifact number is application number + artifact type code (see list below) + sequential letter (A, B, C ...). The first artifact folder for an artifact type receives the letter A, the second B, etc.. Examples: 59123456PA, 59123456PB, 59123456ZA, 59123456ZB

10260878 P A

Indicate quantity of a single type of artifact received but not scanned. Create individual artifact folder/box and artifact number for each Artifact Type.

☑ CD(s) containing:
computer program listing ☑
Doc Code: Computer Artifact Type Code: P

pages of specification and/or sequence listing and/or table ☐
Doc Code: Artifact Artifact Type Code: S

content unspecified or combined ☐
Doc Code: Artifact Artifact Type Code: U

☐ Stapled Set(s) Color Documents or B/W Photographs
Doc Code: Artifact Artifact Type Code: C

☐ Microfilm(s)
Doc Code: Artifact Artifact Type Code: F

☐ Video tape(s)
Doc Code: Artifact Artifact Type Code: V

☐ Model(s)
Doc Code: Artifact Artifact Type Code: M

☐ Bound Document(s)
Doc Code: Artifact Artifact Type Code: B

☐ Confidential Information Disclosure Statement or Other Documents marked Proprietary, Trade Secrets, Subject to Protective Order, Material Submitted under MPEP 724.02, etc.
Doc Code: Artifact Artifact Type Code X

☐ Other, description: _____
Doc Code: Artifact Artifact Type Code: Z

March 8, 2004

OIPE
JUL 15 2003
PATENT & TRADEMARK OFFICE

2154
#2
SP
7-23-03

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of Laurence B. Boucher, et al.		Ser. No:	10/260,878
Filing Date:	September 27, 2002	Examiner:	Unknown
Atty. Docket No:	ALA-006E	GAU:	2154

For: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

RECEIVED
JUL 17 2003
Technology Center 2100

July 11, 2003

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Information Disclosure Statement per 37 C.F.R. §1.98

Sir:

Pursuant to 37 C.F.R. §§ 1.56, 1.97 and 1.98, applicants bring one hundred and forty-six documents listed on the enclosed fourteen-page form PTO-1449 to the attention of the Examiner in the above-identified application.

Citation of these documents shall not be construed as an admission that the documents are prior art with respect to the instant invention, a representation that a search has been made, or an admission that the information cited herein is, or is considered to be, material to patentability as defined in 37 C.F.R. § 1.56(b). Copies of the documents listed on the enclosed fourteen-page form PTO-1449 are not submitted because they were submitted in an earlier application (09/801,488,) which is relied upon for an earlier filing date under 35 U.S.C. §120.

Respectfully submitted,

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on July 11, 2003.

Date: 7-11-03

Mark Lauer

Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE JC16
JUL 15 2003
PATENT & TRADEMARK OFFICE

RECEIVED
JUL 17 2003
Technology Center 2100

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	4,366,538	12/28/82	Johnson et al.	364	200	
	B	4,991,133	05/13/86	Davis et al.	364	900	
	C	5,056,058	10/08/91	Hirata et al.	709	230	
	D	5,097,442	03/17/92	Ward et al.	365	78	
	E	5,163,131	11/10/92	Row et al.	395	200	
	F	5,212,778	05/18/93	Dally et al.	395	400	
	G	5,280,477	01/18/94	Trapp	370	85.1	
	H	5,289,580	02/22/94	Latif et al.	395	275	
	I	5,303,344	04/12/94	Yokoyama et al.	395	275	
	J	5,412,782	05/02/95	Hausman et al.	395	250	
	K	5,448,566	09/05/95	Richter et al.	370	94.1	

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L	WO 00/13091	03/09/00	PCT/US98/24943				
	M	WO 99/65219	12/16/99	PCT/US99/13184				

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	Internet pages entitled "Hardware Assisted Protocol Processing", (which Eugene Feinberg is working on), 1 page, printed 11/25/98.
	O	Zilog product Brief entitled "Z85C30 CMOS SCC Serial Communication Controller", Zilog Inc., 3 pages, 1997.
	P	Internet pages of Xpoint Technologies, Inc. entitled "Smart LAN Work Requests", 5 pages, printed 12/19/97.
	Q	Internet pages entitled: Asante and 100BASE-T Fast Ethernet, 7 pages, printed 5/27/97.

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE
JUL 15 2003
PATENT & TRADEMARK OFFICE

RECEIVED
JUL 17 2003
Technology Center 2100

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	5,485,579	01/16/96	Hitz et al.	395	200.12	
	B	5,506,966	04/09/96	Ban	395	250	
	C	5,511,169	04/23/96	Suda	395	280	
	D	5,548,730	08/20/96	Young et al.	395	280	
	E	5,588,121	12/24/96	Reddin et al.	395	200.15	
	F	5,566,170	10/15/96	Bakke et al	370	60	
	G	5,590,328	12/31/96	Seno et al.	395	675	
	H	5,592,622	01/07/97	Isfeld et al.	395	200.02	
	I	5,629,933	05/13/97	Delp et al.	370	411	
	J	5,634,099	05/27/97	Andrews et al	395	200.07	
	K	5,634,127	05/27/97	Cloud et al.	395	680	

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L	WO 01/05107 A1	01/18/01	PCT/US00/19006				
	M	WO 01/05116 A2	01/18/01	PCT/US00/19243				

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	Internet pages entitled: A Guide to the Paragon XP/S-A7 Supercomputer at Indiana University, 13 pages, printed 12/21/98.
	O	Richard Stevens, "TCP/IP Illustrated, Volume 1, The Protocols", pages 325-326 (1994).
	P	Internet pages entitled: Northridge/Southbridge vs. Intel Hub Architecture, 4 pages, printed 2/19/01.
	Q	Gigabit Ethernet Technical Brief, Achieving End-to-End Performance, Alteon Networks, Inc., First Edition, September 1996.

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

0125 JUL 15 2003 PATENT & TRADEMARK OFFICE

RECEIVED
JUL 17 2003
Technology Center 2100

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	5,642,482	06/24/97	Pardillos	395	200.2	
	B	5,664,114	09/02/97	Krech, Jr. et al.	395	200.64	
	C	5,671,355	09/23/97	Collins	395	200.2	
	D	5,678,060	10/14/97	Yokoyama et al.	709	212	
	E	5,692,130	11/25/97	Shobu et al.	395	200.12	
	F	5,699,317	12/16/97	Sartore et al.	395	230.06	
	G	5,701,434	12/23/97	Nakagawa	395	484	
	H	5,701,516	12/23/97	Cheng et al.	395	842	
	I	5,749,095	05/05/98	Hagersten	711	141	
	J	5,751,715	05/12/98	Chan et al.	370	455	
	K	5,752,078	05/12/98	Delp et al.	395	827	

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L	WO 01/05123 A1	01/18/01	PCT/US00/18976				
	M	WO 01/40960A1	06/07/01	PCT/US00/32660				

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	Internet pages directed to Technical Brief on Alteon Ethernet Gigabit NIC technology, www.alteon.com, 14 pages, printed 3/15/97.
	O	VIA Technologies, Inc. article entitled "VT8501 Apollo MVP4", pages i-iv, 1-11, cover and copyright page, revision 1.3, Feb. 1, 2000.
	P	iReady News Archives article entitled "iReady Rounding Out Management Team with Two Key Executives", http://www.ireadyco.com/archives/keyexec.html, 2 pages, printed 11/28/98.
	Q	"Toshiba Delivers First Chips to Make Consumer Devices Internet-Ready Based On iReady's Design," Press Release October, 1998, 3 pages, printed 11/28/98.

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

JUL 15 2003 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	5,758,084	05/26/98	Silverstein et al.	395	200.58	
	B	5,758,089	05/26/98	Gentry et al.	395	200.64	
	C	5,758,186	05/26/98	Hamilton et al.	395	831	
	D	5,758,194	05/26/98	Kuzma	395	886	
	E	5,771,349	06/23/98	Picazo, Jr. et al.	395	188.01	
	F	5,790,804	08/04/98	Osborne	395	200.75	
	G	5,794,061	08/11/98	Hansen et al.	395	800.01	
	H	5,802,580	09/01/98	McAlpice	711	149	
	I	5,809,328	09/15/98	Nogales et al.	395	825	
	J	5,812,775	09/22/98	Van Seeters et al.	395	200.43	
	K	5,815,646	09/29/98	Purcell et al.	395	163	

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L	WO 01/04770 A2	01/18/01	PCT/US00/18939				
	M	WO/98/19412	05/07/98	PCT/US97/17257				

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	Internet pages from iReady Products, web sitehttp://www.ireadyco.com/products,html, 2 pages, downloaded 11/25/98.
	O	iReady News Archives, Toshiba, iReady shipping Internet chip, 1 page, printed 11/25/98.
	P	Interprophet article entitled "Technology", http://www.interprophet.com/technology.html, 17 pages, printed 3/1/00.
	Q	iReady Corporation, article entitled "The I-1000 Internet Tuner", 2 pages, date unknown.

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE 3600
JUL 15 2003
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	5,878,225	03/02/99	Bilansky et al.	395	200.57	
	B	5,913,028	06/15/99	Wang et al.	395	200.33	
	C	5,930,830	07/27/99	Mendelson et al.	711	171	
	D	5,931,918	08/03/99	Row et al	709	300	
	E	5,935,205	08/10/99	Murayama et al.	709	216	
	F	5,937,169	08/10/99	Connery et al.	395	200.8	
	G	5,941,969	08/24/99	Ram et al.	710	128	
	H	5,941,972	08/24/99	Hoese et al.	710	129	
	I	5,950,203	09/07/99	Stakuis et al.	707	10	
	J	5,991,299	11/23/99	Radogna et al.	370	392	
	K	5,996,024	11/30/99	Blumenau	709	301	

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L	WO/98/50852	11/12/98	PCT/US98/08719				
	M	WO/99/04343	01/28/99	PCT/US98/14729				

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	iReady article entitled "About Us Introduction", Internet pages fromhttp://www.iReadyco.com/about.html, 3 pages, printed 11/25/98.
	O	iReady News Archive article entitled "Revolutionary Approach to Consumer Electronics Internet Connectivity Funded", San Jose, CA, November 20,1997. 2 pages, printed 11/2/98.
	P	iReady News Archive article entitled "Seiko Instruments Inc. (SII) INTRODUCES WORLD'S FIRST INTERNET-READY INTELLIGENT LCD MODULES BASED ON IREADY TECHNOLOGY," Santa Clara, CA and Chiba, Japan, October 26, 1998, 2 pages, printed 11/2/98.
	Q	NEWSwatch article entitled "iReady internet Tuner to Web Enable Devices", Tuesday, November 5, 1996, printed 11/2/98.

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE JC16
JUL 15 2003
PATENT & TRADEMARK OFFICE

RECEIVED
JUL 17 2003
Technology Center 2100

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	6,005,849	12/21/99	Roach et al.	370	276	
	B	6,009,478	12/28/99	Panner et al.	710	5	
	C	6,016,513	01/18/00	Lowe	709	250	
	D	6,026,452	02/15/00	Pitts	710	56	
	E	6,034,963	03/07/00	Minami et al.	370	401	
	F	6,044,438	03/28/00	Olnowich	711	130	
	G	6,047,356	04/04/00	Anderson et al.	711	129	
	H	6,057,863	05/02/00	Olarig	345	520	
	I	6,061,368	05/09/00	Hitzelberger	370	537	
	J	6,065,096	05/16/00	Day et al.	711	114	
	K	6,141,705	10/31/00	Anand et al.	710	15	

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	EETimes article entitled "Tuner for Toshiba, Toshiba Taps iReady for Internet Tuner", by David Lammers, 2 pages, printed 11/02/98.
	O	"Comparison of Novell Netware and TCP/IP Protocol Architectures", by J.S. Carbone, 19 pages, printed 4/10/98.
	P	Adaptec article entitled "AEA-7110C-a DuraSAN product", 11 pages, printed 10/1/01.
	Q	iSCSI HBA article entitled "iSCSI and 2Gigabit fibre Channel Host Bus Adapters from Emulex, QLogic, Adaptec, JNI", 8 pages, printed 10/01/01.

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE
JUL 15 2003
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	6,226,680	05/01/01	Boucher et al.	709	230	
	B	6,246,683	06/12/01	Connery et al.	370	392	
	C	6,247,060	06/12/01	Boucher et al.	709	238	
	D	60/053,240		Jolitz et al.			07/18/97
	E	6,345,301	02/05/02	Burns et al.	709	230	
	F						
	G						
	H						
	I						
	J						
	K						

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	iSCSI HBA article entitled "FCE-3210/6410 32 and 64-bit PCI-to-Fibre Channel HBA", 6 pages, printed 10/01/01.
	O	ISCSI.com article entitled "iSCSI Storage", 2 pages, printed 10/01/01.
	P	"Two-Way TCP Traffic Over Rate Controlled Channels: Effects and Analysis", by Kalampoukas et al., IEEE Transactions on Networking, vol. 6, no. 6, December 1998.
	Q	IReady News article entitled "Toshiba Delivers First Chips to Make Consumer Devices Internet-Ready Based on iReady Design", Santa Clara, CA, and Tokyo, Japan, October 14, 1998, printed 11/2/98.

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE
JUL 15 2003
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	United States Patent Application No. 08/964,304, by Napolitano, et al., entitled "File Array Storage Architecture", filed 11/04/97.
	O	"File System Design For An NFS File Server Appliance", Article by D. Hitz, et al., 13 pages.
	P	Adaptec Press Release article entitled "Adaptec Announces EtherStorage Technology", 2 pages, May 4, 2000, printed 6/14/00.
	Q	Adaptec article entitled "EtherStorage Frequently Asked Questions", 5 pages, printed 7/19/00.

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE SC16
JUL 15 2003
PATENT & TRADEMARK OFFICE

RECEIVED
JUL 17 2003
Technology Center 2100

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	Adaptec article entitled "EtherStorage White Paper", 7 pages, printed 7/19/00.
	O	CIBC World Markets article entitled "Computers; Storage", by J. Berlino et al., 9 pages, dated August 7, 2000.
	P	Merrill Lynch article entitled "Storage Futures", by S. Milunovich, 22 pages, dated May 10, 2000.
	Q	CBS Market Watch article entitled "Montreal Start-Up Battles Data Storage Botttleneck", by S. Taylor, dated March 5, 2000, 2 pages, printed 3/7/00.

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE SC100
JUL 15 2003
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	Internet-draft article entitled "SCSI/TCP (SCSI over TCP)", by J. Satran et al., 38 pages, dated February 2000, printed 5/19/00.
	O	Internet pages entitled "Technical White Paper-Xpoint's Disk to LAN Acceleration Solution for Windows NT Server," 16 pages, printed 6/5/97.
	P	Jato Technologies article entitled "Network Accelerator Chip Architecture," twelve-slide presentation, printed 8/19/98.
	Q	EETimes article entitled "Enterprise System Uses Flexible Spec," dated August 10,1998, printed 11/25/98.

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 268

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE
JUL 15 2003
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	Internet pages entitled "Smart Ethernet Network Interface Cards", which Berend Ozceri is developing, printed 11/25/98.
	O	Internet pages of Xaqti corporation entitled "GigaPower Protocol Processor Product Review," printed 11/25/99.
	P	Internet pages entitled "DART: Fast Application Level Networking via Data-Copy Avoidance," by Robert J. Walsh, printed 6/3/99.
	Q	

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

PTO/SB/08A
JUL 15 2003
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	J							
	K							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	L	Internet pages of InterProphet entitled "Frequently Asked Questions", by Lynne Jolitz, printed 6/14/00.
	M	Internet pages entitled "iReady Products," printed 11/25/98.
	N	Andrew S. Tanenbaum, "Computer Networks," Third Edition, 1996, ISBN 0-13-349945-6.
	O	Form 10-K for Exelan, Inc., for the fiscal year ending December 31, 1987 (10 pages).
	P	Form 10-K for Exelan, Inc., for the fiscal year ending December 31, 1988 (10 pages).
	Q	"Second Supplemental Information Disclosure Statement per 37 C.F.R. §1.97(i)", dated July 29, 2002 relating to Exelan Inc. as submitted in Application Serial No. 09/464,283.

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE
JUL 15 2003
PATENT & TRADEMARK OFFICE

RECEIVED
JUL 17 2003
Technology Center 2100

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	6,173,333	07/18/97	Jolitz et al.			
	B	2001/0025315A1		Jolitz			01/10/01
	C	2001/0004354A1		Jolitz			01/10/01
	D						
	E						
	F						
	G						
	H						
	I						

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	J							
	K							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	L	WindRiver article entitled "Tornado: For Intelligent Network Acceleration", copyright Wind River Systems, 2001, 2 pages.
	M	WindRiver White Paper entitled "Complete TCP/IP Offload for High-Speed Ethernet Networks", Copyright Wind River Systems, 2002, 7 pages.
	N	Intel article entitled "Solving Server Bottlenecks with Intel Server Adapters", Copyright Intel Corporation, 1999, 8 pages.
	O	
	P	

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE IDS
JUL 15 2003
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	5,058,110	10/15/91	Beach et al.	370	85.6	
	B	6,021,446	02/01/00	Gentry, Jr.	709	303	
	C	6,356,951	03/12/02	Gentry, Jr.	709	250	
	D	6,389,468	05/14/02	Muller et al.	709	226	
	E	6,427,169	07/30/02	Elzur	709	224	
	F	6,434,651	08/13/02	Gentry, Jr.	710	260	
	G	6,449,656	09/10/02	Elzur et al.	709	236	
	H	6,453,360	09/17/02	Muller et al.	709	250	
	I						
	J						
	K						

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	Article from Rice University, Department of Computer Science entitled "Lazy Receiver Processing (LRP): A New Network Subsystem Architecture for Server Systems", by Peter Druschel and Gaurav Banga, 15 pages.
	O	Internet RFC/STD/FYI/BCP Archives article with heading "RFC2140" entitled "TCP Control Block Interdependence", web address http://www.faqs.org/rfcs/rfc2140.html, 9 pages, printed 9/20/02.
	P	
	Q	

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 272

OIPE JC10
JUN 10 2004
PATENT & TRADEMARK OFFICE

2154

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of Laurence B. Boucher, et al.		Ser. No:	10/260,878
Filing Date:	September 27, 2002	Examiner:	Unknown
Atty. Docket No:	ALA-006E	GAU:	2154

For: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

June 7, 2004

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Supplemental Information Disclosure Statement per 37 C.F.R. §1.98

Sir:

Pursuant to 37 C.F.R. §§ 1.56, 1.97 (e)(1), 1.98, applicants bring the following document to the Examiner's attention. Included with this letter is one U.S. Patent document that was first cited in a communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this information disclosure statement. Also included is a one-page form PTO-1449 listing this document.

Citation of this document shall not be construed as an admission that the document is prior art with respect to the instant invention, a representation that a search has been made, or an admission that the information cited herein is, or is considered to be, material to patentability as defined in 37 C.F.R. § 1.56(b).

Respectfully submitted,

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Commissioner for Patents, P.O. Box 1450 Alexandria, VA 22313-1450, on June 7, 2004.

Date: 6-7-04

Mark Lauer

Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

O I P E
JUN 10 2004
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	5,598,410	1/28/1997	Stone	370	469	
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	
	O	
	P	
	Q	

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

OIPE JC99
JUN 24 2004
PATENT & TRADEMARK OFFICE

2154
IFW

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of Laurence B. Boucher, et al.		Ser. No:	10/260,878
Filing Date:	September 27, 2002	Examiner:	Unknown
Atty. Docket No:	ALA-006E	GAU:	2154

For: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

June 21, 2004

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

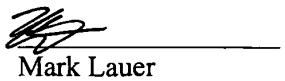
2nd Supplemental Information Disclosure Statement per 37 C.F.R. §1.98

Sir:

Pursuant to 37 C.F.R. §§ 1.56, 1.97 and 1.98, applicants bring the following document to the Examiner's attention. Included is one U.S. Patent reference document and a one-page form PTO-1449 listing this document.

Citation of this document shall not be construed as an admission that the document is prior art with respect to the instant invention, a representation that a search has been made, or an admission that the information cited herein is, or is considered to be, material to patentability as defined in 37 C.F.R. § 1.56(b).

Respectfully submitted,

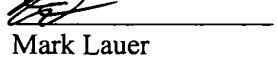


Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Commissioner for Patents, P.O. Box 1450 Alexandria, VA 22313-1450, on June 21, 2004.

Date: 6-21-04



Mark Lauer

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
2nd SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

JUN 24 2004 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	5,517,668	5/14/1996	Szwerinski et al.	395	800	
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	
	O	
	P	
	Q	

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 276

CIFE JC1P4
MAR 09 2005
PATENT & TRADEMARK OFFICE

2154

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of Laurence B. Boucher, et al.		Ser. No:	10/260,878
Filing Date:	September 27, 2002	Examiner:	Unknown
Atty. Docket No:	ALA-006E	GAU:	2154

For: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

February 24, 2005

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

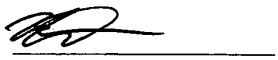
3rd Supplemental Information Disclosure Statement

Sir:

Pursuant to 37 C.F.R. §§ 1.56, 1.97 and 1.98, applicants bring sixteen documents to the Examiner's attention. Included are copies of nine non-patent reference documents, and a one-page form PTO-1449 listing these documents separately from seven U.S. Patent reference documents. Copies of the seven U.S. Patent reference documents are not enclosed.

Citation of these documents shall not be construed as an admission that the documents are prior art with respect to the instant invention, a representation that a search has been made, or an admission that the information cited herein is, or is considered to be, material to patentability as defined in 37 C.F.R. § 1.56(b).

Respectfully submitted,

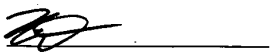


Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Commissioner for Patents, P.O. Box 1450 Alexandria, VA 22313-1450, on February 24, 2005.

Date: 2-24-05



Mark Lauer

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

O1P E
MAR 0 9 2005
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	5,524,250	6/4/96	Chesson et al.	395	775	
	B	5,619,650	4/8/97	Bach et al.	395	200.01	
	C	5,727,142	3/10/98	Chen	395	181	
	D	5,802,258	9/1/98	Chen	395	182.08	
	E	5,898,713	4/27/99	Melzer et al.	371	53	
	F	6,021,507	2/1/00	Chen	714	2	
	G	6,047,323	4/4/00	Krause	709	227	
	H						

OTHER ART—NON PATENT LITERATURE DOCUMENTS

*Examiner Initial	Cite No.	(Including Author, Title, Date, Pertinent Pages, Etc.)
	1	Schwaderer et al., IEEE Computer Society Press publication entitled, "XTP in VLSI Protocol Decomposition for ASIC Implementation", from 15th Conference on Local Computer Networks, 5 pages, Sept. 30 – Oct. 3, 1990.
	2	Beach, Bob, IEEE Computer Society Press publication entitled, "UltraNet: An Architecture for Gigabit Networking", from 15th Conference on Local Computer Networks, 18 pages, Sept. 30 – Oct. 3, 1990.
	3	Chesson et al., IEEE Syposium Record entitled, "The Protocol Engine Chipset", from Hot Chips III, 16 pages, Aug. 26-27, 1991.
	4	Maclean et al., IEEE Global Telecommunications Conference, Globecom '91, presentation entitled, "An Outboard Processor for High Performance Implementation of Transport Layer Protocols", 7 pages, Dec. 2-5, 1991.
	5	Ross et al., IEEE article entitled "FX1000: A high performance single chip Gigabit Ethernet NIC", from Compcon '97 Proceedings, 7 pages, Feb. 23-26, 1997.
	6	Strayer et al., "Ch. 9: The Protocol Engine" from XTP: The Transfer Protocol, 12 pages, July 1992.
	7	Publication entitled "Protocol Engine Handbook", 44 pages, Oct. 1990.
	8	Koufopavlou et al., IEEE Global Telecommunications Conference, Globecom '92, presentation entitled, "Parallel TCP for High Performance Communication Subsystems", 7 pages, Dec. 6-9, 1992.
	9	Lilienkamp et al., Publication entitled "Proposed Host-Front End Protocol", 56 pages, Dec. 1984.

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L5	10	(((session OR application OR presentation OR upper) ADJ layer) NEAR2 header) AND ((tcp OR "transport control") NEAR2 connection) AND (header NEAR2 (template OR default))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/18 09:30
L6	86	(((session OR application OR presentation OR upper) ADJ layer) NEAR2 header) AND ((tcp OR "transport control") NEAR2 connection)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/18 10:14
L7	9	(((transport OR tcp) ADJ layer) NEAR2 header) SAME ((tcp OR "transport control") NEAR2 connection)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/18 10:03
L8	155	(((transport OR tcp) ADJ layer) NEAR2 header) AND ((tcp OR "transport control") NEAR2 connection)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/18 10:03
L9	157	(((session OR application OR presentation OR upper) ADJ layer) NEAR2 header) SAME (destination OR address)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/18 13:23
L13	269	(((session OR application OR presentation OR upper) ADJ layer) NEAR2 header) AND ((destination OR address) WITH header)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/18 11:27
L14	120	(((session OR application OR presentation OR upper) ADJ layer) NEAR2 header) AND ((destination OR address) WITH header) NOT L9	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/18 11:27
L15	1920	(header WITH (template OR default))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/18 13:24
L16	8	((prepend OR append OR attach) WITH header WITH (template OR default))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/18 13:37
L17	121	((write OR form) WITH header WITH (template OR default))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/18 13:37
S5	1	"260878".apn.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/13 13:59

S6	48	("5983271" "6141705" "5937169" "6161123" "5619650" "5841764" "5898713" "5940598" "6219697" "6226680" "6219697" "6226680" "6247060" "6400712" "6405247" "6415313" "6424650" "6591302" "6963921" "6965941" "5684954" "5309437" "5805572" "5706508" "5727142" "5931916" "5941988" "6021507" "6032183" "6073180" "6078733" "6324183" "6356951" "6389468" "6434620" "6453360" "6480489" "6483804" "6606301" "6650640" "6658002" "6721806" "6751665" "6823437" "6907042" "6920493" "6938092" "6947430" "5995741" "6370599").pn.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/13 15:25
S7	50	("6904519" "6208620" "6134245" "6229823" "5307413" "5682534" "5764645" "5793958" "5923659" "6084892" "6091710" "6097719" "6181695" "6195425" "6236652" "6246683" "6262976" "6292479" "6330250" "6430595" "6571272" "6625170" "6697352" "6714541" "6738361" "6771673" "6779033" "6826620" "6850495" "6880017" "6885678" "6956853" "5434976" "5274768" "6065064" "6208651" "5574919" "5706429" "6134244" "6208650" "5235644" "5303344" "5386542" "5594869" "5903724" "5920703" "5983259" "5983274" "6081846" "6085215").pn.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/13 16:28
S8	49	("6088777" "6108782" "6263444" "6334153" "6353619" "6389479" "6393487" "6427173" "6449631" "6470391" "6625662" "6658480" "6798743" "6006268" "6226267" "6226267" "6463470" "6466984" "6675218" "6760304" "5740371" "6535509" "6920484" "6049833" "5619645" "6055237" "6192411" "4893307" "5021949" "5815516" "5950195" "6148410" "5777989" "5867636" "5898830" "6021263" "6052788" "6098108" "6097697" "6119170" "6122276" "6128662" "6151300" "6185617" "6195705" "5278955" "5412654" "5442633" "5550984" "5636371").pn.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/15 16:33

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 280

S9	48	("5650993" "5664116" "5673322" "5699361" "5757924" "5764756" "5774640" "5799016" "5822523" "5838682" "5907610" "5917997" "5931913" "5935215" "5944783" "5951650" "5964891" "6006264" "6018766" "6041041" "6061341" "6075796" "6104716" "6115385" "6119171" "6137792" "6147976" "6151679" "6169795" "6212175" "6226686" "6212175" "6226686" "6233249" "6233626" "6240513" "6249294" "6252857" "6263371" "6266701" "6269099" "6286047" "6314284" "6321267" "6324161" "6324582" "6345301" "6351775" "6353891" "6370144").pn.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/15 16:34
S10	81	("20010004354"|"20010025315"|"4366538"|"4991133"|"5056058"|"5058110"|"5097442"|"5163131"|"5212778"|"5280477"|"5289580"|"5303344"|"5412782"|"5448566"|"5485579"|"5506966"|"5511169"|"5548730"|"5566170"|"5588121"|"5590328"|"5592622"|"5629933"|"5634099"|"5634127"|"5642482"|"5664114"|"5671355"|"5678060"|"5692130"|"5699317"|"5701434"|"5701516"|"5749095"|"5751715"|"5752078"|"5758084"|"5758089"|"5758186"|"5758194"|"5771349"|"5790804"|"5794061"|"5802580"|"5809328"|"5812775"|"5815646"|"5878225"|"5913028"|"5930830"|"5931918"|"5935205"|"5937169"|"5941969"|"5941972"|"5950203"|"5991299"|"5996024"|"6005849"|"6009478"|"6016513"|"6021446"|"6026452"|"6034963"|"6044438"|"6047356"|"6057863"|"6061368"|"6065096"|"6141705"|"6173333"|"6226680"|"6246683"|"6247060"|"6247169"|"6345301"|"6356951"|"6389468"|"6434651"|"6449656"|"6453360").PN.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/15 18:18

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 281

S11	66	("4366538"|"4991133"|"5056058"|"5097442"|"5163131"|"5212778"|"5280477"|"5289580"|"5303344"|"5412782"|"5448566"|"5485579"|"5506966"|"5511169"|"5548730"|"5566170"|"5588121"|"5590328"|"5592622"|"5629933"|"5634127"|"5642482"|"5664114"|"5671355"|"5678060"|"5692130"|"5699317"|"5701516"|"5749095"|"5751715"|"5752078"|"5758084"|"5758089"|"5758194"|"5771349"|"5790804"|"5794061"|"5802580"|"5809328"|"5812775"|"5815646"|"5878225"|"5913028"|"5930830"|"5931918"|"5935205"|"5937169"|"5941972"|"5950203"|"5991299"|"5996024"|"6005849"|"6009478"|"6016513"|"6026452"|"6034963"|"6044438"|"6047356"|"6057863"|"6061368"|"6065096"|"6141705"|"6226680"|"6246683"|"6247060"|"6345301").PN.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/15 18:27
S12	1	("5598410").PN.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/15 18:15
S13	1	("5517668").PN.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/15 18:17
S14	36	(interrupt WITH header) AND (network ADJ layer) AND (transport ADJ layer) AND (header WITH validat$4) AND (header WITH prepend$4)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/16 12:36
S15	28	((network ADJ layer) WITH header) AND ((transport ADJ layer) WITH header) AND (header WITH validat$4) AND (header WITH prepend$4)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/16 12:38
S16	33	((network ADJ layer) WITH header) AND ((transport ADJ layer) WITH header) AND (header WITH validat$4)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/16 12:43
S17	174	((network ADJ layer) WITH header) AND ((transport ADJ layer) WITH header) AND (header WITH (validat$4 OR check OR checksum OR "crc"))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/16 13:04

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 282

S18	89	((network ADJ layer) WITH header) AND ((transport ADJ layer) WITH header) AND (((mac OR "media access" OR datalink OR "data link") ADJ layer) WITH header) AND (header WITH (validat$4 OR check OR checksum OR "crc"))	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/16 14:20
S19	152	((network ADJ layer) WITH header) AND ((transport ADJ layer) WITH header) AND (((mac OR "media access control" OR datalink OR "data link") ADJ layer) WITH header)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/16 19:20
S20	40	("5077732" | "5428615" | "5651002" | "5729543" | "5732081" | "5825774").PN. OR ("5991299"). URPN.	US-PGPUB; USPAT; USOCR	OR	ON	2006/01/16 16:42
S21	45	("5088090" | "5274631" | "5406643" | "5452294" | "5473599" | "5504866" | "5570466" | "5583996").PN. OR ("5845091"). URPN.	US-PGPUB; USPAT; USOCR	OR	ON	2006/01/16 16:50
S22	2427	(network ADJ layer) AND (transport ADJ layer) AND ((mac OR "media access control" OR datalink OR "data link") ADJ layer)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/16 17:16
S23	549	(network ADJ layer) AND (transport ADJ layer) AND ((mac OR "media access control" OR datalink OR "data link") ADJ layer) AND (protocol WITH header WITH layer)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/16 17:29
S24	15	(network ADJ layer) AND (transport ADJ layer) AND ((mac OR "media access control" OR datalink OR "data link") ADJ layer) AND (protocol WITH header WITH layer) AND (inbound SAME outbound SAME header)	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	ON	2006/01/18 09:27

S25	133	("4652874" | "4807111" | "4850042" | "4899333" | "4922503" | "4933938" | "5150358" | "5210746" | "5220562" | "5231633" | "5251205" | "5278830" | "5291482" | "5293379" | "5301333" | "5309437" | "5313454" | "5343471" | "5386413" | "5392432" | "5394402" | "5410540" | "5410722" | "5422838" | "5425028" | "5426736" | "5450399" | "5455820" | "5457681" | "5459714" | "5459717" | "5461611" | "5461624" | "5473607" | "5477537" | "5481540" | "5485455" | "5485578" | "5490139" | "5490252" | "5500860" | "5515376" | "5535202" | "5555405" | "5561666" | "5570365" | "5572522" | "5583981" | "5592476" | "5594727" | "5600641" | "5602841" | "5608726" | "5610905" | "5619500" | "5619661" | "5633865" | "5636371" | "5640605" | "5649109" | "5651002" | "5675741" | "5684800" | "5691984" | "5706472" | "5720032" | "5724358" | "5726977" | "5734865" | "5740171" | "5740175" | "5740375" | "5742604" | "5742760" | "5745048" | "5748905" | "5751967" | "5754540" | "5754801" | "5757771" | "5757795" | "5764634" | "5781549" | "5784573" | "5790546" | "5802047" | "5802052").PN. OR ("5920566").URPN.	US-PGPUB; USPAT; USOCR	OR	ON	2006/01/16 18:03
S26	58	S25 AND header AND layer AND protocol	US-PGPUB; USPAT; USOCR	OR	ON	2006/01/16 18:03

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/260,878	09/27/2002	Laurence B. Boucher	ALA-006E	9902

24501 7590 01/27/2006

MARK A LAUER
6601 KOLL CENTER PARKWAY
SUITE 245
PLEASANTON, CA 94566

EXAMINER	
KUIPER, ERIC J	

ART UNIT	PAPER NUMBER
2154	

DATE MAILED: 01/27/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	10/260,878	BOUCHER ET AL.	
	Examiner	Art Unit	
	Eric Kuiper	2154	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *27 September 2002*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-24* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-24* is/are rejected.

7)☒ Claim(s) *11* is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-24 have been presented for examination.

Claim Objections

2. Claim 11 is objected to because of the following informalities: line 3 contains only the phrase "the media access control layer header," which appears to be a typographical error in the addition of this phrase to the claim. Appropriate correction is required.

Claim Rejections - 35 USC § 102

3. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

4. Claims 1, 2, 6 and 7 are rejected under 35 U.S.C. 102(e) as being anticipated by Hendel et al. (US 5,920,566, hereinafter Hendel).

5. As per claim 1, Hendel teaches a method for network communication (e.g. Hendel, col. 4, lines 53-55), the method comprising:

receiving a plurality of packets from the network, each of the packets including a media access control layer header, a network layer header and a transport layer header (e.g. Hendel, col. 4, lines 56-67; col. 5, lines 1-8);

processing the packets by a first mechanism, so that for each packet the network layer header and the transport layer header are validated without an interrupt dividing the processing of the network layer header and the transport layer header (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16);

sorting the packets, dependent upon the processing, into first and second types of packets, so that the packets of the first type each contain data (e.g. Hendel, col. 5, lines 26-33);

sending the data from each packet of the first type to a destination without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination (e.g. Hendel, col. 13, lines 63-67; col. 14, lines 1-9).

6. As per claim 2, Hendel teaches the method of claim 1, wherein processing the packets by a first mechanism further comprises:

processing the media access control layer header for each packet without an interrupt dividing the processing of the media access control layer header and the network layer header (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16).

7. As per claim 6, Hendel teaches the method of claim 1, wherein sorting the packets includes classifying each of the packets of the first type as having an Internet Protocol (IP) header and a Transport Control Protocol (TCP) (e.g. Hendel, col. 6, lines 50-61).

8. As per claim 7, Hendel teaches the method of claim 1, further comprising:

transmitting a second plurality of packets to the network, each of the second plurality of packets containing a media access control layer header, a network layer header and a transport layer header, including processing the second plurality of packets by the first mechanism, so that for each packet the media access control layer header, the network layer header and the transport layer header are processed without an interrupt dividing the processing of the media access control layer header (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16).

Claim Rejections - 35 USC § 103

9. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

10. The factual inquiries set forth in *Graham* **v.** *John Deere Co.*, 383 U.S. 1, 148 USPQ 459 (1966), that are applied for establishing a background for determining obviousness under 35 U.S.C. 103(a) are summarized as follows:

1. Determining the scope and contents of the prior art.
2. Ascertaining the differences between the prior art and the claims at issue.
3. Resolving the level of ordinary skill in the pertinent art.
4. Considering objective evidence present in the application indicating obviousness or nonobviousness.

11. Claims 3 and 4 are rejected under 35 U.S.C. 103(a) as being unpatentable over Hendel et al. (US 5,920,566, hereinafter Hendel) in view of Ota et al. (US 6,115,615, hereinafter Ota).

12. As per claim 3, Hendel teaches the method of claim 1, but fails to teach the method further comprising: processing an upper layer header of at least one of the packets by a second mechanism, thereby determining the destination, wherein the upper layer header corresponds to a protocol layer above the transport layer.

However, in a similar art, Ota teaches a network communications system that uses an application layer level address to indicate the destination and route packets through the network (e.g. Ota, col. 7, lines 18-25, 40-53).

It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Hendel because of the advantages of using an upper layer header to determine the destination of packets in a network. Network layer and transport layers also generally include addresses or indications of destination for the packets and including this feature into the application layer as well provides another fail-safe step for the network in the even of a failure in some portion of the network. Having fail-safe routes for information decreases the amount of network downtime since routes can be switched almost instantaneously upon the realization of a fault or error. This is a benefit in any communications network system.

13. As per claim 4, Hendel teaches the method of claim 1, but fails to teach the method further comprising: processing an upper layer header of at least one of the packets of the second type by a second mechanism, thereby determining the destination.

However, in a similar art, Ota teaches a network communications system that uses an application layer level address to indicate the destination and route packets through the network (e.g. Ota, col. 7, lines 18-25, 40-53).

It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Hendel for similar reasons as stated above in regards to claim 3.

14. Claim 5 is rejected under 35 U.S.C. 103(a) as being unpatentable over Hendel et al. (US 5,920,566, hereinafter Hendel) in view of Klaus (US 5,892,903, hereinafter Klaus).

15. As per claim 5, Hendel teaches the method of claim 1, further comprising:

processing a transport layer header of another packet by a second mechanism, prior to receiving the plurality of packets from the network (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16).

Hendel fails to teach establishing a Transport Control Protocol (TCP) connection for the packets of the first type.

However, in a similar art, Klaus teaches the use of a transport layer header to create a TCP connection over a network (e.g. Klaus, col. 5, lines 8-23).

It would have been obvious to one skilled in the art at the time the invention was made to combine Klaus with Hendel because of the advantages of using a transport layer header to provide a TCP connection over a network. The use transport layer, included in the well-known OSI model, is advantageous because it provides segregation of communication functions across the various layers of the protocol stack and modularizes the functions required to implement network communication, which simplifies computer communication operation and maintenance (e.g. Klaus, col. 2, lines 14-23). The use of the OSI model also allows for communication across various systems and platforms without the need for conversion or modification of the

communication method. This can greatly increase the efficiency of communication across a network, which is beneficial in any communications network system.

16. Claim 8 is rejected under 35 U.S.C. 103(a) as being unpatentable over Hendel et al. (US 5,920,566, hereinafter Hendel) in view of Radogna et al. (US 5,991,299, hereinafter Radogna).

17. As per claim 8, Hendel teaches the method of claim 1, but fails to teach the method wherein the first mechanism is a sequencer running microcode.

However, in a similar art, Radogna teaches a dedicated sequencer using microcode to perform network communication and header translation and processing (e.g. col. 4, lines 25-30).

It would have been obvious to one skilled in the art at the time the invention was made to combine Radogna with Hendel because of the benefits of using a specialized processor to handle various tasks in a communications system. Using a sequencer for processing header information can greatly accelerate a frame or packet through a network since the central processing unit does not become overburdened when many packets need to be processed. This frees up the central processor to handle other networking tasks, therefore increasing the speed and efficiency of transmissions through the network. The use of software microcode for this processing easily accommodates new protocols and can bypass hardware processing in the event of a hardware failure. These are beneficial in any computer network system.

18. Claims 9, 10, 14, 16-18, 20 and 22 are rejected under 35 U.S.C. 103(a) as being unpatentable over Radogna et al. (US 5,991,299, hereinafter Radogna) in view of Hendel et al. (US 5,920,566, hereinafter Hendel).

19. As per claim 9, Radogna teaches a method for communicating information over a network (e.g. Radogna, col. 2, lines 63-67), the method comprising:

obtaining data from a source allocated by a first processor (e.g. Radogna, col. 3, lines 50-59);

dividing the data into multiple segments (e.g. Radogna, col. 3, lines 50-59);

prepending a packet header to each of the segments by a second processor, thereby forming a packet corresponding to each segment (e.g. Radogna, col. 14, lines 22-36);

transmitting the packets to the network (e.g. Radogna, col. 5, lines 9-17).

Radogna fails to teach the method comprising each packet header containing a media access control layer header, a network layer header and a transport layer header, wherein the prepending of each packet header occurs without an interrupt dividing the prepending of the network layer header and the transport layer header.

However, in a similar art, Hendel teaches a network communications system based on packets utilizing media access control layer headers, network layer headers and transport layer headers, the processing of these headers all occurring without interrupts between each layer (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16).

It would have been obvious to one skilled in the art at the time the invention was made to combine Hendel with Radogna because of the advantages of including headers for each of the

MAC (data link) layer, network layer and transport layer when communicating over a packetized network conforming to the OSI model. The use of each of these layers is well known in the art since the OSI model was developed. Prepending a header associated with each layer is a common method for allowing the network to process the packets layer by layer, in accordance with the OSI model. Performing the processing and prepending of headers without the use of an interrupt between layers provides the benefit of speeding up the entire processing method and increasing the efficiency of packet transmission across a network. This is beneficial in any computer network system.

20. As per claim 10, Radogna and Hendel teach the method of claim 9, wherein prepending a packet header to each of the segments by a second processor further comprises:

prepending the media access control layer header for each packet without an interrupt dividing the prepending of the media access control layer header and the network layer header (e.g. Radogna, col. 14, lines 22-36; Hendel, col. 12, lines 66-67; col. 13, lines 1-16).

21. As per claim 11, Radogna and Hendel teach the method of claim 9, wherein each packet header contains an Internet Protocol (IP) header and a Transport Control Protocol (TCP) header (e.g. Hendel, col. 6, lines 50-61).

22. As per claim 14, Radogna and Hendel teach the method of claim 9, wherein obtaining data from the source in memory allocated by the first processor is performed by a Direct

Memory Access (DMA) unit controlled by the second processor (e.g. Radogna, col. 5, lines 5-17).

23. As per claim 16, Radogna and Hendel teach the method of claim 9, further comprising:

receiving another packet from the network, the other packet containing a receive header including information corresponding to a network layer and a transport layer (e.g. Hendel, col. 4, lines 56-67; col. 5, lines 1-8); and

selecting whether to process the other packet by the first processor or by the second processor (e.g. Hendel, col. 5, lines 26-33).

24. As per claim 17, Radogna teaches a method for communicating information over a network (e.g. Radogna, col. 2, lines 63-67), the method comprising:

providing multiple segments of data (e.g. Radogna, col. 3, lines 50-59);

prepending an outbound packet header to each of the segments, thereby forming an outbound packet corresponding to each segment (e.g. Radogna, col. 14, lines 22-36);

transmitting the outbound packets to the network (e.g. Radogna, col. 5, lines 9-17); and

receiving multiple inbound packets from the network (e.g. Radogna, col. 3, lines 50-59).

Radogna fails to teach the method comprising the outbound packet header containing an outbound media access control layer header, an outbound network layer header and an outbound transport layer header, wherein the prepending of each outbound packet header occurs without an interrupt dividing the prepending of the outbound media access control layer header, the outbound network layer header and the outbound transport layer header; processing the inbound

packets, so that for each packet the inbound network layer header and the inbound transport layer header are validated without an interrupt dividing the processing of the inbound network layer header and the inbound transport layer header.

However, in a similar art, Hendel teaches a network communications system based on packets utilizing media access control layer headers, network layer headers and transport layer headers, the processing and validating of these headers all occurring without interrupts between each layer (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16).

It would have been obvious to one skilled in the art at the time the invention was made to combine Hendel with Radogna because of the advantages of including headers for each of the MAC (data link) layer, network layer and transport layer when communicating over a packetized network conforming to the OSI model. The use of each of these layers is well known in the art since the OSI model was developed. Prepending a header associated with each layer is a common method for allowing the network to process the packets layer by layer, in accordance with the OSI model. Performing the processing and prepending of headers without the use of an interrupt between layers provides the benefit of speeding up the entire processing method and increasing the efficiency of packet transmission across a network. This is beneficial in any computer network system.

25. As per claim 18, Radogna and Hendel teach the method of claim 17, wherein the processing the inbound packets is performed simultaneously with the prepending the outbound packet header to each of the segments (e.g. Radogna, separate processors for receive

functionality and transmit functionality, col. 3, lines 50-59; col. 5, lines 9-17; col. 14; lines 22-36).

26. As per claim 20, Radogna and Hendel teach the method of claim 17, wherein providing multiple segments of data includes dividing a block of data into the segments (e.g. Radogna, col. 3, lines 50-59).

27. As per claim 22, Radogna and Hendel teach the method of claim 17, further comprising:

sending data from each inbound packet to a destination without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination (e.g. Hendel, col. 13, lines 63-67; col. 14, lines 1-9).

28. Claims 12 and 24 are rejected under 35 U.S.C. 103(a) as being unpatentable over Radogna et al. (US 5,991,299, hereinafter Radogna) in view of Hendel et al. (US 5,920,566, hereinafter Hendel) as applied to claims 9 and 17 above, and further in view of Klaus (US 5,892,903, hereinafter Klaus).

29. As per claim 12, Radogna and Hendel teach the method of claim 9, comprising prepending the packet header to each of the segments by the second processor (e.g. Radogna, col. 14, lines 22-36).

Radogna and Hendel fail to teach the method further comprising establishing a Transport Control Protocol (TCP) connection by the first processor and using the connection to prepend the packet header to each of the segments by the second processor.

However, in a similar art, Klaus teaches the use of a transport layer header to create and utilize a TCP connection over a communications network (e.g. Klaus, col. 5, lines 8-23).

It would have been obvious to one skilled in the art at the time the invention was made to combine Klaus with Radogna and Hendel because of the advantages of using a transport layer header to provide a TCP connection over a network. The use transport layer, included in the well-known OSI model, is advantageous because it provides segregation of communication functions across the various layers of the protocol stack and modularizes the functions required to implement network communication, which simplifies computer communication operation and maintenance (e.g. Klaus, col. 2, lines 14-23). The use of the OSI model also allows for communication across various systems and platforms without the need for conversion or modification of the communication method. This can greatly increase the efficiency of communication across a network, which is beneficial in any communications network system.

30. As per claim 24, Radogna and Hendel teach the method of claim 17, further comprising:

processing a transport layer header of another inbound packet, prior to receiving the plurality of packets from the network (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16).

Radogna and Hendel fail to teach the method further comprising establishing a Transport Control Protocol (TCP) connection for the inbound packets.

However, in a similar art, Klaus teaches the use of a transport layer header to create and utilize a TCP connection over a communications network (e.g. Klaus, col. 5, lines 8-23).

It would have been obvious to one skilled in the art at the time the invention was made to combine Klaus with Radogna and Hendel for similar reasons as stated above in regards to claim 12.

31. Claims 15, 21 and 23 are rejected under 35 U.S.C. 103(a) as being unpatentable over Radogna et al. (US 5,991,299, hereinafter Radogna) in view of Hendel et al. (US 5,920,566, hereinafter Hendel) as applied to claims 9, 20 and 17, respectively, above, and further in view of Ota et al. (US 6,115,615, hereinafter Ota).

32. As per claim 15, Radogna and Hendel teach the method of claim 9, but fail to teach the method further comprising prepending an upper layer header to the data, prior to dividing the data into multiple segments.

However, in a similar art, Ota teaches a network communication system that attaches and uses a header in the application layer (e.g. Ota, col. 7, lines 18-25, 40-53).

It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Radogna and Hendel because of the advantages of attaching a header to an upper layer, such as the application layer, along with the other layers well-known by the OSI model. The use of an upper layer header can provide a great deal of flexibility to the system since it is able to transmit more data with the packet itself. The OSI model is designed to attach and process headers from each of the seven layers efficiently to ensure that the data within the

packet is transmitted properly across the network. Including an application layer header further ensures the proper receipt of the data. This is beneficial in any communication network system.

33. As per claim 21, Radogna and Hendel teach the method of claim 20, but fail to teach the method further comprising prepending an upper layer header to the block of data, prior to dividing the block of data into multiple segments.

However, in a similar art, Ota teaches a network communication system that attaches and uses a header in the application layer (e.g. Ota, col. 7, lines 18-25, 40-53).

It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Radogna and Hendel for similar reasons as stated above in regards to claim 15.

34. As per claim 23, Radogna and Hendel teach the method of claim 17, but fail to teach the method further comprising: processing an upper layer header of at least one of the packets by a second mechanism, thereby determining the destination, wherein the upper layer header corresponds to a protocol layer above the transport layer.

However, in a similar art, Ota teaches a network communications system that uses an application layer level address to indicate the destination and route packets through the network (e.g. Ota, col. 7, lines 18-25, 40-53).

It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Hendel because of the advantages of using an upper layer header to determine the destination of packets in a network. Network layer and transport layers also generally

include addresses or indications of destination for the packets and including this feature into the application layer as well provides another fail-safe step for the network in the even of a failure in some portion of the network. Having fail-safe routes for information decreases the amount of network downtime since routes can be switched almost instantaneously upon the realization of a fault or error. This is a benefit in any communications network system.

35. Claims 13 and 19 are rejected under 35 U.S.C. 103(a) as being unpatentable over Radogna et al. (US 5,991,299, hereinafter Radogna) in view of Hendel et al. (US 5,920,566, hereinafter Hendel) as applied to claims 9 and 17 above, and further in view of Hansen et al. (US 5,778,419, hereinafter Hansen).

36. As per claim 13, Radogna and Hendel teach the method of claim 9, but fail to teach the method further comprising creating a template header and forming each packet header based upon the template header.

However, in a similar art, Hansen teaches the use of a header template from which all packet headers are based (e.g. Hansen, col. 6, lines 4-21).

It would have been obvious to one skilled in the art at the time the invention was made to combine Hansen with Radogna and Hendel because of the advantages of using a template when creating a similar header for each packet. A template is a well-known method for creating files, or in this case, a header, which needs to be attached to many packets containing altogether the same, or very similar data. The structure of each packet header should always consist of the same elements in the same arrangement so a processor does not have to locate the information it

needs prior to performing processing functions. When a template is used, a large amount of time can be saved when performing a large number of transmissions, since it is not necessary to create an entire packet header during each iteration. This increases the overall speed and efficiency of the network, which is beneficial in any communication network system.

37. As per claim 19, Radogna and Hendel teach the method of claim 17, but fail to teach the method further comprising creating a template header and using the template header to form each outbound packet header.

However, in a similar art, Hansen teaches the use of a header template from which all packet headers are based (e.g. Hansen, col. 6, lines 4-21).

It would have been obvious to one skilled in the art at the time the invention was made to combine Hansen with Radogna and Hendel for similar reasons as stated above in regards to claim 13.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Eric Kuiper whose telephone number is (571) 272-0953. The examiner can normally be reached on Monday through Friday, 8:00am to 4:30pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, John Follansbee can be reached on (571) 272-3964. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Eric Kuiper
18 January 2006

JOHN FOLLANSBEE
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

Notice of References Cited	Application/Control No. 10/260,878	Applicant(s)/Patent Under Reexamination BOUCHER ET AL.	
	Examiner Eric Kuiper	Art Unit 2154	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
*	A	US-5,920,566	07-1999	Hendel et al.	370/401
*	B	US-5,991,299	11-1999	Radogna et al.	370/392
*	C	US-5,892,903	04-1999	Klaus, Christopher W.	709/227
*	D	US-6,115,615	09-2000	Ota et al.	455/422.1
*	E	US-5,778,419	07-1998	Hansen et al.	711/112
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
3rd SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

MAR 09 2005 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
EJK	A	5,524,250	6/4/96	Chesson et al.	395	775	
EJK	B	5,619,650	4/8/97	Bach et al.	395	200.01	
EJK	C	5,727,142	3/10/98	Chen	395	181	
EJK	D	5,802,258	9/1/98	Chen	395	182.08	
EJK	E	5,898,713	4/27/99	Melzer et al.	371	53	
EJK	F	6,021,507	2/1/00	Chen	714	2	
EJK	G	6,047,323	4/4/00	Krause	709	227	
	H						

OTHER ART—NON PATENT LITERATURE DOCUMENTS

*Examiner Initial	Cite No.	(Including Author, Title, Date, Pertinent Pages, Etc.)
EJK	1	Schwaderer et al., IEEE Computer Society Press publication entitled, "XTP in VLSI Protocol Decomposition for ASIC Implementation", from 15th Conference on Local Computer Networks, 5 pages, Sept. 30 – Oct. 3, 1990.
EJK	2	Beach, Bob, IEEE Computer Society Press publication entitled, "UltraNet: An Architecture for Gigabit Networking", from 15th Conference on Local Computer Networks, 18 pages, Sept. 30 – Oct. 3, 1990.
EJK	3	Chesson et al., IEEE Syposium Record entitled, "The Protocol Engine Chipset", from Hot Chips III, 16 pages, Aug. 26-27, 1991.
EJK	4	Maclean et al., IEEE Global Telecommunications Conference, Globecom '91, presentation entitled, "An Outboard Processor for High Performance Implementation of Transport Layer Protocols", 7 pages, Dec. 2-5, 1991.
EJK	5	Ross et al., IEEE article entitled "FX1000: A high performance single chip Gigabit Ethernet NIC", from Compcon '97 Proceedings, 7 pages, Feb. 23-26, 1997.
EJK	6	Strayer et al., "Ch. 9: The Protocol Engine" from XTP: The Transfer Protocol, 12 pages, July 1992.
EJK	7	Publication entitled "Protocol Engine Handbook", 44 pages, Oct. 1990.
EJK	8	Koufopavlou et al., IEEE Global Telecommunications Conference, Globecom '92, presentation entitled, "Parallel TCP for High Performance Communication Subsystems", 7 pages, Dec. 6-9, 1992
EJK	9	Lilienkamp et al., Publication entitled "Proposed Host-Front End Protocol", 56 pages, Dec. 1984.

Examiner	Date Considered 1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

2nd SUPP
JUN 26 2006
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
EJK	A	5,517,668	5/14/1996	Szwerinski et al.	395	800	
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	
	O	
	P	
	Q	

Examiner	Date Considered
	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE JUN 10 2004 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
ESK	A	5,598,410	1/28/1997	Stone	370	469	
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	N	
	O	
	P	
	Q	

Examiner	Date Considered
[signature]	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE JC15
JUL 15 2003
PATENT & TRADEMARK OFFICE

RECEIVED
JUL 17 2003
Technology Center 2100

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
EJK	A	4,366,538	12/28/82	Johnson et al.	364	200	
EJK	B	4,991,133	05/13/86	Davis et al.	364	900	
EJK	C	5,056,058	10/08/91	Hirata et al.	709	230	
EJK	D	5,097,442	03/17/92	Ward et al.	365	78	
EJK	E	5,163,131	11/10/92	Row et al.	395	200	
EJK	F	5,212,778	05/18/93	Dally et al.	395	400	
EJK	G	5,280,477	01/18/94	Trapp	370	85.1	
EJK	H	5,289,580	02/22/94	Latif et al.	395	275	
EJK	I	5,303,344	04/12/94	Yokoyama et al.	395	275	
EJK	J	5,412,782	05/02/95	Hausman et al.	395	250	
EJK	K	5,448,566	09/05/95	Richter et al.	370	94.1	

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
EJK	L	WO 00/13091	03/09/00	PCT/US98/24943				
EJK	M	WO 99/65219	12/16/99	PCT/US99/13184				

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJK	N	Internet pages entitled "Hardware Assisted Protocol Processing", (which Eugene Feinberg is working on), 1 page, printed 11/25/98.
EJK	O	Zilog product Brief entitled "Z85C30 CMOS SCC Serial Communication Controller", Zilog Inc., 3 pages, 1997.
EJK	P	Internet pages of Xpoint Technologies, Inc. entitled "Smart LAN Work Requests", 5 pages, printed 12/19/97.
EJK	Q	Internet pages entitled: Asante and 100BASE-T Fast Ethernet, 7 pages, printed 5/27/97.

Examiner	Date Considered
[signature]	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE JUL 15 2003 PATENT & TRADEMARK OFFICE

RECEIVED JUL 17 2003 Technology Center 2100

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
EJK	A	5,485,579	01/16/96	Hitz et al.	395	200.12	
EJK	B	5,506,966	04/09/96	Ban	395	250	
EJK	C	5,511,169	04/23/96	Suda	395	280	
EJK	D	5,548,730	08/20/96	Young et al.	395	280	
EJK	E	5,588,121	12/24/96	Reddin et al.	395	200.15	
EJK	F	5,566,170	10/15/96	Bakke et al	370	60	
EJK	G	5,590,328	12/31/96	Seno et al.	395	675	
EJK	H	5,592,622	01/07/97	Isfeld et al.	395	200.02	
EJK	I	5,629,933	05/13/97	Delp et al.	370	411	
EJK	J	5,634,099	05/27/97	Andrews et al	395	200.07	
EJK	K	5,634,127	05/27/97	Cloud et al.	395	680	

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
EJK	L	WO 01/05107 A1	01/18/01	PCT/US00/19006				
EJK	M	WO 01/05116 A2	01/18/01	PCT/US00/19243				

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJK	N	Internet pages entitled: A Guide to the Paragon XP/S-A7 Supercomputer at Indiana University, 13 pages, printed 12/21/98.
EJK	O	Richard Stevens, "TCP/IP Illustrated, Volume 1, The Protocols", pages 325-326 (1994).
EJK	P	Internet pages entitled: Northridge/Southbridge vs. Intel Hub Architecture, 4 pages, printed 2/19/01.
EJK	Q	Gigabit Ethernet Technical Brief, Achieving End-to-End Performance, Alteon Networks, Inc., First Edition, September 1996.

Examiner	Date Considered
[signature]	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

JUL 15 2003 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
EJK	A	5,642,482	06/24/97	Pardillos	395	200.2	
EJK	B	5,664,114	09/02/97	Krech, Jr. et al.	395	200.64	
EJK	C	5,671,355	09/23/97	Collins	395	200.2	
EJK	D	5,678,060	10/14/97	Yokoyama et al.	709	212	
EJK	E	5,692,130	11/25/97	Shobu et al.	395	200.12	
EJK	F	5,699,317	12/16/97	Sartore et al.	395	230.06	
EJK	G	5,701,434	12/23/97	Nakagawa	395	484	
EJK	H	5,701,516	12/23/97	Cheng et al.	395	842	
EJK	I	5,749,095	05/05/98	Hagersten	711	141	
EJK	J	5,751,715	05/12/98	Chan et al.	370	455	
EJK	K	5,752,078	05/12/98	Delp et al.	395	827	

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
EJK	L	WO 01/05123 A1	01/18/01	PCT/US00/18976				
EJK	M	WO 01/40960A1	06/07/01	PCT/US00/32660				

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJK	N	Internet pages directed to Technical Brief on Alteon Ethernet Gigabit NIC technology, www.alteon.com, 14 pages, printed 3/15/97.
EJK	O	VIA Technologies, Inc. article entitled "VT8501 Apollo MVP4", pages i-iv, 1-11, cover and copyright page, revision 1.3, Feb. 1, 2000.
EJK	P	iReady News Archives article entitled "iReady Rounding Out Management Team with Two Key Executives", http://www.ireadyco.com/archives/keyexec.html, 2 pages, printed 11/28/98.
EJK	Q	"Toshiba Delivers First Chips to Make Consumer Devices Internet-Ready Based On iReady's Design," Press Release October, 1998, 3 pages, printed 11/28/98.

Examiner		Date Considered	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE JUL 15 2003 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
EJK	A	5,758,084	05/26/98	Silverstein et al.	395	200.58	
EJK	B	5,758,089	05/26/98	Gentry et al.	395	200.64	
EJK	C	5,758,186	05/26/98	Hamilton et al.	395	831	
EJK	D	5,758,194	05/26/98	Kuzma	395	886	
EJK	E	5,771,349	06/23/98	Picazo, Jr. et al.	395	188.01	
EJK	F	5,790,804	08/04/98	Osborne	395	200.75	
EJK	G	5,794,061	08/11/98	Hansen et al.	395	800.01	
EJK	H	5,802,580	09/01/98	McAlpice	711	149	
EJK	I	5,809,328	09/15/98	Nogales et al.	395	825	
EJK	J	5,812,775	09/22/98	Van Seeters et al.	395	200.43	
EJK	K	5,815,646	09/29/98	Purcell et al.	395	163	

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
EJK	L	WO 01/04770 A2	01/18/01	PCT/US00/18939				
EJK	M	WO/98/19412	05/07/98	PCT/US97/17257				

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJK	N	Internet pages from iReady Products, web sitehttp://www.ireadyco.com/products,html, 2 pages, downloaded 11/25/98.
EJK	O	iReady News Archives, Toshiba, iReady shipping Internet chip, 1 page, printed 11/25/98.
EJK	P	Interprophet article entitled "Technology", http://www.interprophet.com/technology.html, 17 pages, printed 3/1/00.
EJK	Q	iReady Corporation, article entitled "The I-1000 Internet Tuner", 2 pages, date unknown.

Examiner	Date Considered
[signature]	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE JUL 15 2003 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
EJK	A	5,878,225	03/02/99	Bilansky et al.	395	200.57	
EJK	B	5,913,028	06/15/99	Wang et al.	395	200.33	
EJK	C	5,930,830	07/27/99	Mendelson et al.	711	171	
EJK	D	5,931,918	08/03/99	Row et al	709	300	
EJK	E	5,935,205	08/10/99	Murayama et al.	709	216	
EJK	F	5,937,169	08/10/99	Connery et al.	395	200.8	
EJK	G	5,941,969	08/24/99	Ram et al.	710	128	
EJK	H	5,941,972	08/24/99	Hoese et al.	710	129	
EJK	I	5,950,203	09/07/99	Stakuis et al.	707	10	
EJK	J	5,991,299	11/23/99	Radogna et al.	370	392	
EJK	K	5,996,024	11/30/99	Blumenau	709	301	

RECEIVED JUL 17 2003 Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
EJK	L	WO/98/50852	11/12/98	PCT/US98/08719				
EJK	M	WO/99/04343	01/28/99	PCT/US98/14729				

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJK	N	iReady article entitled "About Us Introduction", Internet pages fromhttp://www.iReadyco.com/about.html, 3 pages, printed 11/25/98.
EJK	O	iReady News Archive article entitled "Revolutionary Approach to Consumer Electronics Internet Connectivity Funded", San Jose, CA, November 20,1997. 2 pages, printed 11/2/98.
EJK	P	iReady News Archive article entitled "Seiko Instruments Inc. (SII) INTRODUCES WORLD'S FIRST INTERNET-READY INTELLIGENT LCD MODULES BASED ON IREADY TECHNOLOGY," Santa Clara, CA and Chiba, Japan, October 26, 1998, 2 pages, printed 11/2/98.
EJK	Q	NEWSwatch article entitled "iReady internet Tuner to Web Enable Devices", Tuesday, November 5, 1996, printed 11/2/98.

Examiner		Date Considered	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE JUL 15 2003 PATENT & TRADEMARK OFFICE

RECEIVED JUL 17 2003 Technology Center 2100

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
EJK	A	6,005,849	12/21/99	Roach et al.	370	276	
EJK	B	6,009,478	12/28/99	Panner et al.	710	5	
EJK	C	6,016,513	01/18/00	Lowe	709	250	
EJK	D	6,026,452	02/15/00	Pitts	710	56	
EJK	E	6,034,963	03/07/00	Minami et al.	370	401	
EJK	F	6,044,438	03/28/00	Olnowich	711	130	
EJK	G	6,047,356	04/04/00	Anderson et al.	711	129	
EJK	H	6,057,863	05/02/00	Olarig	345	520	
EJK	I	6,061,368	05/09/00	Hitzelberger	370	537	
EJK	J	6,065,096	05/16/00	Day et al.	711	114	
EJK	K	6,141,705	10/31/00	Anand et al.	710	15	

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJK	N	EETimes article entitled "Tuner for Toshiba, Toshiba Taps iReady for Internet Tuner", by David Lammers, 2 pages, printed 11/02/98.
EJK	O	"Comparison of Novell Netware and TCP/IP Protocol Architectures", by J.S. Carbone, 19 pages, printed 4/10/98.
EJK	P	Adaptec article entitled "AEA-7110C-a DuraSAN product", 11 pages, printed 10/1/01.
EJK	Q	iSCSI HBA article entitled "iSCSI and 2Gigabit fibre Channel Host Bus Adapters from Emulex, QLogic, Adaptec, JNI", 8 pages, printed 10/01/01.

Examiner		Date Considered	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

JUL 15 2003
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
EJK	A	6,226,680	05/01/01	Boucher et al.	709	230	
EJK	B	6,246,683	06/12/01	Connery et al.	370	392	
EJK	C	6,247,060	06/12/01	Boucher et al.	709	238	
EJK	D	60/053,240		Jolitz et al.			07/18/97
EJK	E	6,345,301	02/05/02	Burns et al.	709	230	
	F						
	G						
	H						
	I						
	J						
	K						

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJK	N	iSCSI HBA article entitled "FCE-3210/6410 32 and 64-bit PCI-to-Fibre Channel HBA", 6 pages, printed 10/01/01.
EJK	O	ISCSI.com article entitled "iSCSI Storage", 2 pages, printed 10/01/01.
EJK	P	"Two-Way TCP Traffic Over Rate Controlled Channels: Effects and Analysis", by Kalampoukas et al., IEEE Transactions on Networking, vol. 6, no. 6, December 1998.
EJK	Q	IReady News article entitled "Toshiba Delivers First Chips to Make Consumer Devices Internet-Ready Based on iReady Design", Santa Clara, CA, and Tokyo, Japan, October 14, 1998, printed 11/2/98.

Examiner	Date Considered
[signature]	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE
JUL 15 2003
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJK	N	United States Patent Application No. 08/964,304, by Napolitano, et al., entitled "File Array Storage Architecture", filed 11/04/97.
EJK	O	"File System Design For An NFS File Server Appliance", Article by D. Hitz, et al., 13 pages.
EJK	P	Adaptec Press Release article entitled "Adaptec Announces EtherStorage Technology", 2 pages, May 4, 2000, printed 6/14/00.
EJK	Q	Adaptec article entitled "EtherStorage Frequently Asked Questions", 5 pages, printed 7/19/00.

Examiner		Date Considered	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE SCIEB
JUL 15 2003
PATENT & TRADEMARK OFFICE

RECEIVED
JUL 17 2003
Technology Center 2100

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJK	N	Adaptec article entitled "EtherStorage White Paper", 7 pages, printed 7/19/00.
EJK	O	CIBC World Markets article entitled "Computers; Storage", by J. Berlino et al., 9 pages, dated August 7, 2000.
EJK	P	Merrill Lynch article entitled "Storage Futures", by S. Milunovich, 22 pages, dated May 10, 2000.
EJK	Q	CBS Market Watch article entitled "Montreal Start-Up Battles Data Storage Botttleneck", by S. Taylor, dated March 5, 2000, 2 pages, printed 3/7/00.

Examiner		Date Considered	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE JC16
JUL 15 2003
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

ESK	N	Internet-draft article entitled "SCSI/TCP (SCSI over TCP)", by J. Satran et al., 38 pages, dated February 2000, printed 5/19/00.
ESK	O	Internet pages entitled "Technical White Paper-Xpoint's Disk to LAN Acceleration Solution for Windows NT Server," 16 pages, printed 6/5/97.
ESK	P	Jato Technologies article entitled "Network Accelerator Chip Architecture," twelve-slide presentation, printed 8/19/98.
ESK	Q	EETimes article entitled "Enterprise System Uses Flexible Spec," dated August 10,1998, printed 11/25/98.

Examiner	Date Considered
[signature]	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE
JUL 15 2003
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJK	N	Internet pages entitled "Smart Ethernet Network Interface Cards", which Berend Ozceri is developing, printed 11/25/98.
EJK	O	Internet pages of Xaqti corporation entitled "GigaPower Protocol Processor Product Review," printed 11/25/99.
EJK	P	Internet pages entitled "DART: Fast Application Level Networking via Data-Copy Avoidance," by Robert J. Walsh, printed 6/3/99.
	Q	

Examiner		Date Considered	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE
JUL 15 2003
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	J							
	K							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJK	L	Internet pages of InterProphet entitled "Frequently Asked Questions", by Lynne Jolitz, printed 6/14/00.
EJK	M	Internet pages entitled "iReady Products," printed 11/25/98.
EJK	N	Andrew S. Tanenbaum, "Computer Networks," Third Edition, 1996, ISBN 0-13-349945-6.
EJK	O	Form 10-K for Exelan, Inc., for the fiscal year ending December 31, 1987 (10 pages).
EJK	P	Form 10-K for Exelan, Inc., for the fiscal year ending December 31, 1988 (10 pages).
EJK	Q	"Second Supplemental Information Disclosure Statement per 37 C.F.R. §1.97(i)", dated July 29, 2002 relating to Exelan Inc. as submitted in Application Serial No. 09/464,283.

Examiner	Date Considered
[signature]	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE
JUL 15 2003
PATENT & TRADEMARK OFFICE

RECEIVED
JUL 17 2003
Technology Center 2100

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	6,173,333	07/18/97	Jolitz et al.			
	B	2001/0025315A1		Jolitz			01/10/01
	C	2001/0004354A1		Jolitz			01/10/01
	D						
	E						
	F						
	G						
	H						
	I						

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	J							
	K							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJK	L	WindRiver article entitled "Tornado: For Intelligent Network Acceleration", copyright Wind River Systems, 2001, 2 pages.
EJK	M	WindRiver White Paper entitled "Complete TCP/IP Offload for High-Speed Ethernet Networks", Copyright Wind River Systems, 2002, 7 pages.
EJK	N	Intel article entitled "Solving Server Bottlenecks with Intel Server Adapters", Copyright Intel Corporation, 1999, 8 pages.
	O	
	P	

Examiner		Date Considered	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE JUL 15 2003 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
EJK	A	5,058,110	10/15/91	Beach et al.	370	85.6	
EJK	B	6,021,446	02/01/00	Gentry, Jr.	709	303	
EJK	C	6,356,951	03/12/02	Gentry, Jr.	709	250	
EJK	D	6,389,468	05/14/02	Muller et al.	709	226	
EJK	E	6,427,169	07/30/02	Elzur	709	224	
EJK	F	6,434,651	08/13/02	Gentry, Jr.	710	260	
EJK	G	6,449,656	09/10/02	Elzur et al.	709	236	
EJK	H	6,453,360	09/17/02	Muller et al.	709	250	
	I						
	J						
	K						

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJK	N	Article from Rice University, Department of Computer Science entitled "Lazy Receiver Processing (LRP): A New Network Subsystem Architecture for Server Systems", by Peter Druschel and Gaurav Banga, 15 pages.
EJK	O	Internet RFC/STD/FYI/BCP Archives article with heading "RFC2140" entitled "TCP Control Block Interdependence", web address http://www.faqs.org/rfcs/rfc2140.html, 9 pages, printed 9/20/02.
	P	
	Q	

Examiner	Date Considered
[signature]	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

BIBDATASHEET

Bib Data Sheet

CONFIRMATION NO. 9902

SERIAL NUMBER	FILING OR 371(c) DATE	CLASS	GROUP ART UNIT	ATTORNEY DOCKET NO.
10/260,878	09/27/2002 RULE	709	2154	ALA-006E

APPLICANTS

Laurence B. Boucher, Saratoga, CA;
Stephen E. J. Blightman, San Jose, CA;
Peter K. Craft, San Francisco, CA;
David A. Higgen, Saratoga, CA;
Clive M. Philbrick, San Jose, CA;
Daryl D. Starr, Milpitas, CA;

**** CONTINUING DATA **********

This application is a CIP of 10/092,967 03/06/2002 PAT 6,591,302
which is a CIP of 10/023,240 12/17/2001 PAT 6,965,941
and is a CIP of 09/464,283 12/15/1999 PAT 6,427,173
which is a CIP of 09/439,603 11/12/1999 PAT 6,247,060
which is a CIP of 09/067,544 04/27/1998 PAT 6,226,680
which claims benefit of 60/061,809 10/14/1997
and said 10/092,967 03/06/2002
is a CIP of 09/384,792 08/27/1999 PAT 6,434,620
which claims benefit of 60/098,296 08/27/1998
and is a CIP of 09/141,713 08/28/1998 PAT 6,389,479
which claims benefit of 60/098,296 08/27/1998
and said 10/092,967 03/06/2002
is a CIP of 09/514,425 02/28/2000 PAT 6,427,171
and is a CIP of 09/416,925 10/13/1999 PAT 6,470,415
and is a CIP of 09/675,484 09/29/2000 PAT 6,807,581
and is a CIP of 09/675,700 09/29/2000
and is a CIP of 09/789,366 02/20/2001 PAT 6,757,746
and is a CIP of 09/801,488 03/07/2001 PAT 6,687,758
and is a CIP of 09/802,551 03/09/2001
and is a CIP of 09/802,426 03/09/2001
and is a CIP of 09/802,550 03/09/2001 PAT 6,658,480
and is a CIP of 09/855,979 05/14/2001 *
and is a CIP of 09/970,124 10/02/2001
(*)Data provided by applicant is not consistent with PTO records.

**** FOREIGN APPLICATIONS **********

IF REQUIRED, FOREIGN FILING LICENSE GRANTED
** 11/01/2002

		STATE OR COUNTRY	SHEETS DRAWING	TOTAL CLAIMS	INDEPENDENT CLAIMS
Foreign Priority claimed	☐ yes ☐ no	CA	89	24	3
35 USC 119 (a-d) conditions met	☐ yes ☐ no ☐ Met after Allowance				
Verified and Acknowledged	Examiner's Signature Initials				

ADDRESS

24501		
TITLE Fast-path apparatus for receiving data corresponding to a TCP connection		
FILING FEE RECEIVED 812	FEES: Authority has been given in Paper No. _____ to charge/credit DEPOSIT ACCOUNT No. _____ for following:	☐ All Fees ☐ 1.16 Fees (Filing) ☐ 1.17 Fees (Processing Ext. of time) ☐ 1.18 Fees (Issue) ☐ Other _____ ☐ Credit

Index of Claims

Application/Control No.	Applicant(s)/Patent under Reexamination
10/260,878	BOUCHER ET AL.
Examiner	**Art Unit**
Eric Kuiper	2154

√	Rejected	–	(Through numeral) Cancelled	N	Non-Elected	A	Appeal
=	Allowed	÷	Restricted	I	Interference	O	Objected

Claim		Date
Final	Original	1/16/06
	1	√
	2	√
	3	√
	4	√
	5	√
	6	√
	7	√
	8	√
	9	√
	10	√
	11	√
	12	√
	13	√
	14	√
	15	√
	16	√
	17	√
	18	√
	19	√
	20	√
	21	√
	22	√
	23	√
	24	√
	25	
	26	
	27	
	28	
	29	
	30	
	31	
	32	
	33	
	34	
	35	
	36	
	37	
	38	
	39	
	40	
	41	
	42	
	43	
	44	
	45	
	46	
	47	
	48	
	49	
	50	

Claim		Date
Final	Original	
	51	
	52	
	53	
	54	
	55	
	56	
	57	
	58	
	59	
	60	
	61	
	62	
	63	
	64	
	65	
	66	
	67	
	68	
	69	
	70	
	71	
	72	
	73	
	74	
	75	
	76	
	77	
	78	
	79	
	80	
	81	
	82	
	83	
	84	
	85	
	86	
	87	
	88	
	89	
	90	
	91	
	92	
	93	
	94	
	95	
	96	
	97	
	98	
	99	
	100	

Claim		Date
Final	Original	
	101	
	102	
	103	
	104	
	105	
	106	
	107	
	108	
	109	
	110	
	111	
	112	
	113	
	114	
	115	
	116	
	117	
	118	
	119	
	120	
	121	
	122	
	123	
	124	
	125	
	126	
	127	
	128	
	129	
	130	
	131	
	132	
	133	
	134	
	135	
	136	
	137	
	138	
	139	
	140	
	141	
	142	
	143	
	144	
	145	
	146	
	147	
	148	
	149	
	150	

Search Notes	Application/Control No.	Applicant(s)/Patent under Reexamination
	10/260,878	BOUCHER ET AL.
	Examiner: Eric Kuiper	Art Unit: 2154

SEARCHED			
Class	Subclass	Date	Examiner

INTERFERENCE SEARCHED			
Class	Subclass	Date	Examiner

SEARCH NOTES (INCLUDING SEARCH STRATEGY)		
	DATE	EXMR
Performed Inventor search in PALM	1/13/2006	EJK
Searched EAST Databases See attached Search History	1/16/2006	EJK
Conducted PLUS search	1/6/2006	EJK

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

PTO/SB
JUL 15 2003
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	6,173,333	07/18/97	Jolitz et al.			
	B	2001/0025315A1		Jolitz			01/10/01
	C	2001/0004354A1		Jolitz			01/10/01
	D						
	E						
	F						
	G						
	H						
	I						

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

		Document Number	Date	Country	Class	Subclass	Translation Yes	Translation No
	J							
	K							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJK	L	WindRiver article entitled "Tornado: For Intelligent Network Acceleration", copyright Wind River Systems, 2001, 2 pages.
EJK	M	WindRiver White Paper entitled "Complete TCP/IP Offload for High-Speed Ethernet Networks", Copyright Wind River Systems, 2002, 7 pages.
EJK	N	Intel article entitled "Solving Server Bottlenecks with Intel Server Adapters", Copyright Intel Corporation, 1999, 8 pages.
	O	
	P	

Examiner	Date Considered
	1/13/2006 5/25/06

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

OIPE 4P21
APR 27 2006
PATENT & TRADEMARK OFFICE

TEW 2154

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of Laurence B. Boucher, et al.		Ser. No:	10/260,878
Filing Date:	September 27, 2002	Examiner:	Eric J. Kuiper
Atty. Docket No:	ALA-006E	GAU:	2154

For: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

April 23, 2006

MS Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Amendment

Sir:

In response to an Office Action dated January 27, 2006, please enter the following Amendment to the Claims and consider the following Remarks.

Amendment to the Claims

1. (currently amended) A method for network communication, the method comprising:

receiving a plurality of packets from the network, each of the packets including a media access control layer header, a network layer header and a transport layer header;

processing the packets by a first mechanism, so that for each packet the network layer header and the transport layer header are validated without an interrupt dividing the processing of the network layer header and the transport layer header;

sorting the packets, dependent upon the processing, into first and second types of packets, so that the packets of the first type each contain data;

sending the data from each packet of the first type to a destination in memory allocated to an application without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination.

2. (original) The method of claim 1, wherein processing the packets by a first mechanism further comprises:

processing the media access control layer header for each packet without an interrupt dividing the processing of the media access control layer header and the network layer header.

3. (original) The method of claim 1, further comprising:

processing an upper layer header of at least one of the packets by a second mechanism, thereby determining the destination, wherein the upper layer header corresponds to a protocol layer above the transport layer.

4. (original) The method of claim 1, further comprising:

processing an upper layer header of at least one of the packets of the second type by a second mechanism, thereby determining the destination.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 328

5. (currently amended) The method of claim 1, further comprising:

processing a transport layer header of another packet by a second mechanism, prior to receiving the plurality of packets from the network, thereby establishing a ~~Transport~~ Transmission Control Protocol (TCP) connection for the packets of the first type.

6. (currently amended) The method of claim 1, wherein sorting the packets includes classifying each of the packets of the first type as having an Internet Protocol (IP) header and a ~~Transport~~ Transmission Control Protocol (TCP).

7. (original) The method of claim 1, further comprising:

transmitting a second plurality of packets to the network, each of the second plurality of packets containing a media access control layer header, a network layer header and a transport layer header, including processing the second plurality of packets by the first mechanism, so that for each packet the media access control layer header, the network layer header and the transport layer header are processed without an interrupt dividing the processing of the media access control layer header, the network layer header and the transport layer header.

8. (original) The method of claim 1, wherein the first mechanism is a sequencer running microcode.

9. (currently amended) A method for communicating information over a network, the method comprising:

obtaining data from a source allocated by a first processor;

dividing the data into multiple segments;

prepending a packet header to each of the segments by a second processor, thereby forming a packet corresponding to each segment, each packet header containing a media access control layer header, a network layer header and a transport layer header, wherein the network layer header is Internet Protocol (IP), the transport layer header is

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 329

Transmission Control Protocol (TCP) and the prepending of each packet header occurs without an interrupt dividing the prepending of the network layer header and the transport layer header; and

transmitting the packets to the network.

10. (original) The method of claim 9, wherein prepending a packet header to each of the segments by a second processor further comprises:

prepending the media access control layer header for each packet without an interrupt dividing the prepending of the media access control layer header and the network layer header.

11. (currently amended) The method of claim 9, wherein each packet header contains an Internet Protocol (IP) header and a ~~Transport~~ Transmission Control Protocol (TCP) header.

~~the media access control layer header,~~

12. (currently amended) The method of claim 9, further comprising establishing a ~~Transport~~ Transmission Control Protocol (TCP) connection by the first processor and using the connection to prepend the packet header to each of the segments by the second processor.

13. (original) The method of claim 9, further comprising creating a template header and forming each packet header based upon the template header.

14. (original) The method of claim 9, wherein obtaining data from the source in memory allocated by the first processor is performed by a Direct Memory Access (DMA) unit controlled by the second processor.

15. (original) The method of claim 9, further comprising prepending an upper layer header to the data, prior to dividing the data into multiple segments.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 330

16. (original) The method of claim 9, further comprising:

receiving another packet from the network, the other packet containing a receive header including information corresponding to a network layer and a transport layer; and

selecting whether to process the other packet by the first processor or by the second processor.

17. (original) A method for communicating information over a network, the method comprising:

providing multiple segments of data;

prepending an outbound packet header to each of the segments, thereby forming an outbound packet corresponding to each segment, the outbound packet header containing an outbound media access control layer header, an outbound network layer header and an outbound transport layer header, wherein the prepending of each outbound packet header occurs without an interrupt dividing the prepending of the outbound media access control layer header, the outbound network layer header and the outbound transport layer header;

transmitting the outbound packets to the network;

receiving multiple inbound packets from the network, each of the inbound packets including an inbound media access control layer header, an inbound network layer header and an inbound transport layer header;

processing the inbound packets, so that for each packet the inbound network layer header and the inbound transport layer header are validated without an interrupt dividing the processing of the inbound network layer header and the inbound transport layer header.

18. (original) The method of claim 17, wherein the processing the inbound packets is performed simultaneously with the prepending the outbound packet header to each of the segments.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 331

19. (original) The method of claim 17, further comprising creating a template header and using the template header to form each outbound packet header.

20. (original) The method of claim 17, wherein providing multiple segments of data includes dividing a block of data into the segments.

21. (original) The method of claim 20, further comprising prepending an upper layer header to the block of data, prior to dividing the block of data into multiple segments.

22. (currently amended) The method of claim 17, further comprising:

sending data from each inbound packet to a destination in memory allocated to an application without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination.

23. (currently amended) The method of claim ~~17~~ 22, further comprising:

processing an upper layer header of at least one of the packets by a second mechanism, thereby determining the destination, wherein the upper layer header corresponds to a protocol layer above the transport layer.

24. (currently amended) The method of claim 17, further comprising:

processing a transport layer header of another inbound packet, prior to receiving the plurality of packets from the network, thereby establishing a ~~Transport~~ Transmission Control Protocol (TCP) connection for the inbound packets.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 332

Remarks

I. Claim Objections

Applicants have amended claim 11 to remove the phrase: "the media access control layer header,". Applicants respectfully assert that claim 11, as amended, is no longer objectionable.

II. Claim Rejections

A. 35 U.S.C. §102

The Office Action rejects claims 1, 2, 6 and 7 under 35 U.S.C. §102(e) as being anticipated by U.S. Patent No. 5,920,566 to Hendel et al. (hereinafter "Hendel"). Regarding claim 1, the Office Action states:

> As per claim 1, Hendel teaches a method for network communication (e.g. Hendel, col. 4, lines 53-55), the method comprising:
>
> receiving a plurality of packets from the network, each of the packets including a media access control layer header, a network layer header and a transport layer header (e.g. Hendel, col. 4, lines 56-67; col. 5, lines 1-8);
>
> processing the packets by a first mechanism, so that for each packet the network layer header and the transport layer header are validated without an interrupt dividing the processing of the network layer header and the transport layer header (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16);
>
> sorting the packets, dependent upon the processing, into first and second types of packets, so that the packets of the first type each contain data (e.g. Hendel, col. 5, lines 26-33);
>
> sending the data from each packet of the first type to a destination without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination (e.g. Hendel, col. 13, lines 63-67; col. 14, lines 1-9).

Applicants respectfully disagree with the Office Action assertion that Hendel teaches "processing the packets by a first mechanism, so that for each packet the network layer header and the transport layer header are validated without an interrupt dividing the processing of the network layer header and the transport layer header (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16)." Column 12, lines 66-67 and column 13, lines 1-16 of Hendel state:

> An innovative structure and method for transmitting the packet and control information across the internal link will now be described with

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 333

> reference to FIGS. 8A and 8B. FIG. 8A is a simplified diagram of the packet structure utilized. More particularly, as the inbound subsystem has determined certain information regarding the packet, e.g., routing, it is advantageous to simply convey this information to the outbound subsystem so that subsequent processing, such as the header field replacement, can easily be performed without reperforming the same steps performed by the inbound subsystem. Furthermore, it is desirable to maintain end-to-end error robustness. Thus, the inbound subsystem encapsulates the packet 800 with control information 805 and a cycle redundancy code (CRC) 810. The outbound system receives the encapsulated packet, determines frame validity using CRC 810, strips the CRC 810 and removes the control information 805 to determine the subsequent processing to be performed to output the packet.

This paragraph does not teach any processing of a network layer header or a transport layer header, let alone "processing the packets by a first mechanism, so that for each packet the network layer header and the transport layer header are validated without an interrupt dividing the processing of the network layer header and the transport layer header." As noted in column 2, lines 24-25 of Hendel, "Layer 2 provides for transmission of frames of data and error detection." The "outbound system" that "determines frame validity using CRC 810" appears to be directed to layer 2 rather than layer 3 (network layer) or layer 4 (transport layer), in contrast to claim 1. For example, IP and TCP (network and transport layer protocols) headers each have checksums that would be checked to validate the IP and TCP headers of a packet. Applicants respectfully assert that Hendel does not teach such validation, and further does not teach such validation "without an interrupt dividing the processing of the network layer header and the transport layer header."

Applicants also respectfully disagree with the Office Action assertion that Hendel teaches "sending the data from each packet of the first type to a destination without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination (e.g. Hendel, col. 13, lines 63-67; col. 14, lines 1-9)." Column 13, lines 63-67 and column 14, lines 1-9 of Hendel state:

> The input interface 845 outputs to the cascading input process (CIP) 850 the packet stripped of the CRC and the CIP 850 removes the control information and forwards the packet, stripped of the encapsulating CRC and control information, to the packet memory 855. The control information is stored in the control field 857 corresponding to the packet

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 334

> stored in the memory 855. The output port process 860 retrieves the packet and the control information from the packet memory 855 and based upon the control information, selectively performs modifications to the packet and issues control signals to the output interface 865 (i.e., MAC).

Applicants respectfully assert that this paragraph does not teach "sending the data from each packet of the first type to a destination without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination." As shown in FIG. 8A of Hendel, stripping the "control information 805" and "CRC 810" that "encapsulates the packet 800" leaves the header and data that form the "packet 800" intact.

For at least these reasons, applicants respectfully assert that Hendel does not anticipate claim 1 or any claim that depends from claim 1.

Regarding claim 2, the Office Action states:

> As per claim 2, Hendel teaches the method of claim 1, wherein processing the packets by a first mechanism further comprises:
> processing the media access control layer header for each packet without an interrupt dividing the processing of the media access control layer header and the network layer header (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16).

Column 12, lines 66-67 and column 13, lines 1-16 of Hendel are quoted above. Applicants respectfully assert that this paragraph does not teach any processing of a network layer header, let alone "processing the media access control layer header for each packet without an interrupt dividing the processing of the media access control layer header and the network layer header." For at least this reason, applicants respectfully assert that Hendel does not anticipate claim 2.

Regarding claim 6, the Office Action states:

> As per claim 6, Hendel teaches the method of claim 1, wherein sorting the packets includes classifying each of the packets of the first type as having an Internet Protocol (IP) header and a Transport Control Protocol (TCP) (e.g. Hendel, col. 6, lines 50-61).

Column 6, lines 50-61 of Hendel state:

> The MLDNE's distributed architecture can be configured to route message traffic in accordance with a number of known routing algorithms such as RIP and OSPF. In a preferred embodiment, the MLDNE is configured to handle message traffic using the Internet suite of protocols,

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 335

> and more specifically the Transmission Control Protocol (TCP) and the Internet Protocol (IP) over the Ethernet LAN standard and medium access control (MAC) data link layer. The TCP is also referred to here as an exemplary Layer 4 protocol, while the IP is referred to repeatedly as a Layer 3 protocol. However, other protocols can be used to implement the concepts of the invention.

Applicants respectfully assert that this paragraph does not teach "wherein sorting the packets includes classifying each of the packets of the first type as having an Internet Protocol (IP) header and a Transport Control Protocol (TCP)." For at least this reason, applicants respectfully assert that Hendel does not anticipate claim 6.

Regarding claim 7, the Office Action states:

> As per claim 7, Hendel teaches the method of claim 1, further comprising:
>
> transmitting a second plurality of packets to the network, each of the second plurality of packets containing a media access control layer header, a network layer header and a transport layer header, including processing the second plurality of packets by the first mechanism, so that for each packet the media access control layer header, the network layer header and the transport layer header are processed without an interrupt dividing the processing of the media access control layer header, the network layer header and the transport layer header (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16).

Column 12, lines 66-67 and column 13, lines 1-16 of Hendel are quoted above. Applicants respectfully assert that this paragraph does not teach any processing of a network layer header or a transport layer header, let alone "transmitting a second plurality of packets to the network, each of the second plurality of packets containing a media access control layer header, a network layer header and a transport layer header, including processing the second plurality of packets by the first mechanism, so that for each packet the media access control layer header, the network layer header and the transport layer header are processed without an interrupt dividing the processing of the media access control layer header, the network layer header and the transport layer header." For at least this reason, applicants respectfully assert that Hendel does not anticipate claim 7.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 336

B. 35 U.S.C. §103

The Office Action rejects claims 3 and 4 under 35 U.S.C. §103(a) as being unpatentable over Hendel in view of U.S. Patent No. 6,115,615 to Ota et al. (hereinafter "Ota"). Regarding claim 3, the Office Action states:

> As per claim 3, Hendel teaches the method of claim 1, but fails to teach the method further comprising:
>
> processing an upper layer header of at least one of the packets by a second mechanism, thereby determining the destination, wherein the upper layer header corresponds to a protocol layer above the transport layer.
>
> However, in a similar art, Ota teaches a network communications system that uses an application layer level address to indicate the destination and route packets through the network (e.g. Ota, col. 7, lines 18-25, 40-53).
>
> It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Hendel because of the advantages of using an upper layer header to determine the destination of packets in a network. Network layer and transport layers also generally include addresses or indications of destinations for the packets and including this feature into the application layer as well provides another fail-safe step for the network in the event of a failure in some portion of the network. Having fail-safe routes for information decreases the amount of network downtime since routes can be switched almost instantaneously upon the realization of a fault or error. This is a benefit in any communications network system.

Applicants have amended claim 1 to recite, in part, "sending the data from each packet of the first type to a destination in memory allocated to an application without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination." Applicants respectfully assert that Ota does not teach "sending the data from each packet of the first type to a destination in memory allocated to an application without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination." Ota instead allegedly "gives a unique application layer level address to a mobile station, and regards a network layer level address (IP address in this embodiment) as an address indicating a route." Ota, column 7, lines 40-43. For at least this reason, applicants respectfully assert that claim 3 is nonobvious over the combination of Ota and Hendel proposed by the Office Action.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 337

Regarding claim 4, the Office Action states:

> As per claim 4, Hendel teaches the method of claim 1, but fails to teach the method further comprising:
>
> processing an upper layer header of at least one of the packets of the second type by a second mechanism, thereby determining the destination.
>
> However, in a similar art, Ota teaches a network communications system that uses an application layer level address to indicate the destination and route packets through the network (e.g. Ota, col. 7, lines 18-25, 40-53).
>
> It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Hendel for similar reasons as stated above in regards to claim 3.

Applicants respectfully assert that claim 4 is nonobvious over the combination of Ota and Hendel proposed by the Office Action for similar reasons as stated above in regards to claim 3.

The Office Action rejects claim 5 under 35 U.S.C. §103(a) as being unpatentable over Hendel in view of U.S. Patent No. 5,892,903 to Klaus (hereinafter "Klaus"). Regarding claim 5, the Office Action states:

> As per claim 5, Hendel teaches the method of claim 1, further comprising:
>
> processing a transport layer header of another packet by a second mechanism, prior to receiving the plurality of packets from the network (e.g. Hendel, col. 13, lines 63-67; col. 14, lines 1-9).
>
> Hendel fails to teach establishing a Transport Control Protocol (TCP) connection for the packets of the first type.
>
> However, in a similar art, Klaus teaches the use of a transport layer header to create a TCP connection over a network (e.g. Klaus, col. 5, lines 8-23).
>
> It would have been obvious to one skilled in the art at the time the invention was made to combine Klaus with Hendel because of the advantages of using a transport layer header to provide a TCP connection over a network. The use transport layer, included in the well-known OSI model, is advantageous because it provides segmentation of communication functions across the various layers of the protocol stack and modularizes the functions required to implement network communication, which simplifies computer communication operation and maintenance (e.g. Klaus, col. 2, lines 14-23). The use of the OSI model allows for communication across various systems and platforms without the need for conversion or modification of the communication method. This can greatly increase the efficiency of communication across a network, which is beneficial in any communications network system.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 338

Applicants respectfully disagree with the Office Action assertion that "It would have been obvious to one skilled in the art at the time the invention was made to combine Klaus with Hendel because of the advantages of using a transport layer header to provide a TCP connection over a network."

Hendel is directed to "an apparatus and related method for relaying packets by a multi-layer distributed network element according to known routing protocols." Hendel, column 4, lines 53-55. "The network element should be able to operate at bridge-like speeds, yet be capable of routing packets across different subnetworks and provide upper layer functionalities such as quality of service." Hendel, column 4, lines 47-50. Establishing a TCP connection, which is complicated and performed in software, would contradict Hendel's "need for a network element that can handle changing network conditions such as topology and message traffic yet make efficient use of high performance hardware to switch packets based on their Layer 2, Layer 3, and Layer 4 headers." Hendel, column 4, lines 53-55.

Moreover, as noted in column 3, lines 29-49 of Klaus, "In the TCP/IP protocol, a communication connection is established through a three handshake open network protocol. The first handshake or data message is from a source computer and is typically called a "synchronization" or "sync" message. In response to a sync message, the destination computer transmits a synchronization-acknowledgment ("sync-ack") message. The source computer then transmits an acknowledgment ("ack") message and a communication connection between the source and destination computer is established." This multi-step procedure, performed in software and over a network, would appear to one of ordinary skill to slow the routing and switching of packets that Hendel is directed to.

In addition, it is not asserted in the Office Action, and it is certainly not apparent from the cited references, what computer the "network element" of Hendel would establish a TCP connection with, if combined with Klaus as proposed by the Office Action. Stated differently, applicants respectfully assert that the combination of Klaus and Hendel that is proposed by the Office Action may be inoperable, teaching one of ordinary skill in the art away from making such a combination.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 339

Applicants respectfully assert that these disadvantages of establishing a TCP connection would far outweigh the advantages alleged by the Office Action, which for the most part would not even be applicable to the proposed combination of Klaus and Hendel.

For at least this reason, applicants respectfully assert that claim 5 is nonobvious over the combination of Klaus and Hendel proposed by the Office Action.

The Office Action rejects claim 8 under 35 U.S.C. §103(a) as being unpatentable over Hendel in view of U.S. Patent No. 5,991,299 to Radogna et al. (hereinafter "Radogna"). Regarding claim 8, the Office Action states:

> As per claim 8, Hendel teaches the method of claim 1, but fails to teach the method wherein the first mechanism is a sequencer running microcode.
>
> However, in a similar art, Radogna teaches the use of a dedicated sequencer running microcode to perform network communication and header translation and processing (e.g. col. 4, lines 25-30).
>
> It would have been obvious to one skilled in the art at the time the invention was made to combine Radogna with Hendel because of the benefits of using a specialized processor to handle various tasks in a communications system. Using a sequencer for processing header information can greatly accelerate a frame or packet through a network since the central processing unit does not become overburdened when many packets need to be processed. This frees up the central processor to handle other networking tasks, therefore increasing the speed and efficiency of transmissions through the network. The use of software microcode for this processing easily accommodates new protocols and can bypass hardware processing in the event of a hardware failure. This is beneficial in any communications network system.

Applicants respectfully assert that, *assuming arguendo* Radogna and Hendel were combined as proposed by the Office Action, the resulting device would not be processing the packets by a sequencer running microcode, so that for each packet the network layer header and the transport layer header are validated without an interrupt dividing the processing of the network layer header and the transport layer header, in contrast to claim 8. There is, for example, no teaching in Radogna or Hendel of validating a transport layer header, let alone the limitation of "for each packet the network layer header and the transport layer header are validated without an interrupt dividing the processing of the network layer header and the transport layer header." Radogna, like Hendel, is directed

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 340

to high speed header translation processing for bridges and routers. Validating transport layer headers would not only be a waste of time in such devices, but may cause unnecessary errors if, for example, checksums were removed as is typical for such validation, to be replaced with new checksums on retransmission.

For at least this reason, applicants respectfully assert that claim 8 is nonobvious over the combination of Radogna and Hendel proposed by the Office Action.

The Office Action rejects claims 9, 10, 14, 16-18 and 22 under 35 U.S.C. §103(a) as being unpatentable over Radogna in view of Hendel. Regarding claim 9, the Office Action states:

> As per claim 9, Radogna teaches a method for communicating information over a network (e.g. Radogna, col. 2, lines 63-67), the method comprising:
>
> obtaining data from a source allocated by a first processor (e.g. Radogna, col. 3, lines 50-59);
>
> dividing the data into multiple segments (e.g. Radogna, col. 3, lines 50-59);
>
> prepending a packet header to each of the segments by a second processor, thereby forming a packet corresponding to each segment (e.g. Radogna, col. 14, lines 22-36);
>
> transmitting the packets to the network (e.g. Radogna, col. 5, lines 9-17).
>
> Radogna fails to teach the method comprising each packet header containing a media access control layer header, a network layer header and a transport layer header, wherein the prepending of each packet header occurs without an interrupt dividing the prepending of the network layer header and the transport layer header.
>
> However, in a similar art, Hendel teaches a network communications system teach the method comprising each packet header containing a media access control layer header, a network layer header and a transport layer header, wherein the prepending of each packet header occurs without an interrupt dividing the prepending of the network layer header and the transport layer header (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16).
>
> It would have been obvious to one skilled in the art at the time the invention was made to combine Radogna with Hendel because of the advantages of including headers for each of the MAC (data link) layer, network layer and transport layer when communicating over a packetized network conforming to the OSI model. The use of these layers is well known since the OSI model was developed. Prepending a header associated with each layer is a common method for allowing the network to process the packets layer by layer, in accordance with the OSI model. Performing the processing and prepending of headers without an interrupt

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 341

between layers provides the benefits of speeding up the entire processing method and increasing the efficiency of packet transmission across a network. This is beneficial in any communications network system.

Applicants respectfully disagree with the Office Action assertion that "Hendel teaches a network communications system teach the method comprising each packet header containing a media access control layer header, a network layer header and a transport layer header, wherein the prepending of each packet header occurs without an interrupt dividing the prepending of the network layer header and the transport layer header (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16)." Column 12, lines 66-67 and column 13, lines 1-16 of Hendel state:

> An innovative structure and method for transmitting the packet and control information across the internal link will now be described with reference to FIGS. 8A and 8B. FIG. 8A is a simplified diagram of the packet structure utilized. More particularly, as the inbound subsystem has determined certain information regarding the packet, e.g., routing, it is advantageous to simply convey this information to the outbound subsystem so that subsequent processing, such as the header field replacement, can easily be performed without reperforming the same steps performed by the inbound subsystem. Furthermore, it is desirable to maintain end-to-end error robustness. Thus, the inbound subsystem encapsulates the packet 800 with control information 805 and a cycle redundancy code (CRC) 810. The outbound system receives the encapsulated packet, determines frame validity using CRC 810, strips the CRC 810 and removes the control information 805 to determine the subsequent processing to be performed to output the packet.

This paragraph does not teach any processing of a network layer header or a transport layer header, let alone "processing the packets by a first mechanism, so that for each packet the network layer header and the transport layer header are validated without an interrupt dividing the processing of the network layer header and the transport layer header." As noted in column 2, lines 24-25 of Hendel, "Layer 2 provides for transmission of frames of data and error detection." The "outbound system" that "determines frame validity using CRC 810" appears to be directed to layer 2 rather than layer 3 (network layer) or layer 4 (transport layer), in contrast to claim 1. For example, IP and TCP (network and transport layer protocols) headers each have checksums that would be checked to validate the IP and TCP headers of a packet. Applicants respectfully assert that Hendel does not teach such validation, and further does not teach

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 342

such validation "without an interrupt dividing the processing of the network layer header and the transport layer header." As shown in FIG. 8A of Hendel, adding and stripping the "control information 805" and "CRC 810" that "encapsulates the packet 800" leaves the header and data that form the "packet 800" intact.

For at least these reasons, applicants respectfully assert that claim 9 and any claim that depends from claim 9 is nonobvious over the combination of Radogna and Hendel proposed by the Office Action.

Regarding claim 10, the Office Action states:

> As per claim 10, Radogna and Hendel teach the method of claim 9, wherein prepending a packet header to each of the segments by a second processor further comprises:
>
> prepending the media access control layer header for each packet without an interrupt dividing the prepending of the media access control layer header and the network layer header (e.g. Radogna, col. 14, lines 22-36; Hendel, col. 12, lines 66-67; col. 13, lines 1-16).

Applicants respectfully disagree with the Office Action assertion that "Radogna and Hendel teach the method of claim 9, wherein prepending a packet header to each of the segments by a second processor further comprises: prepending the media access control layer header for each packet without an interrupt dividing the prepending of the media access control layer header and the network layer header (e.g. Radogna, col. 14, lines 22-36; Hendel, col. 12, lines 66-67; col. 13, lines 1-16)." Column 12, lines 66-67 and column 13, lines 1-16 of Hendel are quoted above and do not teach this. Column 14, lines 22-36 of Radogna state:

> The method of operation of the hardware microsequencer 100 and associated support hardware comprising the THP engine is generally illustrated in FIGS. 6a-6c. Frames are delivered to Transmit Segmentation Unit (TSEG) FIFOs 59 from the Buffer RAM 22 based upon per port queues maintained within the MBA as depicted in step 200. The Transmit Segmentation Unit (TSEG) 58 queues transmit vectors from the Master Buffer ASIC (MBA) 32, which indicate where in the Buffer RAM 22, respective segments of transmit frames are stored. The frames are packed into the TSEG FIFO 59 so that there are no spaces between bytes. Information needed by the THP 60 to identify and execute the proper translation routine is contained within the transmit vector which is prepended to each frame presented to the THP for header translation.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 343

As can be seen, this paragraph also does not teach the limitations of claim 10. For at least these reasons, applicants respectfully assert that claim 10 is nonobvious over the combination of Radogna and Hendel proposed by the Office Action.

Regarding claim 14, the Office Action states:

> As per claim 10, Radogna and Hendel teach the method of claim 9, wherein obtaining data from the source in memory allocated by the first processor is performed by a Direct Memory Access (DMA) unit controlled by the second processor (e.g. Radogna, col. 5, lines 5-17).

Applicants respectfully disagree with the Office Action assertion that "Radogna and Hendel teach the method of claim 9, wherein obtaining data from the source in memory allocated by the first processor is performed by a Direct Memory Access (DMA) unit controlled by the second processor. (e.g. Radogna, col. 5, lines 5-17)." Column 14, lines 22-36 of Radogna state:

> The RSEG 50 comprises a DMA controller which controls storage of received frame data within appropriate Buffer RAM 22 locations.
>
> The Transmit ASIC
>
> The transmit ASIC includes a Transmit Segmentation Unit (TSEG) 58, a plurality of Transmit Segment Unit (TSEG) FIFOs 59, a Transmit Header Processor (THP) 60, a Transmit State Machine ("TXSM") 62 and Transmit State Machine FIFOs 64. The TSEG 58 comprises a DMA controller which serves to move frame data segments from locations within the Buffer RAM 22 into an input FIFO designated as the TSEG FIFO 59. The TSEG FIFO 59 comprises an input to the THP 60.

As can be seen, these paragraphs do not teach the limitations of claim 14. That is, although a DMA controller is mentioned, these paragraphs do not teach "wherein obtaining data from the source in memory allocated by the first processor is performed by a Direct Memory Access (DMA) unit controlled by the second processor." For at least this reason, applicants respectfully assert that claim 14 is nonobvious over the combination of Radogna and Hendel proposed by the Office Action.

Regarding claim 16, the Office Action states:

> As per claim 16, Radogna and Hendel teach the method of claim 9, further comprising:
>
> receiving another packet from the network, the other packet containing a receive header including information corresponding to a network layer and a transport layer (e.g. Hendel, col. 4, lines 56-67; col. 5, lines 1-8); and

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 344

selecting whether to process the other packet by the first processor or by the second processor (e.g. Hendel, col. 5, lines 26-33).

Applicants respectfully disagree with the Office Action assertion that "Radogna and Hendel teach the method of claim 9, further comprising: ...selecting whether to process the other packet by the first processor or by the second processor. (e.g. Hendel, col. 5, lines 26-33)." Column 5, lines 26-33 of Radogna instead state:

> When the packet is received over the internal link by a second subsystem, the packet is forwarded to the neighbor node in response to the packet's new first header portion matching a type 1 entry in the second forwarding memory. The type 1 entry in the second subsystem contains the address of the neighbor node or endstation and had been created independently of the matching type 2 entry of the inbound subsystem.

As can be seen, this paragraph does not teach the limitations of claim 16. That is, this paragraph does not teach "selecting whether to process the other packet by the first processor or by the second processor." For at least this reason, applicants respectfully assert that claim 16 is nonobvious over the combination of Radogna and Hendel proposed by the Office Action.

Regarding claim 17, the Office Action states:

> As per claim 17, Radogna teaches a method for communicating information over a network, the method comprising:
>
> providing multiple segments of data (e.g. Radogna, col. 3, lines 50-59);
>
> prepending an outbound packet header to each of the segments, thereby forming an outbound packet corresponding to each segment (e.g. Radogna, col. 14, lines 22-36);
>
> transmitting the outbound packets to the network (e.g. Radogna, col. 5, lines 9-17);
>
> receiving multiple inbound packets from the network, each of the inbound packets including an inbound media access control layer header, an inbound network layer header and an inbound transport layer header (e.g. Radogna, col. 3, lines 50-59).
>
> Radogna fails to teach the method comprising the outbound packet header containing an outbound media access control layer header, an outbound network layer header and an outbound transport layer header, wherein the prepending of each outbound packet header occurs without an interrupt dividing the prepending of the outbound media access control layer header, the outbound network layer header and the outbound transport layer header;

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 345

> However, in a similar art, Hendel teaches a network communications system based on packets utilizing media access control layer headers, network layer headers and transport layer headers, the processing and validating of these headers all occurring without interrupts between each layer (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16).
>
> It would have been obvious to one skilled in the art at the time the invention was made to combine Hendel with Radogna because of the advantages of including headers for each of the MAC (data link) layer, network layer and transport layer when communicating over a packetized network conforming to the OSI model. The use of these layers is well known since the OSI model was developed. Prepending a header associated with each layer is a common method for allowing the network to process the packets layer by layer, in accordance with the OSI model. Performing the processing and prepending of headers without an interrupt between layers provides the benefits of speeding up the entire processing method and increasing the efficiency of packet transmission across a network. This is beneficial in any communications network system.

Applicants respectfully disagree with the Office Action assertion that "Hendel teaches a network communications system based on packets utilizing media access control layer headers, network layer headers and transport layer headers, the processing and validating of these headers all occurring without interrupts between each layer (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16)." As discussed above for claim 1 and claim 9, Hendel teaches no such thing. Moreover, applicants respectfully note that the Office Action does not assert that Hendel and Radogna teach, and Hendel and Radogna do not teach, "processing the inbound packets, so that for each packet the inbound network layer header and the inbound transport layer header are validated without an interrupt dividing the processing of the inbound network layer header and the inbound transport layer header," in contrast to claim 17.

For at least these reasons, applicants respectfully assert that claim 17 and any claim that depends from claim 17 is nonobvious over the combination of Radogna and Hendel proposed by the Office Action.

Regarding claim 18, the Office Action states:

> As per claim 18, Radogna and Hendel teach the method of claim 17, wherein the processing the inbound packets is performed simultaneously with the prepending the outbound packet header to each of the segments (e.g. Radogna, separate processors for receive functionality and transmit functionality, col. 3, lines 50-59; col. 5, lines 9-17; col. 14, lines 22-36).

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 346

As noted above, the proposed combination Radogna and Hendel does not process inbound packets or prepend outbound packet headers, as recited, so *assuming arguendo* that Radogna has separate processors for receive functionality and transmit functionality is immaterial to the claim. For at least this reason, applicants respectfully assert that claim 18 is nonobvious over the combination of Radogna and Hendel proposed by the Office Action.

Regarding claim 22, the Office Action states:

> As per claim 22, Radogna and Hendel teach the method of claim 17, further comprising:
>
> sending data from each inbound packet to a destination without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16; col. 14, lines 1-9).

Applicants respectfully assert that Hendel does not teach, in column 12-14 or elsewhere, "sending data from each inbound packet to a destination without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination," as recited in claim 22. For at least this reason, applicants respectfully assert that claim 22 is nonobvious over the combination of Radogna and Hendel proposed by the Office Action.

The Office Action rejects claims 12 and 24 under 35 U.S.C. §103(a) as being unpatentable over Radogna in view of Hendel and Klaus. Regarding claim 12, the Office Action states:

> As per claim 12, Radogna and Hendel teach the method of claim 9, comprising prepending the packet header to each of the segments by the second processor (e.g. Radogna, col. 14, lines 22-36).
>
> Radogna and Hendel fail to teach the method further comprising establishing a Transport Control Protocol (TCP) connection by the first processor and using the connection to prepend the packet header to each of the segments by the second processor.
>
> However, in a similar art, Klaus teaches the use of a transport layer header to create and utilize a TCP connection over a network (e.g. Klaus, col. 5, lines 8-23).
>
> It would have been obvious to one skilled in the art at the time the invention was made to combine Klaus with Radogna and Hendel because of the advantages of using a transport layer header to provide a TCP connection over a network. The use transport layer, included in the well-known OSI model, is advantageous because it provides segmentation of

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 347

> communication functions across the various layers of the protocol stack and modularizes the functions required to implement network communication, which simplifies computer communication operation and maintenance (e.g. Klaus, col. 2, lines 14-23). The use of the OSI model allows for communication across various systems and platforms without the need for conversion or modification of the communication method. This can greatly increase the efficiency of communication across a network, which is beneficial in any communications network system.

Applicants respectfully disagree with the Office Action assertion that "It would have been obvious to one skilled in the art at the time the invention was made to combine Klaus with Radogna and Hendel because of the advantages of using a transport layer header to provide a TCP connection over a network."

Hendel is directed to "an apparatus and related method for relaying packets by a multi-layer distributed network element according to known routing protocols." Hendel, column 4, lines 53-55. "The network element should be able to operate at bridge-like speeds, yet be capable of routing packets across different subnetworks and provide upper layer functionalities such as quality of service." Hendel, column 4, lines 47-50. Establishing a TCP connection, which is complicated and performed in software, would contradict Hendel's "need for a network element that can handle changing network conditions such as topology and message traffic yet make efficient use of high performance hardware to switch packets based on their Layer 2, Layer 3, and Layer 4 headers." Hendel, column 4, lines 53-55.

Radogna, like Hendel, is directed to high speed header translation processing for bridges and routers. Establishing a TCP connection, which is complicated and performed in software, would contradict Radogna's desire "to be able to perform header translations in a network device, such as a router, at or near the frame reception rate for the device." Radogna, column 1, lines 63-65.

Moreover, as noted in column 3, lines 29-49 of Klaus, "In the TCP/IP protocol, a communication connection is established through a three handshake open network protocol. The first handshake or data message is from a source computer and is typically called a "synchronization" or "sync" message. In response to a sync message, the destination computer transmits a synchronization-acknowledgment ("sync-ack") message. The source computer then transmits an acknowledgment ("ack") message and a

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 348

communication connection between the source and destination computer is established." This multi-step procedure, performed in software and over a network, would appear to one of ordinary skill to slow the routing and switching of packets that Hendel is directed to.

In addition, it is not asserted in the Office Action, and it is certainly not apparent from the cited references, what computer the "network element" of Hendel would establish a TCP connection with, if combined with Radogna and Klaus as proposed by the Office Action. Stated differently, applicants respectfully assert that the combination of Klaus and Radogna and Hendel that is proposed by the Office Action may be inoperable, teaching one of ordinary skill in the art away from making such a combination.

Applicants respectfully assert that these disadvantages of establishing a TCP connection would far outweigh the advantages alleged by the Office Action, which for the most part would not even be applicable to the proposed combination of Klaus and Radogna and Hendel.

For at least this reason, applicants respectfully assert that claim 12 is nonobvious over the combination of Klaus and Radogna and Hendel proposed by the Office Action.

Regarding claim 24, the Office Action states:

> As per claim 24, Radogna and Hendel teach the method of claim 17, further comprising:
>
> processing a transport layer header of another inbound packet, prior to receiving the plurality of packets from the network (e.g. Hendel, col. 12, lines 66-67; col. 13, lines 1-16),
>
> Radogna and Hendel fail to teach the method further comprising establishing a Transport Control Protocol (TCP) connection for the inbound packets.
>
> However, in a similar art, Klaus teaches the use of a transport layer header to create and utilize a TCP connection over a network (e.g. Klaus, col. 5, lines 8-23).
>
> It would have been obvious to one skilled in the art at the time the invention was made to combine Klaus with Radogna and Hendel for similar reasons as stated above in regards to claim 12.

Applicants respectfully assert that Hendel does not teach, in column 12, lines 66-67; column 13, lines 1-16, or elsewhere, "processing a transport layer header of another inbound packet" as recited in claim 24, for similar reasons as stated above in regards to

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 349

claim 12.. Applicants also respectfully assert that it would not have been obvious to one skilled in the art at the time the invention was made to combine Klaus with Radogna and Hendel, for similar reasons as stated above in regards to claim 12. For at least these reasons, applicants respectfully assert that claim 24 is nonobvious over the combination of Radogna and Hendel proposed by the Office Action.

The Office Action rejects claims 15, 21 and 23 under 35 U.S.C. §103(a) as being unpatentable over Radogna in view of Hendel and Ota. Regarding claim 15, the Office Action states:

> As per claim 15, Radogna and Hendel teach the method of claim 9, but fail to teach the method further comprising prepending an upper layer header to the data, prior to dividing the data into multiple segments.
>
> However, in a similar art, Ota teaches a network communication system that attaches and uses a header in the application layer (e.g. Ota, col. 7, lines 18-25, 40-53).
>
> It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Radogna and Hendel because of the advantages of attaching a header to an upper layer, such as the application layer, along with the other layers of the well-known OSI model. The use of an upper layer header can provide a great deal of flexibility to the system since it is able to transmit more data with the packet itself. The OSI model is designed to attach and process headers from each of the seven layers efficiently to ensure that the data within the packet is transmitted properly across the network. Including an application layer header further ensures the proper receipt of the data. This is beneficial in any communications network system.

Applicants respectfully note that the Office Action does not assert, and even the combination of the references proposed by the Office Action would not teach, "prepending an upper layer header to the data, prior to dividing the data into multiple segments," as recited in claim 15. This may be because the advantages alleged by Ota and the Office Action would not work in this case. That is, Ota allegedly "gives a unique application layer level address to a mobile station, and regards a network layer level address (IP address in this embodiment) as an address indicating a route." Ota, column 7, lines 40-43. But should such a "unique application layer level address" be prepended as "an upper layer header to the data, prior to dividing the data into multiple segments," that header would presumably only be attached to the first segment of the multiple segments, after dividing the data into multiple segments. In other words, the upper layer addressing

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 350

scheme proposed by Ota would fail for all but the first packet of multiple packets, resulting in multiple problems and showing how useless the upper layer addressing scheme proposed by Ota really is. Because the OSI model does not have any mechanism for providing upper layer headers to each packet for blocks of data that are divided for transmission over a network, and the addressing scheme of Ota reduces network layer level addresses such as IP addresses as merely "indicating a route," Ota is probably inoperable, teaching one of ordinary skill in the art away from using Ota or combining it with any functional reference.

For at least these reasons, applicants respectfully assert that claim 15 is nonobvious over the combination of Radogna, Hendel and Ota proposed by the Office Action.

Regarding claim 21, the Office Action states:

> As per claim 21, Radogna and Hendel teach the method of claim 20, but fail to teach the method further comprising prepending an upper layer header to the data, prior to dividing the data into multiple segments.
>
> However, in a similar art, Ota teaches a network communication system that attaches and uses a header in the application layer (e.g. Ota, col. 7, lines 18-25, 40-53).
>
> It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Radogna and Hendel for similar reasons as stated above in regards to claim 15.

Applicants respectfully note that the Office Action does not assert, and even the combination of the references proposed by the Office Action would not teach, "prepending an upper layer header to the data, prior to dividing the data into multiple segments," as recited in claim 21. This may be because the advantages alleged by Ota and the Office Action would not work in this case. That is, Ota allegedly "gives a unique application layer level address to a mobile station, and regards a network layer level address (IP address in this embodiment) as an address indicating a route." Ota, column 7, lines 40-43. But should such a "unique application layer level address" be prepended as "an upper layer header to the data, prior to dividing the data into multiple segments," that header would presumably only be attached to the first segment of the multiple segments, after dividing the data into multiple segments. In other words, the upper layer addressing scheme proposed by Ota would fail for all but the first packet of multiple packets,

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 351

resulting in multiple problems and showing how useless the upper layer addressing scheme proposed by Ota really is. Because the OSI model does not have any mechanism for providing upper layer headers to each packet for blocks of data that are divided for transmission over a network, and the addressing scheme of Ota reduces network layer level addresses such as IP addresses as merely "indicating a route," Ota is probably inoperable, teaching one of ordinary skill in the art away from using Ota or combining it with any functional reference.

For at least these reasons, applicants respectfully assert that claim 21 is nonobvious over the combination of Radogna, Hendel and Ota proposed by the Office Action.

Regarding claim 23, the Office Action states:

> As per claim 23, Radogna and Hendel teach the method of claim 17, but fail to teach the method further comprising:
>
> processing an upper layer header of at least one of the packets by a second mechanism, thereby determining the destination, wherein the upper layer header corresponds to a protocol layer above the transport layer.
>
> However, in a similar art, Ota teaches a network communications system that uses an application layer level address to indicate the destination and route packets through the network (e.g. Ota, col. 7, lines 18-25, 40-53).
>
> It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Hendel because of the advantages of using an upper layer header to determine the destination of packets in a network. Network layer and transport layers also generally include addresses or indications of destinations for the packets and including this feature into the application layer as well provides another fail-safe step for the network in the event of a failure in some portion of the network. Having fail-safe routes for information decreases the amount of network downtime since routes can be switched almost instantaneously upon the realization of a fault or error. This is a benefit in any communications network system.

Claim 23 has been amended to depend from claim 22 rather than claim 17, and claim 22 recites, in part, "sending data from each inbound packet to a destination in memory allocated to an application without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination." Applicants respectfully assert that Ota does not teach "sending the data from each packet of the first type to a destination in memory allocated to an application without sending any of the

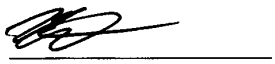
CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 352

media access control layer headers, network layer headers or transport layer headers to the destination." Ota instead allegedly "gives a unique application layer level address to a mobile station, and regards a network layer level address (IP address in this embodiment) as an address indicating a route." Ota, column 7, lines 40-43. For at least this reason, applicants respectfully assert that claim 23 is nonobvious over the combination of Ota and Radogna and Hendel proposed by the Office Action.

III. Conclusion

Applicants have responded to each of the items in the Office Action, and believe that all of the pending claims are in condition for allowance. As such, applicants respectfully solicit a Notice of Allowance.

Respectfully submitted,

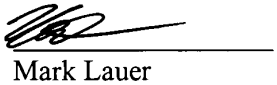


Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the U.S. Postal Service with sufficient postage as first class mail in an envelope addressed to: the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on April 23, 2006.

Date: 4-23-06



Mark Lauer

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 353

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of Laurence B. Boucher, et al.		Ser. No:	10/260,878
Filing Date:	September 27, 2002	Examiner:	Unknown
Atty. Docket No:	ALA-006E	GAU:	2154

For: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

April 23, 2006

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

4th Supplemental Information Disclosure Statement

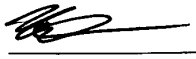
Sir:

Pursuant to 37 C.F.R. §§ 1.56, 1.97 and 1.98, applicants bring thirty-seven reference documents to the Examiner's attention. Included is a two-page form PTO-1449 listing these thirty-seven U.S. Patent reference documents. Copies of the thirty-seven U.S. Patent reference documents are not enclosed. Also enclosed is a check in the amount of $180.00.

Citation of these documents shall not be construed as an admission that the documents are prior art with respect to the instant invention, a representation that a search has been made, or an admission that the information cited herein is, or is considered to be, material to patentability as defined in 37 C.F.R. § 1.56(b).

04/28/2006 EAYALEW1 00000012 10260878
01 FC:1806 180.00 OP

Respectfully submitted,

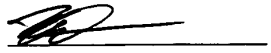


Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Commissioner for Patents, P.O. Box 1450 Alexandria, VA 22313-1450, on April 23, 2006.

Date: 4-23-06



Mark Lauer

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
04th SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

PTO/SB/08A (08-03)
Approved for use through 07/31/2006. OMB 0651-0031
U.S. Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE
APR 27 2006
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	1	2004/0153578	8-04	Elzur			
	2	2004/0054814	3-04	McDaniel			
	3	4,589,063	5/13/86	Shah et al.	710	8	
	4	5,418,912	5-95	Christenson	709	234	
	5	5,535,375	7-96	Eshel	703	27	
	6	5,598,410	1/28/1997	Stone	370	469	
	7	5,633,780	05-1997	Cronin	361	220	
	8	5,682,534	10-97	Kapoor et al.	709	203	
	9	5,742,765	04-1998	Wong et al.	709	230	
	10	5,778,013	7/7/98	Jedwab	714	807	
	11	5,778,419	07-1998	Hansen et al.	711	112	
	12	5,848,293	12-1998	Gentry	710	5	
	13	5,872,919	2-99	Wakeland	709	230	
	14	5,878,225	3-99	Bilansky et al.	709	227	
	15	5,892,903	04-1999	Klaus	709	227	
	16	5,920,566	07-1999	Hendel et al.	370	401	
	17	5,991,299	11-1999	Radogna et al.	370	392	
	18	6,041,058	03-2000	Flanders et al.	370	401	
	19	6,049,528	4-00	Hendel et al.	370	235	
	20	6,078,733	06-2000	Osborne	709	250	
	21	6,097,734	08-2000	Gotesman et al.	370	474	
	22	6,115,615	09-2000	Ota et al.	455	422.1	
	23	6,122,670	09-00	Bennett et al.			
	24	6,223,242	05-2001	Mayer et al.	370	392	
	25	6,246,683	06-2001	Connery et al.	709	250	
	26	6,289,023	09-2001	Dowling et al.	370	419	

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE APR 27 2006 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	27	6,289,023	09-2001	Dowling et al.	370	419	
	28	6,385,647	05-2002	Wills et al.	709	217	
	29	6,487,654	11-02	Dowling			
	30	6,490,631	12-02	Teich et al.	709	250	
	31	6,591,310	7-03	Johnson	710	3	
	32	6,650,640	11-2003	Muller et al.	370	392	
	33	6,681,364	1-04	Calvignac et al.			
	34	6,697,868	02-2004	Craft et al.	709	230	
	35	6,807,581	10-2004	Starr et al.	709	250	
	36	6,912,522	06-05	Edgar			
	37	6,941,386	09-2005	Craft et al.	709	250	
	38						
	39						
	40						
	41						
	42						
	43						
	44						
	45						
	46						
	47						
	48						
	49						
	50						
	51						
	52						

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

PATENT APPLICATION FEE DETERMINATION RECORD
Effective October 1, 2001

Application or Docket Number: 10260878

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS	24	
FOR 09/27/02	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	24 minus 20=	* 4
INDEPENDENT CLAIMS	3 minus 3 =	* 0
MULTIPLE DEPENDENT CLAIM PRESENT		☐

* If the difference in column 1 is less than zero, enter "0" in column 2

SMALL ENTITY TYPE ☐		OR	OTHER THAN SMALL ENTITY	
RATE	FEE		RATE	FEE
BASIC FEE	370.00	OR	BASIC FEE	740.00
X$ 9=		OR	X$18=	72
X42=		OR	X84=	
+140=		OR	+280=	
TOTAL		OR	TOTAL	72

CLAIMS AS AMENDED - PART II

04/27/06

AMENDMENT A

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	* 24	Minus	** 24	= 0
Independent	* 3	Minus	*** 3	= 0
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X42=		OR	X84=	
+140=		OR	+280=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	0

AMENDMENT B

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X42=		OR	X84=	
+140=		OR	+280=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X42=		OR	X84=	
+140=		OR	+280=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 357

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/260,878	09/27/2002	Laurence B. Boucher	ALA-006E	9902

24501 7590 08/09/2006

MARK A LAUER
6601 KOLL CENTER PARKWAY
SUITE 245
PLEASANTON, CA 94566

EXAMINER	
DENNISON, JERRY B	
ART UNIT	**PAPER NUMBER**
2143	

DATE MAILED: 08/09/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

PTO-90C (Rev. 10/03)

Office Action Summary	Application No.	Applicant(s)	
	10/260,878	BOUCHER ET AL.	
	Examiner	Art Unit	
	J. Bret Dennison	2143	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 2 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *27 April 2006*.

2a)☐ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-24* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☐ Claim(s) _____ is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date *4-27-06 3-9-05*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

Requirement for Information under 37 CFR 1.105

Applicant and the assignee of this application are required under 37 CFR § 1.105 to provide the following information that the examiner has determined is reasonably necessary to the examination of this application.

The Applicant is requested to identify any known examples, products, and/or prior art in which, when processing of packets occurs, for each packet, the network layer header and the transport layer header are validated with an interrupt dividing the processing of the network layer header and the transport layer header. The Applicant is requested to explain the meaning of an interrupt with relevance to how it is used in the claimed invention. The Examiner has determined that this identification of examples, products, and/or prior art along with an explanation in reference to the claimed invention is pertinent to the issue of patentability in this case.

The Examiner is authorized to require the submission of this information since the Examiner has determined that such information is relevant to the patentability of the claimed invention. See *Star Fruits S.N.C. v. United States (Fed. Cir. 2005) ("So long as there is some legitimate reason for seeking the information under section 1.105, the applicant has a duty to respond...The Office is authorized under section 1.105 to require any information that is either relevant to patentability under any nonfrivolous legal theory, or is reasonably calculated to lead to such relevant information")*.

The fee and certification requirements of 37 C.F.R. § 1.97 are waived for those documents submitted in reply to this requirement. This waiver extends only to those documents within the scope of this requirement under 37 C.F.R. § 1.105 that are

included in the applicant's first complete communication responding to this requirement. Any supplemental replies subsequent to the first communication responding to this requirement and any information disclosures beyond the scope of this requirement under 37 C.F.R. § 1.105 are subject to the fee and certification requirements of 37 C.F.R. § 1.97.

The applicant is reminded that the reply to this requirement must be made with candor and good faith under 37 CFR 1.56. Where the applicant does not have or cannot readily obtain an item of required information, a statement that the item is unknown or cannot be readily obtained will be accepted as a complete response to the requirement for that item.

This requirement is subject to the provisions of 37 C.F.R. §§ 1.134, 1.135 and 1.136 and has a shortened statutory period of two (2) months. EXTENSIONS OF THIS TIME PERIOD MAY BE GRANTED UNDER 37 CFR 1.136(a).

It is also noted that this requirement is being sent in lieu of an action on the merits. See MPEP 704.14(a) ("A requirement for information under 37 CFR 1.105 is generally prepared as a separate document that may be attached to an Office action on the merits or mailed as a stand alone action. The rule permits a requirement to be included within an Office action, but creating a separate document is preferable because the existence of the requirement is immediately brought to the attention of the recipient and it is more readily routed by the applicant to the parties best able to respond.")

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to J. Bret Dennison whose telephone number is (571) 272-3910. The examiner can normally be reached on M-F 8:30am-5pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, David A Wiley can be reached on (571) 272-3923. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

J. B. D.
Patent Examiner
Art Unit 2143

DAVID WILEY
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
04th SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

U.S.P.T.O. 2006 APR 27 P 4:21 PATENT & TRADEMARK OFFICE WASH. D.C.

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
MB	1	2004/0153578	8-04	Elzur			
	2	2004/0054814	3-04	McDaniel			
	3	4,589,063	5/13/86	Shah et al.	710	8	
	4	5,418,912	5-95	Christenson	709	234	
	5	5,535,375	7-96	Eshel	703	27	
	6	5,598,410	1/28/1997	Stone	370	469	
	7	5,633,780	05-1997	Cronin	361	220	
	8	5,682,534	10-97	Kapoor et al.	709	203	
	9	5,742,765	04-1998	Wong et al.	709	230	
	10	5,778,013	7/7/98	Jedwab	714	807	
	11	5,778,419	07-1998	Hansen et al.	711	112	
	12	5,848,293	12-1998	Gentry	710	5	
	13	5,872,919	2-99	Wakeland	709	230	
	14	5,878,225	3-99	Bilansky et al.	709	227	
	15	5,892,903	04-1999	Klaus	709	227	
	16	5,920,566	07-1999	Hendel et al.	370	401	
	17	5,991,299	11-1999	Radogna et al.	370	392	
	18	6,041,058	03-2000	Flanders et al.	370	401	
	19	6,049,528	4-00	Hendel et al.	370	235	
	20	6,078,733	06-2000	Osborne	709	250	
	21	6,097,734	08-2000	Gotesman et al.	370	474	
	22	6,115,615	09-2000	Ota et al.	455	422.1	
	23	6,122,670	09-00	Bennett et al.			
	24	6,223,242	05-2001	Mayer et al.	370	392	
	25	6,246,683	06-2001	Connery et al.	709	250	
MB	26	6,289,023	09-2001	Dowling et al.	370	419	

Examiner [signature] Date Considered 7/14/06

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE APR 27 2006 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
MD	27	6,289,023	09-2001	Dowling et al.	370	419	
	28	6,385,647	05-2002	Wills et al.	709	217	
	29	6,487,654	11-02	Dowling			
	30	6,490,631	12-02	Teich et al.	709	250	
	31	6,591,310	7-03	Johnson	710	3	
	32	6,650,640	11-2003	Muller et al.	370	392	
	33	6,681,364	1-04	Calvignac et al.			
	34	6,697,868	02-2004	Craft et al.	709	230	
	35	6,807,581	10-2004	Starr et al.	709	250	
	36	6,912,522	06-05	Edgar			
MD	37	6,941,386	09-2005	Craft et al.	709	250	
	38						
	39						
	40						
	41						
	42						
	43						
	44						
	45						
	46						
	47						
	48						
	49						
	50						
	51						
	52						

Examiner MD Date Considered 7/14/06

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
03rd SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

MAR 0 9 2005 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	5,524,250	6/4/96	Chesson et al.	395	775	
	B	5,619,650	4/8/97	Bach et al.	395	200.01	
	C	5,727,142	3/10/98	Chen	395	181	
	D	5,802,258	9/1/98	Chen	395	182.08	
	E	5,898,713	4/27/99	Melzer et al.	371	53	
	F	6,021,507	2/1/00	Chen	714	2	
	G	6,047,323	4/4/00	Krause	709	227	
	H						

OTHER ART—NON PATENT LITERATURE DOCUMENTS

*Examiner Initial	Cite No.	(Including Author, Title, Date, Pertinent Pages, Etc.)
	1	Schwaderer et al., IEEE Computer Society Press publication entitled, "XTP in VLSI Protocol Decomposition for ASIC Implementation", from 15th Conference on Local Computer Networks, 5 pages, Sept. 30 – Oct. 3, 1990.
	2	Beach, Bob, IEEE Computer Society Press publication entitled, "UltraNet: An Architecture for Gigabit Networking", from 15th Conference on Local Computer Networks, 18 pages, Sept. 30 – Oct. 3, 1990.
	3	Chesson et al., IEEE Syposium Record entitled, "The Protocol Engine Chipset", from Hot Chips III, 16 pages, Aug. 26-27, 1991.
	4	Maclean et al., IEEE Global Telecommunications Conference, Globecom '91, presentation entitled, "An Outboard Processor for High Performance Implementation of Transport Layer Protocols", 7 pages, Dec. 2-5, 1991.
	5	Ross et al., IEEE article entitled "FX1000: A high performance single chip Gigabit Ethernet NIC", from Compcon '97 Proceedings, 7 pages, Feb. 23-26, 1997.
	6	Strayer et al., "Ch. 9: The Protocol Engine" from XTP: The Transfer Protocol, 12 pages, July 1992.
	7	Publication entitled "Protocol Engine Handbook", 44 pages, Oct. 1990.
	8	Koufopavlou et al., IEEE Global Telecommunications Conference, Globecom '92, presentation entitled, "Parallel TCP for High Performance Communication Subsystems", 7 pages, Dec. 6-9, 1992
	9	Lilienkamp et al., Publication entitled "Proposed Host-Front End Protocol", 56 pages, Dec. 1984.

Examiner [signature] Date Considered 7/14/06

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

IFW

2143

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of Laurence B. Boucher, et al.		Ser. No:	10/260,878
Filing Date:	September 27, 2002	Examiner:	J. Bret Dennison
Atty. Docket No:	ALA-006E	GAU:	2143

For: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

August 25, 2006

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

5th Supplemental Information Disclosure Statement

Sir:

Pursuant to 37 C.F.R. §§ 1.56, 1.97 and 1.98, applicants bring thirty-five reference documents to the Examiner's attention. Included are copies of two foreign patent reference documents, three non-patent reference documents, and a two-page form PTO-1449 listing these documents separately from the thirty U.S. Patent reference documents. Copies of the thirty U.S. Patent reference documents are not enclosed. Also enclosed is a check in the amount of $180.00.

Applicants respectfully point out that five of the references cited in the enclosed form PTO-1449 have been previously cited in Information Disclosure Statements filed for this Application Ser. No: 10/260,878. Specifically, U.S. Patent Publication numbers 2001/0025315A1 and 2001/0004354A1 and U.S. Patent number 6,173,333 were cited on lines A-C, sheet 13 of 14, on the applicants' Information Disclosure Statement form PTO-1449 filed on July 11, 2003. U.S. Patent number 5,598,410 was cited on line A, sheet 1 of 1, on the applicants' Supplemental Information Disclosure Statement form PTO-1449 filed on June 7, 2004. In addition, U.S. Patent number 5,517,668 was cited on line A, sheet 1 of 1, on the applicants' 2nd Supplemental Information Disclosure Statement form PTO-1449 filed on June 21, 2004. Applicants have not, however, received confirmation from the Patent Office that those references have been considered by the Examiner.

08/30/2006 EFLORES 00000059 10260878
01 FC:1806 180.00 OP

Citation of these documents shall not be construed as an admission that the documents are prior art with respect to the instant invention, a representation that a search has been made, or an admission that the information cited herein is, or is considered to be, material to patentability as defined in 37 C.F.R. § 1.56(b).

Respectfully submitted,

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Commissioner for Patents, P.O. Box 1450 Alexandria, VA 22313-1450, on August 25, 2006.

Date: 8-25-06

Mark Lauer

Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J Bret Dennison
	Attorney Docket No.: ALA-006E

OIPE 5th Floor AUG 28 2006 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	1	2003/0165160	9/4/03	Minami et al.	370	466	
	2	2003/0066011	4/11/02	Oren	714	758	
	3	2001/0053148	12/20/01	Bilic et al.	370	389	
	4	2001/0048681	12/6/01	Bilic et al.	370	389	
	5	2001/0014954	8/16/01	Purcell et al.	714	4	
	6	2001/0014892	8/16/01	Gaither et al.	707	200	
	7	2001/0013059	8/9/01	Dawson et al.	709	217	
	8	2001/0025315A1	09/27/01	Jolitz	707	001	
	9	2001/0004354A1	06/21/01	Jolitz	709	240	
	10	5,058,110	10/15/91	Beach et al.	370	85.6	
	11	5,517,668	5/14/96	Szwerinski et al.	395	800	
	12	5,598,410	1/28/97	Stone	370	469	
	13	5,970,804	10/26/99	Osborne	395	200.75	
	14	6,038,562	3/14/00	Anjur et al.	707	10	
	15	6,067,569	5/23/00	Khaki et al.	709	224	
	16	6,070,200	5/30/00	Gates et al.	710	20	
	17	6,101,555	8/8/00	Goshey et al.	709	321	
	18	6,145,017	11/7/00	Ghaffari	710	5	
	19	6,157,955	12/5/00	Narad et al.	709	228	
	20	6,172,980	1/9/01	Flanders et al.	370	401	
	21	6,173,333	07/18/97	Jolitz et al.	709	240	
	22	6,202,105	3/13/01	Gates et al.	710	20	
	23	6,279,051	8/21/01	Gates et al.	710	20	
	24	6,298,403	10/2/01	Suri et al.	710	100	
	25	6,334,153	12/25/01	Boucher et al.	709	230	
	26	6,345,302	02/05/02	Bennett et al.	709	236	

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
5th SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J. Bret Dennison
	Attorney Docket No.: ALA-006E

OIPE IAP84 AUG 28 2006 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	27	6,389,479	5/14/02	Boucher	709	243	
	28	6,421,742	7/16/02	Tillier	710	1	
	29	6,480,489	11/12/02	Muller et al.	370	389	
	30	6,526,446	2/25/03	Yang et al.	709	230	

Foreign Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	1	WO 01/59966	8/16/01	PCT/US00/06475			
	2	WO 01/86430	11/15/01	PCT/US01/15180			

OTHER ART – NON-PATENT LITERATURE DOCUMENTS

*Examiner Initial	Cite No.	(Including Author, Title, Date, Pertinent Pages, Etc.)
	1	Thia, Y.H. Publication entitled "High-Speed OSI Protocol Bypass Algorithm with Window Flow Control", Protocols for High Speed Networks, pages 53-68, 1993.
	2	Thia, Y.H. Publication entitled " A Reduced Operational Protocol Engine (ROPE) for a multiple-layer bypass architecture", Protocols for High Speed Networks, pages 224-239, 1995.
	3	U.S. Provisional Patent Application No.: 60/283,896, Titled: CRC Calculations for Out of Order PUDs, Filed April 12, 2003, Inventor: Amit Oren, Assignee: Siliquent Technologies Ltd.

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
16 August 2001 (16.08.2001)

PCT

(10) International Publication Number
WO 01/59966 A1

(51) International Patent Classification[7]: H04J 3/22

(21) International Application Number: PCT/US00/06475

(22) International Filing Date: 9 March 2000 (09.03.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/500,119 8 February 2000 (08.02.2000) US

(71) Applicant *(for all designated States except US)*: NISHAN SYSTEMS, INC. [US/US]; 3850 North First Street, San Jose, CA 95134-1702 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: LATIF, Aamer [US/US]; 2047 Blue Ridge Drive, Milpitas, CA 95035 (US). MULLENDORE, Rodney, N. [US/US]; 305 Elan Village Lane, Apt. #434, San Jose, CA 95134 (US). WHITE, Joseph, L. [US/US]; 2000 Crystal Springs Road, Apt. 5-21, San Bruno, CA 94066 (US). UCHINO, Brian, Y. [US/US]; 1348 Marlin Avenue, Foster City, CA 94404 (US).

(74) Agents: GRAY, Gerald, T. et al.; Townsend and Townsend and Crew LLP, Two Embarcadero Center, 8th floor, San Francisco, CA 94111 (US).

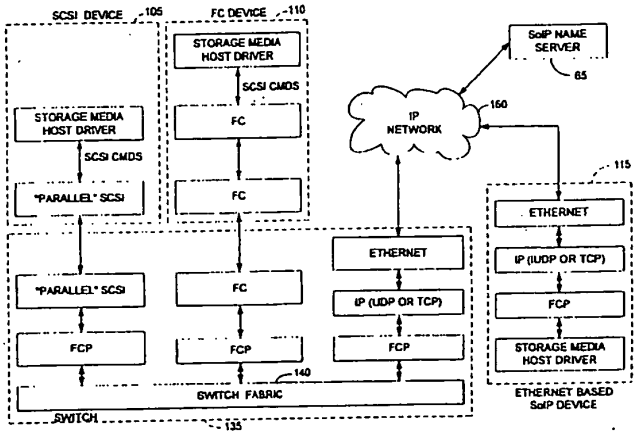
(81) Designated States *(national)*: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR TRANSFERRING DATA BETWEEN DIFFERENT NETWORK DEVICES OVER AN IP NETWORK



(57) Abstract: A method and apparatus for transferring data between IP devices (115) and SCSI (105) or Fibre Channel FC (110) devices over an IP network (160). The device interfaces may be either SCSI, FC or IP interface such as Gigavit Ethernet. Data is switched between SCSI and IP, FC and IP, or between SCSI and FC. Data can also be switched from SCSI to SCSI, IP to IP, or FC to FC. The port interfaces provide the conversion from the input frame format to an internal frame format, which can be routed within the apparatus. The amount of processing performed by each port interface is dependent on the interface type. The apparatus permits rapid transfer of information packets between interfaces at latency levels meeting the stringent requirements for storage protocols. The configuration control can be applied to each port on a switch, and each switch on the network, via an SNMP or Web-based interface.

BEST AVAILABLE COPY

WO 01/59966 A1

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND APPARATUS FOR TRANSFERRING DATA BETWEEN DIFFERENT NETWORK DEVICES OVER AN IP NETWORK

CROSS REFERENCES TO RELATED APPLICATIONS

This application is related to U.S. Provisional Patent Application Serial No. 60/123,606 (Atty. Docket No. 019678-000100), filed March 10, 1999, entitled "METHOD AND APPARATUS FOR TRANSFERRING DATA BETWEEN IP NETWORK DEVICES AND SCSI AND FIBRE CHANNEL DEVICES OVER AN IP NETWORK," the disclosure of which is hereby incorporated by reference in its entirety.

BACKGROUND OF THE INVENTION

The present invention relates to transferring information between storage devices and a network via a switched, packetized communications system. In particular, the present invention relates to methods and apparatus for receiving, translating, and routing data packets between SCSI (Small Computer Systems Interface), Fibre Channel and Ethernet devices in a flexible, programmable manner.

In enterprise computing environments, it is desirable and beneficial to have multiple servers able to directly access multiple storage devices to support high-bandwidth data transfers, system expansion, modularity, configuration flexibility and optimization of resources. In conventional computing environments, such access is typically provided via file system level Local Area Network (LAN) connections, which operate at a fraction of the speed of direct storage connections. As such, access to storage systems is highly susceptible to bottlenecks.

Storage Area Networks (SANs) have been proposed as one method of solving this storage access bottleneck problem. By applying the networking paradigm to storage devices, SANs enable increased connectivity and bandwidth, sharing of resources,

and configuration flexibility. The current SAN paradigm assumes that the entire network is constructed using Fibre Channel switches. Therefore, most solutions involving SANs require implementation of separate networks: one to support the normal LAN and another to support the SAN. The installation of new equipment and technology, such as new equipment at the storage device level (Fibre Channel interfaces), the host/server level (Fibre Channel adapter cards) and the transport level (Fibre Channel hubs, switches and routers), into a mission-critical enterprise computing environment could be described as less than desirable for data center managers, as it involves replication of network infrastructure, new technologies (i.e., Fibre Channel), and new training for personnel. Most companies have already invested significant amounts of money constructing and maintaining their network (e.g., based on Ethernet and/or ATM). Construction of a second high-speed network based on a different technology is a significant impediment to the proliferation of SANs. Therefore, a need exists for a method and apparatus that can alleviate problems with access to storage devices by multiple hosts, while retaining current equipment and network infrastructures, and minimizing the need for new training for data center personnel.

In general, a majority of storage devices currently use "parallel" SCSI or Fibre Channel data transfer protocols whereas most LANs use an Ethernet protocol, such as Gigabit Ethernet. SCSI, Fibre Channel and Ethernet are protocols for data transfer, each of which uses a different individual format for data transfer. For example, SCSI commands were designed to be implemented over a parallel bus architecture and therefore are not packetized. Fibre Channel, like Ethernet, uses a serial interface with data transferred in packets. However, the physical interface and frame formats between Fibre Channel and Ethernet are not compatible. Gigabit Ethernet was designed to be compatible with existing Ethernet infrastructures and is therefore based on an Ethernet packet architecture. Because of these differences there is a need for new methods and apparatus to allow efficient communication between these protocols.

SUMMARY OF THE INVENTION

The present invention solves the above and other problems, thereby advancing the state of the useful arts, by providing methods and apparatus for transferring data between storage device interfaces and network interfaces. In particular, the present

invention brings sophisticated SAN capabilities to existing enterprise computing configurations, without the installation of costly Fibre Channel switches and hubs, by providing the means for Internet Protocol (IP) devices to transparently communicate with SCSI and Fibre Channel devices over an IP network. The present invention accomplishes this through the use of Fibre Channel Protocol (FCP), an industry standard developed for implementation of SCSI commands over a Fibre Channel network. The invention allows the storage devices to retain the use of standard SCSI and Fibre Channel storage interfaces and construct a SAN using a company's existing network infrastructure. Therefore, no changes are required in host bus adapters (HBA) or storage devices (e.g. disk drives, tape drives, etc).

According to the present invention, methods and apparatus are provided for transferring data between IP devices (including, but not limited to, Gigabit Ethernet devices) and SCSI or Fibre Channel devices. The device interfaces may be either SCSI, Fibre Channel or IP interfaces such as Gigabit Ethernet. Data is switched between SCSI and IP, Fibre Channel and IP, or between SCSI and Fibre Channel. Data can also be switched from SCSI to SCSI, Fibre Channel to Fibre Channel and IP to IP. The port interfaces provide the conversion from the input frame format to an internal frame format, which can be routed within the apparatus. The apparatus may include any number of total ports. The amount of processing performed by each port interface is dependent on the interface type. The processing capabilities of the present invention permit rapid transfer of information packets between multiple interfaces at latency levels meeting the stringent requirements for storage protocols. The configuration control can be applied to each port on a switch and, in turn, each switch on the network, via an SNMP or Web-based interface, providing a flexible, programmable control for the apparatus.

According to one aspect of the present invention, a method is provided for routing data packets in a switch device in a network such as a SAN. The method typically comprises the steps of receiving a packet from a first network device at a first port interface of the switch device, wherein the packet is one of a SCSI formatted packet (i.e., SCSI formatted data stream converted into a packet), a Fibre Channel (FC) formatted packet and an Internet protocol (IP) formatted packet, wherein the first port interface is communicably coupled to the first network device, and converting the received packet into a packet having an internal format. The method also typically includes the steps of routing the internal format packet to a second port interface of the

switch device, reconverting the internal format packet to one of a SCSI formatted packet, an FC formatted packet or an IP formatted packet, and transmitting the reconverted packet to a second network device communicably coupled to the second port interface.

According to another aspect of the present invention, a network switch device is provided which typically comprises a first port interface including a means for receiving data packets from a network device, wherein the receiving means receives one of a SCSI formatted packet and a Fibre Channel (FC) formatted packet from a first network device, and a means for converting received packets into packets having an internal format, wherein the received data packet is converted into a first packet having the internal format. The switch device also typically comprises a second port interface including a means for reconverting packets from the internal format to an IP format, wherein the first packet is converted into a packet having an IP format, and a means for transmitting IP packets to a network, wherein the IP formatted packet is transmitted to an IP network. A means for routing the first packet to the second port interface is also provided.

According to yet another aspect of the present invention, a network switch device is provided which typically comprises a first port interface including a means for receiving data packets from an IP network, wherein the first interface means receives a packet in an IP format, and a means for converting received packets into packets having an internal format, wherein the received packet is converted into a first packet having an internal format. The switch device also typically comprises a second port interface including a means for reconverting packets having the internal format to packets having the SCSI format, and a means for transmitting reconverted packets to a SCSI network device. The switch device further typically includes a third port interface having a means for reconverting packets having the internal format to packets having the FC format, and a means for transmitting reconverted packets to a FC network device. A means for routing packets between the first, second and third port interfaces is also typically provided. In operation, wherein if the first packet is routed to the second port interface, the first packet is converted to the SCSI format and transmitted to the SCSI network device, and wherein if the first packet is routed to the third port interface, the first packet is converted to the FC format and transmitted to the FC network device.

According to a further aspect of the present invention, a network switch device is provided for use in a storage area network (SAN). The switch device typically

comprises a first port interface communicably coupled to a SCSI device, wherein the first port interface converts SCSI formatted data packets received from the SCSI device into data packets having an internal format, and wherein the first port interface converts data packets having the internal format into SCSI formatted data packets. The switch device also typically comprises a second port interface communicably coupled to a FC device, wherein the second port interface converts FC formatted data packets received from the FC device into data packets having the internal format, and wherein the second port interface converts data packets having the internal format into FC formatted data packets. The switch device further typically includes a third port interface communicably coupled to a IP device, wherein the third port interface converts IP formatted data packets received from the IP device into data packets having the internal format, and wherein the third port interface converts data packets having the internal format into IP formatted data packets, and a switch fabric for routing data packets having the internal format between the first, second and third port interfaces. In typical operation, when a first one of the SCSI, FC and IP devices sends a first data packet to a second one of the SCSI, FC and IP devices, the port interface coupled to the first device converts the first data packet to a packet having the internal format and routes the internal format packet through the switch fabric to the port interface coupled to the second device, wherein the port interface coupled to the second device reconverts the internal format packet into the format associated with the second device and sends the reconverted packet to the second device.

According to yet a further aspect of the present invention, a network switch device for use in a storage area network (SAN) is provided. The switch may comprise any combination of Fibre Channel, SCSI, Ethernet and Infiniband ports, and may comprise any number of total ports. The switch device typically comprises a first port interface communicably coupled to one of a SCSI device(s), an FC device, or an IP device, a second port interface, wherein the second port interface is configurable to communicate with either a FC device or an Ethernet device, and a switch fabric for routing data packets having the internal format between the first and second port interfaces. In typical operation, when the second port interface is configured to communicate with a FC device, the second port interface converts FC formatted data packets received from the FC device into data packets having an internal format, and wherein the second port interface converts data packets having the internal format received from the switch fabric into FC formatted data packets, and wherein when the

second port interface is configured to communicate with an Ethernet device, the second port interface converts Ethernet formatted data packets received from the Ethernet device into data packets having the internal format, and wherein the second port interface converts data packets having the internal format received from the switch fabric into Ethernet formatted data packets. The second port interface can be either self-configurable or user configurable.

Reference to the remaining portions of the specification, including the drawings and claims, will realize other features and advantages of the present invention. Further features and advantages of the present invention, as well as the structure and operation of various embodiments of the present invention, are described in detail below with respect to the accompanying drawings. In the drawings, like reference numbers indicate identical or functionally similar elements.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates an example of a SAN constructed according to the present invention;

Figure 2 is a block diagram of an overview of the Storage over Internet Protocol (SoIP) implementation;

Figure 3 illustrates the required protocol conversion steps between Fibre Channel, SCSI and IP devices in the apparatus switch fabric according to an embodiment of the present invention;

Figure 4 is an overview of the legacy storage protocol conversion method by which the functionality of the invention is achieved;

Figure 5 is a high level switch diagram outlining the basic architecture of the physical apparatus according to an embodiment of the present invention;

Figures 6a-c illustrate FCP packet encapsulation according to an embodiment of the present invention;

Figure 7 shows the frame flow for the "session" initialization for Fibre Channel devices connected to an SoIP network;

Figures 8 and 9 show the flow of data frames for a node login initiated by FC port A of switch 1 to FC Port B of switch 2 located remotely according to an embodiment of the present invention;

Figure 10 shows the routing of Port Login Request and Response frames for local FC ports according to an embodiment of the present invention;

Figure 11 shows an example of the address domains which exist in a network according to one embodiment of the present invention;

Figures 12a-d illustrate a network architecture and address tables for a Third Party Command example;

Figure 13 illustrates layer 2 FCP packet encapsulation according to an embodiment of the present invention;

Figures 14a-c illustrate examples of UDP Frame demultiplexing according to embodiments of the present invention;

Figure 15 is a high level block diagram which illustrates the basic architecture for a switch port that supports both Fibre Channel and Ethernet according to an embodiment of the present invention;

Figure 16 is a high level block diagram which illustrates the basic architecture for a switch port that supports both Fibre Channel and Ethernet, where two routing blocks are combined into a single block according to an embodiment of the present invention;

Figure 17 is a high level block diagram which illustrates the basic architecture for a switch port that supports both Fibre Channel and Ethernet wherein low-level port interface logic blocks are combined according to an embodiment of the present invention;

Figure 18 is a high level block diagram which illustrates the basic architecture for a switch port that supports both Fibre Channel and Ethernet using a Field Programmable Gate Array (FPGA) according to an embodiment of the present invention;

Figure 19 shows a block diagram of a common FC/Gigabit Ethernet port combined with a GBIC interface according to an embodiment of the present invention; and

Figure 20 illustrates the architecture of an intelligent network interface card (NIC) according to an embodiment of the present invention.

DESCRIPTION OF THE SPECIFIC EMBODIMENTS

Figure 1 illustrates an example of a storage area network (SAN) 10 according to an embodiment of the present invention. As shown, network 10 includes numerous storage devices, such as tape libraries 15, RAID drives 20 and optical drives 25 (e.g., CD, DVD, etc.) and servers 30. The storage devices can be either storage targets (e.g., tape libraries 15, RAID drives 20, etc.) or initiators (e.g., servers 30). Note that a device could be both an initiator and a target. In a preferred embodiment, the invention is implemented in a switching device 35 within network 10. For example, as shown in Figure 1, each switching device 35 is an "edge" switch which provides the connectivity between nodes (i.e., one or more storage devices) and a network 40. In other words, the switch resides on the "edge" of the network where the devices are located. Each edge switch 35 allows connected storage elements to communicate through the edge switch with no traffic being sent to network 40. Each edge switch 35 also allows storage elements connected to different edge switches to communicate with each other through network 40. In a preferred embodiment, network 40 is an Ethernet network, but other networks may be used, for example, Asynchronous Transfer Mode (ATM)-based or FDDI-based networks, or the like.

In one embodiment, a switching device 35 is implemented in an SoIP (Storage over Internet Protocol) storage area network (SAN) as shown in Figure 2. According to the present invention, SoIP is a framework for transporting SCSI commands and data over IP networks using the Fibre Channel Protocol for SCSI (FCP) for communication between IP networked storage devices. A majority of storage devices currently communicate using either a "parallel" SCSI bus or a Fibre Channel serial interface. FCP is an FC-4 Upper Layer Protocol for sending SCSI commands and data over a Fibre Channel network yielding a "serial" SCSI network. The SoIP framework enables FCP for use on an IP network by defining the SoIP protocol. Storage devices and host bus adapters operating the SoIP protocol form a storage area network (SAN) directly on an IP network. This framework offers an enormous advantage in the installation and utility of SANs.

As shown in Figure 2, each SoIP device 50 converts SCSI commands and data into FCP data frames in FCP block 52. The SoIP protocol layer block 54 then encapsulates these FCP frames in multiple IP packets using either the User Datagram

Protocol (UDP) or Transport Control Protocol (TCP). IP port 56 forwards the packet to IP network 60, which routes the IP packets between the devices 50 or to switch 35. IP network 60 is preferably an Ethernet network, but may be based on any IP-compatible media including ATM, FDDI, SONET and the like. The storage name server 65 serves as a database where devices store their own information and retrieve information on other devices in the SoIP network. The SoIP proxy 70 performs protocol conversion between SoIP based on UDP and SoIP based on TCP.

Because the majority of storage devices currently use "parallel" SCSI or Fibre Channel protocols, the transition to SoIP-based SANs may be hampered unless such "legacy" devices can be connected to an SoIP network. For these "legacy" devices, a switch as shown in Figure 3 is provided for connection into an SoIP SAN.

Figure 3 illustrates data exchange between storage devices using a switch 135 according to an embodiment of the present invention. In this embodiment, switch 135 is configured to receive data from different interfaces, each of which has a different data or frame format. SCSI device 105 transmits data using a "parallel" SCSI interface 106, Fibre Channel (FC) device 110 transmits data using Fibre Channel interface 111 and Ethernet device 115 transmits data using Ethernet interface 116. Switch 135 translates data received from a source port in one of the three different formats into an internal format and transfers the data in the internal format through switch fabric 140 to a destination port. The destination port translates the data back into the native format appropriate for the connection thereto.

In this embodiment, each device, e.g., SCSI device 105, FC device 110, Ethernet device 115, or generic IP device 120 (e.g., disk drive, tape drive, server), performs storage operations based on the SCSI Command Set. For Fibre Channel device 110, the SCSI commands and data are converted to FCP and transmitted using Fibre Channel interface 111. For SCSI device 105 the SCSI commands and data are transferred directly using a "parallel" bus 106. In this embodiment, the SCSI port interface 125 of switch 135 acts like a SCSI to FC bridge so that the SCSI port looks like an FC port from the point of view of switch fabric 140. As shown, the SCSI data is preferably converted to FCP, and is not actually transmitted using a Fibre Channel interface. For Ethernet device 115, SCSI commands and data are converted to FCP and then encapsulated in an IP packet using UDP or TCP. The IP packet is then encapsulated in an Ethernet frame and transmitted using Ethernet interface 116. Note that the term "SCSI device" implies a

device with a "parallel SCSI bus" while the term "Fibre Channel device" implies a device with a Fibre Channel interface. Both devices operate as SCSI devices at the command level. Note that SCSI device 105 does not convert the SCSI commands and data to an FCP format. Therefore, it is not possible to transfer data between FC device 110 and SCSI device 105 directly. As shown in Figure 3, it is possible for all devices connected to switch 135 to exchange data frames because the data format of all interfaces into switch fabric 140 are FCP compatible frames. Also note that it is possible to replace Fibre Channel with another interface. For example, Figure 3 shows a storage device constructed using Ethernet in the same manner as a device is constructed with FC. Ethernet simply replaced Fibre Channel as the media for transport. Infiniband may also be implemented, for example in generic IP device 120. As is well known, Infiniband is an I/O interface that merges the work of NGIO (Next Generation I/O) and Future I/O.

Figure 4 illustrates data exchange between Fibre Channel, SCSI and IP devices in switch apparatus 135 according to an embodiment of the present invention. The example in Figure 4 is for an Ethernet based IP network 160, however any other IP networks based on other protocols such as ATM, FDDI, etc. may be used. Similar to the embodiment in Figure 3, Figure 4 shows the protocol translations which occur for each device. SCSI device 105 communicates with switch 135 using SCSI commands directly with no encapsulation of data or commands in data frames. FC device 110 uses the FCP protocol to send SCSI commands and data to switch 135. Switch 135 converts the received data to a common protocol based on FCP to allow the devices to communicate with each other. In addition, switch 135 performs address translation between the Fibre Channel and SCSI addressing schemes to the IP addressing method as will be discussed in more detail below. This is done transparently so that no changes are required in Fibre Channel device 110 or SCSI device 105, or in any host bus adapters, driver software or application software.

Figure 5 is a high-level switch diagram outlining the basic architecture of a physical switch apparatus 235 according to an embodiment of the present invention. In this embodiment, switch 235 includes three main elements: switch fabric 240, management processor 250 and port interfaces 270. Switch fabric 240 provides a high bandwidth mechanism for transferring data between the various port interfaces 270 as well as between port interfaces 270 and management processor 250. Management processor 250 performs management related functions for switch 235 (e.g. switch

initialization, configuration, SNMP, Fibre Channel services, etc.) primarily through management bus 255.

Port interfaces 270 convert data packets from the input frame format (e.g., parallel SCSI, FC, or Ethernet) to an internal frame format. The internal frame format data packets are then routed within switch fabric 240 to the appropriate destination port interface. Port interfaces 270 also determine how packets are routed within the switch. The amount of processing performed by each port interface 270 is dependent on the interface type. SCSI ports 270_1 and 270_2 provide the most processing because the SCSI interface is half-duplex and it is not frame oriented. The SCSI port interfaces 270_1 and 270_2 also emulate the functionality of a SCSI host and/or target. Fibre Channel ports 270_3 and 270_4 require the least amount of processing because the internal frame format is most compatible with Fibre Channel. In essence, IP ports 270_5 and 270_6 (e.g., Ethernet ports) and SCSI ports 270_1 and 270_2 convert data received into an internal frame format before sending the packets through switch fabric 240.

Because FCP frames are not directly compatible with an Ethernet interface as they are with a Fibre Channel interface, the transmission of FCP packets on an Ethernet interface requires that an FCP frame be encapsulated in an Ethernet frame as shown in Figure 6a.

Figure 6a illustrates FCP packet encapsulation in an IP frame carried over an Ethernet frame according to an embodiment of the present invention. Field Definitions for Figure 6a include the following:

DA: Ethernet Destination Address (6 bytes).

SA: Ethernet Source Address (6 Bytes).

TYPE: The Ethernet packet type.

CHECKSUM PAD: An optional 2-byte field which may be used to guarantee that the UDP checksum is correct even when a data frame begins transmission before all of the contents are known. The CHECKSUM PAD bit in the SoIP Header indicates if this field is present.

ETHERNET CRC: Cyclic Redundancy Check (4 bytes).

As shown in Figure 6a, the SoIP Header field contain the following parameters:

CLASS: This 4-bit field indicates the class of service. In one embodiment, only the values 2 or 3 are used.

VERS: This 4-bit field indicates the protocol version of SoIP.

SoIP FLAGS: This 8-bit field contains bits that indicate various parameters for a data frame as shown in Figure 6b.

In Figure 6a, the User Datagram Protocol (UDP) Header is the protocol used within the IP packet. TCP may also be used. The UDP header, defined in RFC 768, is 8 bytes in length consisting of four 16-bit fields as shown in Figure 6c, with the following field definitions:

SOURCE PORT: An optional field. When meaningful, it indicates the port of the sending process, and may be assumed to be the port to which a reply should be addressed in the absence of other information. If not used, a value of zero is inserted.

DESTINATION PORT: has a meaning within the context of a particular internet destination address.

LENGTH: the length, in bytes, of the user datagram including the UDP header and data (thus, if there were no data in the datagram, the length would be 8). For an encapsulated FCP packet, the UDP Length is the sum of the UDP Header Length, FCP Header length, and FCP Payload length and optionally the checksum pad.

CHECKSUM: the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with bytes of zero at the end (if necessary) to make a multiple of 2 bytes.

In one embodiement, a switch 235 encapsulates FC packets into an Ethernet Frame with a "wrapper" around the FC information. The encapsulation of an FCP data frame in an Ethernet packet may require that the FCP data frame be limited in size because the maximum FCP data frame size is 2136 bytes (24 byte header + 2112 byte payload) whereas an Ethernet packet has a maximum size of 1518 bytes. The use of Ethernet Jumbo Frames, which permit packet sizes up to 9 Kbytes to be used, eliminates the need to limit the Fibre Channel frame size. However, support for Ethernet jumbo

frames is limited within the existing network infrastructure. Therefore, FCP data frames need to be limited otherwise a large FCP data frame may need to be "fragmented" into 2 separate Ethernet frames. The Login procedures defined in the Fibre Channel standard allows devices to negotiate the maximum payload with the switch fabric 240. Thus, the switch fabric 240 can respond to a login with a smaller payload size than the maximum (e.g., 1024 bytes). Switch 235 makes use of this fact to limit FC packets to a size which can be encapsulated in an Ethernet packet to eliminate the need for fragmenting FC packets. According to one embodiment, a node's maximum receive data field size is provided to switch fabric 240 during "Fabric Login" and to each destination node during "Port Login." The fabric or node being "logged into" generates a login response which indicates the maximum receive data field size for data frames it is capable of receiving. Note that these values may not be the same. For example, a fabric may have the maximum allowed size of 2112 bytes while a node may limit the maximum size to 1024 bytes (e.g. the Hewlett-Packard Tachyon-Lite Fibre Channel Controller). A source node may not transmit a data frame larger than the maximum frame size as determined for the login response.

Since an encapsulated FCP data frame cannot be larger than the maximum Ethernet packet size, an upper limit is placed on the frame payload size during login by a device. According to one embodiment, the upper limit value is set by determining or discovering the maximum IP datagram size and subtracting 60 bytes to account for the various headers and trailers. For example, for an Ethernet Frame, the upper limit value equals 1440 bytes. That is, the payload for an FCP Frame cannot exceed 1440 bytes in size. This limit is established because an FCP Frame being transported across an IP network will not be allowed to fragment. Allowing IP datagrams to fragment degrades network performance and so most networks rarely fragment. An IP header's Do Not Fragment Flag can be used to prevent the IP layer from fragmenting the datagram. Even with node login setting an appropriate size for the FCP payload, this bit is set to ensure that fragmentation does not occur. According to one embodiment, the payload is padded to a multiple of 4 bytes to make it easy to convert frames being sent to legacy FC devices.

Each switch 235 preferably makes use of the Buffer to Buffer Receive Data Field size to force end nodes to communicate with data frames that will fit within an IP packet carried over an Ethernet link. According to an embodiment of the present invention, one method for enforcing the maximum frame size is to intercept Node Login

and Node Response frames which are redirected to the management processor 250, which adjusts the Buffer to Buffer Receive Data Field Size in the frame as necessary and then routes the modified packet to the original destination.

Following fabric login, a device logs into the name server 65. The login to name server 65 is done to establish the parameters used for communication between the device and name server 65 (e.g. maximum payload size). The devices "register" with name server 65 to provide information for a database, which describes parameters for devices on the network. An initiator can then query the database to determine information about devices in the system thereby eliminating the need to "probe" the system to determine which devices exist. Probing of the network would not be feasible since there are 16 million Fibre Channel addresses. Name server 65 is preferably a node attached to the network, but may be implemented within a switch and can be distributed across the fabric for redundancy and to ensure quick access.

Figure 7 is a flow diagram that illustrates an example of "session" initialization for Fibre Channel devices connected to an SoIP network according to an embodiment of the present invention. During fabric login, switch 235 assigns an IP address to the device from a block of IP addresses assigned to the switch.

Switch 235 uses the IP address assigned to an FC device when packets are transmitted to or received from an IP compatible port. An FC device now has 2 addresses assigned by the SoIP network: the FC address and the IP address. The FC address is used when FC devices communicate in a local Fibre Channel "Island" while the IP address is used when the device communicates across an IP network.

Figures 8 and 9 show the flow of data frames for a Node login initiated by FC port A of switch 1 to FC Port B of switch 2 located remotely from switch 1 according to an embodiment of the present invention. The Login request data frame is redirected by FC port A to the Switch 1 Management Processor which makes any changes required in the Buffer-to-Buffer Receive Data Field Size parameter and then forwards the packet to the original destination port. In the destination switch, (switch 2) there is no redirection of the Login Request packet. As shown in Figure 9, the Login Response frame from FC Port B is redirected to the Switch 2 Management Processor which changes the Buffer-to-Buffer Receive Data Field Size in the response if necessary.

In one embodiment, each Management Processor adjusts its Buffer to Buffer Receive Data Field Size to a value to allow the FCP data frames to fit into an IP

packet which can be transmitted across an Ethernet network without being fragmented. Therefore, each Management Processor may need to perform MTU (Maximum Transmission Unit) discovery to determine a size which does not result in fragmentation of IP packets in the network.

When an FC port performs a Port Login with an FC port which is local (i.e. connected to the same switch), it is not necessary to change the Buffer to Buffer Receive Data Field Size of the Login request or response. This is because, in one embodiment, the switch supports the maximum frame size for transfers between FC ports (on the same switch). However, the FC port interface logic will always redirect the Port Login packets to the switch's Management Processor to simplify the port interface logic. Thus, in this embodiment, the switch looks and acts like an FC switch from the point of view of any FC devices connected thereto. An example of the routing of Port Login Request and Response frames for local FC ports is shown in Figure 10.

According to one embodiment, routing FC Port Login Request/Response packets to the Management Processor allows the Port Login for SCSI ports to be handled by the Management Processor. The Management Processor always handles login for SCSI.

According to one embodiment, an SoIP device is uniquely identified using two parameters: an IP address and an SoIP socket number. Therefore, it is possible for a device to have a unique IP address or for multiple devices to share an IP address. For example, all of the devices on a Fibre Channel arbitrated loop may share an IP address while a server Host Bus Adapter may have a dedicated IP address. In one embodiment, there are two possible modes for assignment of the SoIP socket number: local or global.

A single SoIP device connected directly to an IP network must have a unique IP address in order for the network to be able to route data frames to the device. An IP network will not route traffic based on the SoIP socket number. However, devices connected to a switch (e.g., switch 235) may share an IP address if the switch uses both the IP address and the SoIP socket number when switching data frames.

According to the present invention, an SoIP network SAN with "legacy" Fibre Channel devices attached has different address domains due to the two different address methods used: IP and Fibre Channel. Figure 11 shows an example of the address domains which exist in a network according to one embodiment of the present invention. SoIP devices communicate using IP addresses and the SoIP socket numbers while the

Fibre Channel devices (SCSI devices are treated as Fibre Channel devices by a switch) use Fibre Channel addresses. Each switch 235 performs address translation between the IP and Fibre Channel address domains. Switch 235_1 performs address translation between the IP address domain and FC address domain 1, and Switch 235_2 performs address translation between the IP address domain and FC address domain 2. Each switch 235 assigns an IP address, SoIP socket number and Fibre Channel address to each Fibre Channel device when the device performs a fabric login. A Fibre Channel device only learns about its assigned Fibre Channel address. The assigned IP address, SoIP socket number and Fibre Channel Address are maintained within a translation table (not shown) in the switch. Parallel SCSI devices are assigned their addresses by the switch during initialization of the SCSI port. The Fibre Channel ports direct all Name server requests by a Fibre Channel device to the management processor for processing.

According to one embodiment of the present invention, the management processor converts Fibre Channel Name Server requests into SoIP Name Server requests that are then forwarded to the SoIP Name Server, e.g., implemented in server 280. In one embodiment, the SoIP name server functionality is distributed and thus handled directly by the management processor. Responses from the name server are returned to the management processor where they are converted into Fibre Channel Name Server responses before being forwarded to the port that originated the name server request. When a Fibre Channel device sends data frames to a device not located in its Fibre Channel address domain, switch 235 converts the packet into an SoIP compatible packet. The conversion encapsulates the FCP data frame in an IP data frame as described above. Referring back to Figure 6a, in one embodiment, the IP addresses and SoIP socket numbers are derived by using the Fibre Channel source address (S_ID) and the destination address (D_ID) as "keys" into the IP/Fibre Channel address conversion table on the name server. The Fibre Channel address fields are replaced by the SoIP socket numbers when translating a Fibre Channel data frame to an SoIP data frame. The packet is then transmitted on the IP network and routed using the destination IP address. If the destination device is an SoIP compatible device, the packet is processed directly (i.e., de-encapsulated and processed as an FCP packet) by the destination device. However, if the destination is a Fibre Channel (or parallel SCSI) device, the packet is routed to a switch 235, which receives the packet, de-encapsulates the SoIP packet and replaces the SoIP

socket numbers with the appropriate source and destination Fibre Channel addresses based on the source and destination IP addresses and SoIP socket numbers.

According to one embodiment, local assignment is the preferred method for assigning SoIP socket numbers. In this embodiment, native SoIP devices select their SoIP socket numbers while an SoIP switch (e.g., switch 235) assigns the SoIP socket number for Fibre Channel and SCSI devices attached to the switch. When the SoIP socket number is assigned locally, the value chosen may be any value that results in a unique IP Address/SoIP socket number combination. Devices that share an IP address must be assigned unique SoIP socket numbers in order to create a unique IP Address/SoIP socket number pair. Devices that have a unique IP address may have any desired SoIP socket number. In one embodiment, an SoIP switch assigns the SoIP socket numbers in such a manner as to simplify the routing of received data frames. A switch must also assign a locally significant Fibre Channel address to each "remote" device for use by the local devices in addressing the "remote" devices. These locally assigned addresses are only known by a switch within its Fibre Channel address domain. Thus each switch maintains a set of locally assigned Fibre Channel addresses which correspond to the globally known IP Address/SoIP Port Number pairs defined in the SoIP Name Server.

According to one embodiment, due to the different address domains, each switch 235 intercepts Fibre Channel Extended Link Service requests and responses which have Fibre Channel address information embedded in the payload. Extended Link Service requests and responses are generated infrequently. Therefore, it is acceptable to redirect the Extended Link Service requests to the switch's management processor which makes any necessary changes to the data frame. If an Extended Link Service request/response has no addressing information embedded in the payload, the Management Processor simply retransmits the packet with no modifications.

The IP Address and SoIP socket number assigned to a Fibre Channel or SCSI device are determined by the switch. The assignment of these addresses is implementation dependent. In a preferred embodiment, the SoIP socket number is assigned the device's local Fibre Channel address. In this embodiment, the switch obtains the local Fibre Channel address directly from the received data frame. Alternatively, assignment of the SoIP socket number is based on an incrementing number that can be used as an index into an address table.

In one embodiment, each device is assigned a unique IP address. However, this type of assignment may result in the use of a large number of IP addresses. The use of a single IP address for each device also has implications for routing in the IP network. Therefore, in a preferred embodiment, IP addresses are assigned such that at least a subset of a switch's attached devices share an IP address. For example, an IP address can be assigned to each switch port. Each device attached to that switch port then shares the port's IP address. Thus, an attached Fibre Channel N_Port would have a unique IP address while the devices on a Fibre Channel arbitrated loop attached thereto would share an IP address.

According to one embodiment, Fibre Channel addresses are assigned globally. Globally assigned Fibre Channel addresses provide the maximum compatibility for "legacy" Fibre Channel devices. In this embodiment, the SoIP name server is responsible for managing the allocation of Global Fibre Channel Addresses. A global Fibre Channel address space may need to be supported because in some cases Fibre Channel addresses may be embedded within "third-party" SCSI commands. An example of such a third-party command is COPY. The COPY command instructs another device to copy data. The use of "third-party" commands is rare but when used, either the command would need to be modified for address compatibility or the Fibre Channel addresses would need to be globally assigned.

With reference to the SoIP network shown in Figure 12a, an example third party COPY command will be used to illustrate a problem that occurs with locally assigned Fibre Channel addresses and third-party commands. In this example locally assigned Fibre Channel Addresses are also used as the SoIP socket number. Each device has a unique IP address in this example.

Figure 12b shows the IP Address and SoIP socket number each device has advertised to the Name Server which identifies how the device is addressed within the SoIP network. Each device is uniquely identified by the combination of IP Address and SoIP socket number. Assume that the switches 235_3 and 235_4 and Tape Library C are aware of every device in the system. Tape Library C would then have an address table that is the same as the name server's address table. Switches 235_3 and 235_4 will have assigned local Fibre Channel addresses to each device. Figure 12c illustrates the address table stored on switch 235_3 and Figure 12d illustrates the address table stored on switch

235_4. Because the Fibre Channel addresses are assigned locally, the address assignment is purely arbitrary.

Assume that Server A in local domain 1 sends a COPY command to Server B in local domain 3 indicating that data is to be copied from RAID drive B to Tape Library B, both of which are located in local domain 3. The COPY command will contain the addresses from Server A's perspective. Therefore, referring to Figure 12c, the command received by Server B is COPY from Fibre Channel device 000500 (RAID drive B) to Fibre Channel device 000600 (Tape Library B). However, Server B will interpret the COPY command using the address table of switch 235_4 (Figure 12d) and assume it should copy data from RAID drive A to Tape Library A and not RAID drive B to Tape Library B. Thus, the wrong operation will be performed. As another example, assume that Server B sends a command to Server A to copy from RAID drive A to Tape Library C. The command will be COPY from Fibre Channel address 000500 to 009900 (the addresses are from the perspective of switch 235_4). Server A will assume the command is to copy data from RAID drive B to a nonexistent device because 009900 is not in the address table of switch 235_4.

According to one embodiment, the switch gets around this problem by intercepting each third party command and modifying the embedded Fibre Channel addresses to be compatible with the destination device. However, this requires that the source switch know the assignment of local addresses in the destination switch. While it is possible for a switch to convert the third-party commands, alternative methods are preferred.

According to one alternative method, Fibre Channel addresses are globally assigned for devices that are referenced by Fibre Channel address in third-party commands. The use of a Global Fibre Channel address allows third-party commands to be used with no modification, but sets the total number of devices possible in an SoIP network to the same maximum as a Fibre Channel network. Only those devices that are referenced in a third-party command require a global address, although all devices within an SoIP network can be assigned global addresses.

A Globally Assigned Fibre Channel address is preferably used as the device's SoIP socket number. This simplifies the conversion of "legacy" Fibre Channel data frames to SoIP compatible data frames. Therefore, globally assigning Fibre Channel addresses is equivalent to globally assigning SoIP socket numbers.

Global SoIP socket number allocation is managed by the SoIP Name Server, which allocates Global SoIP socket numbers as requested from a pool of free socket numbers, and deallocates socket numbers (returns them to the free pool) when they are no longer used. The assignment of Global SoIP socket numbers for all devices in an SoIP network is the simplest solution from a management standpoint because it does not require specifying the subset of devices that require a Global SoIP socket number (or alternatively, the devices that can use a local SoIP socket number).

Thus, all devices in an SoIP network either have a locally assigned SoIP socket number or a globally assigned SoIP socket number. All SoIP compatible devices and switches support both modes. Each device or switch determines from the SoIP Name Server which mode is to be supported when it logs into the network. An SoIP Name Server configuration parameter indicates the SoIP socket number allocation mode.

An environment that supports both local and global SoIP socket numbers is not required because it is expected that the need for global SoIP socket numbers will be eliminated due to a new form of Third-Party command format, which embeds World Wide Names in the command instead of the Fibre Channel address. Because World Wide Names are unique, the device receiving the command is able to determine the appropriate address(es) to use from its point of view. One implementation of this new third-party command is the EXTENDED COPY command. Native SoIP devices preferably use the version of third-party commands that embed World Wide Names in the command when SoIP socket numbers are locally assigned.

In one embodiment, when SoIP socket numbers are assigned globally, the requester indicates the minimum number of socket numbers requested and a 24-bit mask defining the boundary. For example, a 16-port switch may request 4096 socket numbers with a bit mask of FFF000 (hex) indicating that the socket numbers should be allocated on a boundary where the lower 12 bits are 0. The switch would then allocate 256 socket numbers to each port (for support of an arbitrated loop). Allocation of socket numbers on a specified boundary allows the switch to allocate socket numbers that directly correlate to port numbers. In the above example, bits 11:8 would identify the port. Native SoIP devices preferably allocate only one global SoIP socket number from the SoIP Name Server.

In one embodiment, the SoIP Name Server also includes a configuration parameter that selects "Maximum Fibre Channel Compatibility" mode which only has

meaning for Global assignment of SoIP socket numbers. Devices are able to query the Name server for the value of this parameter. When enabled, this mode specifies that global SoIP socket numbers are to be allocated in blocks of 65536 (on boundaries of 65536) to switches. This mode is compatible with the existing Fibre Channel modes of address allocation where the lower 8 bits identify the device, the middle 8 bits identify the port and the upper 8 bits identify the switch. SoIP switches check for this mode and, if enabled, request 65536 socket numbers when requesting global SoIP socket numbers. In this mode, Native SoIP devices preferably allocate only one global SoIP socket number from the SoIP Name Server.

According to one embodiment, when operating in a Layer 2 network (e.g., no IP routers), the frame format is modified to simplify the encapsulation logic. A Layer 2 network does not require the IP Header or the UDP header. All frames are forwarded using the physical address (e.g. Ethernet MAC address). A switch then routes frames internally based on the Layer 2 physical address (e.g. Ethernet MAC address) combined with the SoIP socket number. In essence, the Layer 2 physical address replaces the IP address as a parameter in uniquely identifying an SoIP device. Figure 13 shows the frame format for an FCP frame transmitted on Ethernet. An Ethernet Type value 290 is defined specifically for SoIP to allow a station receiving the frame to distinguish the frame from other frame types (e.g., IP). The IP and UDP headers have been removed which reduces the frame overhead. An advantage is that the length and checksum fields in the UDP header no longer need to be generated. The generation of the IP and UDP headers introduces additional latency for the frame transmission because the length and checksum are located at the beginning of the frame. Therefore, it is necessary to buffer the entire frame to determine the length and checksum and write them into the header. For an Ethernet Layer 2 SoIP frame, it is only necessary to determine the amount of padding, if any, added at the end of the frame. The number of PAD bytes must be included in the SoIP Header to allow the PAD bytes to be removed at the receiving station. Since the padding is only required to satisfy a minimum Ethernet frame size of 64 bytes, it is possible to complete the header generation after 64 bytes of the frame (or the entire frame) have been received.

The Layer 2 frame format is similar to the Layer 3 frame format SoIP Frame conversion described above with reference to Figure 6 with the following differences:

a. The IP and UDP headers are no longer present.

b. The Ethernet Type value is different.

c. The CHEKSUM PAD field is replaced by the FC CRC field. The FC CRC field is a 4-byte field containing the Fibre Channel CRC calculated over the FCP header and payload. This field may be inserted by a source when a Fibre Channel data frame is encapsulated with no changes. Thus, the CRC received with the frame is still valid.

d. The CHECKSUM PAD flag is replaced by the FC CRC PRESENT flag. This bit indicates if the FC CRC field is present in the frame. Note that the CHECKSUM PAD field has no meaning since there is no need to calculate a UDP checksum.

e. The FRAME PAD LENGTH may have a non-zero value since the encapsulated frame length may be less than the Ethernet minimum of 64.

The UDP Header contains a Destination Port field and a Source Port field. The normal usage of these fields is to identify the software applications that are communicating with each other. An application requests a port number for use when sending a UDP "datagram". This port number becomes the source port number for each UDP datagram sent by the application. When a UDP datagram is received, the destination port number is used by the UDP layer to determine the application to which the datagram will be forwarded. Figure 14a illustrates "demultiplexing" of UDP datagrams as is typical in the industry.

Figures 14b and 14c illustrate ways to add an SoIP layer according to embodiments of the present invention. Figure 14b illustrates frame demultiplexing when there is a single port number assigned to all SoIP devices. Further demultiplexing is then performed using the SoIP socket number to determine the device. Routing data frames to applications is then performed based on the FCP exchange numbers located in the FCP header. Figure 14c illustrates a similar example, but with separate UDP port numbers assigned to each SoIP device. In this case, it is not necessary to examine the SoIP socket number in order to forward the UDP datagram. (The SoIP socket number and IP address must still uniquely identify the device). The choice of whether to use a single UDP port number for each SoIP device or one UDP port number for all devices is implementation dependent.

The UDP demultiplexing examples illustrated in Figures 14b and 14c are oriented toward a server with one or more host bus adapters (where the host bus adapters are the SoIP devices). A switch is generally less complicated in the sense that data frames are forwarded to end devices and the application layer does not have to be handled.

The addressing mechanisms described above allow software applications to appear as SoIP devices by registering with the name server using a different address. This opens up the possibility for applications to advertise themselves in the name server for use by other applications. An example is a COPY manager that could be used by a higher level backup application.

According to one embodiment, each storage device, when it registers with the name server, must include the UDP port number to use when sending data frames to the device. In a normal UDP application, the destination port would save the source port number for use in sending a reply. However, this mechanism is not feasible for use with "legacy" FC switches since it requires the switch to associate the source port numbers with the exchange ID's. It is much simpler to require a storage device to always use the same UDP port number.

As a result, according to this embodiment, a storage device is identified by 3 parameters in the name server database: IP Address, UDP Port Number, and SoIP socket number. An additional parameter required is the physical address (e.g. Ethernet MAC address) which is determined in the normal manner for IP networks. ARP (address resolution protocol) is preferably used to learn the physical address to use for an IP address. The physical address to use can also be learned when a frame is received from a device. For example, the physical address can be learned when a Port Login request is received. The physical address may not be the physical address of the actual device but the address of an IP router.

The SoIP Name Server (SNS) must have a UDP Port number that is known by all of the SoIP devices within an SoIP network since the port number cannot be learned from another source. This could be a "well-known" port number or a registered port number. This approach is similar to a Domain Name Server (DNS) that has a well-known port number of 53. The assignment of "well-known" port numbers is done by the IANA (Internet Assigned Numbers Authority).

Routing within an IP network is affected by the choice of addressing mode which impacts the ability of switches and routers to determine what constitutes a "conversation". A conversation is a set of data frames that are related and which should arrive in order. However, it is assumed that conversations have no ordering relationship. In other words, the ordering of frames from different conversations can be changed with no effect. For example, assume that frames for 3 conversations (A, B and C) are transmitted in the following order (A1 sent first):

A1 A2 B1 B2 B3 A3 B4 A4 A5 A6 A7 B5 B6 B7 C1 C2 C3 A8.

It is permissible for the frames to be received in any of the following sequences (note that there are many more possible sequences that are acceptable):

A1 A2 A3 A4 A5 A6 A7 A8 B1 B2 B3 B4 B5 B6 B7 C1 C2 C3;
A1 A2 A3 A4 A5 A6 C1 C2 B1 B2 B3 B4 B5 B6 B7 C3 A7 A8; and
C1 C2 A1 C3 A2 B1 B2 A3 A4 A5 A6 B3 B4 B5 A7 A8 B6 B7.

In each of the above sequences, the frames for a particular conversation arrive in order with respect to each other, but out of order with respect to frames from other conversations. The ability to identify different conversations allows load balancing to be performed by allowing traffic to be routed on a conversation basis. Switches and routers can determine conversations based on several parameters within a data frame including Destination/Source addresses, IP Protocol, UDP/TCP Port Numbers, etc. The parameters actually used are dependent on the switch/router implementation.

Storage traffic between the same two devices should be treated as a single conversation. It is not acceptable for storage commands to be received out of order because there may be a relationship between the commands (e.g. ordered queuing). Therefore, it is preferable to select an addressing mechanism that makes a device unique to a switch/router but does not attempt to distinguish commands. Different IP addresses are an ideal choice for distinguishing devices since this method works with all switches and routers. When an IP address is shared, it is preferred that the UDP Port Numbers be unique for the devices sharing the IP address. Thus, devices that share an IP address have the possibility to be treated separately by switches and routers that classify conversations

based on UDP port numbers. It is understood that the discussion of UDP Port Numbers above also applies to TCP Header Port Numbers when SoIP is implemented using TCP instead of UDP.

Figure 15 is a high level block diagram which illustrates the basic architecture for a switch port that supports both Fibre Channel and Gigabit Ethernet according to an embodiment of the present invention. The Fibre Channel and Gigabit Ethernet ports use the same encoding/decoding method (8B/10B) with each port requiring a serializer/deserializer (SERDES) block for converting to/from the high speed serial interface. Therefore, these two interfaces share the 8B/10B block 310 and SERDES block 315 in this embodiment as shown in Figure 15. These two interface types differ in clock speed with Fibre Channel operating at 1062.5 MHz and Gigabit Ethernet operating at 1250 MHz. Higher speed versions of these interfaces are being developed which will also have a different clock speed. Therefore, a multiplexer 345 selects the clock used by the logic based on the port type. In addition, these two interfaces share the switch fabric interface logic block 320 which interfaces with the switch fabric (including the management interface). The MAC blocks (blocks 325 and 330) implement the appropriate protocol state machines for the interface (Fibre Channel or Gigabit Ethernet). The MAC blocks 325 and 330 convert received data into frames which are forwarded to the routing logic blocks 335 and 340, respectively. The MAC blocks 325 and 330 also receive data frames from the routing logic blocks 335 and 340, respectively, which are then transmitted according to the interface's (Fibre Channel or Gigabit Ethernet) protocol. Routing logic blocks 335 and 340 determine where each received frame should be routed based on addressing information within the frame. Routing logic blocks 335 and 340 also perform any modifications to the frames that are required. For example, a routing logic block will remove the SoIP encapsulation from a frame being forwarded to a Fibre Channel port. The routing logic block then sends the frame to the switch fabric with an indication of the destination output ports. Egress data frames (frames from the switch fabric to the output port) are received by a routing logic block and forwarded to the associated MAC. Additional processing may be performed on the frame by the routing logic block before the MAC receives the frame. For example, Ethernet port routing logic block 340 may convert a Fibre Channel frame into an SoIP frame.

According to another embodiment of the present invention as shown in Figure 16, the two routing blocks of Figure 15 are combined into a single routing logic block 350.

This optimization is possible because the routing logic used by these two interfaces is very similar. In one embodiment, routing logic block 350 includes logic blocks which are dependent on the port type and other blocks that are common to both port types. This optimization reduces the number of logic gates required on an ASIC. Routing block 350 determines where a frame is routed based on addressing information within the data frame. This function is known as address resolution and is performed for both Fibre Channel and Gigabit Ethernet data frames. Therefore, address resolution logic can be shared by these two port interfaces though it is necessary for the routing logic to select different data based on the port type. The logic within Routing Logic block 350 can be implemented as hard coded logic or as a programmable method using a network processor, which is designed specifically for processing packets and which can be programmed to route either Fibre Channel frames or Ethernet frames. Therefore, the routing logic hardware can be shared by using different network processor software. In one embodiment, routing logic block 350 also includes an input and output FIFO memory which is shared by the two port interfaces. Additional logic which can be shared include statistics registers and control registers. Statistics registers are used to count the number of frames received, frames transmitted, bytes received, bytes transmitted, etc. A common set of statistics registers can be used. These registers are modified by control signals from each MAC. Control registers determine the operating mode of each MAC. A common set of statistics and control registers reduces the logic required to implement the registers and for interfacing with an external control source such as a switch management CPU.

In another embodiment as shown in Figure 17, the low-level port interface logic (e.g., FC MAC block 325 and Ethernet MAC block 330) is combined into a single MAC block 360. One problem with this approach, however, is that these two logic blocks have little in common. In addition, it is possible to purchase proprietary blocks which implement Gigabit Ethernet MAC and Fibre Channel Port Interface logic. Combining these two blocks would severely hinder the use of these proprietary blocks.

According to another embodiment of the present invention as shown in Figure 18, a Field Programmable Gate Array (FPGA) 370 is used to select the interface protocol supported by the port. The FPGA configuration loaded would be based on the port type. In this embodiment, separate FPGA code is developed for the Fibre Channel and Gigabit Ethernet interfaces. Thus, the FPGA logic can be optimized for the particular interface.

A single hardware design supports both interfaces, with software determining the FPGA code to be downloaded based on the port type.

A common port must also deal with the physical interface external to an ASIC. As is well known, such an interface may include, for example, a copper, multi-mode fiber or single-mode fiber interface. Also, the components are not necessarily the same between Fibre Channel and Ethernet. According to an embodiment of the present invention as shown in Figure 19, a Gigabit Interface Converter (GBIC) 380 is provided to allow a user to select the desired physical interface. A GBIC is a standardized module which has a common form factor and electrical interface and allows any of the many physical interfaces to be installed. GBIC modules are available from many vendors (e.g. HP, AMP, Molex, etc.) and support all of the standard Fibre Channel and Gigabit Ethernet physical interfaces. Figure 14 shows a block diagram of a common FC/Gigabit Ethernet port interface (e.g., as shown in Figures 15, 16, 17 and 18) combined with a GBIC interface according to this embodiment. The ASIC connects to a GBIC connector 385 which allows the user to change GBIC modules. Thus, the user can select the media type by installing the appropriate GBIC 380.

GBIC modules typically contain a serial EEROM whose contents can be read to determine the type of module (e.g. Fibre Channel, Gigabit Ethernet, Infiniband, Copper, Multi-mode, Single-mode, etc.). The GBIC can thus indicate the type of interface, e.g., FC or GE or Infiniband, to use. However, it is possible for the GBIC to support multiple interfaces, for example both FC and GE. Therefore, in one embodiment, the port interface type is user switchable/configurable, and in another embodiment the type of the link interface is automatically determined through added intelligence, for example, through a "handshake".

According to another embodiment of the present invention, an SoIP intelligent network interface card (NIC) 400 is provided as shown in Figure 20. NIC card 400 is able to send and receive both IP and SoIP traffic. In either case, NIC card 400 has the intelligence to determine the type of traffic and direct it accordingly.

The host 410 may issue both storage commands and network commands to NIC card 400 through the PCI interface 420. These commands are sent with a specified address which is used to direct the commands to either the Direct Path or the Storage Traffic Engine. Storage commands are issued via the SCSI Command Set, and Network commands are issued via Winsock and/or TCP/IP.

NIC card 400 directs storage commands to the Storage Traffic Engine 430 based on the specified address. Storage Traffic Engine 430 handles the exchange management and sequence management for the duration of the SCSI operation. SCSI operations are then carried out via SoIP and transmitted to the network 470 via a media access controller (MAC) block 450, which in one embodiment is a Gigabit Ethernet MAC. NIC card 400 directs non-SoIP traffic to the Direct Path 440 based on the specified address. The Direct Path 440 processes the commands and transmits the specified packets to network 470 via block 450. When receiving data from network 470 via MAC 450, NIC 400 demultiplexes the traffic and directs it accordingly. Storage traffic received as SoIP is sent to storage traffic block 430. Non-SoIP traffic is sent directly to the host via direct path 440.

The multiplexer block 460 handles arbitration for the output path when both Direct Path 440 and Storage Traffic Engine 430 simultaneously send traffic to MAC 450. For traffic received from network 470 by MAC 450, Mux block 460 demultiplexes the traffic and sends it accordingly to either Direct Path 440 or Storage Traffic Engine 430.

While the invention has been described by way of example and in terms of the specific embodiments, it is to be understood that the invention is not limited to the disclosed embodiments. To the contrary, it is intended to cover various modifications and similar arrangements as would be apparent to those skilled in the art. Therefore, the scope of the appended claims should be accorded the broadest interpretation so as to encompass all such modifications and similar arrangements.

WHAT IS CLAIMED IS:

1. In a network, a method of routing data packets in a switch device, the method comprising the steps of:

receiving a packet from a first network device at a first port interface of the switch device, wherein the packet is one of a SCSI formatted packet, a Fibre Channel (FC) formatted packet and an Internet protocol (IP) formatted packet, wherein the first port interface is communicably coupled to the first network device;

converting the received packet into a packet having an internal format;

routing the internal format packet to a second port interface of the switch device;

reconverting the internal format packet to one of a SCSI formatted packet, an FC formatted packet and an IP formatted packet; and

transmitting the reconverted packet to a second network device communicably coupled to the second port interface.

2. The method of claim 1, wherein the IP formatted packet is transported over one of an Ethernet protocol, and ATM protocol and a FDDI protocol.

3. The method of claim 1, wherein the second port interface couples the switch device to a network, wherein the step of transmitting includes sending the reconverted packet to the second network device over the network, wherein the reconverted packet is in the IP format.

4. The method of claim 3, wherein the network is an Ethernet network, wherein the IP format is an Ethernet format, and wherein the step of reconverting includes the step of encapsulating the internal format packet in an Ethernet frame.

5. The method of claim 1, wherein the first port interface couples the switch device to a network, wherein the step of receiving includes receiving the packet from the first network device over the network, wherein the received packet is in the IP format.

6. The method of claim 5, wherein the network is an Ethernet network, and wherein the IP format is an Ethernet format.

7. The method of claim 1, wherein the reconverted packet is in a different format than the received packet.

8. The method of claim 1, wherein the first network device is one of a server and a storage device, and wherein the second network device is one of a server and a storage device.

9. The method of claim 1, wherein the internal format is an FCP-based format.

10. A network switch device, comprising:

a) a first port interface including:

means for receiving data packets from a network device, wherein the receiving means receives one of a SCSI formatted packet and a Fibre Channel (FC) formatted packet from a first network device; and

means for converting received packets into packets having an internal format, wherein the received data packet is converted into a first packet having the internal format;

b) a second port interface including:

means for reconverting packets from the internal format to an IP format, wherein the first packet is converted into a packet having an IP format; and

means for transmitting IP packets to a network, wherein the IP formatted packet is transmitted to an IP network; and

c) means for routing the first packet to the second port interface.

11. The switch device of claim 10, wherein the IP network is an Ethernet network and wherein the IP formatted packet is encapsulated in an Ethernet frame.

12. The switch device of claim 10, wherein the internal format is an FCP-based format.

13. A network switch device, comprising:

a) a first port interface including:

means for receiving data packets from an IP network, wherein the first interface means receives a packet in an IP format; and

means for converting received packets into packets having an internal format, wherein the received packet is converted into a first packet having an internal format;

b) a second port interface including:

means for reconverting packets having the internal format to packets having the SCSI format; and

means for transmitting reconverted packets to a SCSI network device;

c) a third port interface including:

means for reconverting packets having the internal format to packets having the FC format; and

means for transmitting reconverted packets to a FC network device; and

d) means for routing packets between the first, second, and third port interfaces, wherein the first packet is routed to one of the second and third port interfaces;

wherein if the first packet is routed to the second port interface, the first packet is converted to the SCSI format and transmitted to the SCSI network device, and wherein if the first packet is routed to the third port interface, the first packet is converted to the FC format and transmitted to the FC network device.

14. The switch device of claim 13, wherein the IP network is an Ethernet network and wherein the IP format is an Ethernet format.

15. The switch device of claim 13, wherein the internal format is one of an FCP-based format andan IP format.

16. A storage area network (SAN) comprising:

a SCSI device capable of receiving and transmitting SCSI formatted data packets;

a fibre channel (FC) device capable of receiving and transmitting FC formatted data packets;

an IP device capable of receiving and transmitting IP formatted data packets; and

a switch device including:

a first port interface communicably coupled to the SCSI device, wherein the first port interface converts SCSI formatted data packets received from the SCSI device into data packets having an internal format, and wherein the first port interface converts data packets having the internal format into SCSI formatted data packets;

a second port interface communicably coupled to the FC device, wherein the second port interface converts FC formatted data packets received from the FC device into data packets having the internal format, and wherein the second port interface converts data packets having the internal format into FC formatted data packets;

a third port interface communicably coupled to the IP device, wherein the third port interface converts IP formatted data packets received from the IP device into data packets having the internal format, and wherein the third port interface converts data packets having the internal format into IP formatted data packets; and

a switch fabric for routing data packets having the internal format between the first, second and third port interfaces;

wherein when a first one of the SCSI, FC and IP devices sends a first data packet to a second one of the SCSI, FC and IP devices, the port interface coupled to the first device converts the first data packet to a packet having the internal format and routes the internal format packet through the switch fabric to the port interface coupled to the second device, wherein the port interface coupled to the second device reconverts the internal format packet into the format associated with the second device and sends the reconverted packet to the second device.

17. The SAN of claim 16, wherein the IP formatted data packets include one of Ethernet formatted data packets, ATM formatted data packets, FDDI formatted data packets, and Infiniband formatted data packets.

18. The SAN of claim 16, wherein the internal format is an FCP-based format.

19. The SAN of claim 16, further including an IP network coupling the IP device to the third port interface.

20. A network switch device for use in a storage area network (SAN), the switch device comprising:

a first port interface communicably coupled to a SCSI device, wherein the first port interface converts SCSI formatted data packets received from the SCSI device into data packets having an internal format, and wherein the first port interface converts data packets having the internal format into SCSI formatted data packets;

a second port interface communicably coupled to a FC device, wherein the second port interface converts FC formatted data packets received from the FC device into data packets having the internal format, and wherein the second port interface converts data packets having the internal format into FC formatted data packets;

a third port interface communicably coupled to a IP device, wherein the third port interface converts IP formatted data packets received from the IP device into data packets having the internal format, and wherein the third port interface converts data packets having the internal format into IP formatted data packets; and

a switch fabric for routing data packets having the internal format between the first, second and third port interfaces;

wherein when a first one of the SCSI, FC and IP devices sends a first data packet to a second one of the SCSI, FC and IP devices, the port interface coupled to the first device converts the first data packet to a packet having the internal format and routes the internal format packet through the switch fabric to the port interface coupled to the second device, wherein the port interface coupled to the second device reconverts the internal format packet into the format associated with the second device and sends the reconverted packet to the second device.

21. The switch device of claim 20, wherein the internal format is an FCP-based format.

22. The switch device of claim 16, wherein an IP network couples the IP device to the third port interface.

23. A network switch device for use in a storage area network (SAN), the switch device comprising:

a first port interface communicably coupled to one of a SCSI device a FC device and an IP device;

a second port interface, wherein the second port interface is configurable to communicate with either a FC device or an Ethernet device; and

a switch fabric for routing data packets having the internal format between the first and second port interfaces;

wherein when the second port interface is configured to communicate with a FC device, the second port interface converts FC formatted data packets received from the FC device into data packets having an internal format, and wherein the second port interface converts data packets having the internal format received from the switch fabric into FC formatted data packets, and wherein when the second port interface is configured to communicate with an Ethernet device, the second port interface converts Ethernet formatted data packets received from the Ethernet device into data packets having the internal format, and wherein the second port interface converts data packets having the internal format received from the switch fabric into Ethernet formatted data packets.

24. The switch device of claim 23, wherein the second port interface is self-configurable based on whether a FC device or an Ethernet device is coupled to the second port interface.

25. The switch device of claim 24, wherein the second port interface includes a means for determining whether an attached device is a FC device or an Ethernet device.

26. The switch device of claim 23, wherein the second port interface is configured by a user to communicate with one of a FC device and an Ethernet device.

27. The switch device of claim 23, wherein the internal format is a FCP format.

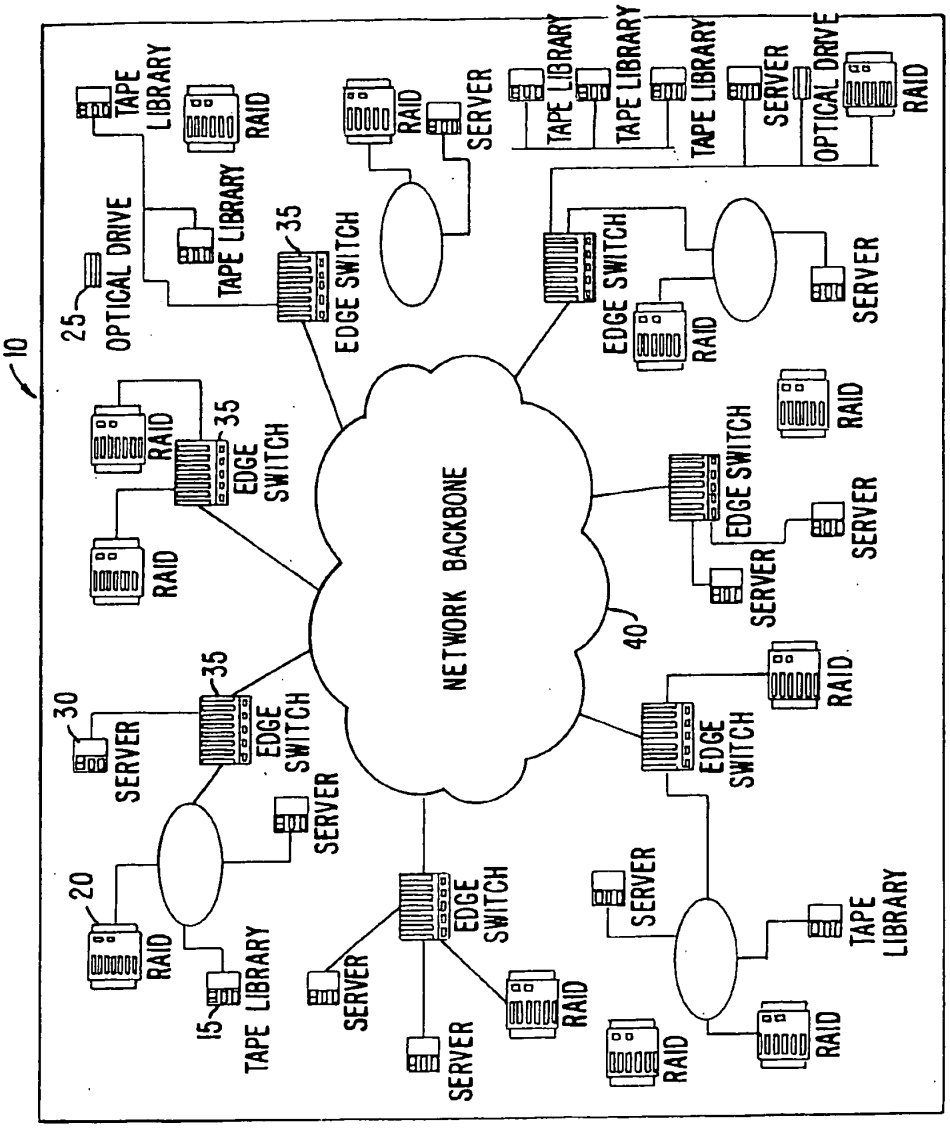


FIG. 1.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 407

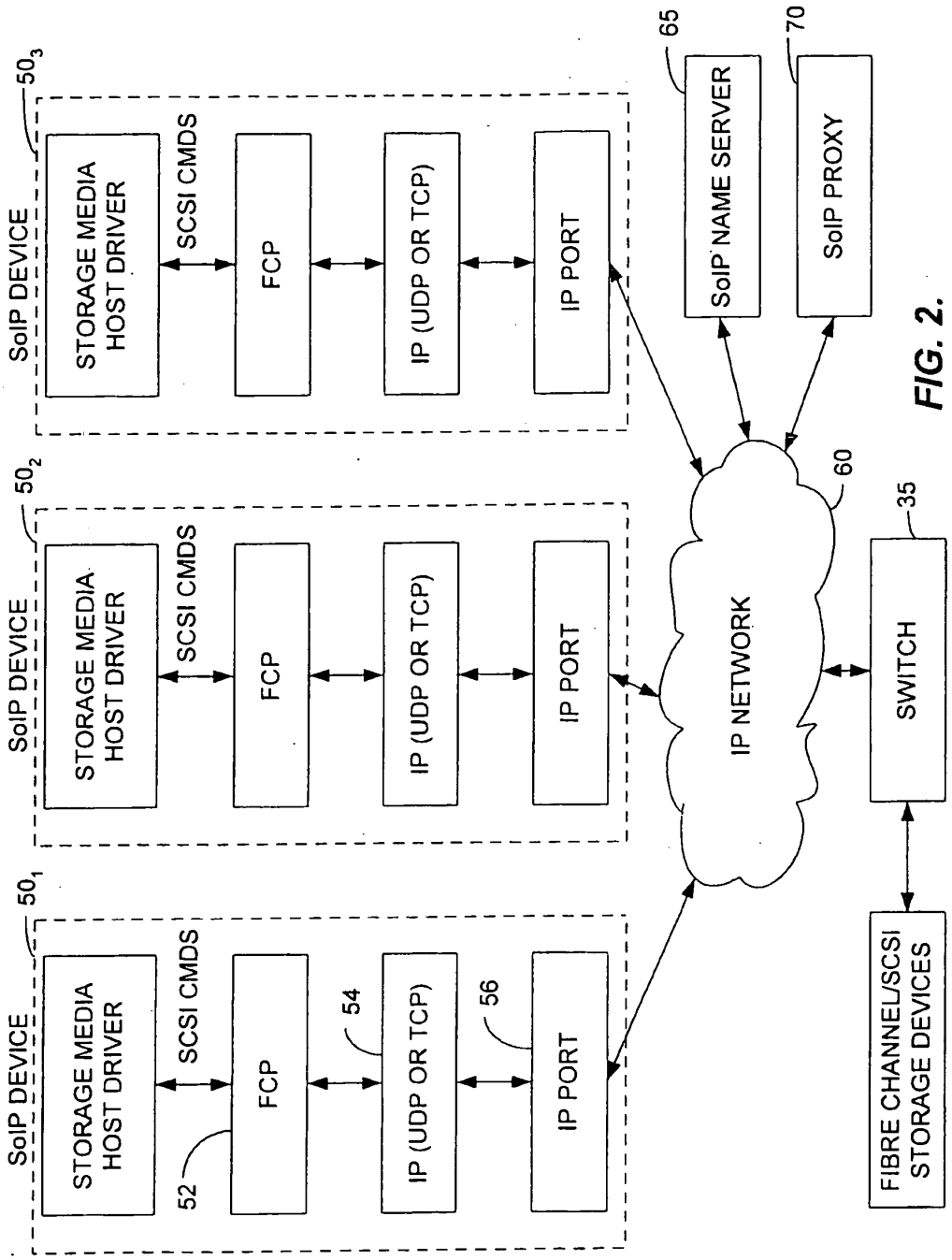


FIG. 2.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 408

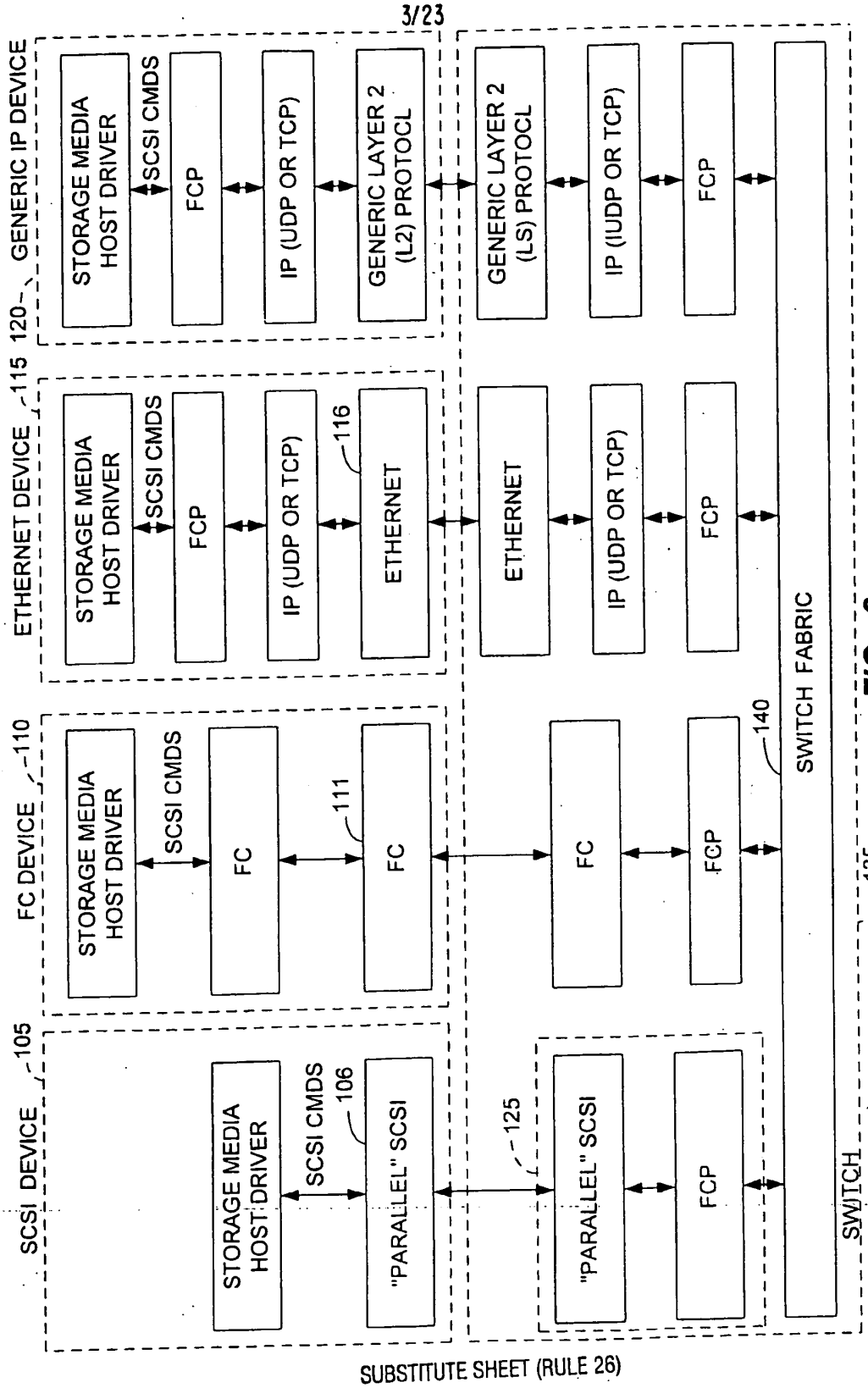


FIG. 3.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 409

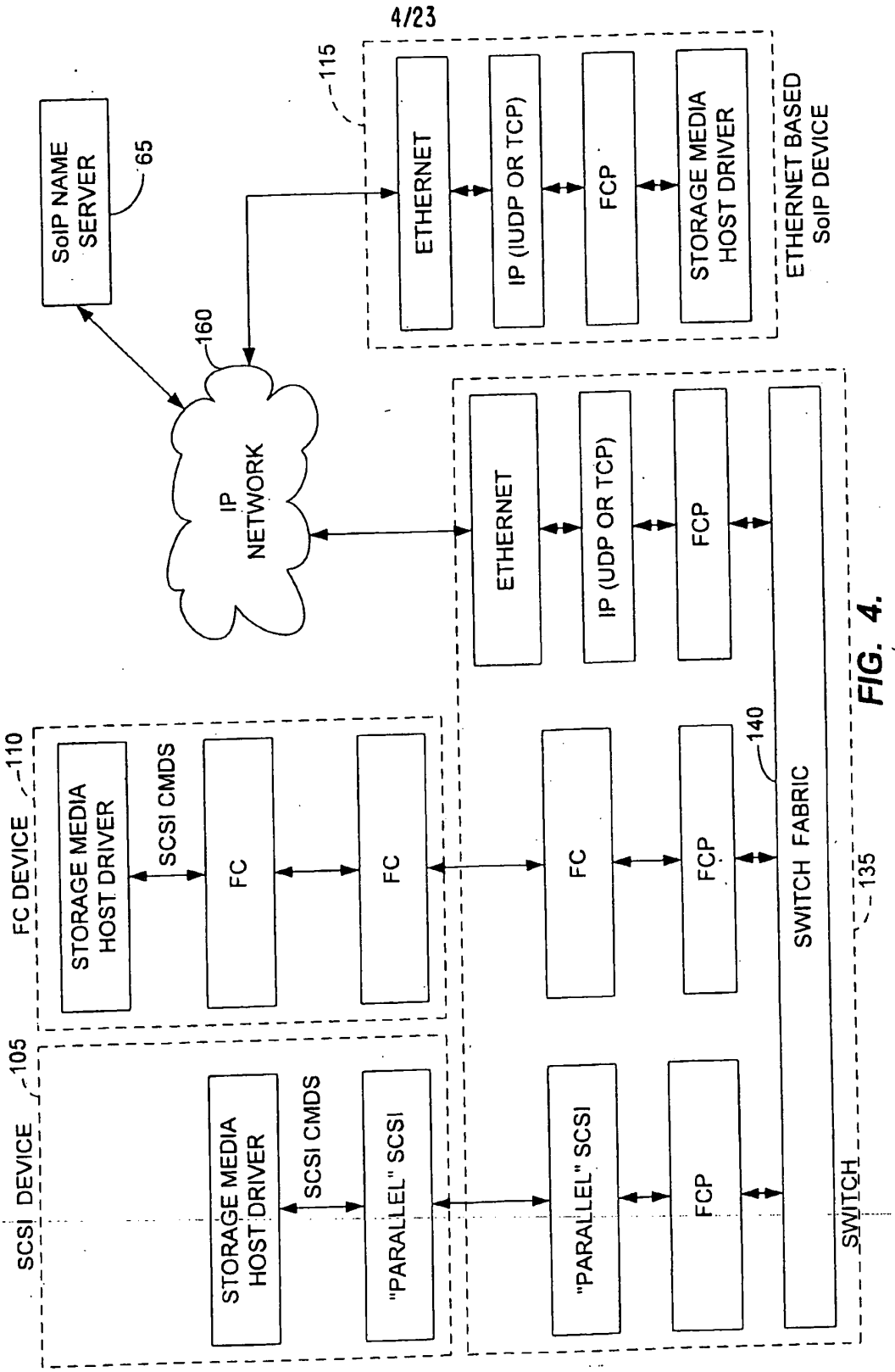


FIG. 4.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 410

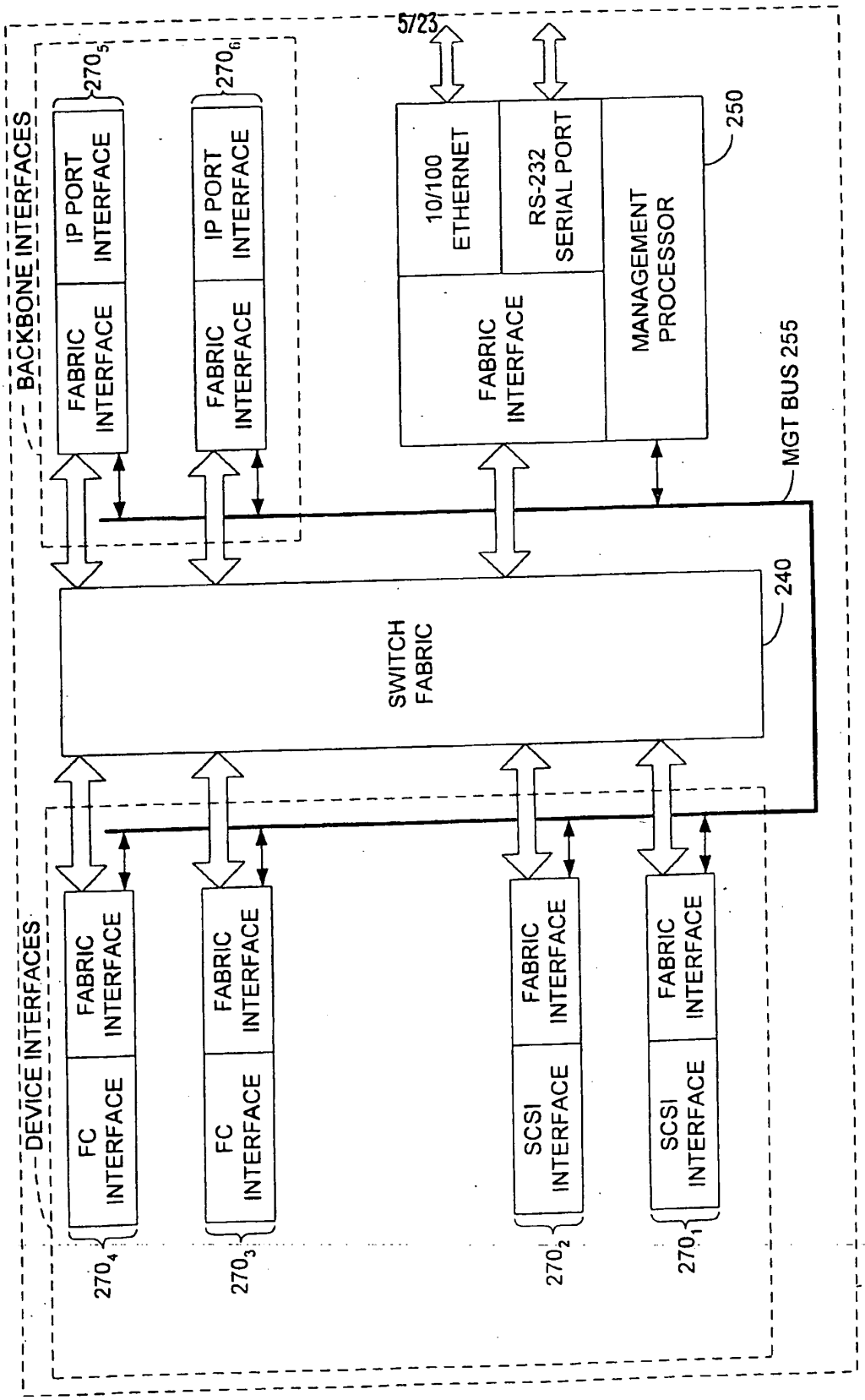


FIG. 5.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 411

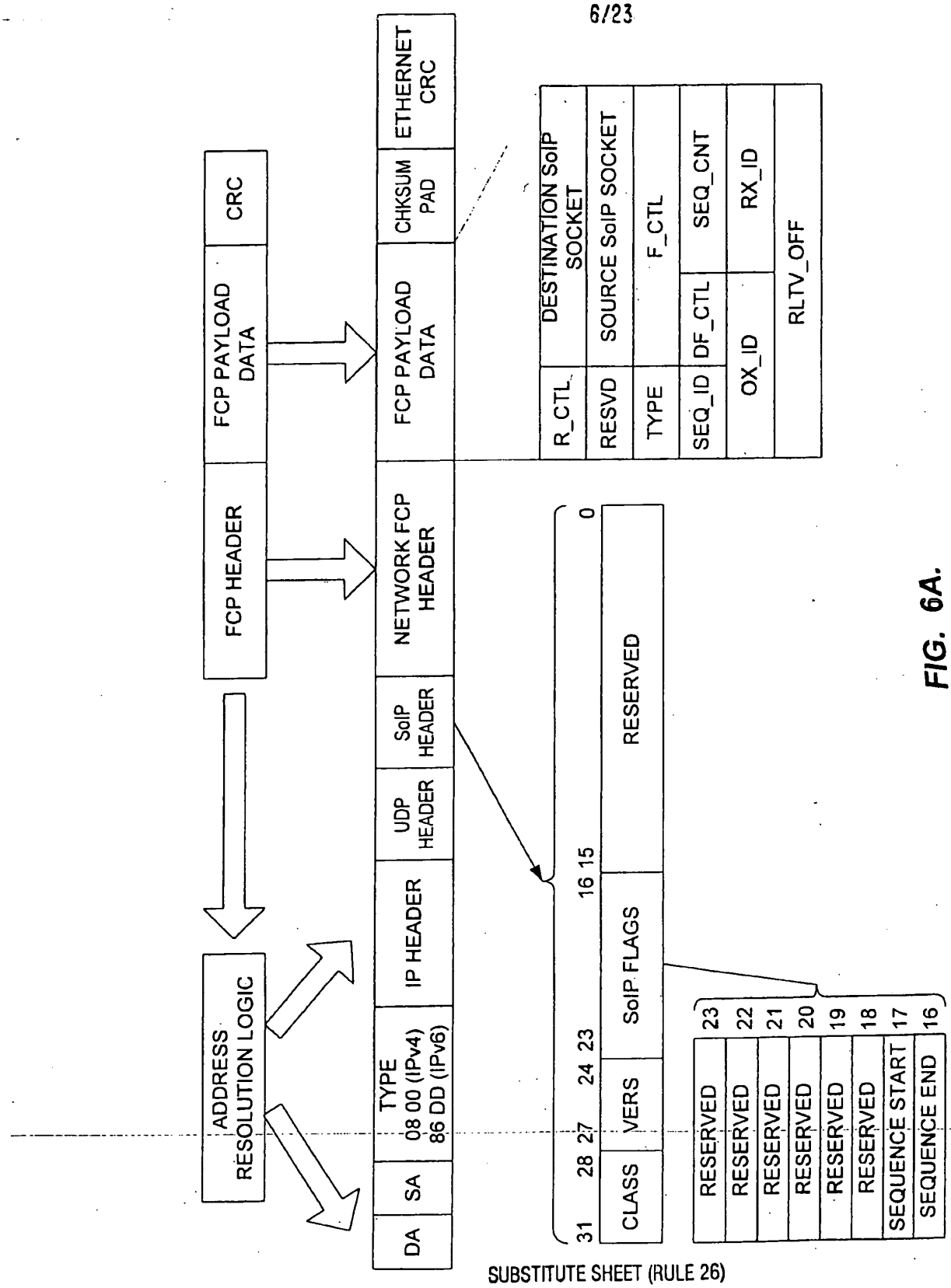


FIG. 6A.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 412

SoIP FLAG	DESCRIPTION
SEQUENCE END	THIS BIT INDICATES IF THIS FRAME TERMINATES A SEQUENCE. THIS CORRESPONDS TO A FIBRE CHANNEL FRAME WHICH ENDS WITH AN EOFt DELIMITER. FOR CLASS 3 SEQUENCES, THE SEQUENCES INITIATOR SETS THIS BIT ON THE LAST FRAME OF THE SEQUENCE. HOWEVER, FOR CLASS 2, THIS BIT IS SET BY THE SEQUENCE RECIPIENT ON THE ACK FRAME THAT TERMINATES THE SEQUENCE 1 = LAST FRAME OF SEQUENCE 0 = NOT LAST FRAME OF A SEQUENCE
SEQUENCE START	THIS BIT INDICATES IF THIS FRAME IS THE FIRST FRAME OF A SEQUENCE. THIS CORRESPONDS TO A FIBRE CHANNEL FRAME BEGINNING EITHER WITH AN SOFi2 OR SOFi3 DELIMITER. 1 = FIRST FRAME OF SEQUENCE 0 = NOT FIRST FRAME OF A SEQUENCE

FIG. 6B.

SOURCE PORT	DESTINATION PORT
LENGTH	CHECKSUM

FIG. 6C.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 413

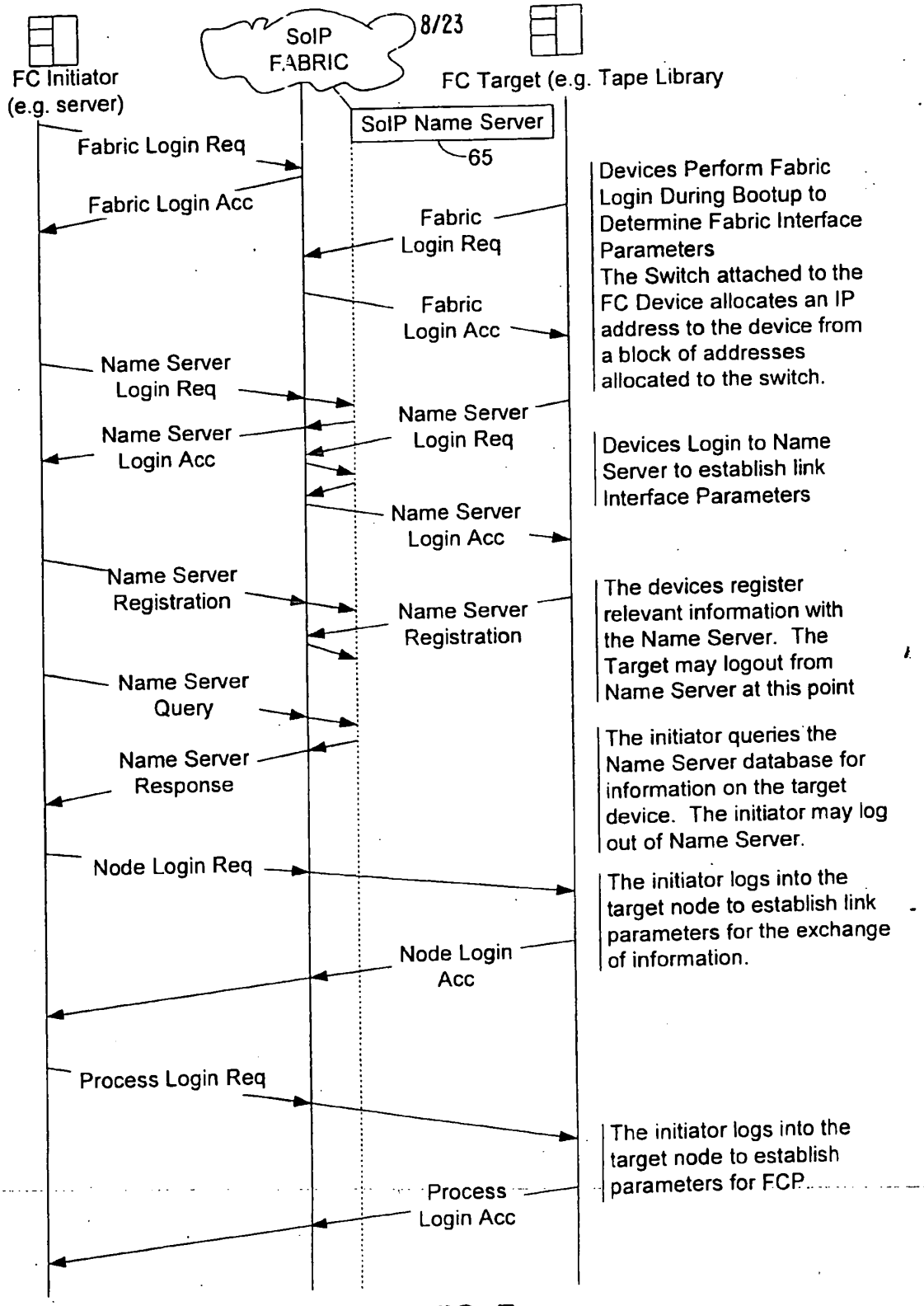


FIG. 7.

SUBSTITUTE SHEET (RULE 26)

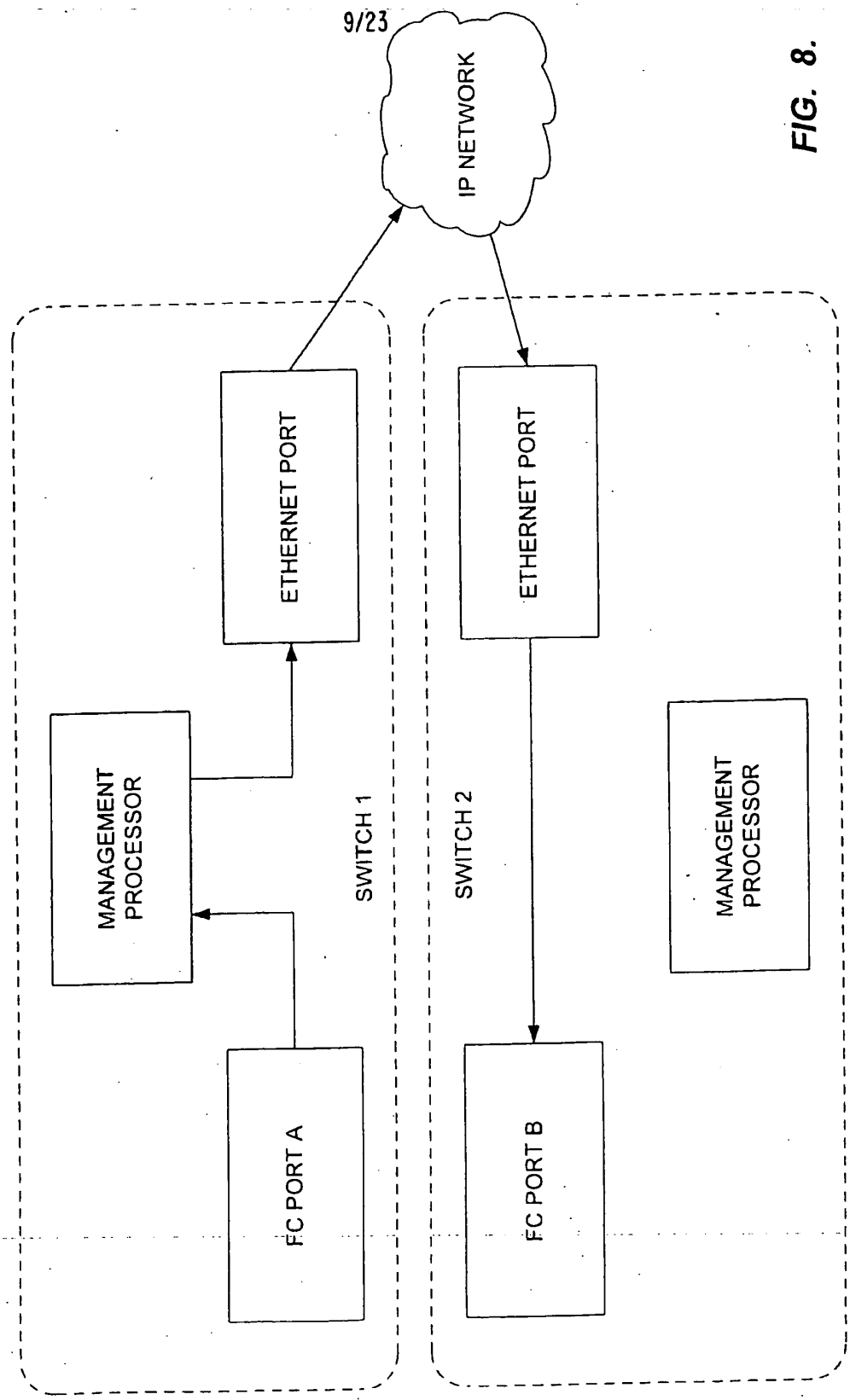


FIG. 8.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 415

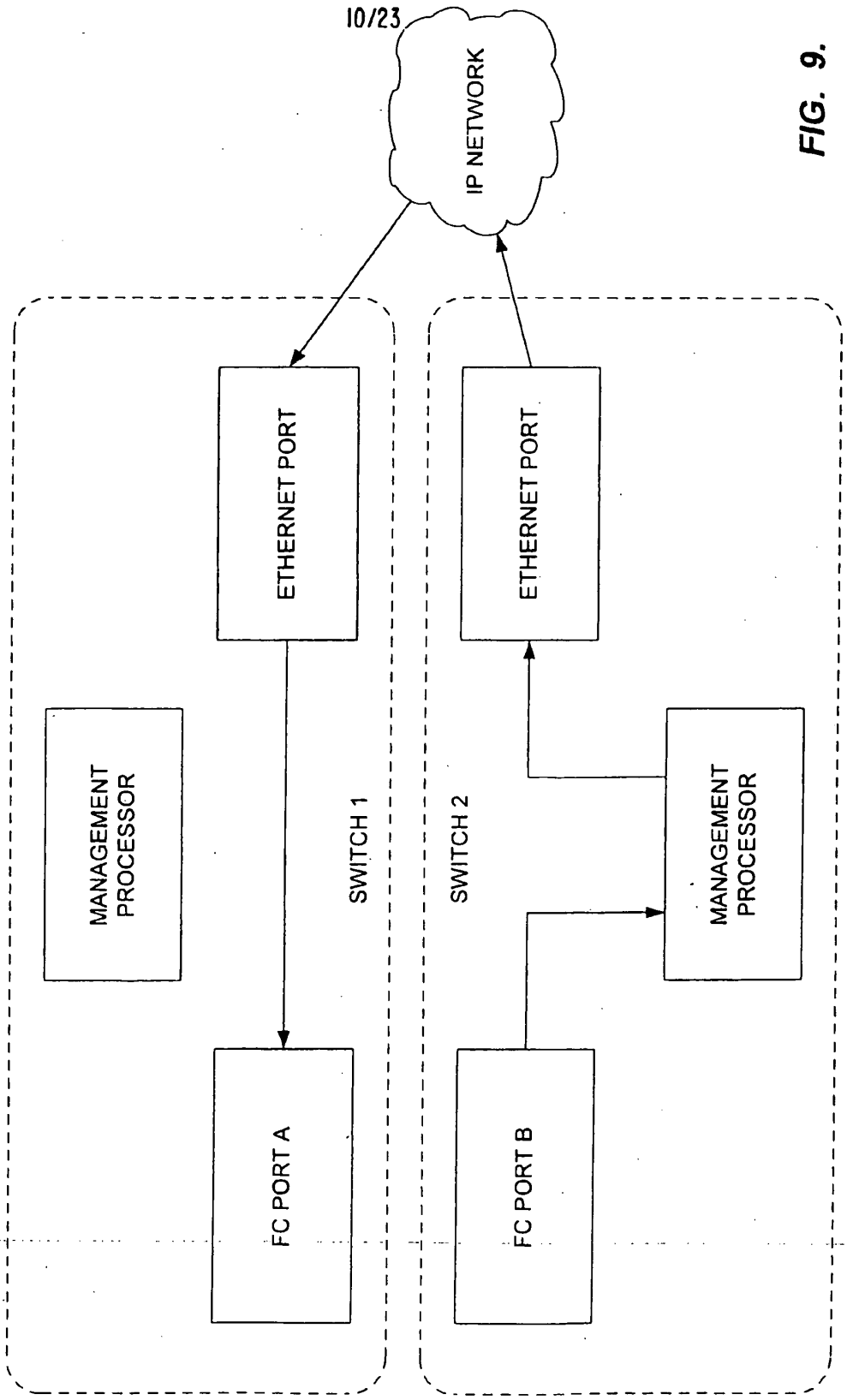


FIG. 9.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 416

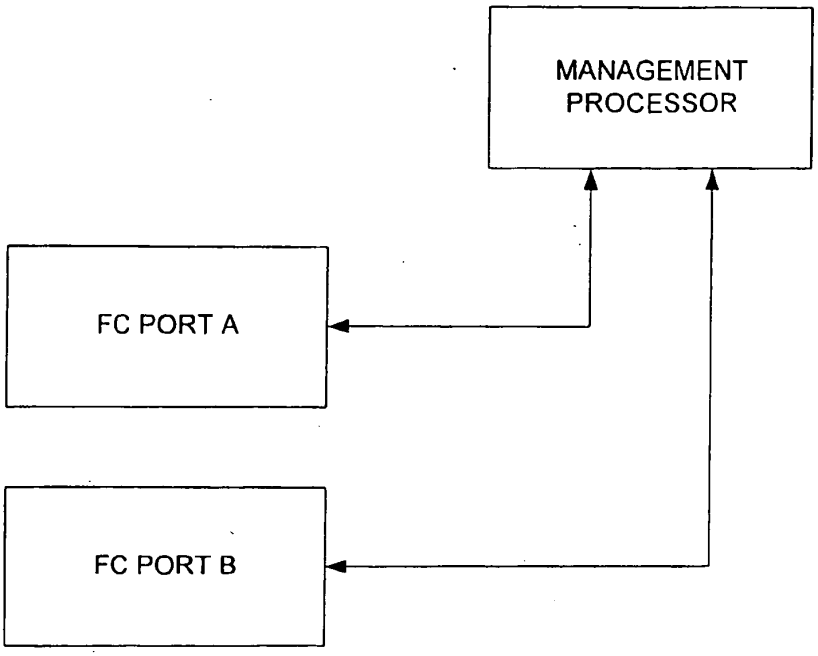


FIG. 10.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 417

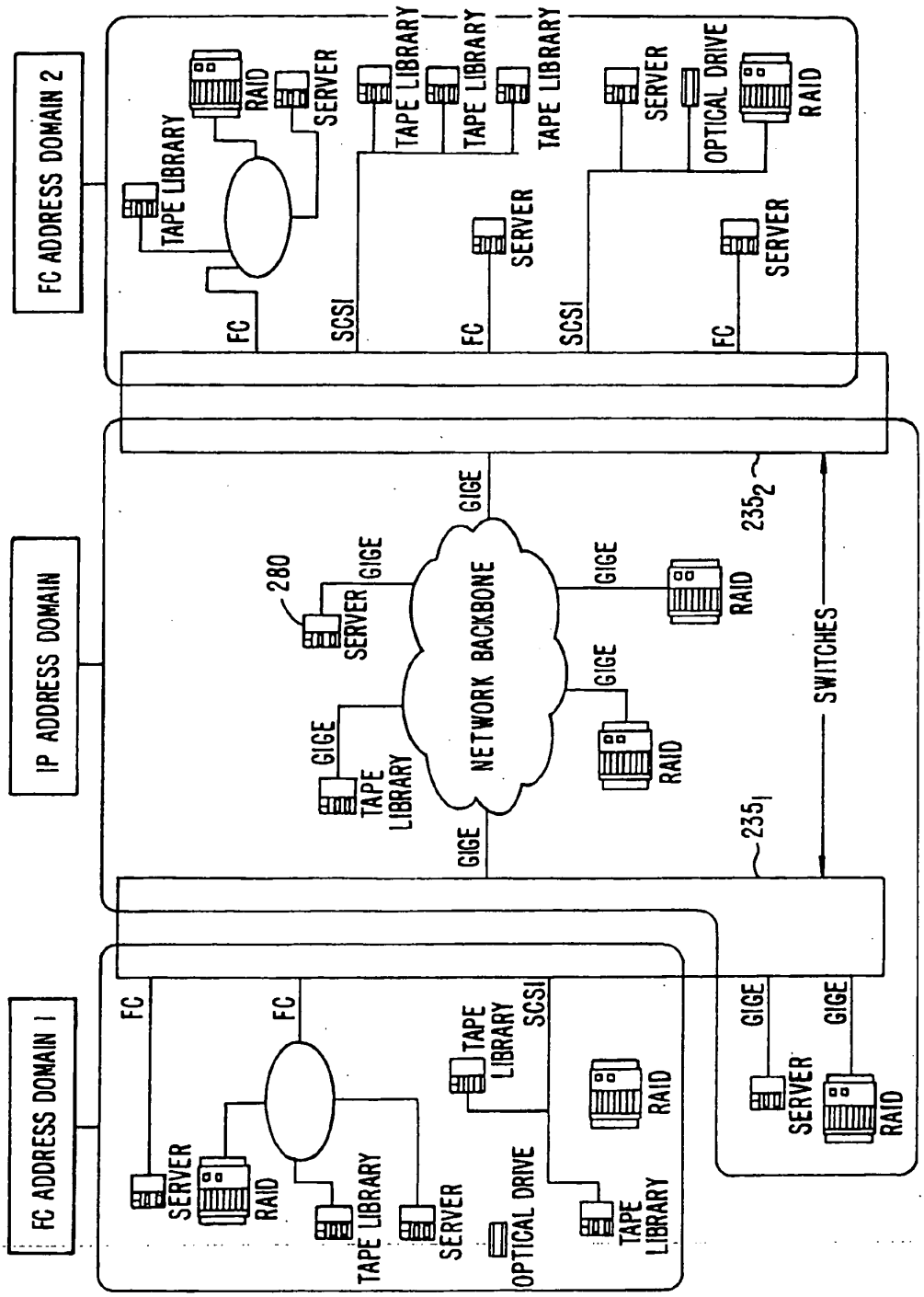


FIG. 11.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 418

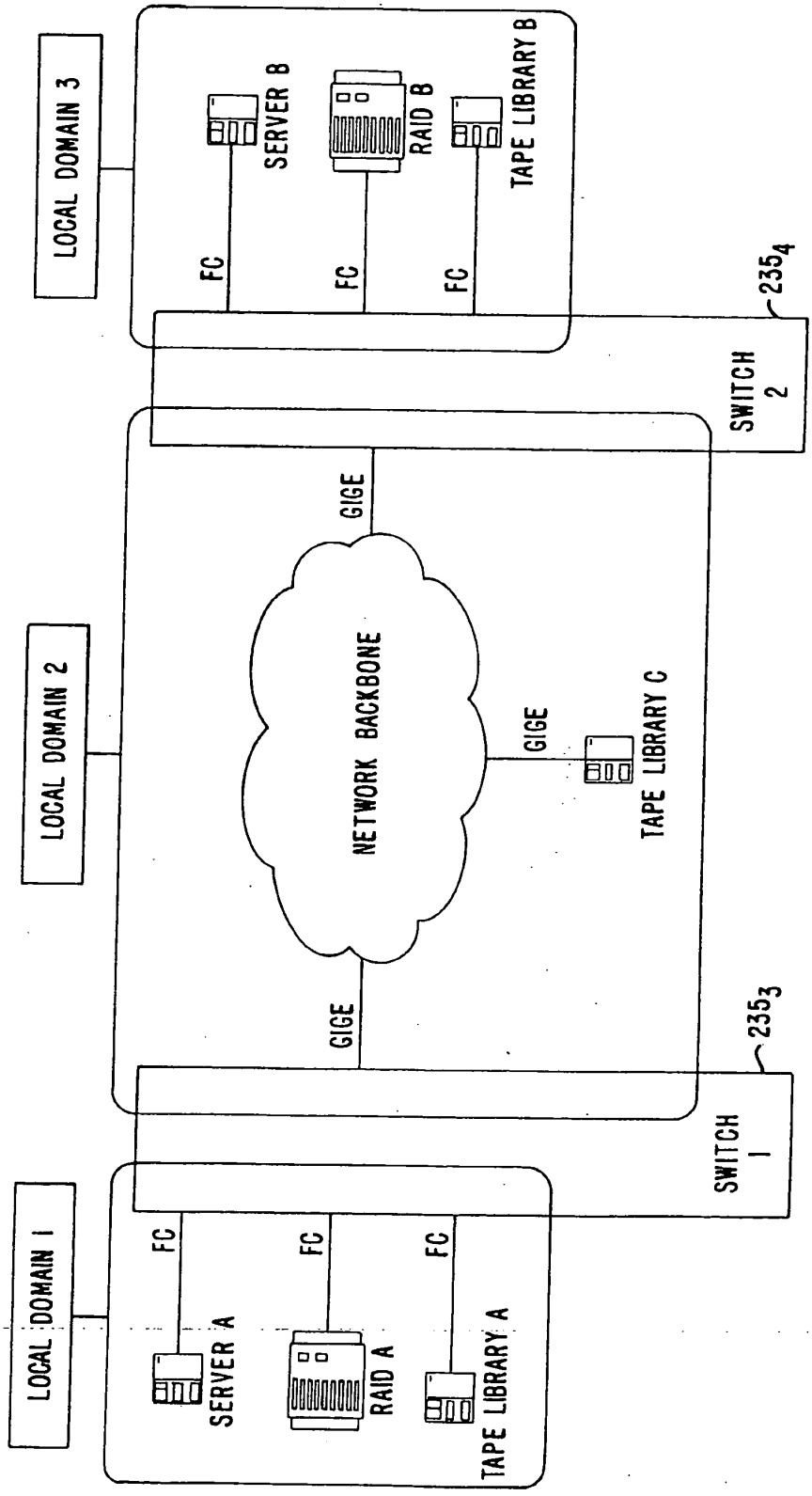


FIG. 12A.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 419

DEVICE NAME	IP ADDRESS	SoIP SOCKET NUMBER
SERVER A	100.100.100.1	000100
RAID A	100.100.100.2	000200
TAPE LIBRARY A	100.100.100.3	000300
SERVER B	100.100.101.1	000100
RAID B	100.100.101.2	000200
TAPE LIBRARY B	100.100.101.3	000300
TAPE LIBRARY C	100.100.102.1	000000

FIG. 12B.

DEVICE NAME	IP ADDRESS	SoIP SOCKET NUMBER	LOCAL FIBRE CHANNEL ADDRESS
SERVER A	100.100.100.1	000100	000100
RAID A	100.100.100.2	000200	000200
TAPE LIBRARY A	100.100.100.3	000300	000300
SERVER B	100.100.101.1	000100	000400
RAID B	100.100.101.2	000200	000500
TAPE LIBRARY B	100.100.101.3	000300	000600
TAPE LIBRARY C	100.100.102.1	000000	000700

FIG. 12C.

DEVICE NAME	IP ADDRESS	SoIP SOCKET NUMBER	LOCAL FIBRE CHANNEL ADDRESS
SERVER A	100.100.100.1	000100	000400
RAID A	100.100.100.2	000200	000500
TAPE LIBRARY A	100.100.100.3	000300	000600
SERVER B	100.100.101.1	000100	000100
RAID B	100.100.101.2	000200	000200
TAPE LIBRARY B	100.100.101.3	000300	000300
TAPE LIBRARY C	100.100.102.1	000000	009900

FIG. 12D.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 420

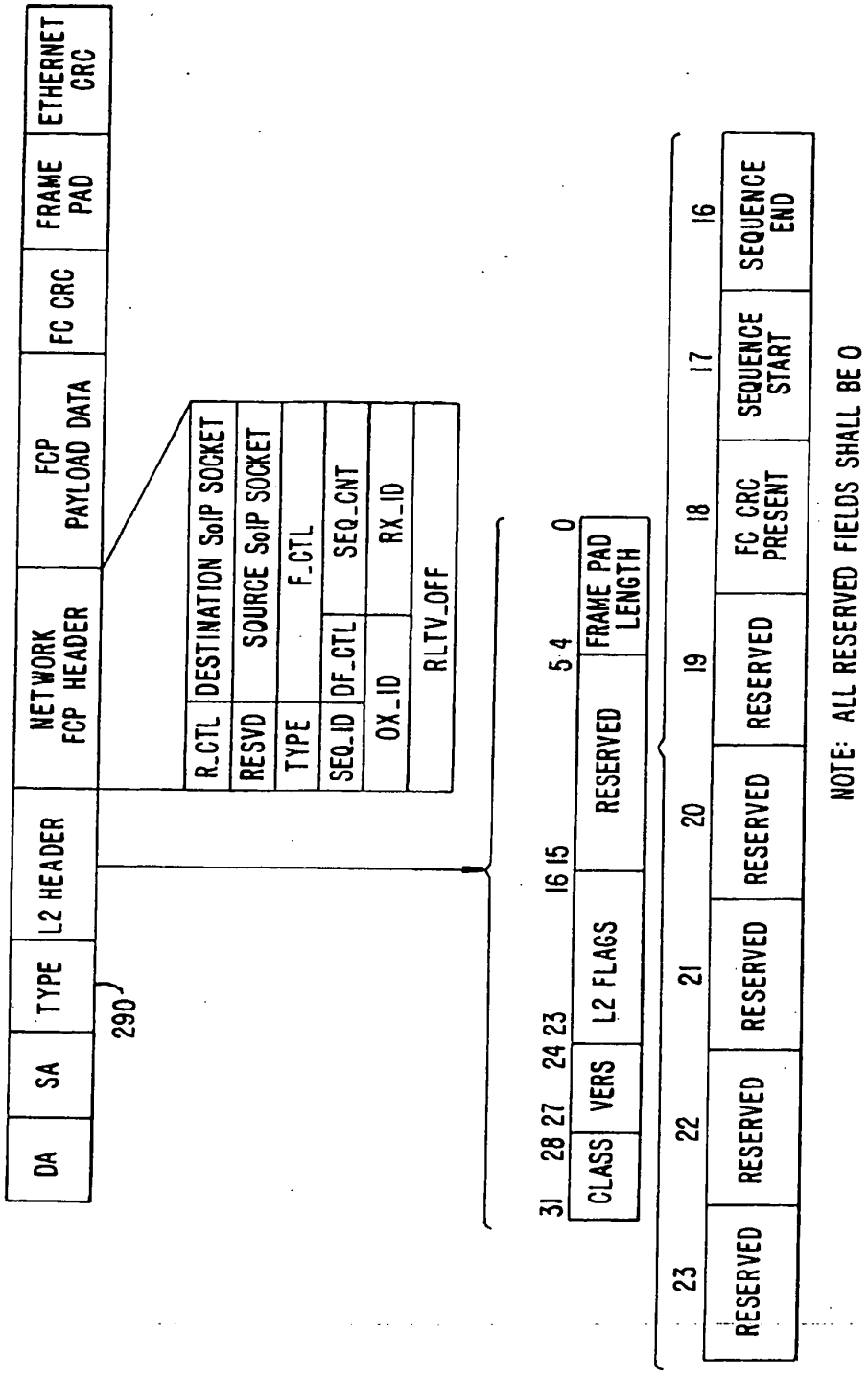


FIG. 13.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 421

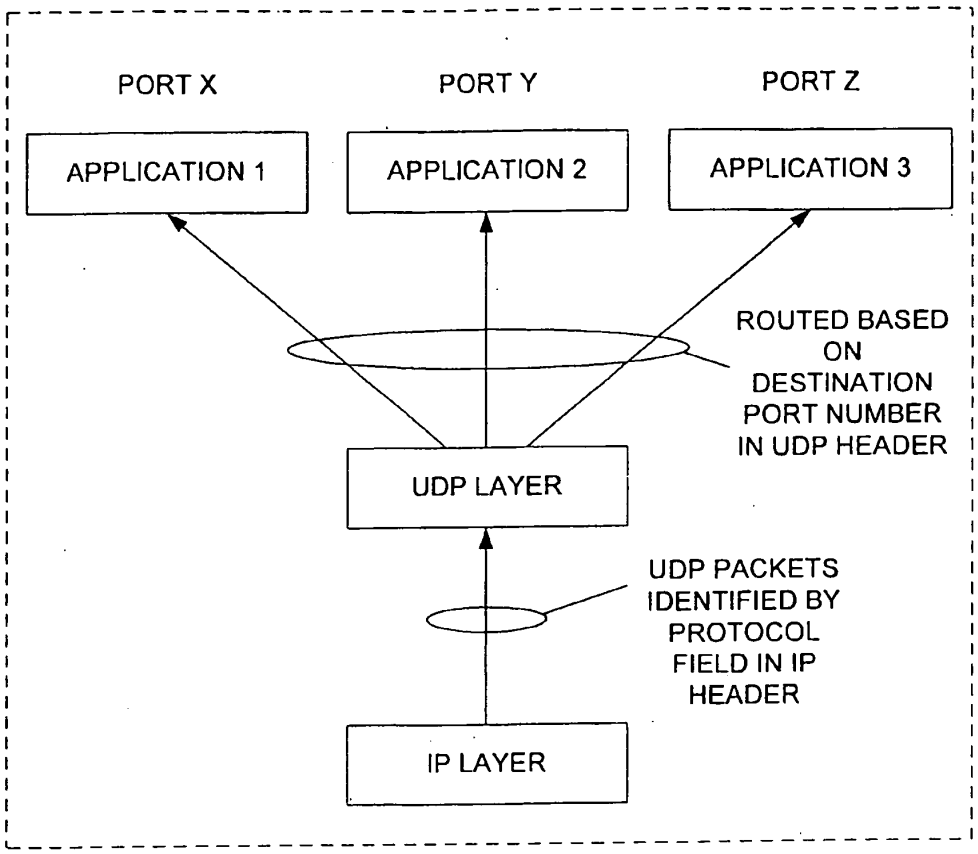


FIG. 14A.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 422

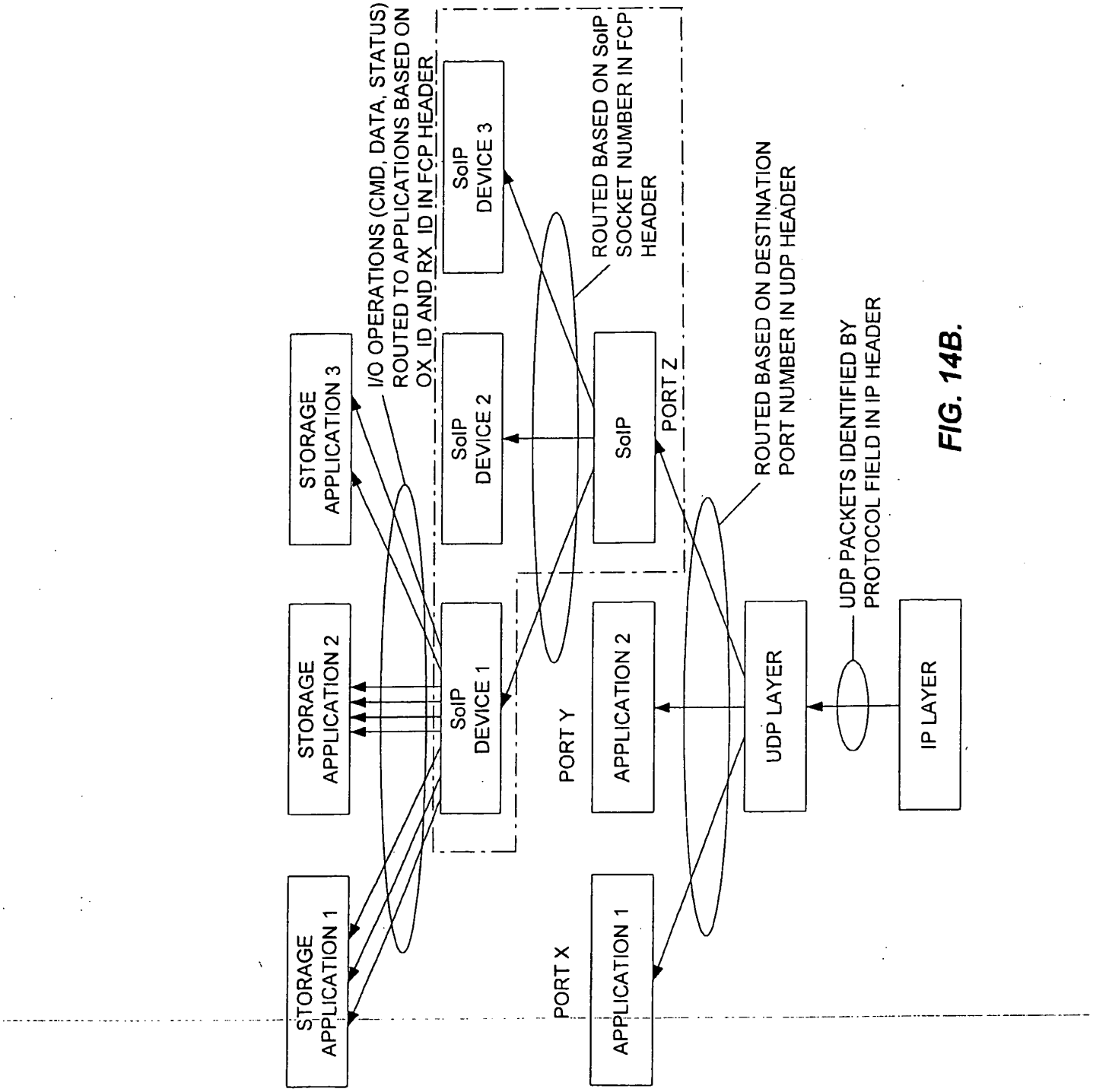


FIG. 14B.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 423

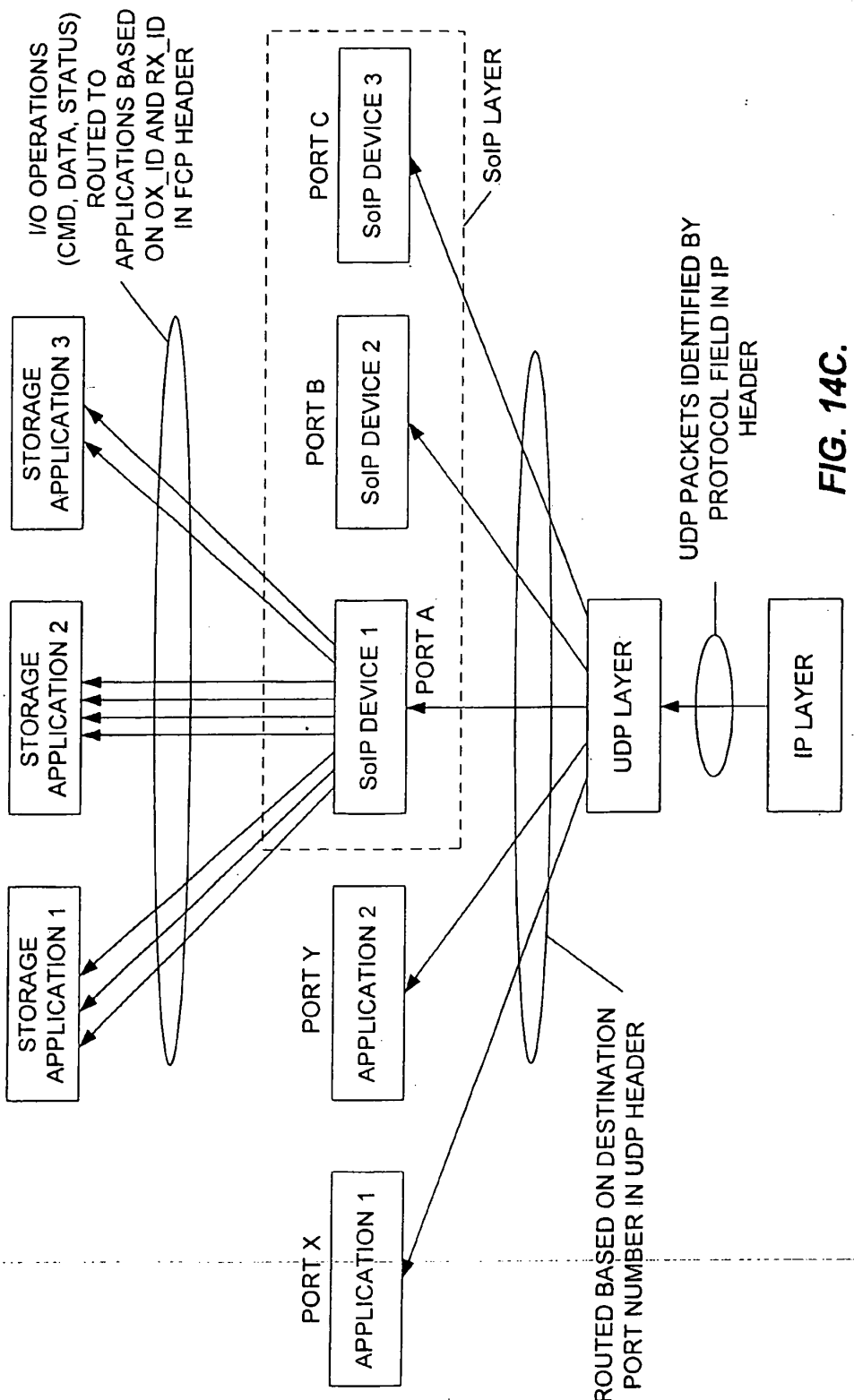


FIG. 14C.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 424

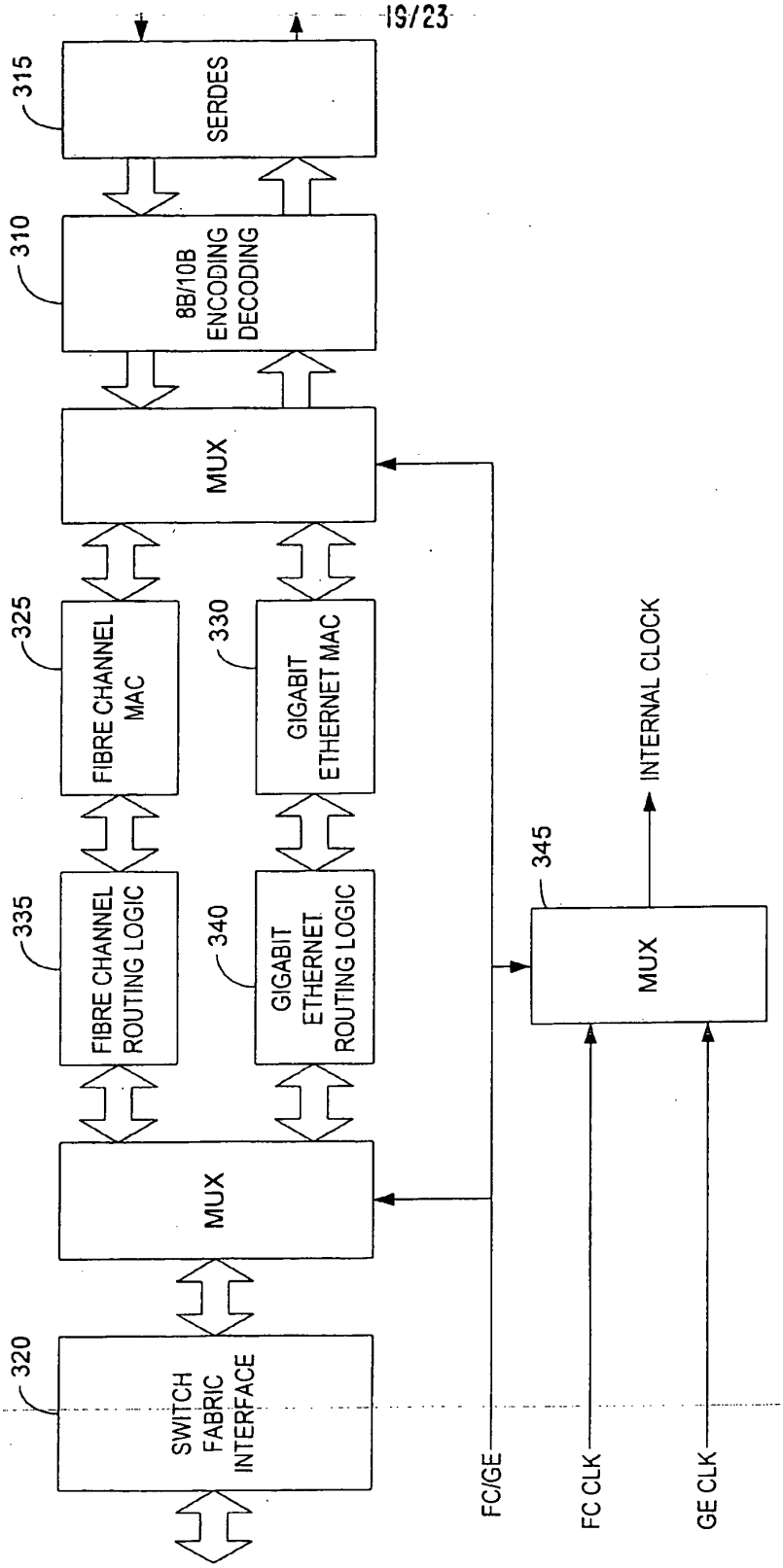


FIG. 15.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 425

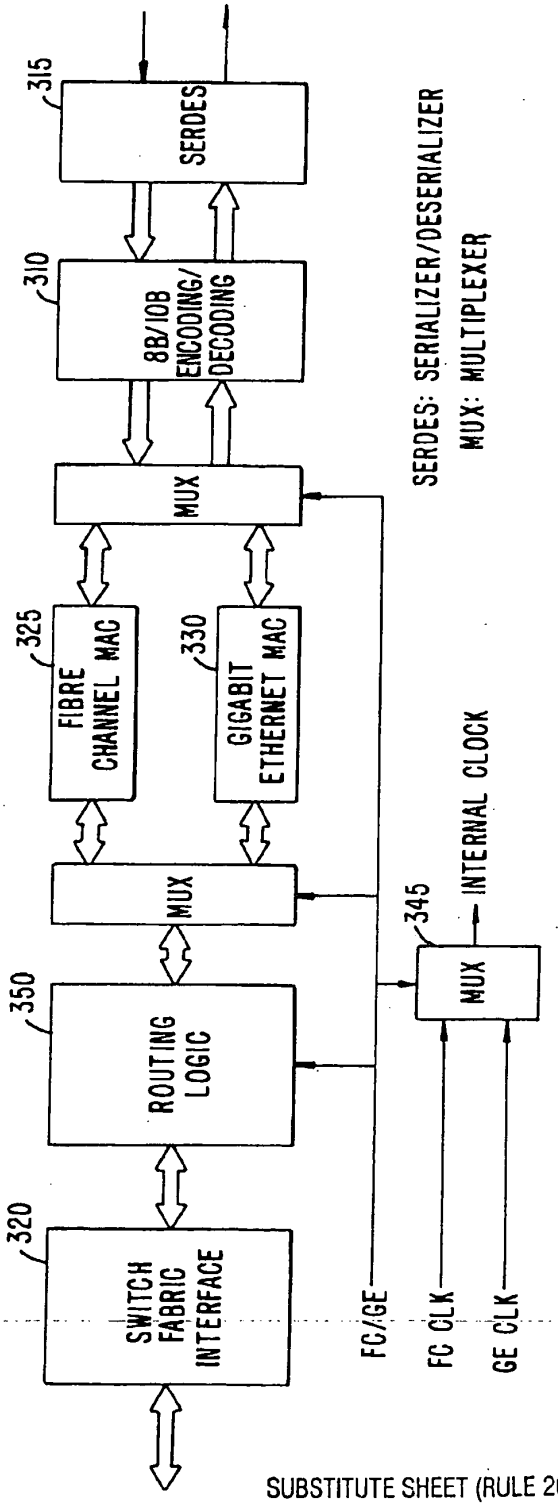


FIG. 16.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 426

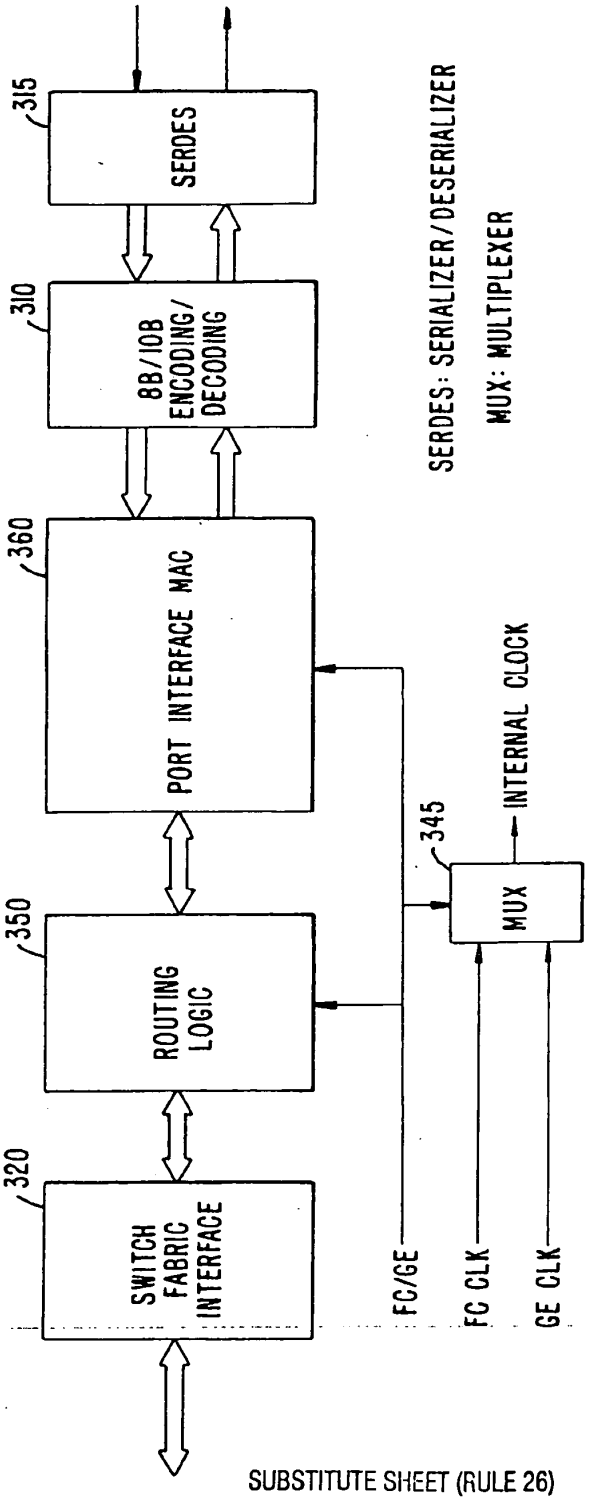


FIG. 17.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 427

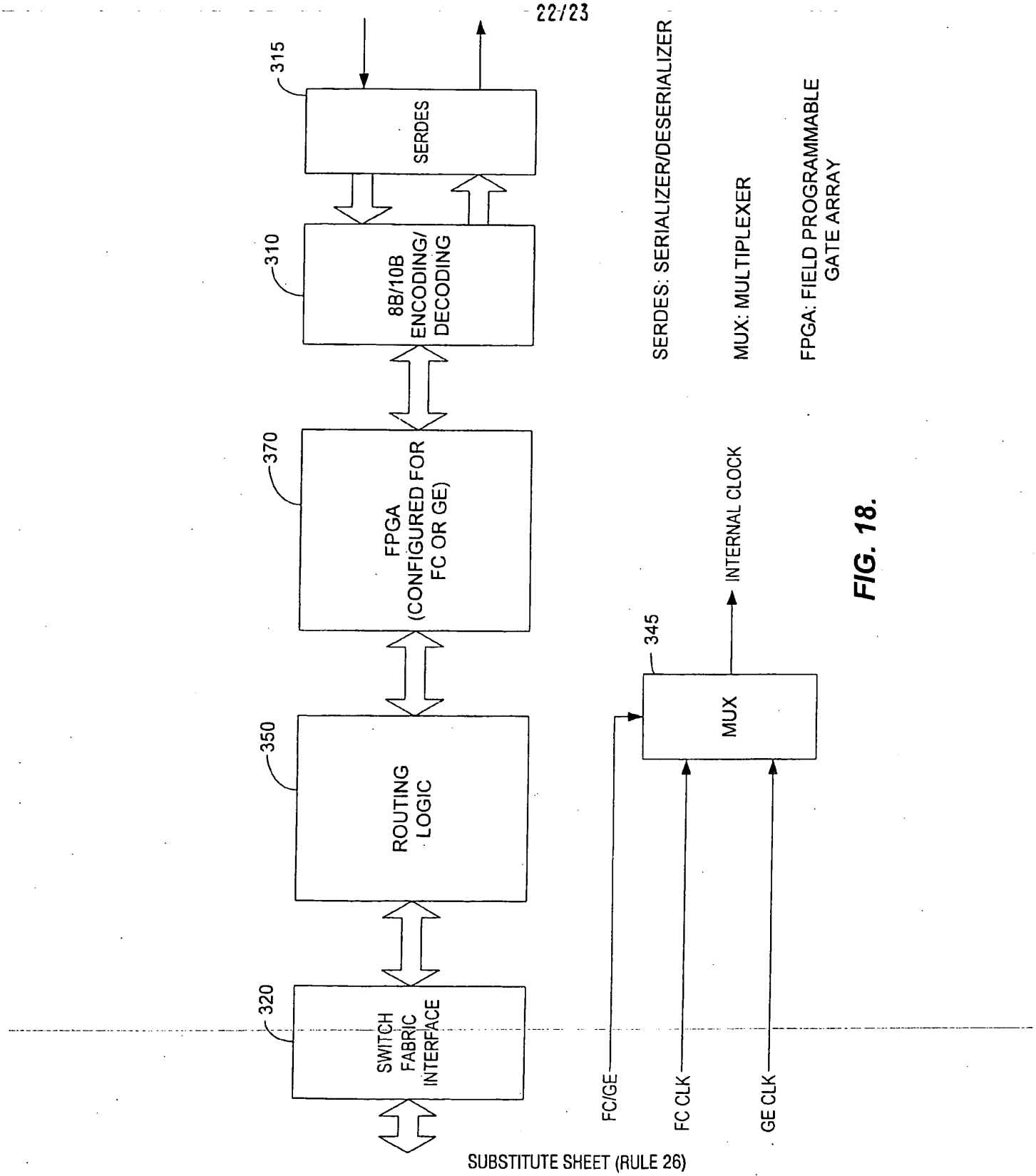


FIG. 18.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 428

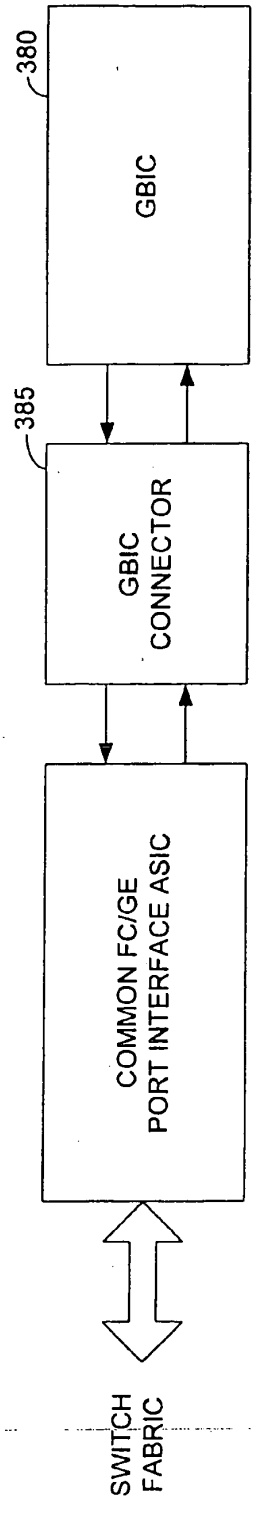


FIG. 19.

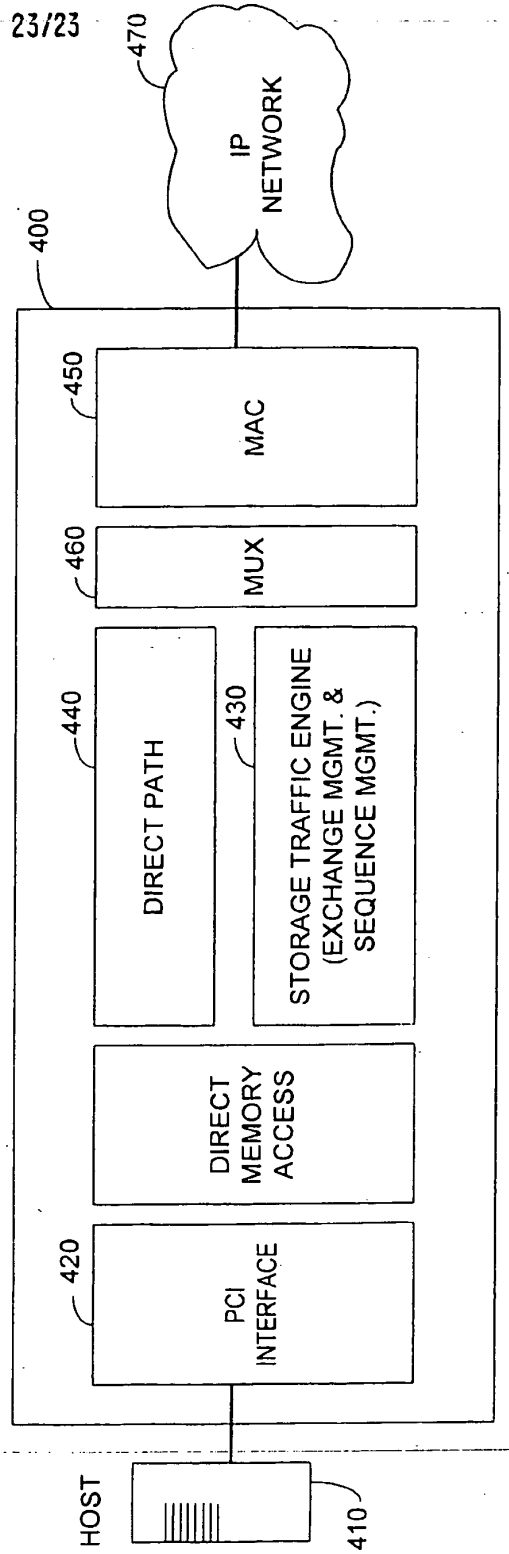


FIG. 20.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 429

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/06475

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) :H04J 3/22
US CL : 370/466
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 370/466, 355, 363.400-405, 465, 469

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EAST
search terms: SCSI, Fibre Channel, IP, conver$ SAME protocol$

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 6,000,020 A (CHIN et al) 07 December 1999, col. 1, line 51 - col. 2, line 9.	1

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search	Date of mailing of the international search report
12 JUNE 2000	11 JUL 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer CHI H. PHAM Telephone No. (703) 305-4378

Form PCT/ISA/210 (second sheet) (July 1998) ★

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
15 November 2001 (15.11.2001)

PCT

(10) International Publication Number
WO 01/86430 A2

(51) International Patent Classification[7]: G06F 9/30

(21) International Application Number: PCT/US01/15180

(22) International Filing Date: 10 May 2001 (10.05.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/203,465 11 May 2000 (11.05.2000) US
60/203,409 11 May 2000 (11.05.2000) US

(71) Applicants *(for all designated States except US)*: **NETOCTAVE, INC.** [US/US]; 507 Airport Boulevard, Suite 111, Morrisville, NC 27560 (US). **HANNA, Michael** [US/US]; 522 N. Person Street, Raleigh, NC 27604 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **BLAKER, David** [US/US]; 109 Hogan Glen Court, Chapel Hill, NC 27516 (US). **SAVARDA, Raymond** [US/US]; 4224 Sancroft Drive, Apex, NC 27502 (US).

(74) Agent: **MYERS BIGEL SIBLEY & SAJOVEC**; P.O. Box 37428, Raleigh, NC 27627 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

BEST AVAILABLE COPY

WO 01/86430 A2

(54) Title: CRYPTOGRAPHIC DATA PROCESSING SYSTEMS, COMPUTER PROGRAM PRODUCTS, AND METHODS OF OPERATING SAME IN WHICH A SYSTEM MEMORY IS USED TO TRANSFER INFORMATION BETWEEN A HOST PROCESSOR AND AN ADJUNCT PROCESSOR

(57) Abstract: Embodiments of cryptographic data processing systems, computer program products, and methods of operating same are provided in which system memory is used to transfer information between a host processor and an adjunct processor.

CRYPTOGRAPHIC DATA PROCESSING SYSTEMS, COMPUTER PROGRAM PRODUCTS, AND METHODS OF OPERATING SAME IN WHICH A SYSTEM MEMORY IS USED TO TRANSFER INFORMATION BETWEEN A HOST PROCESSOR AND AN ADJUNCT PROCESSOR

CROSS-REFERENCE TO PROVISIONAL APPLICATIONS

This application claims the benefit of Provisional Application Serial No. 60/203,409, filed May 11, 2000, entitled *Cryptographic Acceleration Methods and Apparatus*, and Provisional Application Serial No. 60/203,465, filed May 11, 2000, entitled *Methods and Apparatus for Supplying Random Numbers*, the disclosures of which are hereby incorporated herein by reference in their entirety as if set forth fully herein.

BACKGROUND OF THE INVENTION

The present invention relates generally to the field of data processing systems, and, more particularly, to cryptographic data processing systems, computer program products, and methods of operating same.

Signal processors and integrated circuit chips have been developed to accelerate cryptographic operations, such as public key operations. Examples of such chips include, but are not limited to, the Hifn 6500 available from Hifn, Inc., the SafeNet ADSP 2141 available from SafeNet, Inc., and the Rainbow Mykotronx FastMAP available from Rainbow Mykotronx, Inc. Despite the availability of cryptographic accelerator products, there remains room for improvement in the art.

For example, conventional cryptographic data processing systems generally use two main methods for issuing a command to a cryptographic accelerator: The first method involves the provision of a command register on the cryptographic accelerator that a host processor uses to issue a single command. Once the cryptographic

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 432

accelerator completes executing a command, the host processor may issue a new command. After completing a command, the cryptographic accelerator is generally idle until the host processor issues a new command. Unfortunately, the host processor may spend much time interacting directly with the cryptographic accelerator to download data and issue commands. This may reduce the amount of time available to the host processor for attending to other tasks.

The second method allows the host processor to download one or more command sequences to the cryptographic accelerator and then to instruct the cryptographic accelerator to execute one or more of the downloaded command sequences. After completing a command sequence, the cryptographic accelerator is generally idle until the host processor issues a new command. The size of the command sequences may be limited based on the amount of memory that may be placed on the cryptographic accelerator. Like the first method, the host processor may spend much time interacting directly with the cryptographic accelerator to download data and issue command sequences. This may reduce the amount of time available to the host processor for attending to other tasks.

Cryptographic accelerators generally perform operations using one or more operands. These devices may include general-purpose operand storage that comprises fixed length registers to store the operands and results. To execute an instruction, a register number is used to indicate which operand should be used for the operation and where the output should be stored. For example, if the operation were "a + b = c," then part of the instruction would indicate that "a" is in register 7, "b" is in register 1, and "c" should be put into register 2.

Because the registers are fixed in size and the operands and results are variable in size, the size of the operands will always be less than or equal to the register size. As a result, some of the memory in the registers may be wasted. This reduces the number of operands that may be stored on a chip in a given amount of space. In addition, if the cryptographic accelerator is redesigned to accommodate larger operands, then each of the registers may need to be modified. More registers may be designed into a cryptographic accelerator; however, adding more memory to a cryptographic accelerator may reduce the amount of other functionality that may be included and/or increase the cost.

Cryptographic processors and/or other types of signal processors and integrated circuits may use a hardware-based random number generator. Various

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 433

conventional methods may be used to retrieve random numbers from an integrated circuit incorporating a random number generator. One method is for the random number generator to provide one or more data registers that a host processor may read to obtain random numbers. The host processor may tell the random number generator to provide more random data before or after retrieving random data from the registers. The random number generator may generate the random data in the background so that random data may be available when needed by the host processor.

Another method for obtaining random data is for the host processor to request a sample of random data from the random number generator. The host processor may provide the random number generator with a request that specifies an amount of random data and a location in memory where the random data should be placed. The random number generator may then generate the random data and transfer the random data to the requested location in the background.

Unfortunately, by providing random data through data registers on the random number generator or other integrated circuit chip, any buffer management that may be desired is generally performed by the host processor. Moreover, the bus that connects the host processor with the random number generator may be used inefficiently because single data reads are typically used instead of block reads. If a host processor requests a block of random data, however, then the host processor may initiate the data transfers and any desired buffer management that may be desired is generally performed by the host processor. The foregoing operations may be performed in the background and/or a fast host processor may be used; however, a faster host processor may increase system costs.

SUMMARY OF THE INVENTION

Embodiments of the present invention provide cryptographic data processing systems, computer program products, and methods of operating same in which system memory is used to transfer information between a host processor and an adjunct processor. For example, in accordance with embodiments of the present invention, cryptographic data processing systems comprise a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory. One or more operands are downloaded into the local memory from the system memory, using, for example, a load command, and one or more operations are performed on at least one of the downloaded operands to generate

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 434

a result in the local memory. The result that is generated in the local memory is then stored in the system memory, using, for example, a store command. Advantageously, interaction between the host processor and the cryptographic processor may be reduced as the host processor need not consume processing time downloading operands to the cryptographic accelerator processor and/or uploading results from the cryptographic processor to system memory.

In accordance with further embodiments of the present invention, interaction between the host processor and the cryptographic processor may be further reduced and overall system performance improved by providing a command queue in the system memory, loading a command block into the command queue using the host processor, executing the command block using the cryptographic processor, and notifying the host processor that the command block has been executed. By loading commands into a command queue in the system memory where the cryptographic processor may retrieve them for processing, the host processor need not spend time interacting directly with the cryptographic processor.

In accordance with further embodiments of the present invention, the host processor may be notified that the command block has been executed in various ways. For example, the cryptographic processor may invoke an interrupt to notify the host processor that the command block has been executed. In particular embodiments, the interrupt may be invoked if the host processor has requested notification via an interrupt in an interrupt field of the command block. In other embodiments, the cryptographic processor may update a completion field in the command block to notify the host processor that the command block has been executed. In still other embodiments, a periodic interrupt may be defined such that when the interrupt occurs the host processor reads the completion fields of any command blocks.

In accordance with still further embodiments of the present invention, improved utilization of the system memory may be attained by re-using at least a portion of a command block that contains input data to store a result or output that is generated by an adjunct processor, such as the cryptographic processor. For example, a command queue may be provided in the system memory and a command block may be loaded into the command queue using the host processor. The command block comprises an input data field that contains input data. The adjunct processor performs an operation based on the input data to generate a result and this result is stored in the input data field such that at least a portion of the input data is overwritten.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 435

Advantageously, the memory reserved for the command block in the system memory may be reduced because additional storage space need not be reserved to store the result of executing the command block either in the command block or elsewhere in the system memory.

Cryptographic processors and/or other types of signal processors and integrated circuits may use a hardware-based random number generator. In accordance with further embodiments of the present invention, random number samples may be provided for use by the host processor while reducing interaction between the host processor and the cryptographic processor. For example, a random number data queue in the system memory may be provided that has a read address and a write address associated therewith. The cryptographic processor loads a random number sample into the random number data queue at the write address and the host processor reads the random number sample beginning at the read address. The host processor need only interact with the cryptographic processor to update the read address and to check the value of the write address when the read address approaches the last value the host processor has for the write address. In addition, the cryptographic accelerator processor may manage the buffering of the random number samples, which may conserve processor cycles of the host processor.

BRIEF DESCRIPTION OF THE DRAWINGS

Other features of the present invention will be more readily understood from the following detailed description of specific embodiments thereof when read in conjunction with the accompanying drawings, in which:

FIG. 1 is a block diagram that illustrates cryptographic data processing systems, computer program products, and methods of operating same in accordance with embodiments of the present invention;

FIG. 2 is a flowchart that illustrates operations of cryptographic data processing systems and computer program products in accordance with embodiments of the present invention;

FIGS. 3 - 5 are block diagrams that illustrate functional execution units of a cryptographic accelerator processor in accordance with embodiments of the present invention;

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 436

FIG. 6 is a flowchart that illustrates operations of cryptographic data processing systems and computer program products in accordance with further embodiments of the present invention;

FIG. 7 - 8 are block diagrams that illustrate an encryption/authentication command queue and a public key command queue, respectively, in accordance with embodiments of the present invention;

FIGS. 9 - 11 are flowcharts that illustrate operations of cryptographic data processing systems and computer program products in accordance with further embodiments of the present invention;

FIGS. 12A - 12D are block diagrams that illustrate command blocks in accordance with embodiments of the present invention;

FIG. 13 is a flowchart that illustrates operations of cryptographic data processing systems and computer program products in accordance with further embodiments of the present invention;

FIGS. 14A, 14B, and 15 are block diagrams that illustrate command blocks in accordance with further embodiments of the present invention;

FIGS. 16 and 17 are flowcharts that illustrate operations of cryptographic data processing systems and computer program products in accordance with further embodiments of the present invention;

FIG. 18 is a block diagram that illustrates a random number generator data queue in accordance with embodiments of the present invention;

FIG. 19 is a flowchart that illustrates operations of cryptographic data processing systems and computer program products in accordance with further embodiments of the present invention;

FIG. 20 is a block diagram that illustrates a command interface for a conventional application specific integrated circuit;

FIG. 21 is a block diagram that illustrates parallel command interfaces for an application specific integrated circuit in accordance with embodiments of the present invention;

FIG. 22 is a block diagram of a cryptographic accelerator processor in which command interface managers are respectively associated with functional execution units in accordance with embodiments of the present invention; and

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 437

FIG. 23 is a flowchart that illustrates operations of cryptographic data processing systems and computer program products in accordance with further embodiments of the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that there is no intent to limit the invention to the particular forms disclosed, but on the contrary, the invention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the invention as defined by the claims. Like reference numbers signify like elements throughout the description of the figures. It will also be understood that when an element is referred to as being "connected" or "coupled" to another element, it can be directly connected or coupled to the other element or intervening elements may also be present. In contrast, when an element is referred to as being "directly connected" or "directly coupled" to another element, there are no intervening elements present.

The present invention may be embodied as methods, data processing systems, and/or computer program products. Accordingly, the present invention may be embodied in hardware and/or in software (including firmware, resident software, micro-code, *etc.*). Furthermore, the present invention may take the form of a computer program product on a computer-usable or computer-readable storage medium having computer-usable or computer-readable program code embodied in the medium for use by or in connection with an instruction execution system. In the context of this document, a computer-usable or computer-readable medium may be any medium that can contain, store, communicate, propagate, or transport the program for use by or in connection with the instruction execution system, apparatus, or device.

The computer-usable or computer-readable medium may be, for example but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, device, or propagation medium. More specific examples (a nonexhaustive list) of the computer-readable medium would include the following: an electrical connection having one or more wires, a portable computer diskette, a random access memory (RAM), a read-only memory (ROM), an erasable

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 438

programmable read-only memory (EPROM or Flash memory), an optical fiber, and a portable compact disc read-only memory (CD-ROM). Note that the computer-usable or computer-readable medium could even be paper or another suitable medium upon which the program is printed, as the program can be electronically captured, via, for instance, optical scanning of the paper or other medium, then compiled, interpreted, or otherwise processed in a suitable manner, if necessary, and then stored in a computer memory.

Referring now to **FIG. 1**, an exemplary cryptographic data processing system **12**, in accordance with embodiments of the present invention, comprises a cryptographic accelerator processor **14**, a host processor **16**, a cache memory **18**, a system memory **22**, and a system bus controller **24**, such as a north-bridge system controller. The system bus controller **24** couples the host processor **16** to the cache memory **18** and the system memory **22**, and also couples the host processor **16** and the system memory **22** to the cryptographic accelerator processor **14** via a system bus **26**, which may be, for example, a peripheral component interconnect (PCI) bus. The host processor **16** may be, for example, a commercially available or custom microprocessor. The system memory **22** is representative of an overall hierarchy of memory devices containing the software and data used to implement the functionality of the cryptographic data processing system **12**. The system memory **22** may include, but is not limited to, the following types of devices: ROM, PROM, EPROM, EEPROM, flash, SRAM, and DRAM.

In accordance with embodiments of the present invention, the cryptographic accelerator processor **14** comprises a random number generator (RNG) execution unit **28**, an encryption/authentication (E/A) execution unit **32**, and a public key (PK) engine execution unit **34**, which are coupled to a local memory **36** via a local bus **38**. In accordance with particular embodiments of the present invention, the system memory **22** contains a random number (RN) data queue **42**, an E/A command queue **44**, a PK command queue **46**, and data buffer(s) **47**.

Although **FIG. 1** illustrates an exemplary cryptographic data processing system architecture, it will be understood that the present invention is not limited to such a configuration, but is intended to encompass any configuration capable of carrying out operations described herein. Computer program code for carrying out operations of embodiments of the cryptographic data processing system **12** may be written in a high-level programming language, such as C or C++, for development

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 439

convenience. Nevertheless, some modules or routines may be written in assembly language or even micro-code to enhance performance and/or memory usage. It will be further appreciated that the functionality of any or all of the program modules may also be implemented using discrete hardware components, a single application specific integrated circuit (ASIC), or a programmed digital signal processor or microcontroller.

The present invention is described hereinafter with reference to flowchart and/or block diagram illustrations of methods, data processing systems, and/or computer program products in accordance with exemplary embodiments of the invention. It will be understood that each block of the flowchart and/or block diagram illustrations, and combinations of blocks in the flowchart and/or block diagram illustrations, may be implemented by computer program instructions. These computer program instructions may be provided to a processor of a general purpose computer, a special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

These computer program instructions may also be stored in a computer usable or computer-readable memory that may direct a computer or other programmable data processing apparatus to function in a particular manner, such that the instructions stored in the computer usable or computer-readable memory produce an article of manufacture including instructions that implement the function/act specified in the flowchart and/or block diagram block or blocks.

The computer program instructions may also be loaded onto a computer or other programmable data processing apparatus to cause a series of operational steps to be performed on the computer or other programmable apparatus to produce a computer implemented process such that the instructions that execute on the computer or other programmable apparatus provide steps for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

Exemplary operations of cryptographic data processing systems, computer program products, and methods of operating same, in accordance with embodiments of the present invention, will be described hereafter. Referring now to **FIG. 2**, the host processor **16** loads a command block into one of the command queues **44** and **46**

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 440

at block 52. The cryptographic accelerator processor 14 may be notified by the host processor 16 that the command block is available for processing or may periodically access the command queues 44, and/or 46 to determine if a command block is available for processing. The cryptographic accelerator processor 14 downloads the command block from one of the command queues 44 and 46 and executes the command block at block 54. Once the cryptographic accelerator processor 14 completes execution of the command block, the host processor 16 is notified at block 56. Thus, according to embodiments of the present invention, the host processor 16 need not spend time interacting directly with the cryptographic accelerator processor 14 (*e.g.*, issuing a command to the cryptographic accelerator processor 14, waiting for that command to complete, and then issuing another command). Instead, the host processor 16 may load commands into command queues 44 and 46, which may then be processed in background by the cryptographic accelerator processor 14. Moreover, the size and number of command block sequences may be less constrained because the availability of system memory is generally more abundant.

Referring now to FIGS. 3 - 5, the RNG execution unit 28, the E/A execution unit 32, and the PK engine execution unit 34 may use various registers that facilitate communication with the RN data queue 42 and the command queues 44 and 46. For example, as shown in FIG. 3, a control/status register 62, a RN data queue base address register 64, a RN data queue size register 66, and a RN data queue pointer register 68 may be defined for use by the RNG execution unit 28. The control/status register 62 may include a self-test error field, which may be set if the RNG execution unit 28 generates two successive random number samples that are the same, and/or an error flag field, which may be used to notify the host processor 16 of an error on the system bus 26. The RN data queue base address register 64 may be used to hold the base address of the RN data queue 42 in the system memory 22. If the RN data queue 42 does not have a fixed size, then the RN data queue size register 66 may be used to hold the size of the RN data queue 42. The RN data queue pointer register 68 may comprise a read pointer 72 portion and a write pointer 74 portion, which may be used by the RNG execution unit 28 and the host processor 16 as will be discussed in more detail hereinafter.

As shown in FIG. 4, a control/status register 82, an E/A command queue base address register 84, an E/A command queue size register 86, and an E/A command queue pointer register 88 may be defined for use by the E/A execution unit 32. The

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 441

control/status register **82** may include an interrupt flag field, which may be set if the host processor **16** requests an interrupt upon completion of a command block and/or if execution of a command block fails and/or an error flag field, which may be used to notify the host processor **16** of an error on the system bus **26**. The E/A command queue base address register **84** may be used to hold the base address of the E/A command queue **44** in the system memory **22**. If the E/A command queue **44** does not have a fixed size, then the E/A command queue size register **86** may be used to hold the size of the E/A command queue **44**. The E/A command queue pointer register **88** may comprise a read pointer **92** portion and a write pointer **94** portion, which may be used by the E/A execution unit **32** and the host processor **16**, respectively, as will be discussed in more detail hereinafter.

As shown in **FIG. 5**, a control/status register **102**, a PK command queue base address register **104**, a PK command queue size register **106**, and a PK command queue pointer register **108** may be defined for use by the PK engine execution unit **34**. The control/status register **102** may include an interrupt flag field, which may be set if the host processor **16** requests an interrupt upon completion of a command block and/or if execution of a command block fails and/or an error flag field, which may be used to notify the host processor **16** of an error on the system bus **26**. The PK command queue base address register **104** may be used to hold the base address of the PK command queue **46** in the system memory **22**. If the PK command queue **46** does not have a fixed size, then the PK command queue size register **106** may be used to hold the size of the PK command queue **46**. The PK command queue pointer register **108** may comprise a read pointer **112** portion and a write pointer **114** portion, which may be used by the PK engine execution unit **34** and the host processor **16**, respectively, as will be discussed in more detail hereinafter.

Referring now to **FIG. 6**, operations for loading a command block into the E/A command queue **44** and/or the PK command queue **46**, in accordance with embodiments of the present invention, will be described in more detail hereafter. In general, the host processor **16** writes commands into the command queues **44** and **46** beginning at write address locations stored in the write pointers for the respective command queues (*e.g.*, write pointers **94** and **114**). Before writing a command block into a command queue, however, the host processor determines at block **122** whether the write address plus the command block size equals the read address stored in the corresponding read pointer **92** or **112**. If the result determined at block **122** is "Yes,"

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 442

then the host processor **16** postpones loading a new command block into the command queue until the cryptographic accelerator processor **14** has incremented the read address. If, however, the result determined at block **122** is "No," then the host processor **16** loads a command block into the command queue at block **124** at the write address associated with the command queue and then increments the write address at block **126** by an amount corresponding to the size of the loaded command block. The host processor **16** need not check the current read address every time a new command block is loaded. Instead, the host processor **16** may check the read address when the write address is getting close to the last value the host processor **16** has for the read address. Checking the read address may be expensive in terms of processor cycles consumed. By checking the read address only when the read address is getting close to the write address (*e.g.*, within a predefined threshold), host processor 16 cycles may be conserved.

The foregoing operations are illustrated, for example, in **FIGS. 7** and **8**, which show embodiments of the E/A command queue **44** and the PK command queue **46**, respectively. As shown in **FIGS. 7** and **8**, both the E/A command queue **44** and the PK command queue **46** are configured to hold m command blocks, which each comprise eight, thirty-two bit words. The host processor **16** has written a single command block into the first command block position (*i.e.*, the "0" position) and the write address has been incremented to point to the next empty command block slot. The addresses used in **FIGS. 7** and **8** are based on command block slot numbers for purposes of illustration. These addresses may be converted into absolute addresses by multiplying the command block slot number by 256 and adding the resulting product to the respective base addresses for the command queues, which are stored in the E/A command queue base address register **84** and the PK command queue base address register **104**, respectively. Note that the test used at block **122** of **FIG. 6** to determine whether a new command block may be loaded into a command queue implies that if a command queue may hold up to m command blocks, then only m -1 command blocks may be stored in the command queue at the same time.

Referring now to **FIG. 9**, operations for executing a command block that has been loaded into the E/A command queue **44** and/or the PK command queue **46**, in accordance with embodiments of the present invention, will be described in more detail hereafter. At block **132**, the cryptographic accelerator processor **14** determines whether the write address is equal to the read address. Specifically, the E/A execution

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 443

unit **32** determines whether the write address is equal to the read address for the E/A command queue **44** and the PK engine execution unit **34** determines whether the write address is equal to the read address for the PK command queue **46**. If the result determined at block **132** is "Yes," then the cryptographic processor **14** waits until the host processor **16** loads a new command block into the command queue. If, however, the result determined at block **132** is "No," then the cryptographic accelerator processor **14** downloads the command block at the read address associated with the command queue and executes the command block at block **134**. In particular embodiments of the present invention, multiple command blocks may be downloaded for execution on the cryptographic accelerator processor **14** at the same time, which may further improve performance. The cryptographic accelerator processor **14** then increments the read address at block **136** by an amount corresponding to the size of the executed command block.

Returning to **FIGS. 7** and **8**, the read addresses are set to point to the first command block slot, which has been loaded with a command block by the host processor **16**. The E/A execution unit **32** and the PK engine execution unit **34** may read the command blocks loaded in the E/A command queue **44** and the PK command queue **46**, respectively, with only minimal interaction with the host processor **16**, *e.g.*, maintenance of the read pointers **92** and **112**, and the write pointers **94**, and **114**. In general, the cryptographic accelerator processor **14** may continue to execute commands located in a circular command queue in system memory until the read address equals the write address for that command queue.

In accordance with further embodiments of the present invention, interaction between the host processor **16** and the cryptographic accelerator processor **14** may be further reduced and overall system performance improved by including load and store commands in the cryptographic accelerator processor's command set. Referring now to **FIG. 10**, a load command loads one or more operands from the system memory **22** (*e.g.*, the data buffer(s) **47**) to the local memory **36** at block **142**. The cryptographic accelerator processor **14** then performs one or more operations on the operand(s) at block **144** to generate a result that is stored in the local memory **36**. A store command then stores the result in the system memory **22** at block **146**. Advantageously, the host processor **16** need not consume processing time downloading operands to the cryptographic accelerator processor **14** and/or uploading results from the cryptographic accelerator processor **14** into the system memory **22**.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 444

To improve utilization of the chip area used to implement the cryptographic accelerator processor **14**, at least a portion of the operands downloaded from the system memory **22** may be stored in the local memory **36**. Instead of using a register number to identify the location of operands and results, an offset is used that identifies the relative position of the operands and results in the local memory **36**. For example, to perform the operation "a + b = c," a cryptographic accelerator processor **14** instruction may indicate that "a" is at offset 0 relative to a base address of the local memory **36**, "b" is at offset 8 relative to the base address of the local memory **36**, and the result "c" should be placed at offset 122 relative to the base address of the local memory **36**. In accordance with further embodiments of the present invention, the result generated in the local memory **36** may also be stored in a result field of a command block, which is located in one of the command queues **44** and **46** in the system memory **22**. Advantageously, operands and results may be packed together into the local memory **36**, which may conserve storage space. Because there is no wasted space in storing the operands and results in the local memory **36**, memory utilization may be improved. If the cryptographic accelerator processor **14** needs to be redesigned to handle larger operands, then the local memory **36** may be easier to resize than resizing several registers.

In accordance with further embodiments of the present invention, interaction between the host processor **16** and the cryptographic accelerator processor **14** may be further reduced and overall system performance improved by allowing the cryptographic accelerator processor **14** to inform the host processor **16** when command blocks have been executed. Referring now to **FIG. 11**, the host processor **16** loads a command block into one of the command queues **44** and **46** at block **152**. As shown in **FIG. 12A**, the command block may include an interrupt field, which may be set by the host processor **16** to turn an interrupt request on or off. The cryptographic accelerator processor **14** downloads the command block from one of the command queues **44** and **46** and executes the command block at block **154**. The cryptographic accelerator processor **14** may optionally store error information in the command block as shown in **FIG. 12B** at block **156**. The error information may comprise information that is associated with downloading the command block to the cryptographic accelerator processor **14** and/or executing the command block on the cryptographic accelerator processor **14**. At block **158**, if an interrupt has been requested in the interrupt field of the command block, then the cryptographic

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 445

accelerator processor **14** invokes an interrupt to notify the host processor **16** that the command block has completed.

In other embodiments of the present invention, instead of invoking an interrupt to notify the host processor **16** that a command block has been executed, the cryptographic accelerator processor **14** may update a completion field in the command block as shown in **FIG. 12C**. In addition, a periodic interrupt may be defined that upon each occurrence triggers the host processor **16** to check one or more of the command queues **44** and **46** to determine whether any of the command blocks stored therein have been executed by examining their completion fields. In still other embodiments of the present invention, the cryptographic accelerator processor **14** may store the results from executing a command block in the command block as shown in **FIG. 12D**.

In still other embodiments of the present invention, the host processor **16** may set a timer when storing a command block into a command queue **42, 44**. Upon expiration of the timer, the host processor **16** may check to determine whether the command block has been executed. Advantageously, the status of a command block may be determined by the host processor **16** without the need to process an interrupt from the cryptographic accelerator processor **14**.

In accordance with further embodiments of the present invention, improved utilization of the system memory **22** may be attained by re-using at least a portion of a command block that contains input data to store a result or output that is generated by an adjunct processor, such as the cryptographic accelerator processor **14**, upon executing the command block. It is assumed that the size of the result or output is small enough to fit into the portion of the command block containing the input data that is to be overwritten. In addition, the region of the command block in which the result or output is stored should be selected carefully to ensure that the input data that is overwritten is no longer needed by the host processor **16** after the command block has been executed by the adjunct processor.

Referring now to **FIG. 13**, exemplary operations begin at block **162** where the host processor loads a command block that includes input data into one of the command queues **44** or **46** in the system memory **22**. Note that that instead of or in addition to including input data into the command block, the command block may include pointers to input data that reside, for example, in the data buffer(s) 47 in the system memory **22**. An adjunct processor, such as the cryptographic accelerator

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 446

processor **14**, may download the command block and perform one or more operations on the input data to generate a result at block **164**. If the command block includes pointers to input data, then the data are separately downloaded to the cryptographic accelerator processor **14** using the input data pointers. The result is then stored in the command block in the system memory **22** at block **166** such that at least a portion of the input data is overwritten. Advantageously, the memory reserved for the command block in the system memory **22** may be reduced because additional storage space need not be reserved to store the result of executing the command block either in the command block or elsewhere in the system memory **22**.

The foregoing operations are illustrated by way of example in **FIGS. 14A** and **14B**, which show an exemplary command block for decrypting an encrypted packet. Specifically, in **FIG. 14A**, a command block is shown that comprises a field that contains a hash key for the encrypted packet and another field that contains input information. The cryptographic accelerator processor **14** downloads the command block of **FIG. 14A** and performs hash operations using the hash key and input information to generate a hash value. As shown in **FIG. 14B**, this hash value is then stored in the command block in the system memory **22** by overwriting the input information, which is no longer needed once the hash value has been computed. Note that the input information may be one or more pointers to input data stored, for example, in the data buffer(s) **47** in the system memory **22**.

In accordance with further embodiments of the present invention illustrated in **FIG. 15**, the command block may include an input pointer field and/or an output pointer field, which are used to identify the location of the encrypted packet in the system memory **22** and the location where the decrypted packet is to be stored in the system memory **22**. For example, the cryptographic accelerator processor **14** may use the input pointer to download the encrypted packet from the system memory **22** and may then decrypt the encrypted packet using the hash key and input information to generate a hash value as discussed hereinabove. Note that the input information may be one or more pointers to input data stored, for example, in the data buffer(s) **47** in the system memory **22**. The hash value may be attached to the decrypted packet and the decrypted packet with the attached hash value may be stored in the system memory **22** at the address identified by the output pointer field in the command block.

Cryptographic processors and/or other types of signal processors and integrated circuits may use a hardware-based random number generator. The

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 447

cryptographic accelerator processor **14** may include a RNG execution unit **28** that may be used to generate random numbers for use by other execution units of the cryptographic accelerator processor **14** and/or the host processor **16**. Exemplary operations that may be used to reduce interaction between the host processor **16** and the cryptographic accelerator processor **14** and to improve overall system performance will be described hereafter. Referring now to **FIG. 16**, operations begin at block **172** where the cryptographic accelerator processor **14** loads a random number sample into the RN data queue **42** beginning at the write address stored in the write pointer field **74** of the RN data queue pointer register **68** (see **FIG. 3**). At block **174** the host processor **16** reads the random number sample in the RN data queue **42** beginning at the read address stored in the read pointer field **72** of the RN data queue pointer register **68** (see **FIG. 3**). Thus, according to embodiments of the present invention, the host processor **16** need not spend time interacting directly with the cryptographic accelerator processor **14** to request blocks of random data and/or reading random data from, for example, one or more registers on the cryptographic accelerator processor **14** chip.

Referring now to **FIG. 17**, operations for loading a random number sample into the RN data queue **42**, in accordance with embodiments of the present invention, will be described in more detail hereafter. Before writing a random number sample into the RN data queue **42**, the cryptographic accelerator processor **14** determines at block **182** whether the write address plus the random number sample size equals the read address stored in the read pointer field **72**. If the result determined at block **182** is "Yes," then the cryptographic processor **14** postpones loading a new random number sample into the RN data queue **42** until the host processor **16** has incremented the read address. If, however, the result determined at block **182** is "No," then the cryptographic processor **14** loads a random number sample into the RN data queue **42** at block **184** at the write address stored in the write pointer field **74** and then increments the write address at block **186** by an amount corresponding to the size of the loaded random number sample.

Note that in accordance with embodiments of the present invention, the cryptographic processor **14** may include a register and/or may recognize a command block that may be written to the cryptographic processor **14** that allows the host processor **16** to, for example, provide the cryptographic processor **14** with a random

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 448

number seed and/or instruct the cryptographic processor **14** to begin generating random numbers.

The foregoing operations are illustrated, for example, in **FIG. 18**, which shows an exemplary embodiment of the RN data queue **42**. As shown in **FIG. 18**, the RN data queue **42** is configured to hold 512 random number samples, which each comprise 64 bits. The cryptographic processor **14** has written four random number samples into addresses 1 through 4 and the write address has been incremented to point to the next available address, which is empty or contains data that have already been read by the host processor **16**. The addresses shown in **FIG. 18** are based on random number sample units for purposes of illustration. These addresses may be converted into absolute addresses by multiplying the random number sample number by 64 and adding the resulting product to the respective base address for the RN data queue **42**, which is stored in the RN data queue base address register **64**. Note that the test used at block **182** of **FIG. 17** to determine whether a new random number sample may be loaded into the RN data queue **42** implies that if the RN data queue **42** may hold up to m random number samples, then only m - 1 random number samples may be stored in the RN data queue **42** at the same time. Thus, if the RN data queue **42** is filled to its capacity, then it may hold 32,704 bits (511, 64-bit random number samples), which exceeds the 20,000 bits required by the Federal Information Processing Standard (FIPS) 140-1, Security Requirements for Cryptographic Modules issued January 11, 1994.

Referring now to **FIG. 19**, operations for reading a command block that has been loaded into the RN data queue **42**, in accordance with embodiments of the present invention, will be described in more detail hereafter. At block **192**, the host processor **16** determines whether the write address is equal to the read address. If the result determined at block **192** is "Yes," then the host processor **16** waits until the cryptographic accelerator processor **14** loads a new random number sample into the RN data queue **42**. If, however, the result determined at block **192** is "No," then the host processor **16** reads the random number sample at the read address stored in the read pointer field **72** at block **194**. The host processor **16** then increments the read address at block **196** by an amount corresponding to the size of the random number sample. The host processor **16** need not check the current write address every time a new random number sample is read. Instead, the host processor **16** may check the

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 449

write address when the read address is getting close to the last value the host processor **16** has for the write address.

Thus, according to embodiments of the present invention, a cryptographic accelerator processor **14** may provide random number samples for use by a host processor **16** with reduced interaction between the host processor **16** and the cryptographic accelerator processor **14**. In general, the host processor **14** need only interact with the cryptographic accelerator processor **14** to update the read address and to check the value of the write address when the read address approaches the last value the host processor **14** has for the write address. In addition, the cryptographic accelerator processor **14** may manage the buffering of the random number samples, which may conserve processor cycles of the host processor **16** and may reduce transactions on the system bus **26**, which may improve overall system performance.

The performance of cryptographic data processing systems may be affected by the system architecture and the methodology used to perform operations. For example, conventional cryptographic data processing systems may comprise one or more ASICs, such as the ASIC **202** shown in **FIG. 20**. The ASIC **202** comprises a plurality of functional units **204**, **206**, and **208**, which are configured to perform specific operations. As shown in **FIG. 20**, however, input commands are provided to the ASIC **202** serially and then routed to the appropriate functional unit **204**, **206**, and/or **208**. The outputs and/or results of executing the input commands are provided serially as command outputs from the ASIC **202**. Thus, the ASIC **202** typically processes commands sequentially such that a first command must finish before a subsequent command may be processed even if the commands are executed by different functional units.

Referring now to **FIG. 21**, the performance of cryptographic data processing systems may be improved, in accordance with embodiments of the present invention, by providing separate command interfaces that are respectively associated with the functional units such that each functional unit may receive command inputs and may generate command outputs and/or results independently of other functional units. As shown in **FIG. 21**, an ASIC **212** includes a plurality of functional units **214**, **216**, and **218**, which each receive command inputs through its own command interface and generate outputs and/or results that may be communicated to another processor through the command interface. By associating a separate command interface with each functional unit **214**, **216**, and **218**, the functional units **214**, **216**, and **218** may

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 450

operate independently and in parallel, thereby improving the performance of a cryptographic data processing system.

Referring now to **FIG. 22**, the functional units **214**, **216**, and **218** may comprise the E/A execution unit **32**, the RNG execution unit **28**, and the PK engine execution unit **34**. The E/A execution unit **32** comprises a command interface manager **222**, the RNG execution unit **28** comprises a command interface manager **224**, and the PK engine execution unit **34** comprises a command interface manager **226**. These respective command interface managers **222**, **224**, and **226** may be used to receive input command blocks from the E/A command queue **44**, to transmit random number samples to the RN data queue **42**, and to receive input command blocks from the PK command queue **46**, respectively, and to allow the respective execution units **28**, **32**, and **34** to perform operations in parallel.

Referring now to **FIG. 23**, operations of cryptographic data processing systems in which command interface managers are respectively associated with a plurality of functional units, in accordance with embodiments of the present invention, will be described hereafter. Operations begin at block **232** where one or more command blocks are provided to each of the functional units, such as, for example, by providing command blocks in the E/A command queue **44** and the PK command queue **46** for the E/A execution unit **32** and the PK engine execution unit **34**, respectively. At block **234**, the command blocks are simultaneously executed by the functional units by accessing the command blocks in parallel through, for example, the command interface manager **222** and the command interface manager **226**, which are associated with the E/A execution unit **32** and the PK engine execution unit **34**, respectively.

Note that command blocks may be provided to the cryptographic processor **14** in serial fashion over the system bus **24**. Nevertheless, the cryptographic processor **14** may distribute command blocks to the command interface managers **222**, **224**, and **226** associated with the execution units **32**, **28**, and **34**, which may then process the command blocks in parallel.

For purposes of illustration, exemplary embodiments of the present invention have been discussed hereinabove in which operations related to random number generation, encryption/authentication, and public key generation are performed in parallel based on functional units defined therefor. It will be understood that the operations that may be performed in parallel may be adjusted based on requirements

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 451

and/or needs. Moreover, commands may be provided to the command interface managers in a variety of ways. A processor may write commands directly to the command interface managers or, alternatively, commands may be stored in a memory and the command interface managers may be provided with the addresses where they may retrieve the stored commands for execution.

In summary, by performing operations in parallel using a plurality of functional units, the total number of operations that may be performed may be increased and the average latency for completing operations may be reduced.

The flowcharts of **FIGS. 2, 6, 9 - 11, 13, 16, 17, 19,** and **23** illustrate the architecture, functionality, and operations of possible embodiments of the cryptographic data processing system **12** of **FIG. 1**. In this regard, each block may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function(s). It should also be noted that in some alternative embodiments, the functions noted in the blocks may occur out of the order noted in **FIGS. 2, 6, 9 - 11, 13, 16, 17, 19,** and **23**. For example, two blocks shown in succession may in fact be executed substantially concurrently or the blocks may sometimes be executed in the reverse order, depending on the functionality involved.

In concluding the detailed description, it should be noted that many variations and modifications can be made to the preferred embodiments without substantially departing from the principles of the present invention. All such variations and modifications are intended to be included herein within the scope of the present invention, as set forth in the following claims.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 452

CLAIMS

We claim:

1. A method of operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the method comprising:

loading at least one operand from the system memory to the local memory;

performing at least one operation on the at least one operand to generate a result in the local memory; and

storing the result generated in the local memory in the system memory.

2. A method as recited in Claim 1, wherein performing the at least one operation, and storing the result are performed by the cryptographic processor without interaction with the host processor.

3. A cryptographic data processing system, comprising:

a host processor;

a system memory coupled to the host processor; and

a cryptographic processor that comprises a local memory and is coupled to the host processor and the system memory, the cryptographic processor being programmed to load at least one operand from the system memory to the local memory, perform at least one operation on the at least one operand to generate a result in the local memory, and store the result generated in the local memory in the system memory.

4. A method of operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the method comprising:

providing a command queue in the system memory;

loading a command block into the command queue using the host processor;

executing the command block using the cryptographic processor; and

notifying the host processor that the command block has been executed.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 453

5. A method as recited in Claim 4, further comprising:

providing a read address for the command queue and a write address for the command queue;

wherein loading the command block into the command queue using the host processor comprises loading the command block into the command queue using the host processor beginning at the write address, and wherein executing the command block using the cryptographic processor comprises executing the command block using the cryptographic processor beginning at the read address.

6. A method as recited in Claim 5, wherein loading the command block into the command queue using the host processor beginning at the write address comprises:

determining if the write address plus an amount corresponding to a size of a single command block equals the read address; and

loading the command block into the command queue using the host processor beginning at the write address if the write address plus the amount corresponding to the size of the single command block does not equal the read address.

7. A method as recited in Claim 6, further comprising:

incrementing the write address by the amount corresponding to the size of a single command block using the host processor after loading the command block into the command queue using the host processor beginning at the write address if the write address plus the amount corresponding to the size of the single command block does not equal the read address.

8. A method as recited in Claim 5, wherein executing the command block using the cryptographic processor beginning at the read address comprises:

determining whether the read address is equal to the write address; and

executing the command block using the cryptographic processor beginning at the read address if the read address is not equal to the write address.

9. A method as recited in Claim 8, further comprising:

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 454

incrementing the read address by an amount corresponding to a size of a single command block using the cryptographic processor after executing the command block using the cryptographic processor beginning at the read address.

10. A method as recited in Claim 4, wherein notifying the host processor that the command block has been executed comprises invoking an interrupt using the cryptographic processor after executing the command block.

11. A method as recited in Claim 4, wherein notifying the host processor that the command block has been executed comprises updating a completion field in the command block using the cryptographic processor.

12. A method as recited in Claim 11, further comprising:

providing a periodic interrupt; and

reading the completion field using the host processor upon invocation of the periodic interrupt.

13. A method as recited in Claim 4, wherein notifying the host processor that the command block has been executed comprises:

setting a timer after loading the command block into the command queue using the host processor; and

checking whether the command block has been executed after expiration of the timer.

14. A method as recited in Claim 4, further comprising:

loading at least one operand from the command queue to the local memory;

performing at least one operation on the at least one operand to generate a result in the local memory; and

storing the result generated in the local memory in the command queue.

15. A method of operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that is coupled to the host processor and the system memory, the method comprising:

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 455

providing a command queue in the system memory;

loading a command block into the command queue using the host processor;

setting a value of an interrupt field in the command block to request an interrupt when the command block has been executed;

executing the command block using the cryptographic processor; and

invoking an interrupt using the cryptographic processor after executing the command block if the interrupt field in the command block is set to the value to request the interrupt.

16. A method as recited in Claim 15, further comprising:

storing error information in the command block that is associated with executing the command block using the cryptographic processor.

17. A method of operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that is coupled to the host processor and the system memory, the method comprising:

providing a command queue in the system memory;

loading a command block into the command queue using the host processor;

executing the command block using the cryptographic processor; and

updating a completion field in the command block using the cryptographic processor.

18. A method as recited in Claim 17, further comprising:

providing a periodic interrupt; and

reading the completion field using the host processor upon invocation of the periodic interrupt.

19. A method as recited in Claim 17, further comprising:

storing error information in the command block that is associated with executing the command block using the cryptographic processor.

20. A method of operating a data processing system that comprises a host processor, a system memory coupled to the host processor, and an adjunct processor

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 456

integrated circuit that is coupled to the host processor and the system memory, the method comprising:

providing a command queue in the system memory;

loading a command block into the command queue using the host processor, the command block comprising an input data field that contains input data;

performing an operation based on the input data using the adjunct processor to generate a result; and

storing the result in the input data field such that at least a portion of the input data is overwritten.

21. A method as recited in Claim 20, wherein the data processing system comprises a cryptographic data processing system, the adjunct processor integrated circuit comprises a cryptographic processor integrated circuit, and performing the operation based on the input data comprises:

performing a hash operation based on the input data using the cryptographic processor to generate a hash value.

22. A method as recited in Claim 21, wherein storing the result in the input data field comprises:

storing the hash value in the input data field such that the at least a portion of the input data is overwritten.

23. A method as recited in Claim 21, wherein the command block further comprises an input pointer field that contains an address in the system memory of an incoming packet and wherein performing the hash operation comprises:

performing the hash operation based on the input data and the incoming packet using the cryptographic processor to generate the hash value.

24. A method as recited in Claim 23, wherein the command block further comprises an output pointer field that contains an address in the system memory for storing a decrypted packet, the method further comprising:

decrypting the incoming packet using the cryptographic processor to generate the decrypted packet;

attaching the hash value to the decrypted packet; and

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 457

means for providing a periodic interrupt; and

means for reading the completion field using the host processor upon invocation of the periodic interrupt.

42. A cryptographic data processing system as recited in Claim 33, wherein the means for notifying the host processor that the command block has been executed comprises:

means for setting a timer after loading the command block into the command queue using the host processor; and

means for checking whether the command block has been executed after expiration of the timer.

43. A cryptographic data processing system as recited in Claim 33, further comprising:

means for loading at least one operand from the command queue to the local memory;

means for performing at least one operation on the at least one operand to generate a result in the local memory; and

means for storing the result generated in the local memory in the command queue.

44. A cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that is coupled to the host processor and the system memory, the system further comprising:

means for providing a command queue in the system memory;

means for loading a command block into the command queue using the host processor;

means for setting a value of an interrupt field in the command block to request an interrupt when the command block has been executed;

means for executing the command block using the cryptographic processor; and

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 458

storing the decrypted packet with the attached hash value at the address in the system memory contained in the output pointer field.

25. A method of operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the method comprising:

providing a command queue in the system memory;

providing a read address for the command queue and a write address for the command queue;

loading a random number sample into the command queue using the cryptographic processor beginning at the write address; and

reading the random number sample using the host processor beginning at the read address.

26. A method as recited in Claim 25, wherein loading the random number sample into the command queue using the cryptographic processor beginning at the write address comprises:

determining if the write address plus an amount corresponding to a size of a single random number sample equals the read address; and

loading the random number sample into the command queue using the cryptographic processor beginning at the write address if the write address plus the amount corresponding to the size of the single random number sample does not equal the read address.

27. A method as recited in Claim 26, further comprising:

incrementing the write address by the amount corresponding to the size of a single random number sample using the cryptographic processor after loading the random number sample into the command queue using the cryptographic processor beginning at the write address if the write address plus the amount corresponding to the size of the single random number sample does not equal the read address.

28. A method as recited in Claim 25, wherein reading the random number sample using the host processor beginning at the read address comprises:

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 459

determining whether the read address is equal to the write address; and

reading the random number sample using the host processor beginning at the read address if the read address is not equal to the write address.

29. A method as recited in Claim 28, further comprising:

incrementing the read address by an amount corresponding to a size of a single random number sample using the host processor after reading the random number sample using the host processor beginning at the read address.

30. A method of operating a data processing system that comprises a host processor, a system memory coupled to the host processor, and an adjunct processor integrated circuit that is coupled to the host processor and the system memory, the method comprising:

transferring information between the host processor and the adjunct processor using the system memory.

31. A cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the system further comprising:

means for loading at least one operand from the system memory to the local memory;

means for performing at least one operation on the at least one operand to generate a result in the local memory; and

means for storing the result generated in the local memory in the system memory.

32. A cryptographic data processing system as recited in Claim 31, wherein the means for performing the at least one operation, and the means for storing the result execute without interaction with the host processor.

33. A cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 460

processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the system further comprising:

means for providing a command queue in the system memory;

means for loading a command block into the command queue using the host processor;

means for executing the command block using the cryptographic processor; and

means for notifying the host processor that the command block has been executed.

34. A cryptographic data processing system as recited in Claim 33, further comprising:

means for providing a read address for the command queue and a write address for the command queue;

wherein the means for loading the command block into the command queue using the host processor comprises means for loading the command block into the command queue using the host processor beginning at the write address, and wherein the means for executing the command block using the cryptographic processor comprises means for executing the command block using the cryptographic processor beginning at the read address.

35. A cryptographic data processing system as recited in Claim 34, wherein the means for loading the command block into the command queue using the host processor beginning at the write address comprises:

means for determining if the write address plus an amount corresponding to a size of a single command block equals the read address; and

means for loading the command block into the command queue using the host processor beginning at the write address if the write address plus the amount corresponding to the size of the single command block does not equal the read address.

36. A cryptographic data processing system as recited in Claim 35, further comprising:

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 461

means for incrementing the write address by the amount corresponding to the size of a single command block using the host processor if the write address plus the amount corresponding to the size of the single command block does not equal the read address, the means for incrementing being responsive to the means for loading the command block into the command queue using the host processor beginning at the write address.

37. A cryptographic data processing system as recited in Claim 34, wherein the means for executing the command block using the cryptographic processor beginning at the read address comprises:

means for determining whether the read address is equal to the write address; and

means for executing the command block using the cryptographic processor beginning at the read address if the read address is not equal to the write address.

38. A cryptographic data processing system as recited in Claim 37, further comprising:

means for incrementing the read address by an amount corresponding to a size of a single command block using the cryptographic processor, the means for incrementing being responsive to the means for executing the command block using the cryptographic processor beginning at the read address.

39. A cryptographic data processing system as recited in Claim 33, wherein the means for notifying the host processor that the command block has been executed comprises means for invoking an interrupt using the cryptographic processor after executing the command block.

40. A cryptographic data processing system as recited in Claim 33, wherein the means for notifying the host processor that the command block has been executed comprises means for updating a completion field in the command block using the cryptographic processor.

41. A cryptographic data processing system as recited in Claim 40, further comprising:

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 462

is coupled to the host processor and the system memory, the system further comprising:

means for providing a command queue in the system memory;

means for loading a command block into the command queue using the host processor, the command block comprising an input data field that contains input data;

means for performing an operation based on the input data using the adjunct processor to generate a result; and

means for storing the result in the input data field such that at least a portion of the input data is overwritten.

50. A data processing system as recited in Claim 49, wherein the data processing system comprises a cryptographic data processing system, the adjunct processor integrated circuit comprises a cryptographic processor integrated circuit, and the means for performing the operation based on the input data comprises:

means for performing a hash operation based on the input data using the cryptographic processor to generate a hash value.

51. A data processing system as recited in Claim 50, wherein the means for storing the result in the input data field comprises:

means for storing the hash value in the input data field such that the at least a portion of the input data is overwritten.

52. A data processing system as recited in Claim 50, wherein the command block further comprises an input pointer field that contains an address in the system memory of an incoming packet and wherein the means for performing the hash operation comprises:

means for performing the hash operation based on the input data and the incoming packet using the cryptographic processor to generate the hash value.

53. A data processing system as recited in Claim 52, wherein the command block further comprises an output pointer field that contains an address in the system memory for storing a decrypted packet, the data processing system further comprising:

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 463

means for decrypting the incoming packet using the cryptographic processor to generate the decrypted packet;

means for attaching the hash value to the decrypted packet; and

means for storing the decrypted packet with the attached hash value at the address in the system memory contained in the output pointer field.

54. A cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the system further comprising:

means for providing a command queue in the system memory;

means for providing a read address for the command queue and a write address for the command queue;

means for loading a random number sample into the command queue using the cryptographic processor beginning at the write address; and

means for reading the random number sample using the host processor beginning at the read address.

55. A cryptographic data processing system as recited in Claim 54, wherein the means for loading the random number sample into the command queue using the cryptographic processor beginning at the write address comprises:

means for determining if the write address plus an amount corresponding to a size of a single random number sample equals the read address; and

means for loading the random number sample into the command queue using the cryptographic processor beginning at the write address if the write address plus the amount corresponding to the size of the single random number sample does not equal the read address.

56. A cryptographic data processing system as recited in Claim 55, further comprising:

means for incrementing the write address by the amount corresponding to the size of a single random number sample using the cryptographic processor if the write address plus the amount corresponding to the size of the single random number sample does not equal the read address, the means for incrementing being responsive

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 464

to the means for loading the random number sample into the command queue using the cryptographic processor beginning at the write address.

57. A cryptographic data processing system as recited in Claim 54, wherein the means for reading the random number sample using the host processor beginning at the read address comprises:

means for determining whether the read address is equal to the write address; and

means for reading the random number sample using the host processor beginning at the read address if the read address is not equal to the write address.

58. A cryptographic data processing system as recited in Claim 57, further comprising:

means for incrementing the read address by an amount corresponding to a size of a single random number sample using the host processor, the means for incrementing being responsive to the means for reading the random number sample using the host processor beginning at the read address.

59. A computer program product for operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the computer program product comprising:

a computer readable program medium having computer readable program code embodied therein, the computer readable program code comprising:

computer readable program code for loading at least one operand from the system memory to the local memory;

computer readable program code for performing at least one operation on the at least one operand to generate a result in the local memory; and

computer readable program code for storing the result generated in the local memory in the system memory.

60. A computer program product as recited in Claim 59, wherein the computer readable program code for performing the at least one operation, and the

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 465

computer readable program code for storing the result execute without interaction with the host processor.

61. A computer program product for operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the computer program product comprising:

a computer readable program medium having computer readable program code embodied therein, the computer readable program code comprising:

computer readable program code for providing a command queue in the system memory;

computer readable program code for loading a command block into the command queue using the host processor;

computer readable program code for executing the command block using the cryptographic processor; and

computer readable program code for notifying the host processor that the command block has been executed.

62. A computer program product as recited in Claim 61, further comprising:

computer readable program code for providing a read address for the command queue and a write address for the command queue;

wherein the computer readable program code for loading the command block into the command queue using the host processor comprises computer readable program code for loading the command block into the command queue using the host processor beginning at the write address, and wherein the computer readable program code for executing the command block using the cryptographic processor comprises computer readable program code for executing the command block using the cryptographic processor beginning at the read address.

63. A computer program product as recited in Claim 62, wherein the computer readable program code for loading the command block into the command queue using the host processor beginning at the write address comprises:

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 466

computer readable program code for determining if the write address plus an amount corresponding to a size of a single command block equals the read address; and

computer readable program code for loading the command block into the command queue using the host processor beginning at the write address if the write address plus the amount corresponding to the size of the single command block does not equal the read address.

64. A computer program product as recited in Claim 63, further comprising:

computer readable program code for incrementing the write address by the amount corresponding to the size of a single command block using the host processor if the write address plus the amount corresponding to the size of the single command block does not equal the read address, the computer readable program code for incrementing being responsive to the computer readable program code for loading the command block into the command queue using the host processor beginning at the write address.

65. A computer program product as recited in Claim 62, wherein the computer readable program code for executing the command block using the cryptographic processor beginning at the read address comprises:

computer readable program code for determining whether the read address is equal to the write address; and

computer readable program code for executing the command block using the cryptographic processor beginning at the read address if the read address is not equal to the write address.

66. A computer program product as recited in Claim 65, further comprising:

computer readable program code for incrementing the read address by an amount corresponding to a size of a single command block using the cryptographic processor, the computer readable program code for incrementing being responsive to the computer readable program code for executing the command block using the cryptographic processor beginning at the read address.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 467

67. A computer program product as recited in Claim 61, wherein the computer readable program code for notifying the host processor that the command block has been executed comprises computer readable program code for invoking an interrupt using the cryptographic processor after executing the command block.

68. A computer program product as recited in Claim 61, wherein the computer readable program code for notifying the host processor that the command block has been executed comprises computer readable program code for updating a completion field in the command block using the cryptographic processor.

69. A computer program product as recited in Claim 68, further comprising:

computer readable program code for providing a periodic interrupt; and

computer readable program code for reading the completion field using the host processor upon invocation of the periodic interrupt.

70. A method as recited in Claim 61, wherein the computer readable program code for notifying the host processor that the command block has been executed comprises:

computer readable program code for setting a timer after loading the command block into the command queue using the host processor; and

computer readable program code for checking whether the command block has been executed after expiration of the timer.

71. A computer program product as recited in Claim 61, further comprising:

computer readable program code for loading at least one operand from the command queue to the local memory;

computer readable program code for performing at least one operation on the at least one operand to generate a result in the local memory; and

computer readable program code for storing the result generated in the local memory in the command queue.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 468

72. A computer program product for operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that is coupled to the host processor and the system memory, the computer program product comprising:

a computer readable program medium having computer readable program code embodied therein, the computer readable program code comprising:

computer readable program code for providing a command queue in the system memory;

computer readable program code for loading a command block into the command queue using the host processor;

computer readable program code for setting a value of an interrupt field in the command block to request an interrupt when the command block has been executed;

computer readable program code for executing the command block using the cryptographic processor; and

computer readable program code for invoking an interrupt using the cryptographic processor after executing the command block if the interrupt field in the command block is set to the value to request the interrupt.

73. A computer program product as recited in Claim 72, further comprising:

computer readable program code for storing error information in the command block that is associated with executing the command block using the cryptographic processor.

74. A computer program product for operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that is coupled to the host processor and the system memory, the computer program product comprising:

a computer readable program medium having computer readable program code embodied therein, the computer readable program code comprising:

computer readable program code for providing a command queue in the system memory;

computer readable program code for loading a command block into the command queue using the host processor;

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 469

computer readable program code for executing the command block using the cryptographic processor; and

computer readable program code for updating a completion field in the command block using the cryptographic processor.

75. A computer program product as recited in Claim 74, further comprising:

computer readable program code for providing a periodic interrupt; and

computer readable program code for reading the completion field using the host processor upon invocation of the periodic interrupt.

76. A computer program product as recited in Claim 74, further comprising:

computer readable program code for storing error information in the command block that is associated with executing the command block using the cryptographic processor.

77. A computer program product for operating a data processing system that comprises a host processor, a system memory coupled to the host processor, and an adjunct processor integrated circuit that is coupled to the host processor and the system memory, the computer program product comprising:

a computer readable program medium having computer readable program code embodied therein, the computer readable program code comprising:

computer readable program code for providing a command queue in the system memory;

computer readable program code for loading a command block into the command queue using the host processor, the command block comprising an input data field that contains input data;

computer readable program code for performing an operation based on the input data using the adjunct processor to generate a result; and

computer readable program code for storing the result in the input data field such that at least a portion of the input data is overwritten.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 470

78. A computer program product as recited in Claim 77, wherein the data processing system comprises a cryptographic data processing system, the adjunct processor integrated circuit comprises a cryptographic processor integrated circuit, and the computer readable program code for performing the operation based on the input data comprises:

computer readable program code for performing a hash operation based on the input data using the cryptographic processor to generate a hash value.

79. A computer program product as recited in Claim 78, wherein the computer readable program code for storing the result in the input data field comprises:

computer readable program code for storing the hash value in the input data field such that the at least a portion of the input data is overwritten.

80. A computer program product as recited in Claim 78, wherein the command block further comprises an input pointer field that contains an address in the system memory of an incoming packet and wherein the computer readable program code for performing the hash operation comprises:

computer readable program code for performing the hash operation based on the input data and the incoming packet using the cryptographic processor to generate the hash value.

81. A computer program product as recited in Claim 80, wherein the command block further comprises an output pointer field that contains an address in the system memory for storing a decrypted packet, the computer program product further comprising:

computer readable program code for decrypting the incoming packet using the cryptographic processor to generate the decrypted packet;

computer readable program code for attaching the hash value to the decrypted packet; and

computer readable program code for storing the decrypted packet with the attached hash value at the address in the system memory contained in the output pointer field.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 471

means for invoking an interrupt using the cryptographic processor after executing the command block if the interrupt field in the command block is set to the value to request the interrupt.

45. A cryptographic data processing system as recited in Claim 44, further comprising:

means for storing error information in the command block that is associated with executing the command block using the cryptographic processor.

46. A cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that is coupled to the host processor and the system memory, the system further comprising:

means for providing a command queue in the system memory;

means for loading a command block into the command queue using the host processor;

means for executing the command block using the cryptographic processor; and

means for updating a completion field in the command block using the cryptographic processor.

47. A cryptographic data processing system as recited in Claim 46, further comprising:

means for providing a periodic interrupt; and

means for reading the completion field using the host processor upon invocation of the periodic interrupt.

48. A cryptographic data processing system as recited in Claim 46, further comprising:

means for storing error information in the command block that is associated with executing the command block using the cryptographic processor.

49. A data processing system that comprises a host processor, a system memory coupled to the host processor, and an adjunct processor integrated circuit that

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 472

82. A computer program product for operating cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the computer program product comprising:

a computer readable program medium having computer readable program code embodied therein, the computer readable program code comprising:

computer readable program code for providing a command queue in the system memory;

computer readable program code for providing a read address for the command queue and a write address for the command queue;

computer readable program code for loading a random number sample into the command queue using the cryptographic processor beginning at the write address; and

computer readable program code for reading the random number sample using the host processor beginning at the read address.

83. A computer program product as recited in Claim 82, wherein the computer readable program code for loading the random number sample into the command queue using the cryptographic processor beginning at the write address comprises:

computer readable program code for determining if the write address plus an amount corresponding to a size of a single random number sample equals the read address; and

computer readable program code for loading the random number sample into the command queue using the cryptographic processor beginning at the write address if the write address plus the amount corresponding to the size of the single random number sample does not equal the read address.

84. A computer program product as recited in Claim 83, further comprising:

computer readable program code for incrementing the write address by the amount corresponding to the size of a single random number sample using the cryptographic processor if the write address plus the amount corresponding to the size of the single random number sample does not equal the read address, the computer

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 473

readable program code for incrementing being responsive to the computer readable program code for loading the random number sample into the command queue using the cryptographic processor beginning at the write address.

85. A computer program product as recited in Claim 82, wherein the computer readable program code for reading the random number sample using the host processor beginning at the read address comprises:

computer readable program code for determining whether the read address is equal to the write address; and

computer readable program code for reading the random number sample using the host processor beginning at the read address if the read address is not equal to the write address.

86. A computer program product as recited in Claim 85, further comprising:

computer readable program code for incrementing the read address by an amount corresponding to a size of a single random number sample using the host processor, the computer readable program code for incrementing being responsive to the computer readable program code for reading the random number sample using the host processor beginning at the read address.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 474

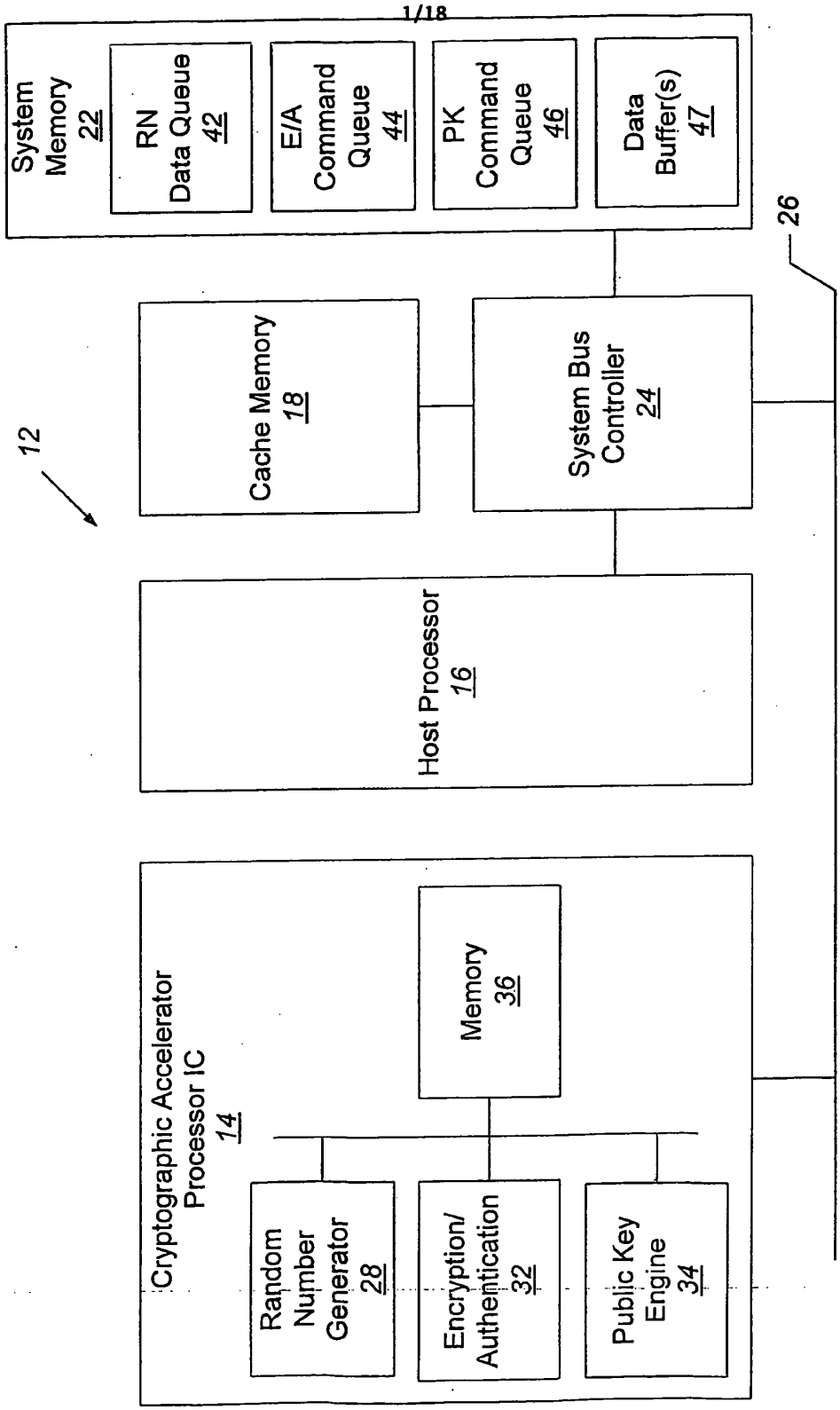

System Memory 22
RN Data Queue 42
E/A Command Queue 44
PK Command Queue 46
Data Buffer(s) 47
26
12
Cache Memory 18
System Bus Controller 24
Host Processor 16
Cryptographic Accelerator Processor IC 14
Memory 36
Random Number Generator 28
Encryption/ Authentication 32
Public Key Engine 34

FIG. 1

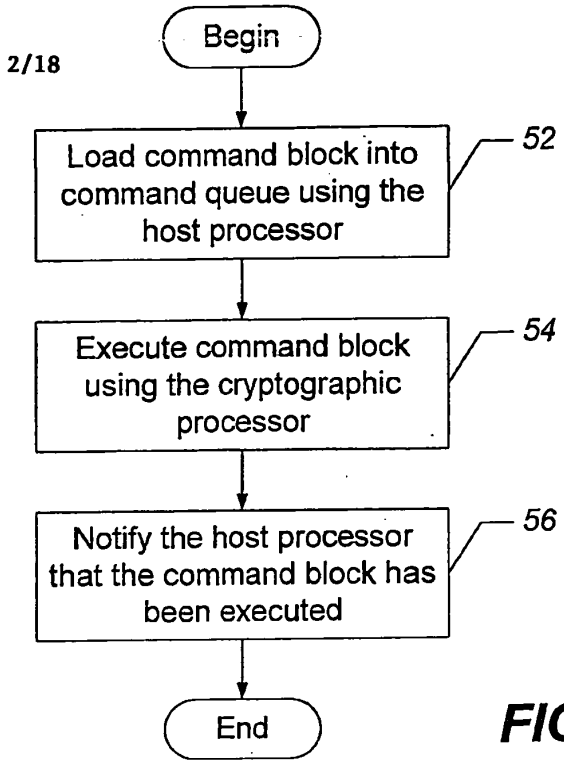

Begin
Load command block into command queue using the host processor
52
Execute command block using the cryptographic processor
54
Notify the host processor that the command block has been executed
56
End

FIG. 2

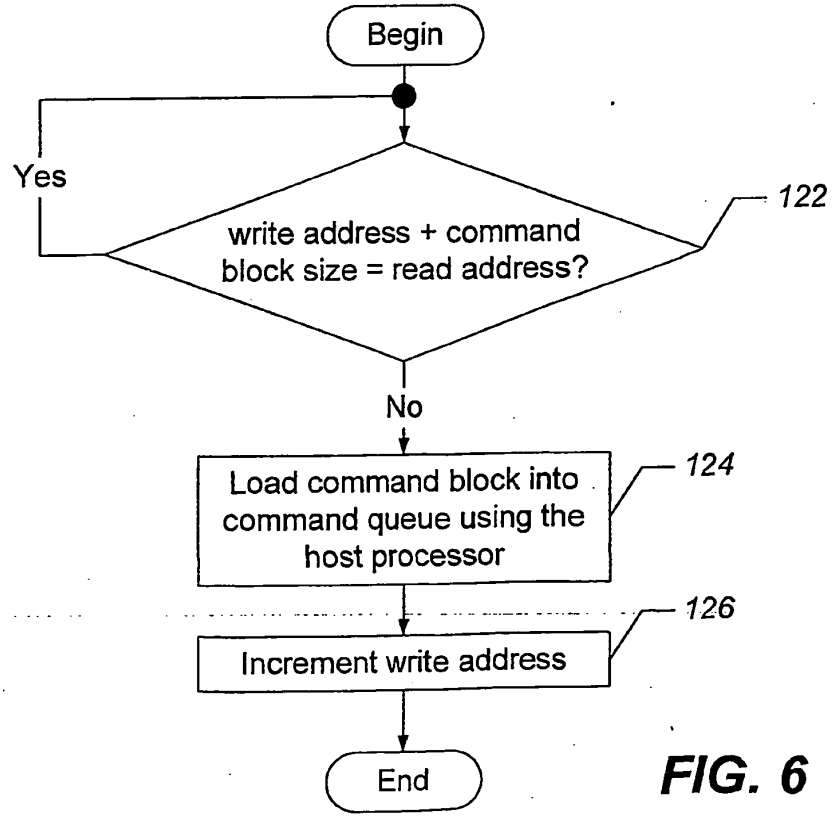

Begin
Yes
write address + command block size = read address?
122
No
Load command block into command queue using the host processor
124
Increment write address
126
End

FIG. 6

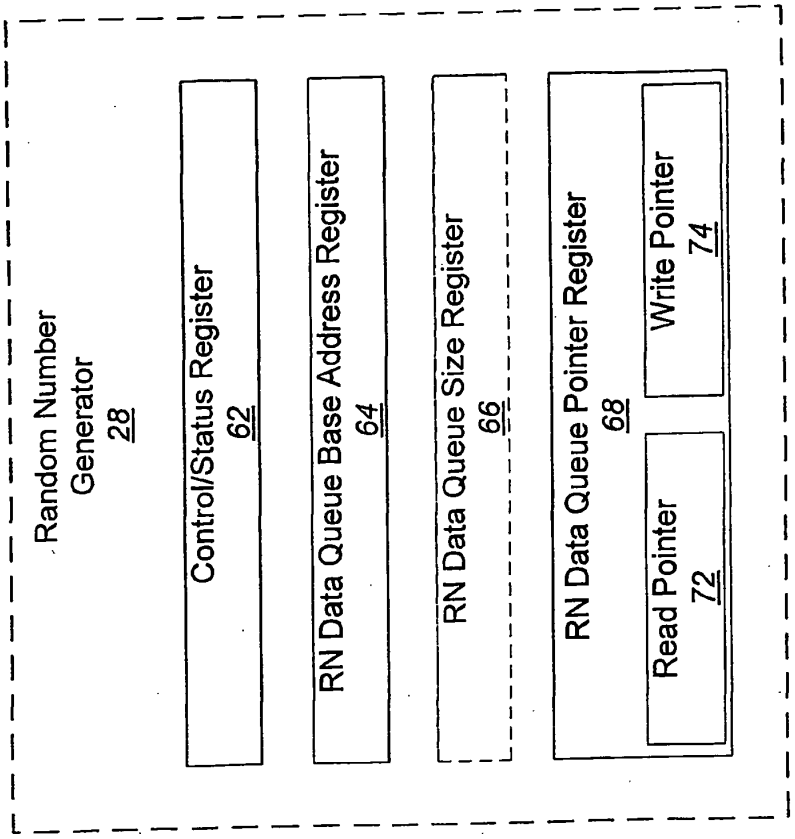

Random Number Generator 28
Control/Status Register 62
RN Data Queue Base Address Register 64
RN Data Queue Size Register 66
RN Data Queue Pointer Register 68
Read Pointer 72
Write Pointer 74

FIG. 3

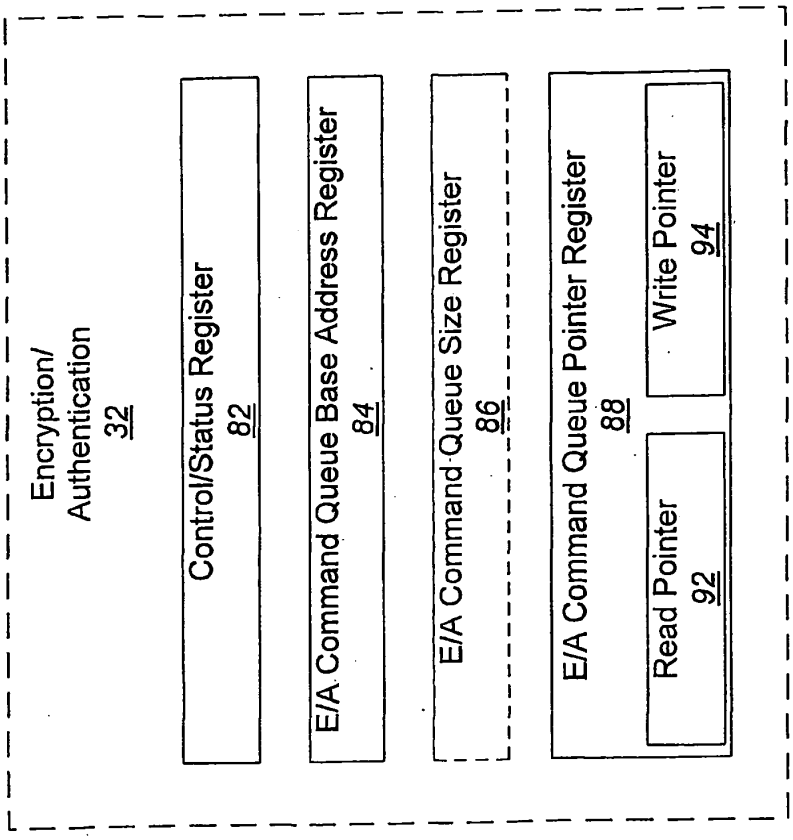

Encryption/
Authentication
32
Control/Status Register
82
E/A Command Queue Base Address Register
84
E/A Command Queue Size Register
86
E/A Command Queue Pointer Register
88
Read Pointer
92
Write Pointer
94

FIG. 4

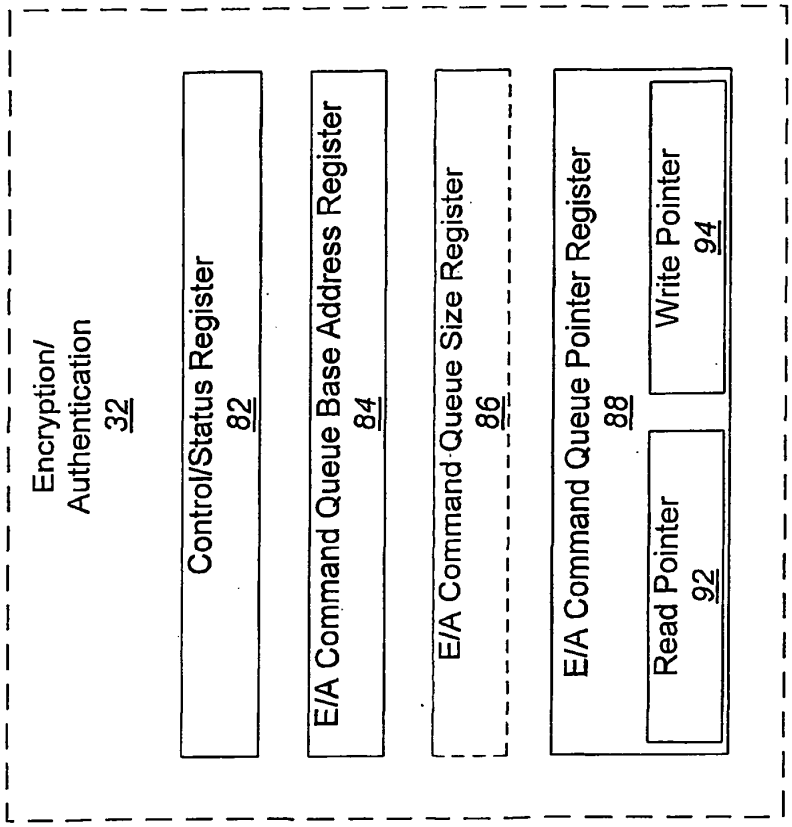

Encryption/
Authentication
32
Control/Status Register
82
E/A Command Queue Base Address Register
84
E/A Command Queue Size Register
86
E/A Command Queue Pointer Register
88
Read Pointer
92
Write Pointer
94

FIG. 4

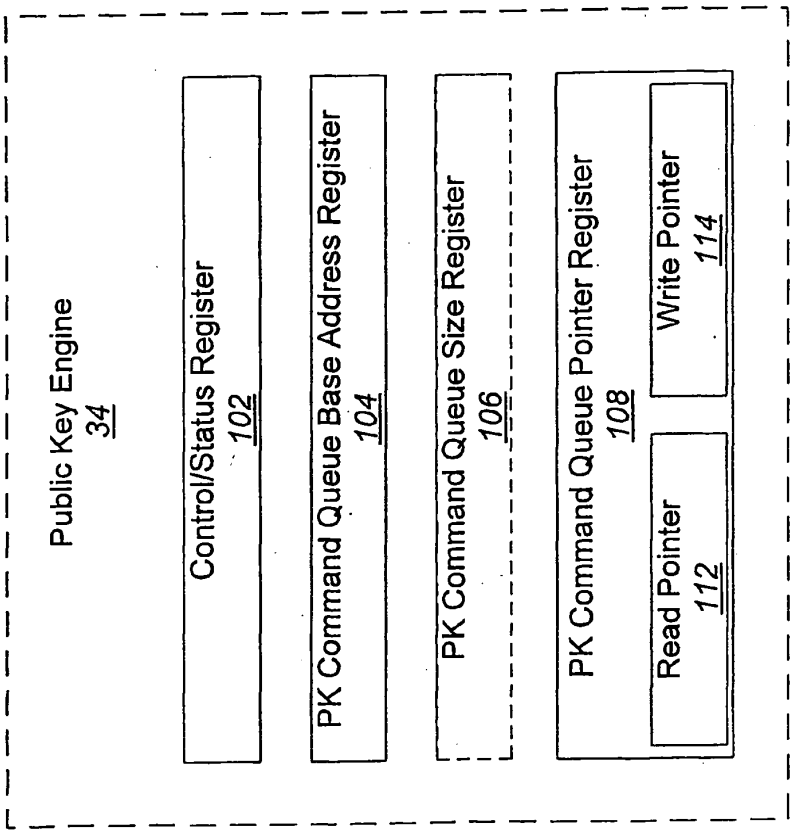

Public Key Engine
34
Control/Status Register
102
PK Command Queue Base Address Register
104
PK Command Queue Size Register
106
PK Command Queue Pointer Register
108
Read Pointer
112
Write Pointer
114

FIG. 5

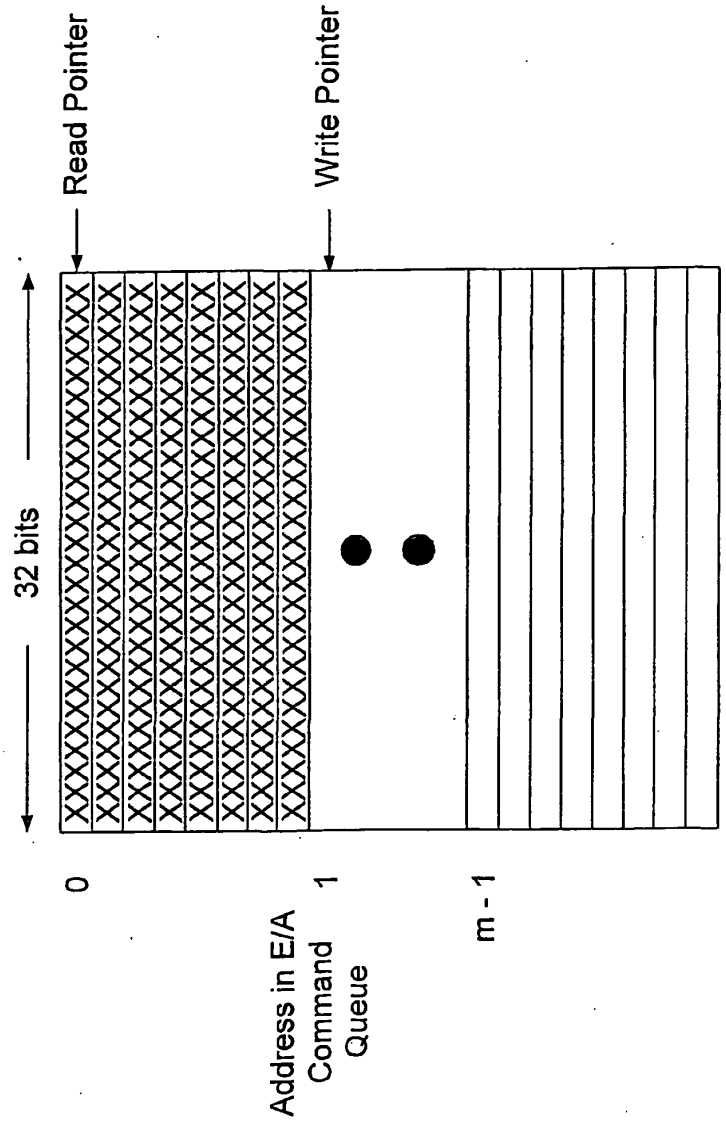

Read Pointer
Write Pointer
32 bits
0
1
m - 1
Address in E/A Command Queue

FIG. 7

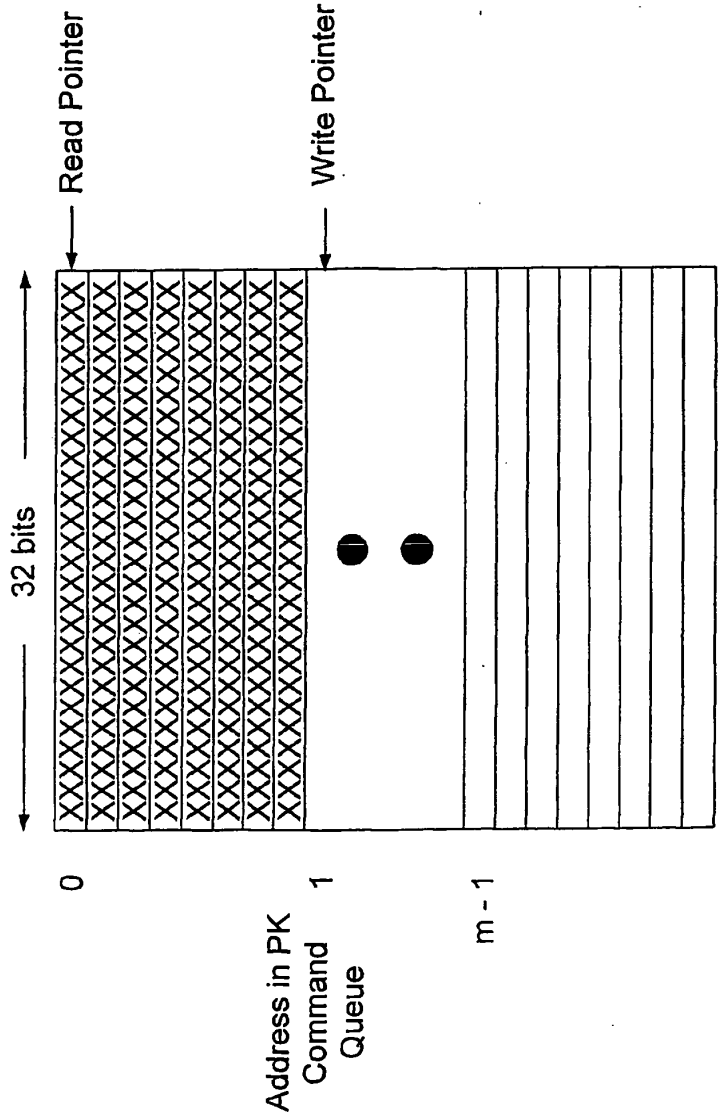

Read Pointer
Write Pointer
32 bits
0
1
m - 1
Address in PK Command Queue

FIG. 8

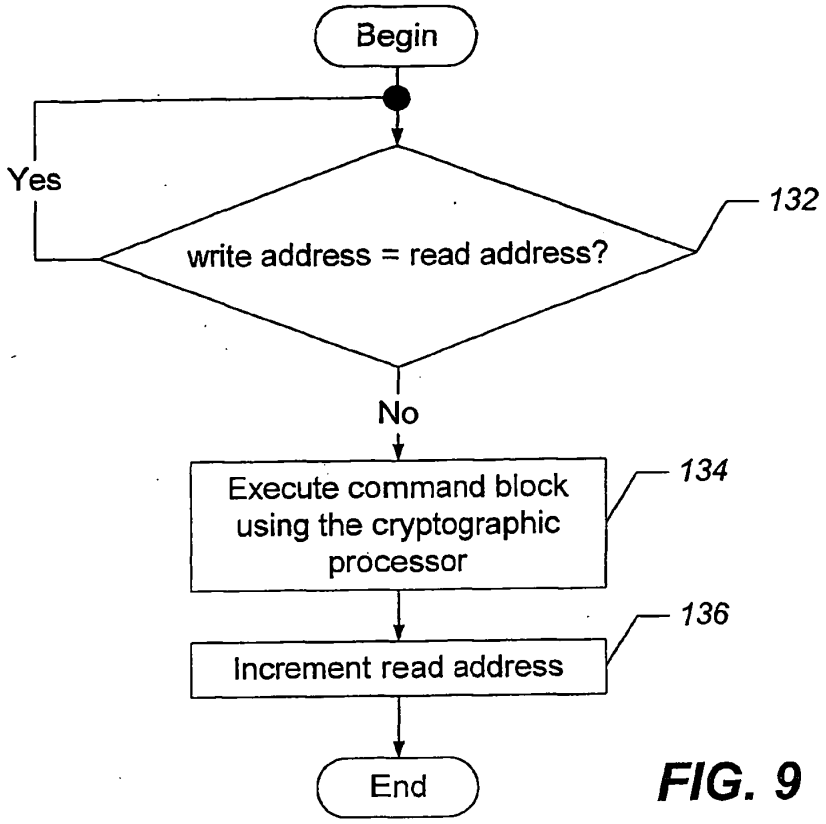

Begin
Yes
write address = read address?
132
No
Execute command block using the cryptographic processor
134
Increment read address
136
End

FIG. 9

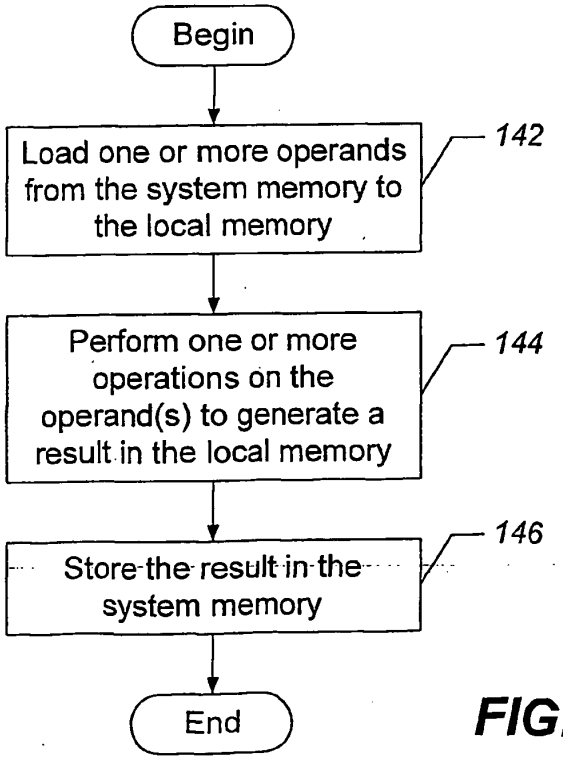

Begin
Load one or more operands from the system memory to the local memory
142
Perform one or more operations on the operand(s) to generate a result in the local memory
144
Store the result in the system memory
146
End

FIG. 10

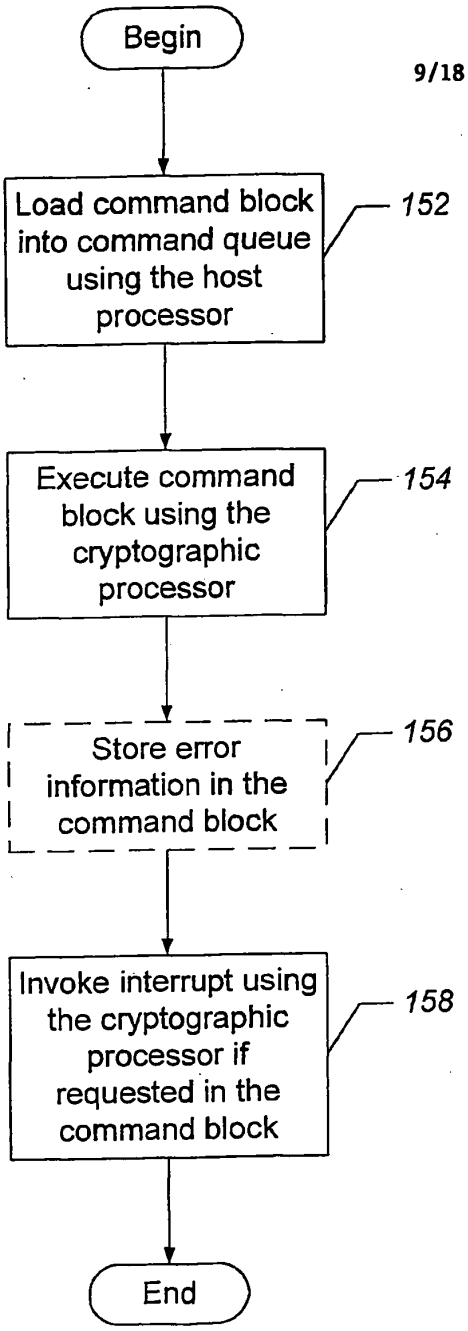

Begin
Load command block into command queue using the host processor
152
Execute command block using the cryptographic processor
154
Store error information in the command block
156
Invoke interrupt using the cryptographic processor if requested in the command block
158
End

FIG. 11

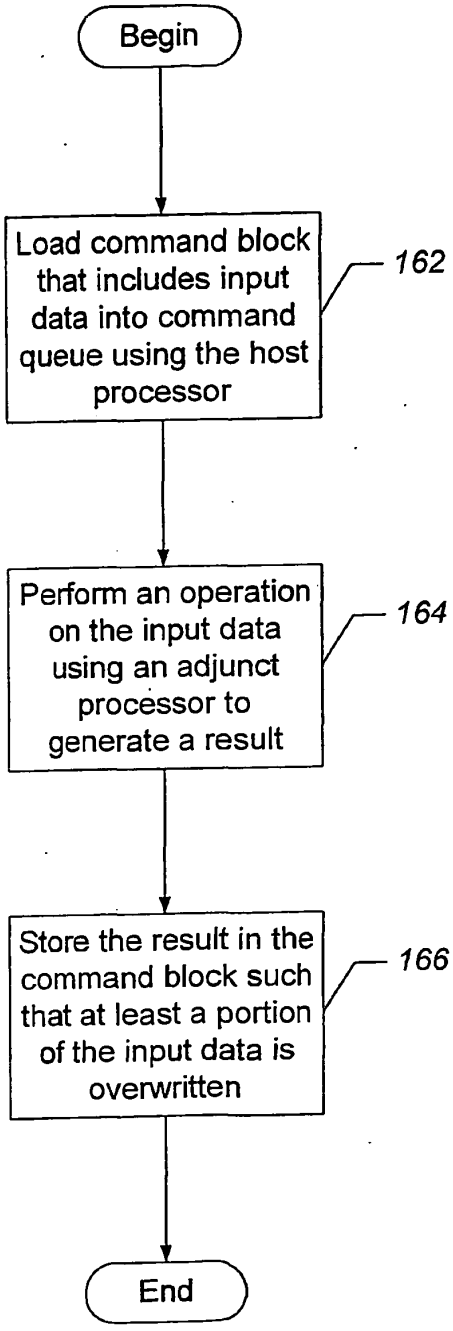

Begin
Load command block that includes input data into command queue using the host processor
162
Perform an operation on the input data using an adjunct processor to generate a result
164
Store the result in the command block such that at least a portion of the input data is overwritten
166
End

FIG. 13

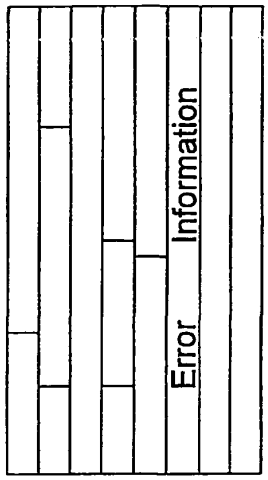


FIG. 12B

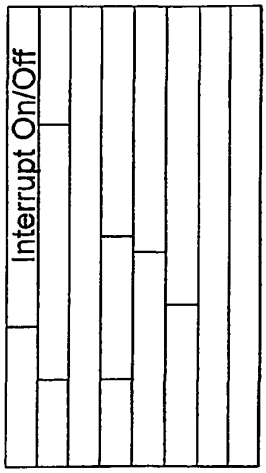


FIG. 12A

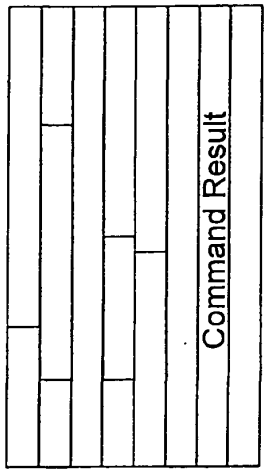


FIG. 12D

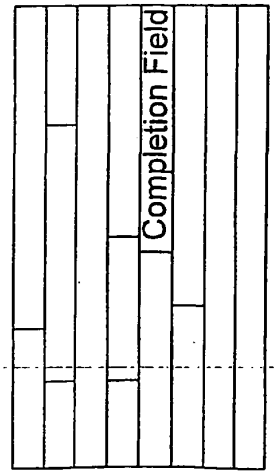


FIG. 12C

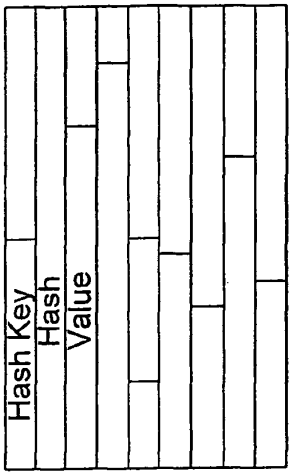


FIG. 14B

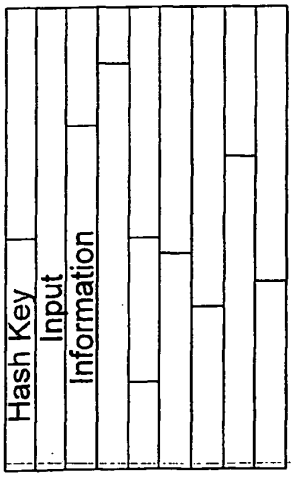


FIG. 14A

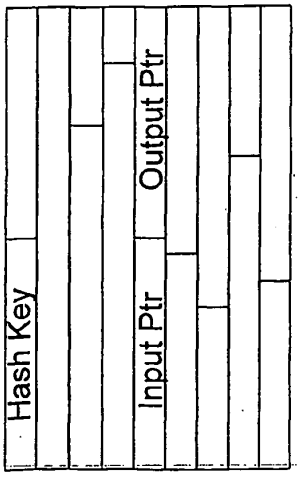


FIG. 15

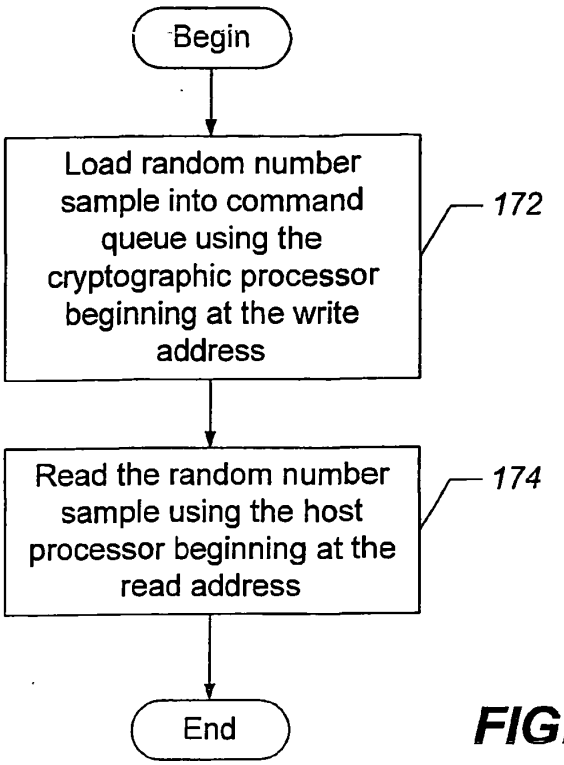

Begin
Load random number sample into command queue using the cryptographic processor beginning at the write address
172
Read the random number sample using the host processor beginning at the read address
174
End

FIG. 16

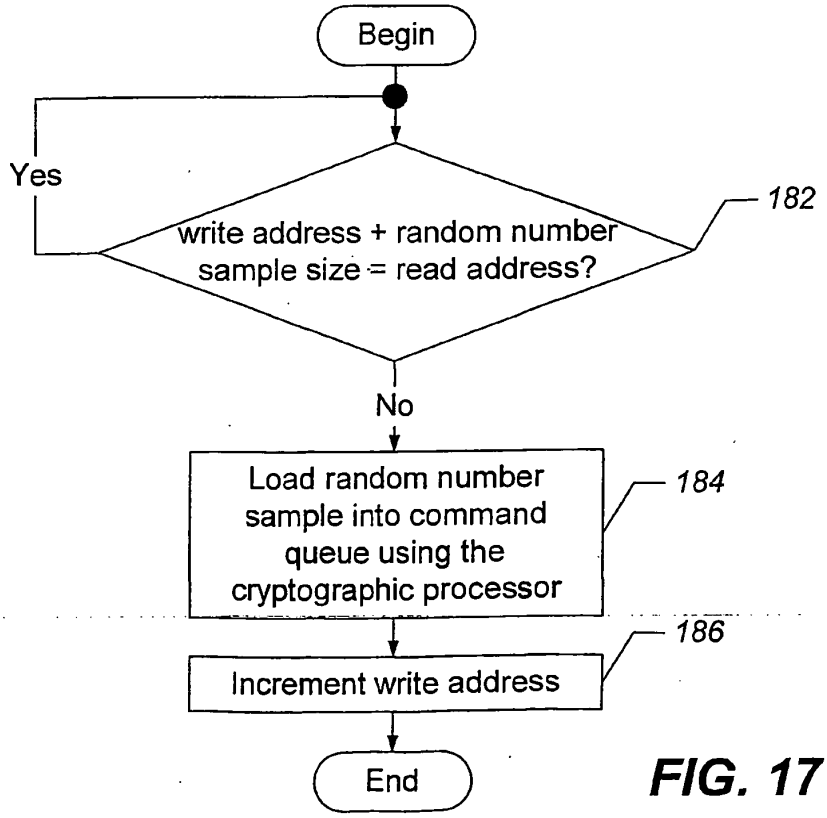

Begin
Yes
write address + random number sample size = read address?
182
No
Load random number sample into command queue using the cryptographic processor
184
Increment write address
186
End

FIG. 17

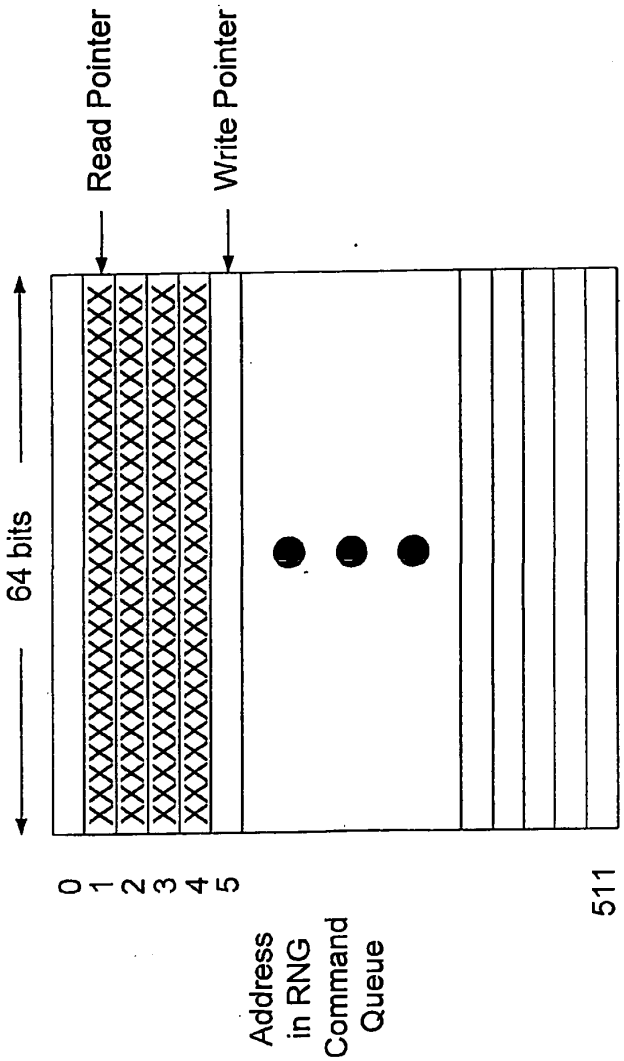

Read Pointer
Write Pointer
64 bits
0
1
2
3
4
5
511
Address in RNG Command Queue

FIG. 18

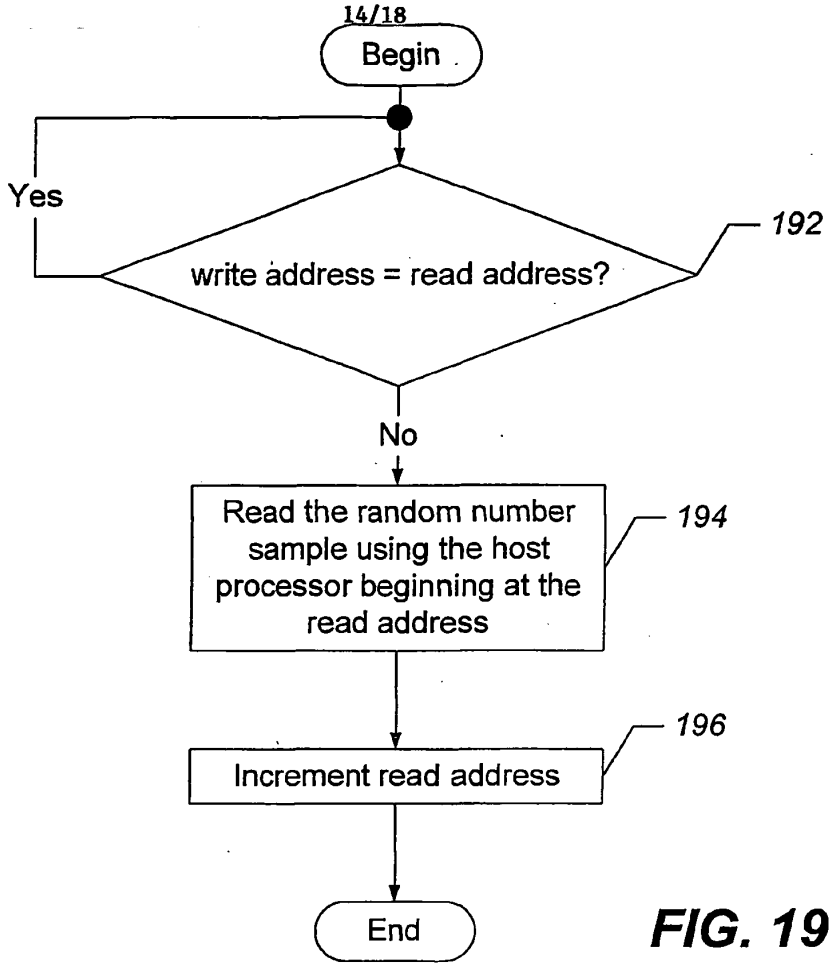

Begin
Yes
write address = read address?
192
No
Read the random number sample using the host processor beginning at the read address
194
Increment read address
196
End

FIG. 19

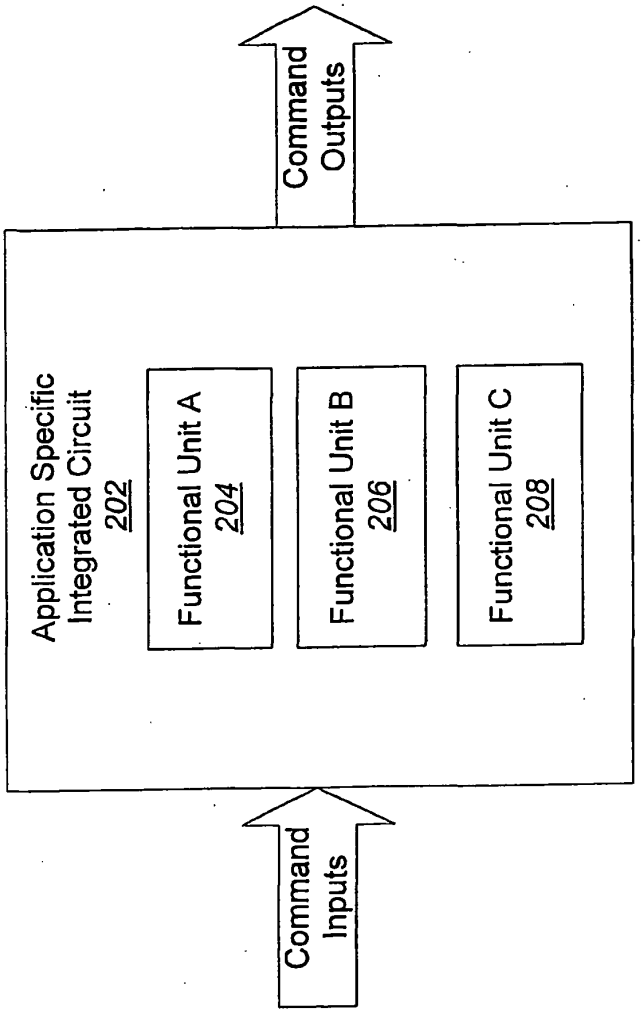


FIG. 20
(Prior Art)

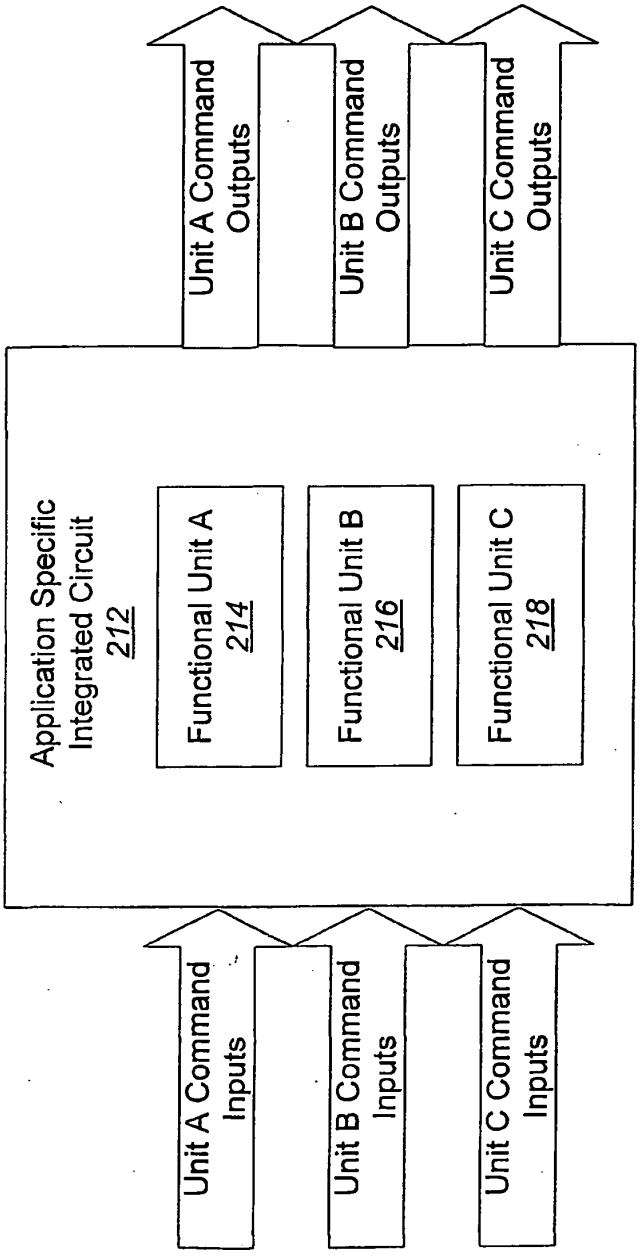

Unit A Command Outputs
Unit B Command Outputs
Unit C Command Outputs
Application Specific Integrated Circuit 212
Functional Unit A 214
Functional Unit B 216
Functional Unit C 218
Unit A Command Inputs
Unit B Command Inputs
Unit C Command Inputs

FIG. 21

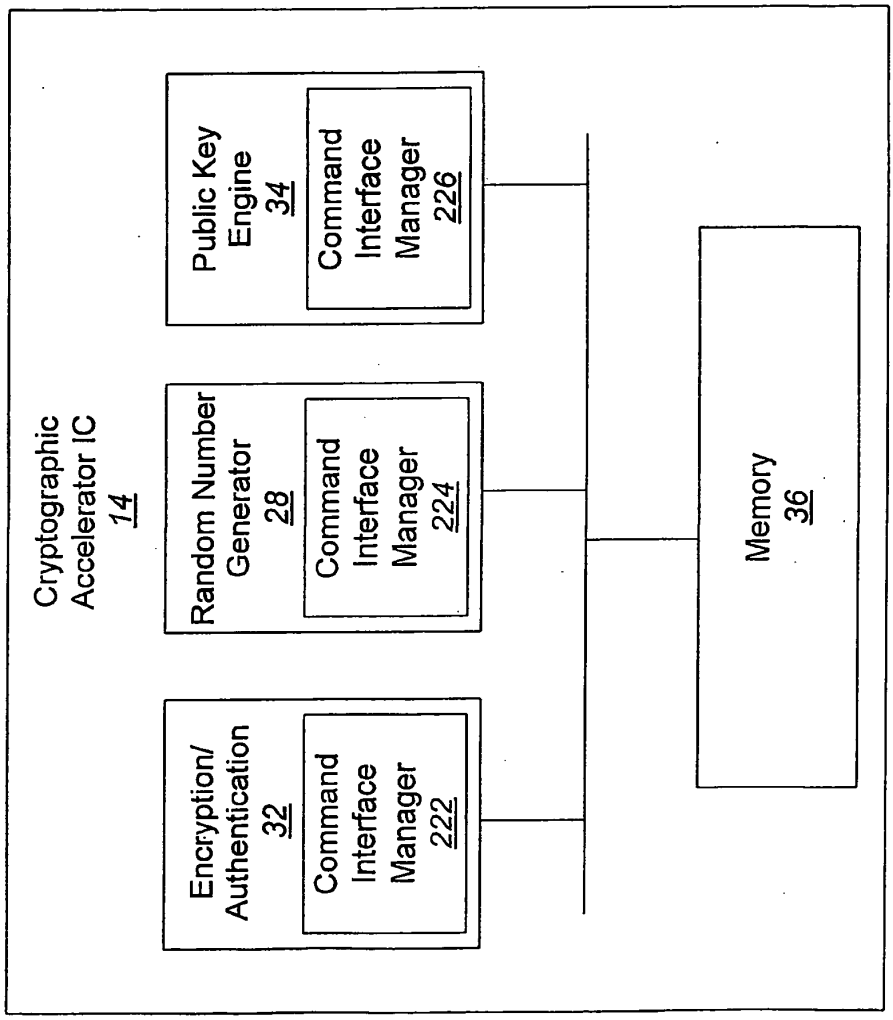

Cryptographic Accelerator IC 14
Encryption/ Authentication 32
Command Interface Manager 222
Random Number Generator 28
Command Interface Manager 224
Public Key Engine 34
Command Interface Manager 226
Memory 36

FIG. 22

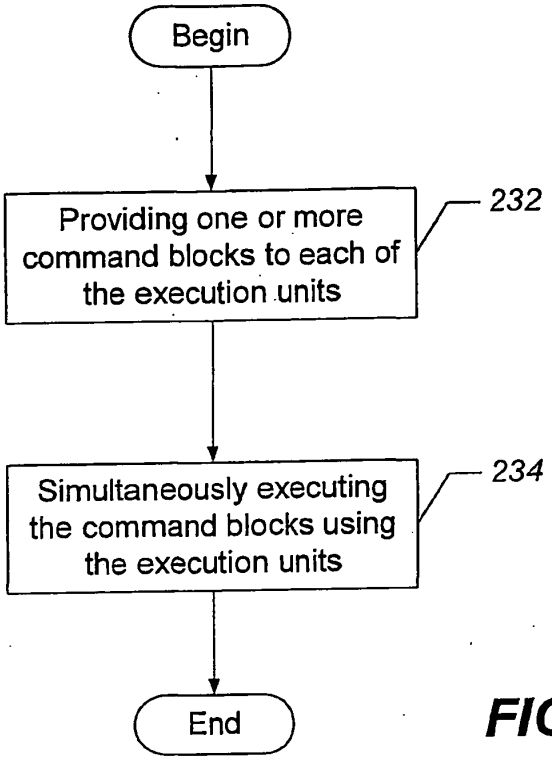

Begin
Providing one or more command blocks to each of the execution units
232
Simultaneously executing the command blocks using the execution units
234
End

FIG. 23

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
15 November 2001 (15.11.2001)

PCT

(10) International Publication Number
WO 01/086430 A3

(51) International Patent Classification[7]: G06F 9/38

(21) International Application Number: PCT/US01/15180

(22) International Filing Date: 10 May 2001 (10.05.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/203,465 11 May 2000 (11.05.2000) US
60/203,409 11 May 2000 (11.05.2000) US

(71) Applicants *(for all designated States except US)*: **NETOCTAVE, INC.** [US/US]; 507 Airport Boulevard, Suite 111, Morrisville, NC 27560 (US). **HANNA, Michael** [US/US]; 522 N. Person Street, Raleigh, NC 27604 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **BLAKER, David** [US/US]; 109 Hogan Glen Court, Chapel Hill, NC 27516 (US). **SAVARDA, Raymond** [US/US]; 4224 Sancroft Drive, Apex, NC 27502 (US).

(74) Agent: **MYERS BIGEL SIBLEY & SAJOVEC**; P.O. Box 37428, Raleigh, NC 27627 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

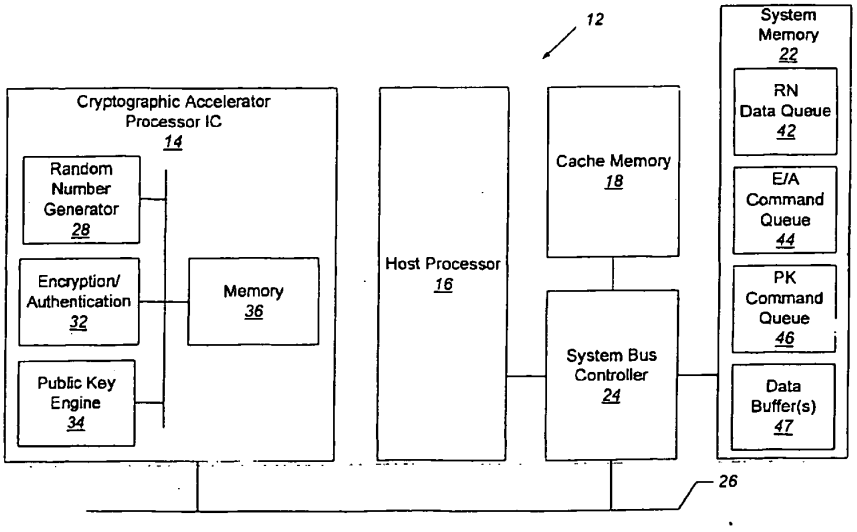
(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

(88) Date of publication of the international search report:
17 October 2002

[Continued on next page]

(54) Title: CRYPTOGRAPHIC DATA PROCESSING SYSTEMS, COMPUTER PROGRAMS, AND METHODS OF OPERATING SAME



(57) Abstract: Embodiments of cryptographic data processing systems, computer program products, and methods of operating same are provided in which system memory is used to transfer information between a host processor and an adjunct processor.

WO 01/086430 A3

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 01/15180

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/38

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 560 020 A (IBM) 15 September 1993 (1993-09-15) the whole document	1-3, 30-32, 59,60
X	EP 0 668 560 A (IBM) 23 August 1995 (1995-08-23) column 7, line 9 - line 23 column 11, line 55 - line 4 column 12, line 24 - line 38	1-3, 30-32, 59,60

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
19 April 2002	10/05/2002

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Moraiti, M

Form PCT/ISA/210 (second sheet) (July 1992)

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 496

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 01/15180

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 945 788 A (TEXAS INSTRUMENTS INC) 29 September 1999 (1999-09-29)	4-10,14, 20, 33-39, 43,49, 59-67, 71,77
A	abstract	22, 25-29, 54-58, 82-86
Y	column 4, line 39 -column 6, line 55 column 13, line 16 - line 33	40,68
P,X	US 6 075 546 A (HUSSAIN ZAHID S ET AL) 13 June 2000 (2000-06-13) abstract	15,17, 44,46, 72,74
P,Y	column 2, line 7 -column 3, line 4	40,68
X	US 5 706 489 A (CHI CHI-HUNG ET AL) 6 January 1998 (1998-01-06) abstract column 4, line 48 - line 55; figures 2,3 column 5, line 38 -column 6, line 3	11-13, 17,18, 33, 40-42, 46,47, 68-70, 74,75
A	US 4 763 242 A (LEE RUBY B ET AL) 9 August 1988 (1988-08-09) column 7, line 1 - line 23	16,19, 45,48, 73,76
P,A	WO 01 05086 A (BROADCOM CORP ;LAW PATRICK (US); LIN DERRICK (US); TARDO JOE (US);) 18 January 2001 (2001-01-18)	
A	EP 0 395 958 A (HITACHI LTD) 7 November 1990 (1990-11-07)	

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 497

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 01/15180

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

see additional sheet

1. ☒ As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☒ No protest accompanied the payment of additional search fees.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 498

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. Claims: 1-3,30-32,59,60

   Trasferring information result between a host processor and an adjunct processor

2. Claims: 4-19,33-48,61-76

   Notifying host processor that command block has been executed by the co-processor

3. Claims: 20-24,49-52,77-86

   Overwriting result

4. Claims: 25-29,54-58,82-86

   Command queue with read and write address

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 01/15180

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
EP 0560020	A	15-09-1993	EP	0560020 A2	15-09-1993
			JP	6083578 A	25-03-1994
EP 0668560	A	23-08-1995	CA	2137488 A1	19-08-1995
			EP	0668560 A2	23-08-1995
			JP	2633488 B2	23-07-1997
			JP	7239783 A	12-09-1995
			US	5655146 A	05-08-1997
EP 0945788	A	29-09-1999	EP	0945788 A2	29-09-1999
			JP	11272631 A	08-10-1999
			US	6256724 B1	03-07-2001
US 6075546	A	13-06-2000	US	6331857 B1	18-12-2001
US 5706489	A	06-01-1998	NONE		
US 4763242	A	09-08-1988	JP	2685747 B2	03-12-1997
			JP	62097039 A	06-05-1987
WO 0105086	A	18-01-2001	AU	6342200 A	30-01-2001
			AU	6342500 A	30-01-2001
			AU	7051400 A	30-01-2001
			EP	1192781 A2	03-04-2002
			EP	1192782 A2	03-04-2002
			EP	1192783 A2	03-04-2002
			WO	0105086 A2	18-01-2001
			WO	0105089 A2	18-01-2001
			WO	0105087 A2	18-01-2001
EP 0395958	A	07-11-1990	JP	2278475 A	14-11-1990
			DE	69032172 D1	30-04-1998
			DE	69032172 T2	29-10-1998
			DE	69033846 D1	06-12-2001
			DE	69033846 T2	11-04-2002
			EP	1158462 A1	28-11-2001
			EP	0395958 A2	07-11-1990
			EP	0807900 A1	19-11-1997
			US	6229543 B1	08-05-2001

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record.

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☑ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☑ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.

OIPE IAP28
OCT 0 4 2006
PATENT & TRADEMARK OFFICE

1FW 2143

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of Laurence B. Boucher, et al.		Ser. No:	10/260,878
Filing Date:	September 27, 2002	Examiner:	J. B. Dennison
Atty. Docket No:	ALA-006E	GAU:	2143

For: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

October 2, 2006

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Reply to Requirement for Information under 35 CFR 1.105

Sir:

The following Remarks are in response to a Requirement for Information dated August 9, 2006.

Remarks

A. The Requirement for Information under 35 CFR 1.105

The Requirement for Information under 35 CFR 1.105 (hereinafter "the Requirement") states, on page 2:

> Applicant and the assignee of this application are required under 37 CFR 1.105 to provide the following information that the examiner has determined is reasonably necessary to the examination of this application.
>
> The Applicant is requested to identify any known examples, products, and/or prior art in which, when processing of packets occurs, for each packet, the network layer header and the transport layer header are validated with an interrupt dividing the processing of the network layer and the transport layer header. The Applicant is requested to explain the meaning of an interrupt with relevance to how it is used in the claimed invention. The Examiner has determined that this identification of examples, products, and/or prior art along with an explanation in reference to the claimed invention is pertinent to the issue of patentability in this case.
>
> The Examiner is authorized to require the submission of this information since the Examiner has determined that such information is relevant to the patentability of the claimed invention. See *Star Fruits S.N.C. v. United States (Fed. Cir. 2005)* (*"So long as there is some legitimate reason for seeking the information under section 1.105, the applicant has a duty to respond...The Office is authorized under section 1.105 to require any information that is either relevant to patentability under any nonfrivilous legal theory, or is reasonably calculated to lead to such relevant information"*).

B. Applicants' Reply

Although applicants disagree with some of the Examiner's requests and their underlying reasoning and legal theory, applicants nonetheless respond to the best of their ability and with candor to the Requirement.

1. Identification of any known examples, products, and/or prior art in which, when processing of packets occurs, for each packet, the network layer header and the transport layer header are validated with an interrupt dividing the processing of the network layer and the transport layer header.

Applicants note that much of the information sought by the Examiner has already been provided in the present application and so applicants begin by identifying, for the Examiner's benefit, portions of the application that he may find relevant to the questions

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 503

he has posed. In that regard, applicants note that the present application claims the benefit of and incorporates by reference the Provisional Application entitled "INTELLIGENT NETWORK INTERFACE CARD AND SYSTEM FOR PROTOCOL PROCESSING," Serial No. 60/061,809, filed on October 14, 1997 (hereinafter "the '809 app."). Because the Examiner may not have ready access to that application, applicants are enclosing a copy of that application with this Reply.

In response to the Examiner's request "to identify any known examples, products, and/or prior art in which, when processing of packets occurs, for each packet, the network layer header and the transport layer header are validated with an interrupt dividing the processing of the network layer and the transport layer header," applicants direct the Examiner's attention to the following sections of the '809 app. beginning on page 2, line 34 – page 3, line 9 of the '809 app., which states:

> **1.3 Too Many Interrupts**
>
> A 64k SMB request (write or read-reply) is typically made up of 44 TCP segments when running over Ethernet (1500 byte MTU). Each of these segments may result in an interrupt to the CPU. Furthermore, since TCP must acknowledge all of this incoming data, it's possible to get another 44 transmit-complete interrupts as a result of sending out the TCP acknowledgements. While this is possible, it is not terribly likely. Delayed ACK timers allow us to acknowledge more than one segment at a time. And delays in interrupt processing may mean that we are able to process more than one incoming network frame per interrupt. Nevertheless, even if we assume 4 incoming frames per input, and an acknowledgement for every 2 segments (as is typical per the ACK-every-other-segment property of TCP), we are still left with 33 interrupts per 64k SMB request.
>
> Interrupts tend to be very costly to the system. Often when a system is interrupted, important information must be flushed or invalidated from the system cache so that the interrupt routine instructions, and needed data can be pulled into the cache. Since the CPU will return to its prior location after the interrupt, it is likely that the information flushed from the cache will immediately need to be pulled back into the cache.
>
> What's more, interrupts force a pipeline flush in today's advanced processors. While the processor pipeline is an extremely efficient way of improving CPU performance, it can be expensive to get going after it has been flushed.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 504

Finally, each of these interrupts results in expensive register accesses across the peripheral bus (PCI). This is discussed more in the following section.

The '809 app. continues to describe the problem of interrupts in the prior art, stating on page 3, lines 11-31:

> **1.4 Inefficient Use of the Peripheral Bus (PCI)**
>
> We noted earlier that when the CPU has to access system memory, it may be stalled for several hundred nanoseconds. When it has to read from PCI, it may be stalled for many microseconds. This happens every time the CPU takes an interrupt from a standard NIC. The first thing the CPU must do when it receives one of these interrupts is to read the NIC Interrupt Status Register (ISR) from PCI to determine the cause of the interrupt. The most troubling thing about this is that since interrupt lines are shared on PC-based systems, we may have to perform this expensive PCI read even when the interrupt is not meant for us!
>
> There are other peripheral bus inefficiencies as well. Typical NICs operate using descriptor rings. When a frame arrives, the NIC reads a receive descriptor from system memory to determine where to place the data. Once the data has been moved to main memory, the descriptor is then written back out to system memory with status about the received frame. Transmit operates in a similar fashion. The CPU must notify that NIC that it has a new transmit. The NIC will read the descriptor to locate the data, read the data itself, and then write the descriptor back with status about the send. Typically on transmits the NIC will then read the next expected descriptor to see if any more data needs to be sent. In short, each receive or transmit frame results in 3 or 4 separate PCI reads or writes (not counting the status register read).

The '809 app. on page 3, lines 43-44 of the '809 app., under the heading "**Summary of the Invention**", notes the reduction in interrupts from the prior art by stating:

> 3. Interrupts are reduced to as little as 4 interrupts per 64k SMB read and 2 per 64k SMB write.

Page 9, lines 4-15 of the '809 app. further explains this reduction in interrupts from the prior art by stating:

> **2.2.3 Affect on interrupts**
>
> Note that when we receive a large SMB transaction, for example, that there are two interactions between the INIC and the host. The first in

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 505

which the INIC indicates a small amount of the transaction to the host, and the second in which the host provides the memory location(s) in which the INIC places the remainder of the data. This results in only two interrupts from the INIC. The first when it indicates the small amount of data and the second after it has finished filling in the host memory given to it. A drastic reduction from the 33/64k SMB request that we estimate at the beginning of this section.

On transmit, we actually only receive a single interrupt when the send command that has been given to the INIC completes.

2.2.4 Transport Layer Interface Summary

Having now established our interaction with Microsoft's TDI interface, we have achieved our goal of landing most of our data directly into its final destination in host memory. We have also managed to transmit all data from its original location on host memory. And finally, we have reduced our interrupts to 2 per 64k SMB read and 1 per 64k SMB write. The only thing that remains in our list of objectives is to design an efficient host (PCI) interface.

The '809 app. further describes the interrupts in the prior art, stating on page 9, line 38 – page 10, line 2:

2.3.1.1 **Memory-based status register**

The only PCI read that is required by most NICs is the read of the interrupt status register. This register gives the host CPU information about what event has caused an interrupt (if any). In the design of our INIC we have elected to place this necessary status register into host memory. Thus, when an event occurs on the INIC, it writes the status register to an agreed upon location in host memory. The corresponding driver on the host reads this local register to determine the cause of the interrupt. The interrupt lines are held high until the host clears the interrupt by writing to the INIC's Interrupt Clear Register. Shadow registers are maintained on the INIC to ensure that events are not lost.

The '809 app. describes an example of interrupts that may be avoided according to the present specification in comparison with the prior art, stating on page 11, lines 4 – 34:

2.4.1 Fast-path 56k NetBIOS session message

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 506

Let's say a 56k NetBIOS session message is received on the INIC. The first segment will contain the NetBIOS header, which contains the total NetBIOS length. A small chunk of this first segment is provided to the host by filling in a small receive buffer, modifying the interrupt status register on the host, and raising the appropriate interrupt line. Upon receiving the interrupt, the host will read the ISR, clear it by writing back to the INIC's Interrupt Clear Register, and will then process its small receive buffer queue looking for receive buffers to be processed. Upon finding the small buffer, it will indicate the small amount of data up to the client to be processed by NetBIOS. It will also, if necessary, replenish the receive buffer pool on the INIC by passing off a pages worth of small buffers. Meanwhile, the NetBIOS client will allocate a memory pool large enough to hold the entire NetBIOS message, and will pass this address or set of addresses down to the transport driver. The transport driver will allocate an INIC command buffer, fill it in with the list of addresses, set the command type to tell the INIC that this is where to put the receive data, and then pass the command off to the INIC by writing to the command register. When the INIC receives the command buffer, it will DMA the remainder of the NetBIOS data, as it is received, into the memory address or addresses designated by the host. Once the entire NetBIOS transaction is complete, the INIC will complete the command by writing to the response buffer with the appropriate status and command buffer identifier.

In this example, we have two interrupts, and all but a couple hundred bytes are DMA'd directly to their final destination. On PCI we have two interrupt status register writes, two interrupt clear register writes, a command register write, a command read, and a response buffer write.

With a standard NIC this would result in an estimated 30 interrupts, 30 interrupt register reads, 30 interrupt clear writes, and 58 descriptor reads and writes. Plus the data will get moved anywhere from 4 to 8 times across the system memory bus.

The '809 app. describes another example of interrupts that may be avoided according to the present specification in comparison with the prior art, stating on page 11, line 35 – page 12, line 8:

2.4.2 Slow-path receive

If the INIC receives a frame that does not contain a TCP segment for one of its TCB's, it simply passes it to the host as if it were a dumb NIC. If the frame fits into a small buffer (~200 bytes or less), then it simply fills in the small buffer with the data and notifies the host. Otherwise it places the data in a large buffer, writes the address of the large buffer into a small

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 507

buffer, and again notifies the host. The host, having received the interrupt and found the completed small buffer, checks to see if the data is contained in the small buffer, and if not, locates the large buffer. Having found the data, the host will then pass the frame upstream to be processed by the standard protocol stack. It must also replenish the INIC's small and large receive buffer pool if necessary.

With the INIC, this will result in one interrupt, one interrupt status register write and one interrupt clear register write as well as a possible small and or large receive buffer register write. The data will go through the normal path although if it is TCP data then the host will not have to perform the checksum.

With a standard NIC this will result in a single interrupt, an interrupt status register read, an interrupt clear register write, and a descriptor read and write. The data will get processed as it would by the INIC, except for a possible extra checksum.

The '809 app. describes yet another example of interrupts that may be avoided according to the present specification in comparison with the prior art, stating on page 12, lines 9 – 29:

2.4.3 Fast-path 400 byte send

In this example, lets assume that the client has a small amount of data to send. It will issue the TDI Send to the transport driver which will allocate a command buffer, fill it in with the address of the 400 byte send, and set the command to indicate that it is a transmit. It will then pass the command off to the INIC by writing to the command register. The INIC will then DMA the 400 bytes into its own memory, prepare a frame with the appropriate checksums and headers, and send the frame out on the wire. After it has received the acknowledgement it will then notify the host of the completion by writing to a response buffer.

With the INIC, this will result in one interrupt, one interrupt status register write, one interrupt clear register write, a command buffer register write a command buffer read, and a response buffer write. The data is DMA'd directly from the system memory.

With a standard NIC this will result in a single interrupt, an interrupt status register read, an interrupt clear register write, and a descriptor read and write. The data would get moved across the system bus a minimum of 4 times. The resulting TCP ACK of the data, however, would add yet another interrupt, another interrupt status register read, interrupt clear

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 508

register write, a descriptor read and write, and yet more processing by the host protocol stack.

Further description of the problem of interrupts that may be avoided according to the present invention can be found in the instant specification, for example on page 7, lines 14 – 21, which states:

> This fast-path bypasses conventional protocol processing of headers that accompany the data. The fast-path employs a specialized microprocessor designed for processing network communication, avoiding the delays and pitfalls of conventional software layer processing, such as repeated copying and interrupts to the CPU. In effect, the fast-path replaces the states that are traditionally found in several layers of a conventional network stack with a single state machine encompassing all those layers, in contrast to conventional rules that require rigorous differentiation and separation of protocol layers.

Continuing, the instant specification further states, on page 16, lines 3 – 8:

> Both the input and output fast-paths attain a huge reduction in interrupts by functioning at an upper layer level, i.e., session level or higher, and interactions between the network microprocessor and the host occur using the full transfer sizes which that upper layer wishes to make. For fast-path communications, an interrupt only occurs (at the most) at the beginning and end of an entire upper-layer message transaction, and there are no interrupts for the sending or receiving of each lower layer portion or packet of that transaction.

The instant specification also offers an exemplary explanation of processing packets by a first mechanism, so that for each packet the network layer header and the transport layer header are validated without an interrupt dividing the processing of the network layer header and the transport layer header. For example, on page 17, line 24 – page 20, line 2, the instant specification states:

> FIG. 8 shows that the fly-by sequencer 178 has several tiers, with each tier generally focusing on a particular portion of the packet header and thus on a particular protocol layer, for generating status pertaining to that layer. The fly-by sequencer 178 in this embodiment includes a media access control sequencer 191, a network sequencer 192, a transport sequencer 194 and a session sequencer 195. Sequencers pertaining to higher protocol layers can additionally be provided. The fly-by sequencer 178 is reset by the packet control sequencer 176 and given pointers by the packet control sequencer that tell the fly-by sequencer whether a given byte is available from the assembly register 174. The media access

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 509

control sequencer 191 determines, by looking at bytes 0-5, that a packet is addressed to host 152 rather than or in addition to another host. Offsets 12 and 13 of the packet are also processed by the media access control sequencer 191 to determine the type field, for example whether the packet is Ethernet or 802.3. If the type field is Ethernet those bytes also tell the media access control sequencer 191 the packet's network protocol type. For the 802.3 case, those bytes instead indicate the length of the entire frame, and the media access control sequencer 191 will check eight bytes further into the packet to determine the network layer type.

For most packets the network sequencer 192 validates that the header length received has the correct length, and checksums the network layer header. For fast-path candidates the network layer header is known to be IP or IPX from analysis done by the media access control sequencer 191. Assuming for example that the type field is 802.3 and the network protocol is IP, the network sequencer 192 analyzes the first bytes of the network layer header, which will begin at byte 22, in order to determine IP type. The first bytes of the IP header will be processed by the network sequencer 192 to determine what IP type the packet involves. Determining that the packet involves, for example, IP version 4, directs further processing by the network sequencer 192, which also looks at the protocol type located ten bytes into the IP header for an indication of the transport header protocol of the packet. For example, for IP over Ethernet, the IP header begins at offset 14, and the protocol type byte is offset 23, which will be processed by network logic to determine whether the transport layer protocol is TCP, for example. From the length of the network layer header, which is typically 20-40 bytes, network sequencer 192 determines the beginning of the packet's transport layer header for validating the transport layer header. Transport sequencer 194 may generate checksums for the transport layer header and data, which may include information from the IP header in the case of TCP at least.

Continuing with the example of a TCP packet, transport sequencer 194 also analyzes the first few bytes in the transport layer portion of the header to determine, in part, the TCP source and destination ports for the message, such as whether the packet is NetBios or other protocols. Byte 12 of the TCP header is processed by the transport sequencer 194 to determine and validate the TCP header length. Byte 13 of the TCP header contains flags that may, aside from ack flags and push flags, indicate unexpected options, such as reset and fin, that may cause the processor to categorize this packet as an exception. TCP offset bytes 16 and 17 are the checksum, which is pulled out and stored by the hardware logic 171 while the rest of the frame is validated against the checksum.

Session sequencer 195 determines the length of the session layer header, which in the case of NetBios is only four bytes, two of which tell the length of the NetBios payload data, but which can be much larger for other protocols. The session sequencer 195 can also be used to

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 510

categorize the type of message as read or write, for example, for which the fast-path may be particularly beneficial. Further upper layer logic processing, depending upon the message type, can be performed by the hardware logic 171 of packet control sequencer 176 and fly-by sequencer 178. Thus hardware logic 171 intelligently directs hardware processing of the headers by categorization of selected bytes from a single stream of bytes, with the status of the packet being built from classifications determined on the fly. Once the packet control sequencer 176 detects that all of the packet has been processed by the fly-by sequencer 178, the packet control sequencer 176 adds the status information generated by the fly-by sequencer 178 and any status information generated by the packet control sequencer 176, and prepends (adds to the front) that status information to the packet, for convenience in handling the packet by the processor 170. The additional status information generated by the packet control sequencer 176 includes media access controller 172 status information and any errors discovered, or data overflow in either the assembly register or DRAM buffer, or other miscellaneous information regarding the packet. The packet control sequencer 176 also stores entries into a receive buffer queue and a receive statistics queue via the queue manager 184.

An advantage of processing a packet by hardware logic 171 is that the packet does not, in contrast with conventional sequential software protocol processing, have to be stored, moved, copied or pulled from storage for processing each protocol layer header, offering dramatic increases in processing efficiency and savings in processing time for each packet. The packets can be processed at the rate bits are received from the network, for example 100 megabits/second for a 100 baseT connection. The time for categorizing a packet received at this rate and having a length of sixty bytes is thus about 5 microseconds. The total time for processing this packet with the hardware logic 171 and sending packet data to its host destination via the fast-path may be about 16 microseconds or less, assuming a 66 MHz PCI bus, whereas conventional software protocol processing by a 300 MHz Pentium II® processor may take as much as 200 microseconds in a busy device. More than an order of magnitude decrease in processing time can thus be achieved with fast-path 159 in comparison with a high-speed CPU employing conventional sequential software protocol processing, demonstrating the dramatic acceleration provided by processing the protocol headers by the hardware logic 171 and processor 170, without even considering the additional time savings afforded by the reduction in CPU interrupts and host bus bandwidth savings.

Continuing, the instant specification further states, on page 21, lines 8 – 10 and page 22, lines 2 – 5, respectfully:

When INIC microcode or comparator circuits detect a match with the CCB, a DMA controller places the data from the packet in the

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 511

destination 168, without any interrupt by the CPU, protocol processing or copying...

Similarly, the fast-path 237 provides an avenue to send data directly from the source 222 to any of the network lines by processor 230 division of the data into packets and addition of full headers for network transmission, again minimizing CPU processing and interrupts.

Similarly, the instant specification further states, on page 22, lines 15 – 18:

SMB and server/redirector are conventionally serviced by the transport layer; in the present invention SMB and redirector can instead be serviced by the INIC. In this case, sending data by the DMA controllers from the INIC buffers when receiving a large SMB transaction may greatly reduce interrupts that the host must handle. Moreover, this DMA generally moves the data to its final destination in the file device cache. An SMB transmission of the present invention follows essentially the reverse of the above described SMB receive, with data transferred from the host to the INIC and stored in buffers, while the associated protocol headers are prepended to the data in the INIC, for transmission via a network line to a remote host. Processing by the INIC of the multiple packets and multiple TCP, IP, NetBios and SMB protocol layers via custom hardware and without repeated interrupts of the host can greatly increase the speed of transmitting an SMB message to a network line.

In addition to the disclosure of the present application that is quoted above, applicants direct the Examiner's attention to the 6th Supplemental Information Disclosure Statement that accompanies this Reply. That disclosure was considered in Application Serial No. 09/802,551 (now U.S. Patent No. 7,076,568), which was relied upon for priority in the present application and so the accompanying PTO Form 1449 was indicated as having been considered by the initial Examiner of the present application.

Other IDS documents that have been cited in the present application may be relevant to the Examiner's inquiry, including any IDS submitted herewith. Applicants respectfully assert that they have attempted to submit any documents of which they are aware that may be relevant to patentability of the pending claims. Applicants further note that it has been nearly nine years since the filing date of the '809 application, upon which priority is claimed, and so examples and products that are available today, which applicants could perhaps obtain today with the expenditure of time and money, do not

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 512

appear to be relevant to patentability of the pending claims, particularly given the rapid change that is known to occur in computer networking area.

2. Explanation of the meaning of an interrupt with relevance to how it is used in the claimed invention.

Applicants respectfully object to the Requirement's statement that: "The Applicant is requested to explain the meaning of an interrupt with relevance to how it is used in the claimed invention." Applicants respectfully assert that, although 37 CFR §1.105 requires certain facts to be presented if they are known by applicants and requested by the Examiner, nowhere does that section or *Star Fruits* require the submission by applicants of a legal opinion as called for by the Examiner in this Requirement. Applicants nonetheless respond to this statement under the applicable legal standard, as detailed below.

37 CFR §1.105 requires the submission of factual information, not opinions, in contrast to the Requirement's request for interpretation of a claim term. Each of the items listed in 37 CFR §1.105(1)(a) calls for a fact, not an opinion. Those facts involve: (i) *Commercial databases*: The existence of any particularly relevant commercial database...(ii) *Search*: Whether a search of the prior art was made...(iii) *Related information*: A copy of any (document)... that relates to the claimed invention ...(iv) *Information used to draft application*: A copy of any (document)... that was used to draft the application...(v) *Information used in invention process*: A copy of any (document)... that was used in the invention process...(vi) *Improvements*: ...identification of what is being improved...(vii) *In Use:* Identification of any use of the claimed invention...(viii) *Technical information known to applicant.* Technical information known to applicant...other factual information...

37 CFR §1.105 also only provides that requirements for factual information may be presented, in contrast to the Requirement's request for applicants' interpretation of a claim term. That is, 37 CFR §1.105(3) states:

> Requirements for *factual* information known to applicant may be presented in any appropriate manner, for example:
> (i) A requirement for *factual* information;

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 513

(ii) Interrogatories in the form of specific questions seeking applicant's *factual* knowledge; or

(iii) Stipulations as to *facts* with which the applicant may agree or disagree. (*emphasis added*)

The Requirement's request "to explain the meaning of an interrupt with relevance to how it is used in the claimed invention" requires applicants to formulate and express an opinion as to the meaning of claim terms, rather than provide factual information that may be readily available to applicants'. Applicants have presented the readily available factual information responsive the Requirement's request regarding examples, etc, of an interrupt, as discussed above, and respectfully submit that several problems exist with respect to the Requirement's request to submit an opinion with regard to that claim term.

Juxtaposed with the difficulty involved in formulating such an opinion, which would likely involve interpretation of other claim terms and claims, is the lack of relevance of such an opinion under the applicable laws, as will be discussed below. Thus, the Requirement's request "to explain the meaning of an interrupt with relevance to how it is used in the claimed invention" has a high cost and little benefit.

In patent litigation, the interpretation of claim terms, also known as claim construction, is a legal exercise performed by a judge rather than a factual exercise performed by a jury. As noted in *Cybor Corp. v. FAS Techs., Inc.*, 138 F.3d 1448, 1454-55 (Fed. Cir. 1998) (*en banc*), "claim construction is purely a matter of law." See also *Markman v. Westview Instruments*, 517 U.S. 370, 372 (U.S. 1996) "the construction of a patent, including terms of art within its claim, is exclusively within the province of the court." The Requirement's request "to explain the meaning of an interrupt with relevance to how it is used in the claimed invention" would purportedly require applicants to perform an exercise in claim construction.

The leading case on claim construction, *Phillips v. AWH Corp.*, 415 F.3d 1303, 1313 (Fed. Cir. 2005) (*en banc*), provides instruction on what forms of evidence should be used to interpret the meaning of claim terms. Applicants respectfully submit that nowhere does *Phillips* state that one should solicit or consider the opinion of the applicants as to the meaning of a claim term.

According to *Phillips* at 1313, "the ordinary and customary meaning of a claim term is the meaning that the term would have to a person of ordinary skill in the art in

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 514

question at the time of the invention, i.e., as of the effective filing date of the patent application." Applicants do not, at this time, know "the meaning that the term (interrupt) would have to a person of ordinary skill in the art in question at the time of the invention, i.e., as of the effective filing date of the patent application." Applicants respectfully submit that their level of skill in the art may have been higher than that of a person of ordinary skill in the art in question at the time of the invention, as applicants have been granted multiple patents on inventions having that effective filing date. Applicants do not, at this time, know what the level of skill in the art for a person of ordinary skill in the art in question at the time of the invention would have been and so cannot readily express an opinion as to the meaning of the claim term, as requested by the Requirement.

Phillips goes on to say that "Importantly, the person of ordinary skill in the art is deemed to read the claim term not only in the context of the particular claim in which the disputed term appears, but in the context of the entire patent, including the specification." *Id.* Applicants have above identified numerous examples of the specification that relate to the claim term in question. Applicants respectfully submit that this factual information should help to explain the meaning of the claim term "interrupt" as requested by the Requirement.

In addition to the fact that 37 CFR §1.105 provides for factual inquiries, not opinions, and that the opinion that matters in interpreting claim terms is that of a hypothetical person of ordinary skill in the art at the time as of the effective filing date of the patent application, rather than the opinion of the inventors, and that the inventers cannot readily ascertain what that person would have thought the claim term means and so cannot offer an opinion as to what the claim term means, applicants respectfully submit that the Requirement is asking for information in a manner that Congress recently considered to be inappropriate. That is, the Patent Office recently proposed rule changes, and a member of Congress introduced legislation, that proposed to revise 37 CFR §1.105 to require inventors to provide a statement explaining the meaning of prior art being submitted with a patent application. Although such a requirement would be less onerous than a requirement to explain the meaning of claim terms such as is being requested here, applicants believe that the legislation was withdrawn due to lack of support. This

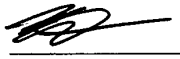
CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 515

provides an additional and independent reason why the Requirement's request for applicants to explain the meaning of a claim term is inappropriate.

III. Conclusion

Applicants have responded to each of the items in the Requirement, including responding to the Requirement's request for an opinion as to a claim term and arguing that the Requirement's request for such an opinion is inappropriate. Should the Examiner have any further questions regarding this application, he is respectfully requested to telephone the undersigned.

Respectfully submitted,

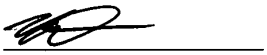


Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the U.S. Postal Service with sufficient postage as first class mail in an envelope addressed to: the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on October 2, 2006.

Date: 10-2-06



Mark Lauer

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 516

OIPE
OCT 04 2006
PATENT & TRADEMARK OFFICE

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of Laurence B. Boucher et al.		Ser. No:	10/260,878
Filing Date:	September 27, 2002	Examiner:	J. B. Dennison
Atty. Docket No:	ALA-006E	GAU:	2143

For: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

October 2, 2006

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sixth Supplemental Information Disclosure Statement per 37 C.F.R. §1.98

Sir:

Pursuant to 37 C.F.R. §§1.56, 1.97 and 1.98, and the Examiner's Requirement for Information under 37 CFR §1.105 mailed on August 9, 2006, the undersigned and Applicants bring the following information to the Examiner's attention in the above-identified application. As indicated in Paragraph 4 on Page 2 of the Examiner's Requirement for Information, the fee and certification requirements of 37 C.F.R. §1.97 are waived as this Supplemental Information Disclosure Statement as the cited documents are submitted in reply to the Examiner's request for information.

The undersigned has not been able to verify the following information with a high degree of certainty. Applicants and the undersigned, therefore, do not and cannot admit or represent that the information set forth below is either true or accurate or complete. Parts of the following information may indeed be wrong. The undersigned sets forth the information in the following numbered paragraphs upon information and belief. If the Examiner would like to discuss anything set forth in this Information Disclosure Statement, the Examiner is requested to contact the undersigned. The undersigned would gladly discuss these matters with the Examiner in further detail.

1. Prior to 1995, a company called Excelan, Inc. made and sold network interface cards (NIC cards) in the United States. One such Excelan NIC card was an expansion card for an IBM PC host computer. The host computer ran a Microsoft operating system. The expansion card coupled the personal computer to an Ethernet local area network (LAN) running TCP/IP. Along with the expansion cards, Excelan provided certain software that executed on the host computer to support the expansion card. In addition to making and selling such NIC cards for IBM personal computers, Excelan also made and sold NIC cards for interfacing numerous other hardware platforms to LANs. For additional detail, see the following documents that are submitted to the Examiner along with this Information Disclosure Statement:

1) The first ten pages of Form 10-K for Excelan, Inc. for the fiscal year ending December 31, 1987.

2) The first ten pages of Form 10-K for Excelan, Inc. for the fiscal year ending December 31, 1988.

2. In an attempt to determine more detail on how the Excelan NIC cards and associated software worked, the undersigned contacted several individuals previously employed by Excelan including Kanwal Rekhi (Excelan's former CEO), Mukesh Sundaram (Excelan's former Chief Technology Officer), and Bruce Steinback. These individuals are not employed or affiliated with Alacritech, Inc. (the assignee of the present application). These individuals were somewhat hesitant to talk at length about Excelan with the undersigned. Consequently, the information received from these individuals was limited.

3. Upon information and belief, Excelan NIC cards provided an interface for TCP/IP communication between an Ethernet network and a host computer. The undersigned at this time does not know whether software executing on the host computers that worked with the Excelan NIC cards included a stack of protocol processing layers.

4. It is the undersigned's present understanding that the Excelan NIC cards and associated software did not provide both: 1) a "fast-path" whereby the card performed TCP and IP protocol processing for some network traffic, and 2) a "slow-path" whereby the card forwarded the other network traffic to a protocol processing stack on the host computer such that the host computer performed TCP and IP protocol processing on that other network traffic.

5. Upon information and belief, control of a TCP connection was not passed from the host computer to the Excelan NIC card, nor was control of a TCP connection passed from the Excelan NIC card to the host computer.

6. Neither the undersigned nor Applicants at this time know whether the Excelan NIC cards had a hardware queue manager.

7. Neither the undersigned nor Applicants at this time know whether the Excelan NIC cards involved dedicated hardware circuitry that performed TCP and/or IP protocol processing, or whether a general purpose processor on the Excelan NIC cards performed all TCP and IP protocol processing.

8. Neither the undersigned nor Applicants at this time know exactly how the Excelan NIC cards and the associated host computer moved data from incoming TCP/IP packets from the NIC card to a location where an application running on the host computer could use the data. For example, neither the undersigned nor Applicants know whether such data was moved directly from the card to a target buffer on the host computer identified by an application program, or whether such data was moved from the card to a first location on the host computer and then the host subsequently moved the data from that first location to a target buffer on the host computer identified by an application program. Data could also have been moved from the card to a target buffer in another indirect fashion.

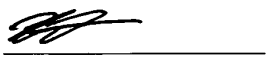
CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 519

9. The undersigned, in an attempt to determine more about how the Excelan NIC cards operated, found several Excelan NIC cards for sale on the Internet. The undersigned purchased the Excelan NIC cards. Due to the considerable time and cost involved, neither the undersigned nor Applicants nor anyone else at Alacritech has to date attached the Excelan NIC cards to an appropriate working host computer and begun the time-consuming task of attempting to determine more details on how the host computer and Excelan NIC cards actually work together.

10. Upon information and belief, Excelan was acquired by Novell, Inc. and as such is no longer in business.

If the Examiner would like to discuss any aspect of this application, including any information set forth in this Information Disclosure Statement, the Examiner is requested to contact the undersigned at the number below.

Respectfully submitted,

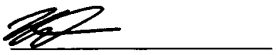


Mark Lauer
Reg. No. 36,578
7041 Koll Center Parkway
Suite 280
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231, on October 2, 2006.

Date: 10-2-06



Mark Lauer

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 520

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
6th SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J. B. Dennison
	Attorney Docket No.: ALA-006E

OIPE
OCT 04 2006
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	1						
	2						
	3						
	4						
	5						
	6						
	7						
	8						
	9						
	10						
	11						
	12						
	13						
	14						
	15						

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	A	Form 10-K for Exelan, Inc., for the fiscal year ending December 31, 1987 (10 pages).
	B	Form 10-K for Exelan, Inc., for the fiscal year ending December 31, 1988 (10 pages).
	C	
	D	
	E	
	F	

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

PATENT APPLICATION FEE DETERMINATION RECORD
Effective October 1, 2001

Application or Docket Number: 10260378

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS	24	
FOR 09/27/02	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	24 minus 20=	* 4
INDEPENDENT CLAIMS	3 minus 3 =	* 0
MULTIPLE DEPENDENT CLAIM PRESENT		/ ☐

* If the difference in column 1 is less than zero, enter "0" in column 2

SMALL ENTITY TYPE ☐			OR	OTHER THAN SMALL ENTITY	
RATE	FEE			RATE	FEE
BASIC FEE	370.00		OR	BASIC FEE	740.00
X$ 9=			OR	X$18=	72
X42=			OR	X84=	
+140=			OR	+280=	/
TOTAL			OR	TOTAL	112

CLAIMS AS AMENDED - PART II

04/27/06

AMENDMENT A

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	* 24	Minus	** 24	= 0
Independent	* 3	Minus	*** 3	= 0
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM ☐				

SMALL ENTITY			OR	OTHER THAN SMALL ENTITY	
RATE	ADDITIONAL FEE			RATE	ADDITIONAL FEE
X$ 9=			OR	X$18=	
X42=			OR	X84=	
+140=			OR	+280=	
TOTAL ADDIT. FEE			OR	TOTAL ADDIT. FEE	0

10/04/06

AMENDMENT B

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	* 24	Minus	** 24	=
Independent	* 3	Minus	*** 3	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM ☐				

RATE	ADDITIONAL FEE		OR	RATE	ADDITIONAL FEE
X$ 9=			OR	X$18=	
X42=			OR	X84=	
+140=			OR	+280=	
TOTAL ADDIT. FEE			OR	TOTAL ADDIT. FEE	

AMENDMENT C

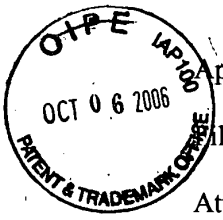
	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM ☐				

RATE	ADDITIONAL FEE		OR	RATE	ADDITIONAL FEE
X$ 9=			OR	X$18=	
X42=			OR	X84=	
+140=			OR	+280=	
TOTAL ADDIT. FEE			OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 522

TFW 2143



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of Laurence B. Boucher et al.		Ser. No:	10/260,878
Filing Date:	September 27, 2002	Examiner:	J. Bret Dennison
Atty. Docket No:	ALA-006E	GAU:	2143

For: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

October 3, 2006

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

7th Supplemental Information Disclosure Statement

Sir:

Pursuant to 37 C.F.R. §§ 1.56, 1.97 and 1.98, applicants bring nineteen reference documents to the Examiner's attention. Included is a one-page form PTO-1449 listing the nineteen U.S. Patent reference documents. Copies of the nineteen U.S. Patent reference documents are not enclosed. Also enclosed is a check in the amount of $180.00.

Citation of these documents shall not be construed as an admission that the documents are prior art with respect to the instant invention, a representation that a search has been made, or an admission that the information cited herein is, or is considered to be, material to patentability as defined in 37 C.F.R. § 1.56(b).

10/10/2006 SFELEKE1 00000012 10260878
01 FC:1806 180.00 OP

Respectfully submitted,

Lara Connors
Patent Agent Reg. No. 32,432
Silicon Edge Law Group LLP
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Commissioner for Patents, P.O. Box 1450 Alexandria, VA 22313-1450, on October 3, 2006.

Date: 3 Oct 06 Lara Connors

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
7th SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J. B. Dennison
	Attorney Docket No.: ALA-006E

OIPE
OCT 0 6 2006
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	1	4,700,185	10/87	Balph et al.	370	451	
	2	5,485,455	01/96	Dobbins et al.	370	255	
	3	5,485,460	01/96	Schrier et al.	703	227	
	4	5,574,919	11/96	Netravali et al.	712	220	
	5	5,699,350	0612/97	Kraslavsky	370	254	
	6	5,768,618	06/98	Erickson et al.	710	9	
	7	5,828,835	10/98	Isfeld et al.	709	200	
	8	5,987.022	11/99	Geiger et al.	370	349	
	9	5.996,013	11/99	Delp et al.	709	226	
	10	6,078,564	06/00	Lakshman et al.	370	235	
	11	6,141,701	10/00	Whitney	710	5	
	12	6,181,705	01/01	Branstad et al.	370	412	
	13	6,473,425	10/02	Bellaton et al.	370	392	
	14	6,487,202	11/02	Klausmeier et al.	370	395.1	
	15	6,765,901	07/04	Johnson et al.	370	352	
	16	6,965,941	11/05	Boucher at al.	709	230	
	17	2002/0073223	06/02	Darnell et al.	709	232	
	18	2003/0110344	06/03	Szczepanek et al.	711	100	
	19	2004/0213290	10/04	Johnson et al.	370	469	
	20						
	21						
	22						
	23						
	24						
	25						
	26						
	27						
	28						

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

EAST Search History

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
S1	1	"6105122".pn. and virtual	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/20 16:18
S2	7234	(storage adj area adj network)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/20 16:18
S3	284	(manag$3 management) near3 (S2 OR san) same switch$3	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/20 16:20
S4	33220488	@ad<"20020506" @pd<"20020506" @rlad<"20020506"	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/20 16:20
S5	130	S3 and S4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/22 15:50
S6	89	S5 and virtual	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/20 16:33
S7	27	lan same S3	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/20 16:33

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 525

EAST Search History

S8	7	S7 and S4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/20 16:40
S9	1	"20040073677".pn. and S8 and virtual	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/20 18:00
S10	0	scalable near4 "i/o" same (add$3 near2 server$1)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/20 18:01
S11	0	scalable same "i/o" same (add$3 near2 server$1)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/20 18:03
S12	855	shared adj ("i/o" "input/output")	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/20 18:02
S13	30	shared adj ("i/o" "input/output") adj subsystem	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/20 18:02
S14	0	S13 same (add$3 near2 server$1)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/20 18:03

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 526

EAST Search History

S15	11	S13 and (add$3 near2 server$1)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/20 18:03
S16	581	load adj balanc$3 with proxy	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/22 14:05
S17	33221526	@ad<"20020506" @pd<"20020506" @rlad<"20020506"	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/22 14:05
S18	378	S16 and S17	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/22 14:06
S19	1	S16 same servers same (storage adj area adj network$3)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/22 14:07
S20	378	S16 and S17	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/22 14:07
S21	0	S19 and S17	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/22 14:07

EAST Search History

S22	16	S16 same servers and (storage adj area adj network$3)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/22 14:07
S23	4	S22 and S17	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/22 14:09
S24	0	"donaghue.xp" and infiniband	US-PGPUB; USPAT	OR	OFF	2006/11/22 15:31
S25	0	"donaghue$.xp" and infiniband	US-PGPUB; USPAT	OR	OFF	2006/11/22 15:31
S26	0	"donaghue$.xa" and infiniband	US-PGPUB; USPAT	OR	OFF	2006/11/22 15:31
S27	0	"donaghue-larry.xp" and infiniband	US-PGPUB; USPAT	OR	OFF	2006/11/22 15:32
S28	0	infiniband and "donaghue.xp"	US-PGPUB; USPAT	OR	OFF	2006/11/22 15:32
S29	0	"donaghue-Larry.xp"	US-PGPUB; USPAT	OR	OFF	2006/11/22 15:32
S30	0	"donaghue.xp"	US-PGPUB; USPAT	OR	OFF	2006/11/22 15:32
S31	4	"7107359".pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/22 15:51
S32	28	("20010053148" | "20020071450" | "20030070014" | "20030091037" | "5367643" | "5634015" | "6188690" | "6243787" | "6400730" | "6421711" | "6545981" | "6557060" | "6591310" | "6594701" | "6611879" | "6628609" | "6668299" | "6678782" | "6690757" | "6691198" | "6694392" | "6751238" | "6775719" | "6778548" | "6831916" | "6859867" | "6889380" | "6937611").PN.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/11/22 15:54

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 528

EAST Search History

S33	1	jaroenchonwanit$.xp. and unisys.as.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/11/22 15:56
S34	2	jaroenchonwanit$.xp. and virtual adj lan	US-PGPUB; USPAT; USOCR	OR	OFF	2006/11/22 15:55
S35	9	(virtual adj lan vlan)and unisys.as.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/11/22 15:56
S36	12	("6289388").URPN.	USPAT	OR	OFF	2006/11/22 16:00
S37	3	("6810431").URPN.	USPAT	OR	OFF	2006/11/22 16:02
S38	38	("3400372" | "4155117" | "4414620" | "5093780" | "5117486" | "5247616" | "5321817" | "5371852" | "5379296" | "5381534" | "5459836" | "5528765" | "5561806" | "5581709" | "5581741" | "5612953" | "5630061" | "5634015" | "5640541" | "5648965" | "5655140" | "5669002" | "5701423" | "5754799" | "5774665" | "5790548" | "5815668" | "5841990" | "5867648" | "5909546" | "5911776" | "5912891" | "5913028" | "5923654" | "5925097" | "5991817" | "6003105").PN.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/11/22 16:12
S39	382	single adj network adj interface	US-PGPUB; USPAT; USOCR	OR	OFF	2006/11/22 16:12
S40	67	single adj network adj interface adj card	US-PGPUB; USPAT; USOCR	OR	OFF	2006/11/22 16:12
S41	190	single adj (nic network adj interface adj card)	US-PGPUB; USPAT; USOCR	OR	OFF	2006/11/22 16:12
S42	33221526	@ad<"20020506" @pd<"20020506" @rlad<"20020506"	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2006/11/22 16:12
S43	97	S41 and S42	US-PGPUB; USPAT; USOCR	OR	OFF	2006/11/22 16:16
S44	0	"10035664".pn.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/11/22 16:31

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 529

EAST Search History

S45	1	"6487608".pn. and address$3	US-PGPUB; USPAT; USOCR	OR	OFF	2006/11/22 16:31

EAST Search History

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	0	"5717691".pn. and simultan$5	US-PGPUB; USPAT; USOCR	OR	OFF	2006/12/29 15:49
L2	0	"5717691".pn. and simultaneous$5	US-PGPUB; USPAT; USOCR	OR	OFF	2006/12/29 15:49
L3	1	"5717691".pn. and "same time"	US-PGPUB; USPAT; USOCR	OR	OFF	2006/12/29 15:49
S1	2684	(session same transport same network) same layer	US-PGPUB; USPAT	OR	OFF	2006/12/28 16:33
S2	2560249	@ad<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/12/28 16:59
S3	426	S1 and S2	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:25
S4	210	S1 same without	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:26
S5	37	S4 and S2	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:43
S6	125206	(intelligent smart) (nic network adj interface adj card)	US-PGPUB; USPAT	OR	OFF	2006/08/17 12:50
S7	108	(intelligent smart) adj (nic network adj interface adj card)	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:45
S8	3	S7 and S2	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:43
S9	1802	(nic network adj interface adj card) with processing	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:44
S10	2	S9 same S1	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:49
S11	1	"20060092934" and session	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:50
S12	1	"6,591,302".pn.	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:52
S13	47	alacritech.as.	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:52
S14	46	S13 and session	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:57
S15	1	"20020091844".pn.	US-PGPUB; USPAT	OR	OFF	2006/06/28 15:45
S16	9	offload$3 same S1	US-PGPUB; USPAT	OR	OFF	2006/06/28 15:47
S17	0	S16 and S2	US-PGPUB; USPAT	OR	OFF	2006/06/28 15:47

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 531

EAST Search History

S18	0	neurauter-g.xa. and S13	US-PGPUB; USPAT	OR	OFF	2006/06/28 15:47
S19	112	(intelligent smart) adj ((nic) (network adj interface adj card)(network adj card))	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:47
S20	2560249	@ad<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:45
S21	4	S19 and S20	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:45
S22	108	(intelligent smart) adj (nic network adj interface adj card)	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:46
S23	3	S22 and S20	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:45
S24	1	S21 not S23	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:46
S25	144	offload$3 near10 ((nic) (network adj interface adj card)(network adj card))	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:48
S26	1	S25 and S20	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:47
S27	3262	bypass$3 near4 (network transport)	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:47
S28	18	S27 same ((nic) (network adj interface adj card)(network adj card))	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:54
S29	0	S28 and S20	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:54
S30	1568	709/250.ccls.	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:54
S31	336	S30 and S20	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:54
S32	76	S31 and network and transport	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:54
S33	17	S32 and ((nic) (network adj interface adj card)(network adj card))	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:56
S34	10	S33 and session	US-PGPUB; USPAT	OR	OFF	2006/06/29 11:10
S35	1	"09970124"	US-PGPUB; USPAT	OR	OFF	2006/06/29 11:11
S36	1	"20020091844".pn. and (network same transport)	US-PGPUB; USPAT	OR	OFF	2006/06/29 11:13
S37	56	craft-peter$.inv.	US-PGPUB; USPAT	OR	OFF	2006/06/29 11:12

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 532

EAST Search History

S38	53	S37 and (network same transport)	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:46
S39	113	("20010004354"|"20010013059"|"20010014892"|"20010014954"|"20010025315"|"20010048681"|"20010053148"|"4336538"|"4589063"|"4991133"|"5056058"|"5058110"|"5097442"|"5163131"|"5212778"|"5280477"|"5289580"|"5303344"|"5412782"|"5418912"|"5448566"|"5485579"|"5506966"|"5511169"|"5524250"|"5535375"|"5548730"|"5566170"|"5588121"|"5590328"|"5592622"|"5598410"|"5619650"|"5629933"|"5634099"|"5634127"|"5642482"|"5664114"|"5671355"|"5678060"|"5682534"|"5692130"|"5699317"|"5701434"|"5701516"|"5727142"|"5749095"|"5751715"|"5752078"|"5758084"|"5758089"|"5758186"|"5758194"|"5771349"|"5778013"|"5790804"|"5794061"|"5802258"|"5802580"|"5809328"|"5812775"|"5815646"|"5872919"|"5878225"|"5898713"|"5913028"|"5930830"|"5931918"|"5935205"|"5937169"|"5941969"|"5941972"|"5950203"|"5970804"|"5991299"|"5996024"|"6005849"|"6009478"|"6016513"|"6021446"|"6021507"|"6026452"|"6034963"|"6044438"|"6047323"|"6047356"|"6049528"|"6057863"|"6061368"|"6065096"|"6067569"|"6070200"|"6101555"|"6141705"|"6145017"|"6157955"|"6172980"|"6173333"|"6202105"|"6226680"|"6246683"|"6247060"|"6279051"|"6298403"|"6345301"|"6356951"|"6389468"|"6427169"|"6434651"|"6449656"|"6453360"|"6490631"|"6591310").PN.	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:46
S40	1809	(nic network adj interface adj card) with processing	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:46
S41	10	S40 and S39	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:46
S42	2691	(session same transport same network) same layer	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:46
S43	12	S42 and S39	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:47
S44	2560249	@ad<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:47
S45	8	S43 and S44	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:56

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 533

EAST Search History

S46	76	S39 and S44	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:56
S47	13	("6226680" "6389479" "6434620" "6470415" "6247060" "6427173" "6427171" "6807581" "6334153" "6757746" "6687758" "6658480" "6393487").pn.	US-PGPUB; USPAT	OR	OFF	2006/06/30 14:05
S48	14	("6226680" "6389479" "6434620" "6470415" "6247060" "6427173" "6427171" "6807581" "6334153" "6757746" "6687758" "6658480" "6393487" "7076568" "7042898" "6393487").pn.	US-PGPUB; USPAT	OR	OFF	2006/06/30 14:07
S49	2560281	@ad<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:14
S50	2560281	S49	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:14
S51	1581	709/250.ccls.	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:15
S52	1938	709/230.ccls.	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:15
S53	26732	header$1 with (send$3 transmit$5)	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:16
S54	5919	S49 and S53	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:16
S55	405	S53 and S52	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:16
S56	53	S55 and S49	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:18
S57	40914	((media medium) adj access adj layer$1) mac	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:23
S58	947603	network transport	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:23
S59	603407	network	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:23
S60	443079	transport	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:24
S61	10455	S57 and S59 and S60	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:24
S62	1335	S57 same S59 same S60 same layer$1	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:24
S63	2560281	@ad<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:24
S64	92	S62 and S63	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:24

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 534

EAST Search History

S65	4	S64 and 709/250.ccls.	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:25
S66	1	offload$3 and S64	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:25
S67	16	offload$3 same S62	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:25
S68	0	S67 and S63	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:25
S69	129196	(intelligent smart) (nic network adj interface adj card)	US-PGPUB; USPAT	OR	OFF	2006/08/17 12:50
S70	6580839	@ad<"19971014" @pd<"19971014" @rlad<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/08/17 12:50
S71	190	single adj (nic network adj interface adj card)	US-PGPUB; USPAT	OR	OFF	2006/12/28 16:36
S72	1	"5920566".pn.	US-PGPUB; USPAT	OR	OFF	2006/12/28 16:36
S73	1	"20040064578".pn.	US-PGPUB; USPAT	OR	OFF	2006/12/28 16:38
S74	2560534	@ad<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:06
S75	18	S71 and S74	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:00
S76	2	S71 same process$3 adj network	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:00
S77	0	S75 and S76	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:00
S78	7885843	@ad<"20020306" @rlad<"20020306" @pd<"20020306"	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:03
S79	139053	(intelligent smart) (nic network adj interface adj card)	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:03
S80	90582	S79 and S78	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:04
S81	118	(intelligent smart) adj (nic network adj interface adj card)	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:04
S82	81	S81 and S78	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:04
S83	2981	(session same transport same network) same layer	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:04
S84	40	S82 and S83	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:04
S85	6583981	@ad<"19971014" @rlad<"19971014" @pd<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:06

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 535

EAST Search History

S86	8	S81 and "154"	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:06
S87	4	S81 and S85	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:14

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 536

EAST Search History

S88	165	("20010004354"|"20010004354"|"20010013059"|"20010014892"|"20010014954"|"20010025315"|"20010025315"|"20010048681"|"20010053148"|"20020073223"|"20030066011"|"20030110344"|"20030165160"|"20040054814"|"20040153578"|"20040213290"|"4366538"|"4589063"|"4700185"|"4991133"|"5056058"|"5058110"|"5097442"|"5163131"|"5212778"|"5280477"|"5289580"|"5303344"|"5412782"|"5418912"|"5448566"|"5485455"|"5485460"|"5485579"|"5506966"|"5511169"|"5517668"|"5524250"|"5535375"|"5548730"|"5566170"|"5574919"|"5588121"|"5590328"|"5592622"|"5598410"|"5619650"|"5629933"|"5633780"|"5634099"|"5634127"|"5642482"|"5664114"|"5671355"|"5678060"|"5682534"|"5692130"|"5699317"|"5699350"|"5701434"|"5701516"|"5727142"|"5742765"|"5749095"|"5751715"|"5752078"|"5758084"|"5758089"|"5758186"|"5758194"|"5768618"|"5771349"|"5778013"|"5778419"|"5790804"|"5794061"|"5802258"|"5802580"|"5809328"|"5812775"|"5815646"|"5828835"|"5848293"|"5872919"|"5878225"|"5892903"|"5892903"|"5898713"|"5913028"|"5920566"|"5930830"|"5931918"|"5935205"|"5937169"|"5941969"|"5941972"|"5950203"|"5970804"|"5987022"|"5991299"|"5996013"|"5996024"|"6005849"|"6009478"|"6016513"|"6021446"|"6021507"|"6026452"|"6034963"|"6038562"|"6041058"|"6044438"|"6047323"|"6047356"|"6049528"|"6057863"|"6061368"|"6065096"|"6067569"|"6070200"|"6078564"|"6078733"|"6097734"|"6101555"|"6115615"|"6122670"|"6141701"|"6141705"|"6145017"|"6157955"|"6172980"|"6173333"|"6181705"|"6202105"|"6223242"|"6226680"|"6246683"|"6247060"|"6247169"|"6279051"|"6289023"|"6298403"|"6334153"|"6345301"|"6345302"|"6356951"|"6385647"|"6389468"|"6389479"|"6421742"|"6434651"|"6449656"|"6453360"|"6473425"|"6480489"|"6487202"|"6487654"|"6490631"|"6526446"|"6591310"|"6650640"|"6681364"|"6697868"|"6765901"|"6807581"|"6912522"|"6941386"|"6965941").PN.	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:14

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 537

EAST Search History

S89	0	"5717691".pn. and S88	US-PGPUB; USPAT	OR	OFF	2006/12/28 19:10
S90	12	DIGHE-RAJIV$.inv.	US-PGPUB; USPAT	OR	OFF	2006/12/28 19:10
S91	26008	"12" and network near2 card	US-PGPUB; USPAT	OR	OFF	2006/12/28 19:10
S92	2	S90 and network near2 card	US-PGPUB; USPAT	OR	OFF	2006/12/28 19:11
S93	36	("5717691").URPN.	USPAT	OR	OFF	2006/12/28 19:11
S94	6583981	@ad<"19971014" @rlad<"19971014" @pd<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/12/28 19:11
S95	11	S93 and S94	USPAT	OR	OFF	2006/12/28 19:12
S96	6	("5548587" | "5555244" | "5594732" | "5617539" | "5623495" | "5623605").PN.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/12/29 13:03
S97	27	dennison-j$.xa.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/12/29 13:04
S98	0	bilgrami-a$.xa.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/12/29 13:04
S99	0	bilgrami.xa.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/12/29 13:04

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/260,878	09/27/2002	Laurence B. Boucher	ALA-006E	9902

24501 7590 01/16/2007

MARK A LAUER
6601 KOLL CENTER PARKWAY
SUITE 245
PLEASANTON, CA 94566

EXAMINER	
DENNISON, JERRY B	
ART UNIT	PAPER NUMBER
2143	

SHORTENED STATUTORY PERIOD OF RESPONSE	MAIL DATE	DELIVERY MODE
3 MONTHS	01/16/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

Office Action Summary	Application No.	Applicant(s)	
	10/260,878	BOUCHER ET AL.	
	Examiner	Art Unit	
	J. Bret Dennison	2143	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *10/4/2006*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-24* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-24* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date 03/21/05, 04/27/06, 08/28/06, 10/04/06, 10/06/06
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This Action is in response to Applicant's Reply for Application Number 10/260,878 received on 04 October 2006.

2. Claims 1-24 have been presented for examination.

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

Claims 1-7, 9-12, 14, 16-18, 20, 22, and 24 are rejected under 35 U.S.C. 102(b) as being anticipated by Dighe et al. (U.S. 5,717,691).

3. Regarding claim 1, Dighe disclosed a Multimedia Computer and Communications Platform (MCCP) that plays the role of an intelligent network interface card for running multimedia applications on existing workstations, which supports a wide range of network-edge products, such as a simple Video On Demand receiver, an advanced multimedia workstation and a small scale server (Dighe, col. 2, lines 13-20).

Therefore, Dighe disclosed a method for network communication, the method comprising:

receiving a plurality of packets from the network, each of the packets including a media access control layer header, a network layer header and a transport layer header

(Dighe, Abstract, Dighe disclosed receiving media streams, col. 2, lines 40-45, Dighe disclosed that the MCCP includes a communications processor that terminates all the networking functions including transport processing, therefore receiving TCP/IP packets, col. 5, lines 24-25);

processing the packets by a first mechanism, so that for each packet the network layer header and the transport layer header are validated without an interrupt dividing the processing of the network layer header and the transport layer header (Dighe, col. 3, lines 30-35, Dighe disclosed the MCCP handling all networking-related tasks and pass only the application-level data to media processing devices via shared memory interface);

sorting the packets, dependent upon the processing, into first and second types of packets, so that the packets of the first type each contain data (Dighe, col. 4, lines 37-55, Dighe disclosed the MCCP in connection with various multimedia devices, therefore requiring the sorting of packets based on the processing of different types of multimedia for the various devices, Dighe also disclosed ATM segmentation and reassembly);

sending the data from each packet of the first type to a destination without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination (Dighe, col. 3, lines 30-35, Dighe disclosed the MCCP handling all networking-related tasks and pass only the application-level data to media processing devices via shared memory interface).

4. Regarding claim 2, Dighe disclosed the limitations, substantially as claimed, as described in claim 1, including wherein processing the packets by a first mechanism further comprises:

processing the media access control layer header for each packet without an interrupt dividing the processing of the media access control layer header and the network layer header (Dighe, col. 3, lines 30-35, Dighe disclosed the MCCP handling all networking-related tasks and pass only the application-level data to media processing devices via shared memory interface).

5. Regarding claim 3, Dighe disclosed the limitations, substantially as claimed, as described in claim 1, including processing an upper layer header of at least one of the packets by a second mechanism, thereby determining the destination, wherein the upper layer header corresponds to a protocol layer above the transport layer (Dighe, col. 3, lines 33-35).

6. Regarding claim 4, Dighe disclosed the limitations, substantially as claimed, as described in claim 1, including processing an upper layer header of at least one of the packets of the second type by a second mechanism, thereby determining the destination (Dighe, col. 3, lines 33-35).

7. Regarding claim 5, Dighe disclosed the limitations, substantially as claimed, as described in claim 1, including processing a transport layer header of another packet by

a second mechanism, prior to receiving the plurality of packets from the network, thereby establishing a Transmission Control Protocol (TCP) connections for the packets of the first type (Dighe, col. 5, lines 20-30).

8. Regarding claim 6, Dighe disclosed the limitations, substantially as claimed, as described in claim 1, including wherein sorting the packets includes classifying each of the packets of the first type as having an Internet Protocol (IP) header and a Transmission Control Protocol (TCP) (Dighe, col. 5, lines 20-30).

9. Regarding claim 7, Dighe disclosed the limitations, substantially as claimed, as described in claim 1, including transmitting a second plurality of packets to the network, each of the second plurality of packets containing a media access control layer header, a network layer header and a transport layer header, including processing the second plurality of packets by the first mechanism, so that for each packet the media access control layer header, the network layer header and the transport layer header are processed without an interrupt dividing the processing of the media access control layer header (Dighe, col. 3, lines 30-35, Dighe disclosed the MCCP handling all networking-related tasks and pass only the application-level data to media processing devices via shared memory interface).

10. Regarding claim 9, Dighe disclosed a method for communicating information over a network, the method comprising:

obtaining data from a source allocated by a first processor;

dividing the data into multiple segments;

prepending a packet header to each of the segments by a second processor, thereby forming a packet corresponding to each segment, each packet header containing a media access control layer header, a network layer header and a transport layer header, wherein the network layer header is Internet Protocol (IP), the transport layer header is Transmission Control Protocol (TCP) and the prepending of each packet header occurs without an interrupt dividing the prepending of the network layer header and the transport layer header;

transmitting the packets to the network (Dighe, col. 5, lines 20-40, Dighe disclosed the communications processor (CP) in the MCCP handles all the network related functions as well as the overall management of the MCCP, the CP offloading transport protocols from the host CPU, such as TCP/IP. Some of the functions typically performed by the CP include call-setup signaling and all the necessary functions to load state memory needed for the ATM function as well as the management of data movement from the network to the devices as well as from the devices to the network, Dighe, see Abstract, Dighe disclosed media stream distribution and media stream adaptation. Therefore, Dighe disclosed obtaining media streaming data from various media devices, including all networking protocols to the packets and transmitting the packets to the network).

11. Regarding claim 10, Dighe disclosed the limitations, substantially as claimed, as described in claim 9, including wherein prepending a packet header to each of the segments by a second processor further comprises prepending the media access control layer header for each packet without an interrupt dividing the prepending of the media access control layer header and the network layer header (Dighe, col. 5, lines 20-40).

12. Regarding claim 11, Dighe disclosed the limitations, substantially as claimed, as described in claim 9, including wherein each packet header contains an Internet Protocol (IP) header and a Transmission Control Protocol (TCP) header (Dighe, col. 5, lines 24-26).

13. Regarding claim 12, Dighe disclosed the limitations, substantially as claimed, as described in claim 9, including establishing a Transmission Control Protocol (TCP) connection by the first processor and using the connection to prepend the packet header to each of the segments by the second processor (Dighe, col. 5, lines 20-35).

14. Regarding claim 14, Dighe disclosed the limitations, substantially as claimed, as described in claim 9, including wherein obtaining data from the source in memory allocated by the first processor is performed by a Direct Memory Access (DMA) unit controlled by the second processor (Dighe, col. 4, lines 50-53).

15. Regarding claim 16, Dighe disclosed the limitations, substantially as claimed, as described in claim 9, including receiving another packet from the network, the other packet containing a receive header including information corresponding to a network layer and a transport layer; and

selecting whether to process the other packet by the first processor or by the second processor (Dighe, col. 3, lines 30-41).

16. Claim 17 includes limitations that are substantially similar to the limitations of claims 1 and 9, and is therefore rejected under the same rationale.

17. Regarding claim 18, Dighe disclosed the limitations, substantially as claimed, as described in claim 17, including wherein the processing the inbound packets is performed simultaneously with the prepending the outbound packet header to each of the segments (Dighe, col. 4, lines 44-55, col. 7, lines 55-65).

18. Regarding claim 20, Dighe disclosed the limitations, substantially as claimed, as described in claim 17, including wherein providing multiple segments of data includes dividing a block of data into the segments (Dighe, col. 3, lines 39-41, Dighe discloses the use of media streaming).

19. Regarding claim 22, Dighe disclosed the limitations, substantially as claimed, as described in claim 17, including sending data from each inbound packet to a destination

in memory allocated to an application without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination (Dighe, col. 5, lines 20-35).

20. Regarding claim 24, Dighe disclosed the limitations, substantially as claimed, as described in claim 17, including processing a transport layer header of another packet, prior to receiving the plurality of packets from the network, thereby establishing a Transmission Control Protocol (TCP) connection for the inbound packets (Dighe, col. 5, lines 20-30).

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

Claim 8 is rejected under 35 U.S.C. 103(a) as being unpatentable over Dighe et al. (U.S. 5,717,691) in view of Radogna et al. (US 5,991,299).

21. As per claim 8, Dighe disclosed the limitations, substantially as claimed, as described in claim 1, but did not explicitly state the method wherein the first mechanism is a sequencer running microcode.

However, in a similar art, Radogna teaches a dedicated sequencer using microcode to perform network communication and header translation and processing (e.g. col. 4, lines 25-30).

It would have been obvious to one skilled in the art at the time the invention was made to combine Radogna with Dighe because of the benefits of using a specialized processor to handle various tasks in a communications system. Using a sequencer for processing header information can greatly accelerate a frame or packet through a network since the central processing unit does not become overburdened when many packets need to be processed. This frees up the central processor to handle other networking tasks, therefore increasing the speed and efficiency of transmissions through the network. The use of software microcode for this processing easily accommodates new protocols and can bypass hardware processing in the event of a hardware failure. These are beneficial in any computer network system.

Claims 13 and 19 are rejected under 35 U.S.C. 103(a) as being unpatentable over Dighe et al. (U.S. 5,717,691) in view of Hansen et al. (US 5,778,419).

22. As per claim 13, Dighe disclosed the limitations, substantially as claimed, as described in claim 9, but did not explicitly state the method further comprising creating a template header and forming each packet header based upon the template header.

However, in a similar art, Hansen teaches the use of a header template from which all packet headers are based (e.g. Hansen, col. 6, lines 4-21).

It would have been obvious to one skilled in the art at the time the invention was made to combine Hansen with Dighe because of the advantages of using a template when creating a similar header for each packet. A template is a well-known method for creating files, or in this case, a header, which needs to be attached to many packets containing altogether the same, or very similar data. The structure of each packet header should always consist of the same elements in the same arrangement so a processor does not have to locate the information it needs prior to performing processing functions. When a template is used, a large amount of time can be saved when performing a large number of transmissions, since it is not necessary to create an entire packet header during each iteration. This increases the overall speed and efficiency of the network, which is beneficial in any communication network system.

23. As per claim 19, Dighe disclosed the limitations, substantially as claimed, as described in claim 17, but did not explicitly state the method further comprising creating a template header and using the template header to form each outbound packet header.

However, in a similar art, Hansen teaches the use of a header template from which all packet headers are based (e.g. Hansen, col. 6, lines 4-21).

It would have been obvious to one skilled in the art at the time the invention was made to combine Hansen with Dighe for similar reasons as stated above in regards to claim 13.

Claims 15, 21 and 23 are rejected under 35 U.S.C. 103(a) as being unpatentable over Dighe et al. (U.S. 5,717,691) in view of Ota et al. (US 6,115,615).

24. As per claim 15, Dighe disclosed the limitations, substantially as claimed, as described in claim 9, but did not explicitly state the method further comprising prepending an upper layer header to the data, prior to dividing the data into multiple segments.

However, in a similar art, Ota teaches a network communication system that attaches and uses a header in the application layer (e.g. Ota, col. 7, lines 18-25, 40-53).

It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Dighe because of the advantages of attaching a header to an upper layer, such as the application layer, along with the other layers well-known by the OSI model. The use of an upper layer header can provide a great deal of flexibility to the system since it is able to transmit more data with the packet itself. The OSI model is designed to attach and process headers from each of the seven layers efficiently to ensure that the data within the packet is transmitted properly across the network. Including an application layer header further ensures the proper receipt of the data. This is beneficial in any communication network system.

25. As per claim 21, Dighe disclosed the limitations, substantially as claimed, as described in claim 20, but did not explicitly state the method further comprising

prepending an upper layer header to the block of data, prior to dividing the block of data into multiple segments.

However, in a similar art, Ota teaches a network communication system that attaches and uses a header in the application layer (e.g. Ota, col. 7, lines 18-25, 40-53).

It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Dighe for similar reasons as stated above in regards to claim 15.

26. As per claim 23, Dighe disclosed the limitations, substantially as claimed, as described in claim 20, but did not explicitly state the method further comprising: processing an upper layer header of at least one of the packets by a second mechanism, thereby determining the destination, wherein the upper layer header corresponds to a protocol layer above the transport layer.

However, in a similar art, Ota teaches a network communications system that uses an application layer level address to indicate the destination and route packets through the network (e.g. Ota, col. 7, lines 18-25, 40-53).

It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Dighe because of the advantages of using an upper layer header to determine the destination of packets in a network. Network layer and transport layers also generally include addresses or indications of destination for the packets and including this feature into the application layer as well provides another fail-safe step for the network in the even of a failure in some portion of the network. Having

fail-safe routes for information decreases the amount of network downtime since routes can be switched almost instantaneously upon the realization of a fault or error. This is a benefit in any communications network system.

Conclusion

Examiner's Note: Examiner has cited particular columns and line numbers in the references applied to the claims above for the convenience of the applicant. Although the specified citations are representative of the teachings of the art and are applied to specific limitations within the individual claim, other passages and figures may apply as well. It is respectfully requested from the applicant in preparing responses, to fully consider the references in entirety as potentially teaching all or part of the claimed invention, as well as the context of the passage as taught by the prior art or disclosed by the Examiner.

In the case of amending the claimed invention, Applicant is respectfully requested to indicate the portion(s) of the specification which dictate(s) the structure relied on for proper interpretation and also to verify and ascertain the metes and bounds of the claimed invention.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to J. Bret Dennison whose telephone number is (571) 272-3910. The examiner can normally be reached on M-F 8:30am-5pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, David A Wiley can be reached on (571) 272-3923. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

J. B. D.
Patent Examiner
Art Unit 2143

JEFFREY PWU
PRIMARY EXAMINER

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
3rd SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

MAR 0 9 2005 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A	5,524,250	6/4/96	Chesson et al.	395	775	
	B	5,619,650	4/8/97	Bach et al.	395	200.01	
	C	5,727,142	3/10/98	Chen	395	181	
	D	5,802,258	9/1/98	Chen	395	182.08	
	E	5,898,713	4/27/99	Melzer et al.	371	53	
	F	6,021,507	2/1/00	Chen	714	2	
	G	6,047,323	4/4/00	Krause	709	227	
	H						

OTHER ART—NON PATENT LITERATURE DOCUMENTS

*Examiner Initial	Cite No.	(Including Author, Title, Date, Pertinent Pages, Etc.)
	1	Schwaderer et al., IEEE Computer Society Press publication entitled, "XTP in VLSI Protocol Decomposition for ASIC Implementation", from 15th Conference on Local Computer Networks, 5 pages, Sept. 30 – Oct. 3, 1990.
	2	Beach, Bob, IEEE Computer Society Press publication entitled, "UltraNet: An Architecture for Gigabit Networking", from 15th Conference on Local Computer Networks, 18 pages, Sept. 30 – Oct. 3, 1990.
	3	Chesson et al., IEEE Syposium Record entitled, "The Protocol Engine Chipset", from Hot Chips III, 16 pages, Aug. 26-27, 1991.
	4	Maclean et al., IEEE Global Telecommunications Conference, Globecom '91, presentation entitled, "An Outboard Processor for High Performance Implementation of Transport Layer Protocols", 7 pages, Dec. 2-5, 1991.
	5	Ross et al., IEEE article entitled "FX1000: A high performance single chip Gigabit Ethernet NIC", from Compcon '97 Proceedings, 7 pages, Feb. 23-26, 1997.
	6	Strayer et al., "Ch. 9: The Protocol Engine" from XTP: The Transfer Protocol, 12 pages, July 1992.
	7	Publication entitled "Protocol Engine Handbook", 44 pages, Oct. 1990.
	8	Koufopavlou et al., IEEE Global Telecommunications Conference, Globecom '92, presentation entitled, "Parallel TCP for High Performance Communication Subsystems", 7 pages, Dec. 6-9, 1992.
	9	Lilienkamp et al., Publication entitled "Proposed Host-Front End Protocol", 56 pages, Dec. 1984.

Examiner	Date Considered 12/29/06

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
04 SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

APR 27 2006
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	1	2004/0153578	8-04	Elzur			
	2	2004/0054814	3-04	McDaniel			
	3	4,589,063	5/13/86	Shah et al.	710	8	
	4	5,418,912	5-95	Christenson	709	234	
	5	5,535,375	7-96	Eshel	703	27	
	6	5,598,410	1/28/1997	Stone	370	469	
	7	5,633,780	05-1997	Cronin	361	220	
	8	5,682,534	10-97	Kapoor et al.	709	203	
	9	5,742,765	04-1998	Wong et al.	709	230	
	10	5,778,013	7/7/98	Jedwab	714	807	
	11	5,778,419	07-1998	Hansen et al.	711	112	
	12	5,848,293	12-1998	Gentry	710	5	
	13	5,872,919	2-99	Wakeland	709	230	
	14	5,878,225	3-99	Bilansky et al.	709	227	
	15	5,892,903	04-1999	Klaus	709	227	
	16	5,920,566	07-1999	Hendel et al.	370	401	
	17	5,991,299	11-1999	Radogna et al.	370	392	
	18	6,041,058	03-2000	Flanders et al.	370	401	
	19	6,049,528	4-00	Hendel et al.	370	235	
	20	6,078,733	06-2000	Osborne	709	250	
	21	6,097,734	08-2000	Gotesman et al.	370	474	
	22	6,115,615	09-2000	Ota et al.	455	422.1	
	23	6,122,670	09-00	Bennett et al.			
	24	6,223,242	05-2001	Mayer et al.	370	392	
	25	6,246,683	06-2001	Connery et al.	709	250	
	26	6,289,023	09-2001	Dowling et al.	370	419	

Examiner _____ Date Considered 12/29/06

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE APR 27 2006 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	27	6,289,023	09-2001	Dowling et al.	370	419	
	28	6,385,647	05-2002	Wills et al.	709	217	
	29	6,487,654	11-02	Dowling			
	30	6,490,631	12-02	Teich et al.	709	250	
	31	6,591,310	7-03	Johnson	710	3	
	32	6,650,640	11-2003	Muller et al.	370	392	
	33	6,681,364	1-04	Calvignac et al.			
	34	6,697,868	02-2004	Craft et al.	709	230	
	35	6,807,581	10-2004	Starr et al.	709	250	
	36	6,912,522	06-05	Edgar			
	37	6,941,386	09-2005	Craft et al.	709	250	
	38						
	39						
	40						
	41						
	42						
	43						
	44						
	45						
	46						
	47						
	48						
	49						
	50						
	51						
	52						

Examiner [signature] Date Considered 12/29/06

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J Bret Dennison
	Attorney Docket No.: ALA-006E

OIPE 510 AUG 28 2006 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	1	2003/0165160	9/4/03	Minami et al.	370	466	
	2	2003/0066011	4/11/02	Oren	714	758	
	3	2001/0053148	12/20/01	Bilic et al.	370	389	
	4	2001/0048681	12/6/01	Bilic et al.	370	389	
	5	2001/0014954	8/16/01	Purcell et al.	714	4	
	6	2001/0014892	8/16/01	Gaither et al.	707	200	
	7	2001/0013059	8/9/01	Dawson et al.	709	217	
	8	2001/0025315A1	09/27/01	Jolitz	707	001	
	9	2001/0004354A1	06/21/01	Jolitz	709	240	
	10	5,058,110	10/15/91	Beach et al.	370	85.6	
	11	5,517,668	5/14/96	Szwerinski et al.	395	800	
	12	5,598,410	1/28/97	Stone	370	469	
	13	5,970,804	10/26/99	Osborne	395	200.75	
	14	6,038,562	3/14/00	Anjur et al.	707	10	
	15	6,067,569	5/23/00	Khaki et al.	709	224	
	16	6,070,200	5/30/00	Gates et al.	710	20	
	17	6,101,555	8/8/00	Goshey et al.	709	321	
	18	6,145,017	11/7/00	Ghaffari	710	5	
	19	6,157,955	12/5/00	Narad et al.	709	228	
	20	6,172,980	1/9/01	Flanders et al.	370	401	
	21	6,173,333	07/18/97	Jolitz et al.	709	240	
	22	6,202,105	3/13/01	Gates et al.	710	20	
	23	6,279,051	8/21/01	Gates et al.	710	20	
	24	6,298,403	10/2/01	Suri et al.	710	100	
	25	6,334,153	12/25/01	Boucher et al.	709	230	
	26	6,345,302	02/05/02	Bennett et al.	709	236	

Examiner [signature] Date Considered 12/29/06

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
5th SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J. Bret Dennison
	Attorney Docket No.: ALA-006E

OIPE IAP84 AUG 28 2006 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	27	6,389,479	5/14/02	Boucher	709	243	
	28	6,421,742	7/16/02	Tillier	710	1	
	29	6,480,489	11/12/02	Muller et al.	370	389	
	30	6,526,446	2/25/03	Yang et al.	709	230	

Foreign Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	1	WO 01/59966	8/16/01	PCT/US00/06475			
	2	WO 01/86430	11/15/01	PCT/US01/15180			

OTHER ART – NON-PATENT LITERATURE DOCUMENTS

*Examiner Initial	Cite No.	(Including Author, Title, Date, Pertinent Pages, Etc.)
	1	Thia, Y.H. Publication entitled "High-Speed OSI Protocol Bypass Algorithm with Window Flow Control", Protocols for High Speed Networks, pages 53-68, 1993.
	2	Thia, Y.H. Publication entitled " A Reduced Operational Protocol Engine (ROPE) for a multiple-layer bypass architecture", Protocols for High Speed Networks, pages 224-239, 1995.
	3	U.S. Provisional Patent Application No.: 60/283,896, Titled: CRC Calculations for Out of Order PUDs, Filed April 12, 2003, Inventor: Amit Oren, Assignee: Siliquent Technologies Ltd.

Examiner	Date Considered 12/22/06

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
6th SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J. B. Dennison
	Attorney Docket No.: ALA-006E

OIPE
OCT 0 4 2006
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	1						
	2						
	3						
	4						
	5						
	6						
	7						
	8						
	9						
	10						
	11						
	12						
	13						
	14						
	15						

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

	A	Form 10-K for Exelan, Inc., for the fiscal year ending December 31, 1987 (10 pages).
	B	Form 10-K for Exelan, Inc., for the fiscal year ending December 31, 1988 (10 pages).
	C	
	D	
	E	
	F	

Examiner [signature] Date Considered 12/29/06

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
7th SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J. B. Dennison
	Attorney Docket No.: ALA-006E

OIPE
OCT 06 2006
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	1	4,700,185	10/87	Balph et al.	370	451	
	2	5,485,455	01/96	Dobbins et al.	370	255	
	3	5,485,460	01/96	Schrier et al.	703	227	
	4	5,574,919	11/96	Netravali et al.	712	220	
	5	5,699,350	0612/97	Kraslavsky	370	254	
	6	5,768,618	06/98	Erickson et al.	710	9	
	7	5,828,835	10/98	Isfeld et al.	709	200	
	8	5,987.022	11/99	Geiger et al.	370	349	
	9	5.996,013	11/99	Delp et al.	709	226	
	10	6,078,564	06/00	Lakshman et al.	370	235	
	11	6,141,701	10/00	Whitney	710	5	
	12	6,181,705	01/01	Branstad et al.	370	412	
	13	6,473,425	10/02	Bellaton et al.	370	392	
	14	6,487,202	11/02	Klausmeier et al.	370	395.1	
	15	6,765,901	07/04	Johnson et al.	370	352	
	16	6,965,941	11/05	Boucher at al.	709	230	
	17	2002/0073223	06/02	Darnell et al.	709	232	
	18	2003/0110344	06/03	Szczepanek et al.	711	100	
	19	2004/0213290	10/04	Johnson et al.	370	469	
	20						
	21						
	22						
	23						
	24						
	25						
	26						
	27						
	28						

Examiner [signature] Date Considered 12/29/06

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

Notice of References Cited	Application/Control No. 10/260,878	Applicant(s)/Patent Under Reexamination BOUCHER ET AL.	
	Examiner J. Bret Dennison	Art Unit 2143	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
*	A	US-5,548,587	08-1996	Bailey et al.	370/395.7
*	B	US-5,974,466	10-1999	Mizutani et al.	709/236
*	C	US-5,717,691	02-1998	Dighe et al.	370/401
	D	US-			
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

Index of Claims

Application/Control No.	Applicant(s)/Patent under Reexamination
10/260,878	BOUCHER ET AL.
Examiner	**Art Unit**
J. Bret Dennison	2143

√	Rejected	–	(Through numeral) Cancelled	N	Non-Elected	A	Appeal
=	Allowed	÷	Restricted	I	Interference	O	Objected

Claim		Date								
Final	Original	12/29/06								
	1	√								
	2	√								
	3	√								
	4	√								
	5	√								
	6	√								
	7	√								
	8	√								
	9	√								
	10	√								
	11	√								
	12	√								
	13	√								
	14	√								
	15	√								
	16	√								
	17	√								
	18	√								
	19	√								
	20	√								
	21	√								
	22	√								
	23	√								
	24	√								
	25									
	26									
	27									
	28									
	29									
	30									
	31									
	32									
	33									
	34									
	35									
	36									
	37									
	38									
	39									
	40									
	41									
	42									
	43									
	44									
	45									
	46									
	47									
	48									
	49									
	50									

Claim		Date								
Final	Original									
	51									
	52									
	53									
	54									
	55									
	56									
	57									
	58									
	59									
	60									
	61									
	62									
	63									
	64									
	65									
	66									
	67									
	68									
	69									
	70									
	71									
	72									
	73									
	74									
	75									
	76									
	77									
	78									
	79									
	80									
	81									
	82									
	83									
	84									
	85									
	86									
	87									
	88									
	89									
	90									
	91									
	92									
	93									
	94									
	95									
	96									
	97									
	98									
	99									
	100									

Claim		Date								
Final	Original									
	101									
	102									
	103									
	104									
	105									
	106									
	107									
	108									
	109									
	110									
	111									
	112									
	113									
	114									
	115									
	116									
	117									
	118									
	119									
	120									
	121									
	122									
	123									
	124									
	125									
	126									
	127									
	128									
	129									
	130									
	131									
	132									
	133									
	134									
	135									
	136									
	137									
	138									
	139									
	140									
	141									
	142									
	143									
	144									
	145									
	146									
	147									
	148									
	149									
	150									

Search Notes	Application/Control No.	Applicant(s)/Patent under Reexamination
	10/260,878	BOUCHER ET AL.
	Examiner	Art Unit
	J. Bret Dennison	2143

SEARCHED			
Class	Subclass	Date	Examiner

INTERFERENCE SEARCHED			
Class	Subclass	Date	Examiner

SEARCH NOTES (INCLUDING SEARCH STRATEGY)		
	DATE	EXMR
Performed Inventor search in PALM	1/13/2006	EJK
Searched EAST Databases See attached Search History	1/16/2006	EJK
Conducted PLUS search	1/6/2006	EJK
Spoke with John Follansbee	12/29/2006	JBD

TFW 2154

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of Laurence B. Boucher, et al.		Ser. No:	10/260,878
Filing Date:	September 27, 2002	Examiner:	J. Bret Dennison
Atty. Docket No:	ALA-006E	GAU:	2154

For: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

April 16, 2007

MS Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Second Amendment

Sir:

In response to an Office Action dated January 16, 2007, please enter the following Amendment to the Claims and consider the following Remarks.

Amendment to the Claims

1. (currently amended) A method for network communication, the method comprising:

receiving a plurality of packets from the network, each of the packets including a media access control layer header, a network layer header and a transport layer header;

processing the packets by a first mechanism, so that for each packet the network layer header and the transport layer header are validated without an interrupt dividing the processing of the network layer header and the transport layer header;

sorting the packets, dependent upon the processing, into first and second types of packets, so that the packets of the first type each contain data;

sending, by the first mechanism, the data from each packet of the first type to a destination in memory allocated to an application without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination.

2. (original) The method of claim 1, wherein processing the packets by a first mechanism further comprises:

processing the media access control layer header for each packet without an interrupt dividing the processing of the media access control layer header and the network layer header.

3. (original) The method of claim 1, further comprising:

processing an upper layer header of at least one of the packets by a second mechanism, thereby determining the destination, wherein the upper layer header corresponds to a protocol layer above the transport layer.

4. (original) The method of claim 1, further comprising:

processing an upper layer header of at least one of the packets of the second type by a second mechanism, thereby determining the destination.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 566

5. (previously presented) The method of claim 1, further comprising:

processing a transport layer header of another packet by a second mechanism, prior to receiving the plurality of packets from the network, thereby establishing a Transmission Control Protocol (TCP) connection for the packets of the first type.

6. (previously presented) The method of claim 1, wherein sorting the packets includes classifying each of the packets of the first type as having an Internet Protocol (IP) header and a Transmission Control Protocol (TCP).

7. (original) The method of claim 1, further comprising:

transmitting a second plurality of packets to the network, each of the second plurality of packets containing a media access control layer header, a network layer header and a transport layer header, including processing the second plurality of packets by the first mechanism, so that for each packet the media access control layer header, the network layer header and the transport layer header are ~~processed without an interrupt dividing the processing of the media access control layer header, the network layer header and the transport layer header~~ prepended at one time as a packet header.

8. (original) The method of claim 1, wherein the first mechanism is a sequencer running microcode.

9. (currently amended) A method for communicating information over a network, the method comprising:

obtaining data from a source in memory allocated by a first processor;

dividing the data into multiple segments;

prepending a packet header to each of the segments by a second processor, thereby forming a packet corresponding to each segment, each packet header containing a media access control layer header, a network layer header and a transport layer header, wherein the network layer header is Internet Protocol (IP), the transport layer header is Transmission Control Protocol (TCP) and the ~~prepending of each packet header occurs~~

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 567

~~without an interrupt dividing the prepending of~~ media access control layer header, the network layer header and the transport layer header are prepended at one time as a sequence of bits during the prepending of each packet header; and

transmitting the packets to the network.

10. (currently amended) The method of claim 9, wherein ~~prepending a packet header to each of the segments by a second processor further comprises:~~

~~prepending the media access control layer header for each packet without an interrupt dividing the prepending of the media access control layer header and the network layer header~~ each packet header is formed based upon a block of information created by the first processor.

11. (currently amended) The method of claim 9, ~~wherein each packet header contains an Internet Protocol (IP) header and a Transmission Control Protocol (TCP) header~~ further comprising:

receiving another packet from the network, the other packet containing a receive header including information corresponding to a network layer and a transport layer; and

determining, by the second processor, whether the other packet corresponds to the same TCP connection as the transmitted packets.

12. (previously presented) The method of claim 9, further comprising establishing a Transmission Control Protocol (TCP) connection by the first processor and using the connection to prepend the packet header to each of the segments by the second processor.

13. (original) The method of claim 9, further comprising creating a template header and forming each packet header based upon the template header.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 568

14. (original) The method of claim 9, wherein obtaining data from the source in memory allocated by the first processor is performed by a Direct Memory Access (DMA) unit controlled by the second processor.

15. (original) The method of claim 9, further comprising prepending an upper layer header to the data, prior to dividing the data into multiple segments.

16. (original) The method of claim 9, further comprising:

receiving another packet from the network, the other packet containing a receive header including information corresponding to a network layer and a transport layer; and

selecting whether to process the other packet by the first processor or by the second processor.

17. (currently amended) A method for communicating information over a network, the method comprising:

~~providing multiple segments of data;~~

providing, by a first mechanism, a block of data and a Transmission Control Protocol (TCP) connection;

dividing, by a second mechanism, the block of data into multiple segments;

prepending, by the second mechanism, an outbound packet header to each of the segments, thereby forming an outbound packet corresponding to each segment, the outbound packet header containing an outbound media access control layer header, an outbound ~~network layer~~ Internet Protocol (IP) header and an outbound ~~transport layer~~ TCP header, wherein the prepending of each outbound packet header occurs without an interrupt dividing the prepending of the outbound media access control layer header, the outbound ~~network layer~~ (IP) header and the outbound ~~transport layer~~ TCP header; and

transmitting the outbound packets to the network[[;]] .

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 569

~~receiving multiple inbound packets from the network, each of the inbound packets including an inbound media access control layer header, an inbound network layer header and an inbound transport layer header;~~

~~processing the inbound packets, so that for each packet the inbound network layer header and the inbound transport layer header are validated without an interrupt dividing the processing of the inbound network layer header and the inbound transport layer header.~~

18. (currently amended) The method of claim 17, further comprising:

receiving multiple inbound packets from the network, each of the inbound packets including an inbound media access control layer header, an inbound IP header and an inbound TCP header;

processing the inbound packets, so that for each packet the inbound IP header and the inbound TCP header are validated without an interrupt dividing the processing of the inbound network layer header and the inbound transport layer header;

wherein the processing the inbound packets is performed simultaneously with the prepending the outbound packet header to each of the segments.

19. (original) The method of claim 17, further comprising creating a template header and using the template header to form each outbound packet header.

20. (currently amended) The method of claim 17, wherein ~~providing multiple segments of data includes dividing a block of data into the segments~~ the TCP connection is passed from the first mechanism to the second mechanism.

21. (original) The method of claim 20, further comprising prepending an upper layer header to the block of data, prior to dividing the block of data into multiple segments.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 570

22. (currently amended) The method of claim 17, further comprising:

receiving multiple inbound packets from the network, each of the inbound packets including an inbound media access control layer header, an inbound IP header and an inbound TCP header;

processing the inbound packets, so that for each packet the inbound IP header and the inbound TCP header are validated without an interrupt dividing the processing of the inbound network layer header and the inbound transport layer header; and

sending data from each inbound packet to a destination in memory allocated to an application without sending any of the media access control layer headers, ~~network layer~~ IP headers or ~~transport layer~~ TCP headers to the destination.

23. (currently amended) The method of claim 22, further comprising:

processing an upper layer header of at least one of the packets by ~~a~~ the second mechanism, thereby determining the destination, wherein the upper layer header corresponds to a protocol layer above the transport layer.

24. (previously presented) The method of claim 17, further comprising:

processing a transport layer header of another inbound packet, prior to receiving the plurality of packets from the network, thereby establishing a Transmission Control Protocol (TCP) connection for the inbound packets.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 571

Remarks

I. 35 U.S.C. §102

The Office Action rejects claims 1-7, 9-12, 14, 16-1 8, 20, 22, and 24 under 35 U.S.C. §102(b) as being anticipated by U.S. Patent No. 5,717,691 to Dighe et al. (hereinafter "Dighe"). Regarding claim 1, the Office Action states:

> Dighe disclosed a Multimedia Computer and Communications Platform (MCCP) that plays the role of an intelligent network interface card for running multimedia applications on existing workstations, which supports a wide range of network-edge products, such as a simple Video On Demand receiver, an advanced multimedia workstation and a small scale server (Dighe, col. 2, lines 13-20).
>
> Therefore, Dighe disclosed a method for network communication, the method comprising:
>
> receiving a plurality of packets from the network, each of the packets including a media access control layer header, a network layer header and a transport layer header (Dighe, Abstract, Dighe disclosed receiving media streams, col. 2, lines 40-45, Dighe disclosed that the MCCP includes a communications processor that terminates all the networking functions including transport processing, therefore receiving TCP/IP packets, col. 5, lines 24-25);
>
> processing the packets by a first mechanism, so that for each packet the network layer header and the transport layer header are validated without an interrupt dividing the processing of the network layer header and the transport layer header (Dighe, col. 3, lines 30-35, Dighe disclosed the MCCP handling all networking-related tasks and pass only the application-level data to media processing devices via shared memory interface);
>
> sorting the packets, dependent upon the processing, into first and second types of packets, so that the packets of the first type each contain data (Dighe, col. 4, lines 37-55, Dighe disclosed the MCCP in connection with various multimedia devices, therefore requiring the sorting of packets based on the processing of different types of multimedia for the various devices, Dighe also disclosed ATM segmentation and reassembly);
>
> sending the data from each packet of the first type to a destination without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination (Dighe, col. 3, lines 30-35, Dighe disclosed the MCCP handling all networking-related tasks and pass only the application-level data to media processing devices via shared memory interface).

Applicants respectfully note that the Office Action does not assert, and Dighe does not disclose, "sending the data from each packet of the first type to a destination in memory allocated to an application," in contrast to claim 1. For clarification, applicants

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 572

have amended claim 1 to recite in part: "sending, by the first mechanism, the data from each packet of the first type to a destination in memory allocated to an application."

Moreover, applicants respectfully assert that, without teaching how to send data to a destination in memory allocated to an application, it is at best unclear how Dighe would send data using a "shared memory interface" to a portion of memory allocated to an application as proposed in the Office Action, especially because no "shared memory interface" is actually disclosed in Dighe. Should the examiner disagree with this statement, he is respectfully requested to point out exactly what in Dighe is the "shared memory interface." Moreover, although Dighe states that its "concept aims" for laudatory improvements, such as "pass(ing) application-level data only to media processing devices" (Dighe, column 3, lines 30-35), Dighe does not state how those conceptual "aims" would be achieved and therefore does teach one of ordinary skill in the art how to practice these conceptual "aims." Should the examiner disagree with this statement, he is respectfully requested to point out where Dighe teaches that "sending, by the first mechanism, the data from each packet of the first type to a destination in memory allocated to an application," as recited in claim 1. Because this limitation is not taught or suggested in Dighe, applicants respectfully assert that claim 1 and all the claims that depend from it are patentable over Dighe.

Regarding claim 3, the Office Action states:

> Dighe disclosed the limitations, substantially as claimed, as described in claim 1, including processing an upper layer header of at least one of the packets by a second mechanism, thereby determining the destination, wherein the upper layer header corresponds to a protocol layer above the transport layer (Dighe,col. 3, lines 33-35).

Applicants respectfully disagree with this assertion. Instead, Column 3, lines 30-35 (and surrounding text) of Dighe state:

> In contrast to existing ATM host interface card architectures that rely on the host CPU for many communication functions, the MCCP concept aims to handle all networking-related tasks and pass application-level data only to media processing devices via a shared memory interface. In this way, application devices and software can be developed with a uniform "local processing" paradigm independent of network specific interfaces (which are often subject to change).

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 573

Applicants respectfully request the Examiner to point out where Dighe teaches "processing an upper layer header of at least one of the packets by a second mechanism, thereby determining the destination, wherein the upper layer header corresponds to a protocol layer above the transport layer," as recited in claim 3. Because this limitation is not taught in Dighe, applicants respectfully assert that claim 3 is patentable over Dighe.

Regarding claim 4, the Office Action states:

> Dighe disclosed the limitations, substantially as claimed, as described in claim 1, including processing an upper layer header of at least one of the packets of the second type by a second mechanism, thereby determining the destination (Dighe, col. 3, lines 33-35).

Applicants respectfully disagree with this assertion. Column 3, lines 30-35 of Dighe are quoted above and do not teach "processing an upper layer header of at least one of the packets of the second type by a second mechanism, thereby determining the destination," in contrast to claim 4. Because this limitation is not taught in Dighe, applicants respectfully assert that claim 4 is patentable over Dighe.

Regarding claim 7, the Office Action states:

> Dighe disclosed the limitations, substantially as claimed, as described in claim 1, including transmitting a second plurality of packets to the network, each of the second plurality of packets containing a media access control layer header, a network layer header and a transport layer header, including processing the second plurality of packets by the first mechanism, so that for each packet the media access control layer header, the network layer header and the transport layer header are processed without an interrupt dividing the processing of the media access control layer header (Dighe, col. 3, lines 30-35, Dighe disclosed the MCCP handling all networking related tasks and pass only the application-level data to media processing devices via shared memory interface).

Applicants have amended claim 7 to recite that "for each packet the media access control layer header, the network layer header and the transport layer header are prepended at one time as a packet header." Applicants respectfully assert that Dighe does not teach or suggest this limitation, and for at least this reason claim 7 is patentable over Dighe.

Regarding claim 9, the Office Action states:

> Dighe disclosed a method for communicating information over a network, the method comprising:

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 574

obtaining data from a source allocated by a first processor;

dividing the data into multiple segments;

prepending a packet header to each of the segments by a second processor, thereby forming a packet corresponding to each segment, each packet header containing a media access control layer header, a network layer header and a transport layer header, wherein the network layer header is Internet Protocol (IP), the transport layer header is Transmission Control Protocol (TCP) and the prepending of each packet header occurs without an interrupt dividing the prepending of the network layer header and the transport layer header;

transmitting the packets to the network (Dighe, col. 5, lines 20-40, Dighe disclosed the communications processor (CP) in the MCCP handles all the network related functions as well as the overall management of the MCCP, the CP offloading transport protocols from the host CPU, such as TCPIIP. Some of the functions typically performed by the CP include call-setup signaling and all the necessary functions to load state memory needed for the ATM function as well as the management of data movement from the network to the devices as well as from the devices to the network, Dighe, see Abstract, Dighe disclosed media stream distribution and media stream adaptation. Therefore, Dighe disclosed obtaining media streaming data from various media devices, including all networking protocols to the packets and transmitting the packets to the network).

Applicants have amended claim 9 to recite that "the media access control layer header, the network layer header and the transport layer header are prepended at one time as a sequence of bits during the prepending of each packet header." Applicants respectfully assert that Dighe does not teach or suggest this limitation, and for at least this reason claim 9 and all the claims that depend from claim 9 are patentable over Dighe.

Regarding claim 10, the Office Action states:

> Dighe disclosed the limitations, substantially as claimed, as described in claim 9, including wherein prepending a packet header to each of the segments by a second processor further comprises prepending the media access control layer header for each packet without an interrupt dividing the prepending of the media access control layer header and the network layer header (Dighe, col. 5, lines 20-40).

Applicants have amended claim 10 to recite that "each packet header is formed based upon a block of information created by the first processor." Applicants respectfully assert that Dighe does not teach or suggest this limitation, and for at least this reason claim 10 is patentable over Dighe.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 575

Regarding claim 11, the Office Action states:

> Dighe disclosed the limitations, substantially as claimed, as described in claim 9, including wherein each packet header contains an Internet Protocol (IP) header and a Transmission Control Protocol (TCP) header (Dighe, col. 5, lines 24-26).

Applicants have amended claim 11 to recite that "receiving another packet from the network, the other packet containing a receive header including information corresponding to a network layer and a transport layer; and determining, by the second processor, whether the other packet corresponds to the same TCP connection as the transmitted packets." Applicants respectfully assert that Dighe does not teach or suggest this limitation, and for at least this reason claim 11 is patentable over Dighe.

Regarding claim 12, the Office Action states:

> Dighe disclosed the limitations, substantially as claimed, as described in claim 9, including establishing a Transmission Control Protocol (TCP) connection by the first processor and using the connection to prepend the packet header to each of the segments by the second processor (Dighe, col. 5, lines 20-35).

Applicants respectfully assert that Dighe does not teach or suggest this limitation, either in the cited portion or anywhere else, and for at least this reason claim 12 is patentable over Dighe.

Regarding claim 14, the Office Action states:

> Dighe disclosed the limitations, substantially as claimed, as described in claim 9, including wherein obtaining data from the source in memory allocated by the first processor is performed by a Direct Memory Access (DMA) unit controlled by the second processor (Dighe, col. 4, lines 50-53.

Applicants respectfully assert that Dighe does not teach or suggest this limitation, either in the cited portion or anywhere else, and for at least this reason claim 14 is patentable over Dighe.

Regarding claim 16, the Office Action states:

> Dighe disclosed the limitations, substantially as claimed, as described in claim 9, including receiving another packet from the network, the other packet containing a receive header including information corresponding to a network layer and a transport layer; and

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 576

selecting whether to process the other packet by the first processor or by the second processor (Dighe, col. 3, lines 30-41).

Applicants respectfully assert that Dighe does not teach or suggest this limitation, either in the cited portion or anywhere else, and for at least this reason claim 16 is patentable over Dighe.

Regarding claim 17, the Office Action states:

> Claim 17 includes limitations that are substantially similar to the limitations of claims 1 and 9, and is therefore rejected under the same rationale.

Applicants have amended claim 17 to recite:

> providing, by a first mechanism, a block of data and a Transmission Control Protocol (TCP) connection;
>
> dividing, by a second mechanism, the block of data into multiple segments;
>
> prepending, by the second mechanism, an outbound packet header to each of the segments, thereby forming an outbound packet corresponding to each segment, the outbound packet header containing an outbound media access control layer header, an outbound Internet Protocol (IP) header and an outbound TCP header, wherein the prepending of each outbound packet header occurs without an interrupt dividing the prepending of the outbound media access control layer header, the outbound (IP) header and the outbound TCP header; and
>
> transmitting the outbound packets to the network.

Applicants respectfully assert that Dighe does not teach or suggest these limitations, and for at least this reason claim 17 and all the claims that depend from claim 17 are patentable over Dighe.

Regarding claim 18, the Office Action states:

> Dighe disclosed the limitations, substantially as claimed, as described in claim 17, including wherein the processing the inbound packets is performed simultaneously with the prepending the outbound packet header to each of the segments (Dighe, col. 4, lines 44-55, col. 7, lines 55-65).

Applicants respectfully disagree with this assertion, and note that the "outbound packet header" and the "inbound packets" recited in claim 18 both involve IP and TCP, which is quite different than the "ATM cells" and "ATM network" disclosed in the cited portion of Dighe.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 577

Regarding claim 20, the Office Action states:

> Dighe disclosed the limitations, substantially as claimed, as described in claim 17, including wherein providing multiple segments of data includes dividing a block of data into the segments (Dighe, col. 3, lines 39-41, Dighe discloses the use of media streaming).

Applicants have amended claim 20 to recite that "the TCP connection is passed from the first mechanism to the second mechanism." Applicants respectfully assert that Dighe does not teach or suggest this limitation, and for at least this reason claim 20 is patentable over Dighe.

Regarding claim 22, the Office Action states:

> Dighe disclosed the limitations, substantially as claimed, as described in claim 17, including sending data from each inbound packet to a destination in memory allocated to an application without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination (Dighe, cot. 5, lines 20-35).

Applicants respectfully assert that Dighe does not teach or suggest this limitation, either in the cited portion or anywhere else, and for at least this reason claim 22 is patentable over Dighe.

Regarding claim 24, the Office Action states:

> Dighe disclosed the limitations, substantially as claimed, as described in claim 17, including processing a transport layer header of another packet, prior to receiving the plurality of packets from the network, thereby establishing a Transmission Control Protocol (TCP) connection for the inbound packets (Dighe, col. 5, lines 20-30).

Applicants respectfully assert that Dighe does not teach or suggest this limitation, either in the cited portion or anywhere else, and for at least this reason claim 24 is patentable over Dighe.

II. 35 U.S.C. §103

A. Claim 8

The Office Action rejects claim 8 under 35 U.S.C. §103(a) as being unpatentable over Dighe in view of U.S. Patent No. 5,991,299 to Radogna et al. (hereinafter "Radogna"), stating:

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 578

> As per claim 8, Dighe disclosed the limitations, substantially as claimed, as described in claim 1, but did not explicitly state the method wherein the first mechanism is a sequencer running microcode.
>
> However, in a similar art, Radogna teaches a dedicated sequencer using microcode to perform network communication and header translation and processing (e.g. col. 4, lines 25-30).
>
> It would have been obvious to one skilled in the art at the time the invention was made to combine Radogna with Dighe because of the benefits of using a specialized processor to handle various tasks in a communications system. Using a sequencer for processing header information can greatly accelerate a frame or packet through a network since the central processing unit does not become overburdened when many packets need to be processed. This frees up the central processor to handle other networking tasks, therefore increasing the speed and efficiency of transmissions through the network. The use of software microcode for this processing easily accommodates new protocols and can bypass hardware processing in the event of a hardware failure. These are beneficial in any computer network system.

Applicants initially disagree with the Office Action assertion that "Dighe disclosed the limitations, substantially as claimed, as described in claim 1," for the reasons stated above with regard to the alleged anticipation of claim 1 by Dighe.

Moreover, applicants respectfully assert that Radogna does not teach or suggest "sending, by the first mechanism, the data from each packet of the first type to a destination in memory allocated to an application without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination," as recited in claim 1, "wherein the first mechanism is a sequencer running microcode," as recited in claim 8. For at least these reasons, applicants respectfully assert that claim 8 is patentable over Dighe and Radogna.

In addition, applicants respectfully disagree with the Office Action assertion that "These are beneficial in any computer network system." Because this assertion appears to come from the personal knowledge of the Examiner, applicants respectfully request that he provide a supporting affidavit as required by 37 CFR 1.104(d)(2).

B. Claims 13 and 19

The Office Action rejects claims 13 and 19 under 35 U.S.C. §103(a) as being unpatentable over Dighe in view of U.S. Patent No. 5,778,419 to Hansen et al. (hereinafter "Hansen"). Regarding claim 13, the Office Action states:

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 579

> As per claim 13, Dighe disclosed the limitations, substantially as claimed, as described in claim 9, but did not explicitly state the method further comprising creating a template header and forming each packet header based upon the template header.
>
> However, in a similar art, Hansen teaches the use of a header template from which all packet headers are based (e.g. Hansen, col. 6, lines 4-21).
>
> It would have been obvious to one skilled in the art at the time the invention was made to combine Hansen with Dighe because of the advantages of using a template when creating a similar header for each packet. A template is a well-known method for creating files, or in this case, a header, which needs to be attached to many packets containing altogether the same, or very similar data. The structure of each packet header should always consist of the same elements in the same arrangement so a processor does not have to locate the information it needs prior to performing processing functions. When a template is used, a large amount of time can be saved when performing a large number of transmissions, since it is not necessary to create an entire packet header during each iteration. This increases the overall speed and efficiency of the network, which is beneficial in any communication network system.

Applicants initially disagree with the Office Action assertion that "Dighe disclosed the limitations, substantially as claimed, as described in claim 9," for the reasons stated above with regard to the alleged anticipation of claim 9 by Dighe.

Moreover, applicants respectfully assert that Hansen involves DRAM rather than networking, and that the use of a template header for DRAM, which is stateless, does not teach or suggest the use of a template header for a statefull protocol such as TCP. The difficulty of maintaining the correct state for all the packets would be exacerbated by the limitations of independent claim 9 that "obtaining data from a source in memory allocated by a first processor," and "prepending a packet header to each of the segments by a second processor." Neither Dighe nor Hansen teaches or suggests any means for overcoming these difficulties, which would have dissuaded one of ordinary skill in the art from making the modification proposed in the Office Action. For at least these reasons, applicants respectfully assert that claim 13 is patentable over Dighe.

Regarding claim 19, the Office Action states:

> As per claim 19, Dighe disclosed the limitations, substantially as claimed, as described in claim 17, but did not explicitly state the method further comprising creating a template header to form each outbound packet header.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 580

However, in a similar art, Hansen teaches the use of a header template from which all packet headers are based (e.g. Hansen, col. 6, lines 4-21).

It would have been obvious to one skilled in the art at the time the invention was made to combine Hansen with Dighe for similar reasons as stated above in regards to claim 13.

Applicants initially disagree with the Office Action assertion that "Dighe disclosed the limitations, substantially as claimed, as described in claim 17," for the reasons stated above with regard to the alleged anticipation of claim 17 by Dighe.

Moreover, applicants respectfully assert that Hansen involves DRAM rather than networking, and that the use of a template header for DRAM, which is stateless, does not teach or suggest the use of a template header for a statefull protocol such as TCP. The difficulty of maintaining the correct state for all the packets would be exacerbated by the limitations of independent claim 9 that "obtaining data from a source in memory allocated by a first processor," and "prepending a packet header to each of the segments by a second processor." Neither Dighe nor Hansen teaches or suggests any means for overcoming these difficulties, which would have dissuaded one of ordinary skill in the art from making the modification proposed in the Office Action. For at least these reasons, applicants respectfully assert that claim 19 is patentable over Dighe.

C. Claims 15, 21 and 23

The Office Action rejects claims 15, 21 and 23 under 35 U.S.C. §103(a) as being unpatentable over Dighe in view of U.S. Patent No. 6,115,615 to Ota et al. (hereinafter "Ota"). Regarding claim 15, the Office Action states:

As per claim 15, Dighe disclosed the limitations, substantially as claimed, as described in claim 9, but did not explicitly state the method further comprising prepending an upper layer header to the data, prior to dividing the data into multiple segments.

However, in a similar art, Ota teaches a network communication system that attaches and uses a header in the application layer (e.g. Ota, col. 7, lines 18-25, 40-53).

It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Dighe because of the advantages of attaching a header to an upper layer, such as the application layer, along with the other layers well-known by the OSI model. The use of an upper layer header can provide a great deal of flexibility to the system since it is able to transmit more data with the packet itself. The OSI model is designed to attach and process headers from each of the seven layers

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 581

> efficiently to ensure that the data within the packet is transmitted properly across the network. Including an application layer header further ensures the proper receipt of the data. This is beneficial in any communication network system.

Applicants initially disagree with the Office Action assertion that "Dighe disclosed the limitations, substantially as claimed, as described in claim 9," for the reasons stated above with regard to the alleged anticipation of claim 9 by Dighe.

Moreover, applicants respectfully assert that Ota does not teach or suggest, "prepending an upper layer header to the data, prior to dividing the data into multiple segments," as recited in claim 15. This may be because the advantages alleged by Ota and the Office Action would not work in this case. That is, Ota allegedly "gives a unique application layer level address to a mobile station, and regards a network layer level address (IP address in this embodiment) as an address indicating a route." Ota, column 7, lines 40-43. But should such a "unique application layer level address" be prepended as "an upper layer header to the data, prior to dividing the data into multiple segments," that header would presumably only be attached to the first segment of the multiple segments, after dividing the data into multiple segments. In other words, the upper layer addressing scheme proposed by Ota would fail for all but the first packet of multiple packets, resulting in multiple problems and showing how useless the upper layer addressing scheme proposed by Ota really is. Because the OSI model does not have any mechanism for providing upper layer headers to each packet for blocks of data that are divided for transmission over a network, and the addressing scheme of Ota reduces network layer level addresses such as IP addresses as merely "indicating a route," Ota is probably inoperable, teaching one of ordinary skill in the art away from using Ota or combining it with any functional reference.

In addition, applicants respectfully disagree with the Office Action assertion that "This is beneficial in any computer network system." Because this assertion appears to come from the personal knowledge of the Examiner, applicants respectfully request that he provide a supporting affidavit as required by 37 CFR 1.104(d)(2).

For at least these reasons, applicants respectfully assert that claim 15 is nonobvious over the combination of Dighe and Ota proposed by the Office Action.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 582

Regarding claim 21, the Office Action states:

> As per claim 21, Dighe disclosed the limitations, substantially as claimed, as described in claim 20, but did not explicitly state the method further comprising prepending an upper layer header to the data, prior to dividing the data into multiple segments.
>
> However, in a similar art, Ota teaches a network communication system that attaches and uses a header in the application layer (e.g. Ota, col. 7, lines 18-25, 40-53).
>
> It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Dighe for similar reasons as stated above in regards to claim 13.

Applicants initially disagree with the Office Action assertion that "Dighe disclosed the limitations, substantially as claimed, as described in claim 20," for the reasons stated above with regard to the alleged anticipation of claim 20 by Dighe.

Moreover, applicants respectfully disagree with the Office Action assertion that it "would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Dighe for similar reasons as stated above in regards to claim 13," for at least the reasons stated above with regard to claim 13.

For at least these reasons, applicants respectfully assert that claim 21 was nonobvious over the combination of Dighe and Ota proposed by the Office Action.

Applicants have, however, amended claim 21 to recite that "the TCP connection is passed from the first mechanism to the second mechanism," another limitation that is not found in any of the cited references.

Regarding claim 23, the Office Action states:

> As per claim 23, Dighe disclosed the limitations, substantially as claimed, as described in claim 20, but did not explicitly state the method further comprising: processing an upper layer header of at least one of the packets by a second mechanism, thereby determining the destination, wherein the upper layer header corresponds to a protocol layer above the transport layer.
>
> However, in a similar art, Ota teaches a network communication system that attaches and uses a header in the application layer (e.g. Ota, col. 7, lines 18-25, 40-53).
>
> It would have been obvious to one skilled in the art at the time the invention was made to combine Ota with Dighe because of the advantages of using an upper layer header to determine the destination of packets in a network. Network layer and transport layers also generally include addresses or indications of destination for the packets and including this

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 583

> feature into the application layer as well provides another failsafe step for the network in the even of a failure in some portion of the network. Having fail-safe routes for information decreases the amount of network downtime since routes can be switched almost instantaneously upon the realization of a fault or error. This is a benefit in any communications network system.

Applicants initially disagree with the Office Action assertion that "Dighe disclosed the limitations, substantially as claimed, as described in claim 20," for the reasons stated above with regard to the alleged anticipation of claim 20 by Dighe.

Moreover, applicants respectfully note that claim 23 depends from claim 22, which recites, in part, "sending data from each inbound packet to a destination in memory allocated to an application without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination." Applicants respectfully assert that Ota does not teach "sending the data from each packet of the first type to a destination in memory allocated to an application without sending any of the media access control layer headers, network layer headers or transport layer headers to the destination." Ota instead allegedly "gives a unique application layer level address to a mobile station, and regards a network layer level address (IP address in this embodiment) as an address indicating a route." Ota, column 7, lines 40-43. For at least this reason, applicants respectfully assert that claim 23 is nonobvious over the combination of Ota and Radogna and Hendel proposed by the Office Action.

In addition, applicants respectfully disagree with the Office Action assertion that "This is beneficial in any computer network system." Because this assertion appears to come from the personal knowledge of the Examiner, applicants respectfully reiterate the request that he provide a supporting affidavit as required by 37 CFR 1.104(d)(2).

For at least these reasons, applicants respectfully assert that claim 23 is nonobvious over the combination of Dighe and Ota proposed by the Office Action.

III. Conclusion

Applicants have responded to each of the items in the Office Action, and believe that all of the pending claims are in condition for allowance. As such, applicants respectfully solicit a Notice of Allowance.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 584

Respectfully submitted,

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the U.S. Postal Service with sufficient postage as first class mail in an envelope addressed to: the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on April 16, 2007.

Date: 4-16-07

Mark Lauer

Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 585

PTO/SB/06 (07-06)
Approved for use through 1/31/2007. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PATENT APPLICATION FEE DETERMINATION RECORD Substitute for Form PTO-875	Application or Docket Number 10/260,878	Filing Date 09/27/2002	☐ To be Mailed

APPLICATION AS FILED – PART I

	(Column 1)	(Column 2)	SMALL ENTITY ☐		OR	OTHER THAN SMALL ENTITY	
FOR	NUMBER FILED	NUMBER EXTRA	RATE ($)	FEE ($)		RATE ($)	FEE ($)
☐ BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A	N/A			N/A	
☐ SEARCH FEE (37 CFR 1.16(k), (i), or (m))	N/A	N/A	N/A			N/A	
☐ EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A	N/A			N/A	
TOTAL CLAIMS (37 CFR 1.16(i))	minus 20 =	*	X $ =		OR	X $ =	
INDEPENDENT CLAIMS (37 CFR 1.16(h))	minus 3 =	*	X $ =			X $ =	
☐ APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is $250 ($125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).						
☐ MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))							
* If the difference in column 1 is less than zero, enter "0" in column 2.			TOTAL			TOTAL	

APPLICATION AS AMENDED – PART II

AMENDMENT	04/19/2007	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	SMALL ENTITY RATE ($)	ADDITIONAL FEE ($)	OR	OTHER THAN SMALL ENTITY RATE ($)	ADDITIONAL FEE ($)
	Total (37 CFR 1.16(i))	* 21	Minus	** 24	= 0	X $ =		OR	X $50=	0
	Independent (37 CFR 1.16(h))	* 3	Minus	***3	= 0	X $ =		OR	X $200=	0
	☐ Application Size Fee (37 CFR 1.16(s))									
	☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
						TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	0

AMENDMENT		(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	RATE ($)	ADDITIONAL FEE ($)		RATE ($)	ADDITIONAL FEE ($)
	Total (37 CFR 1.16(i))	*	Minus	**	=	X $ =		OR	X $ =	
	Independent (37 CFR 1.16(h))	*	Minus	***	=	X $ =		OR	X $ =	
	☐ Application Size Fee (37 CFR 1.16(s))									
	☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
						TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.

** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".

*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".

The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

Legal Instrument Examiner:
peggy s. yarborough

This collection of information is required by 37 CFR 1.16. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

EAST Search History

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	1493	protocol near10 process$4 same interrupt	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/07/05 10:48
L2	6587487	@ad<"19971014" @rlad<"19971014" @pd<"19971014"	US-PGPUB; USPAT	OR	OFF	2007/07/05 10:48
L3	466	1 and 2	US-PGPUB; USPAT	OR	OFF	2007/07/05 10:48
L4	1778	709/250.ccls.	US-PGPUB; USPAT	OR	OFF	2007/07/05 10:48
L5	22	3 and 4	US-PGPUB; USPAT	OR	OFF	2007/07/05 10:50
L6	0	BOUCHER-LAURENCE.inv.	US-PGPUB; USPAT	OR	OFF	2007/07/05 10:50
L7	86	BOUCHER-L$.inv.	US-PGPUB; USPAT	OR	OFF	2007/07/05 10:50
L8	268	(intelligent smart) near10 (nic network adj interface adj card)	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/07/05 10:54
L9	22	8 and 2	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/07/05 10:53
L10	2	9 and 4	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/07/05 10:51

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 587

EAST Search History

L11	0	protocol adj process$3 near5 (nic network adj interface adj card) same application adj data	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/07/05 10:54
L12	0	protocol near3 process$3 near5 (nic network adj interface adj card) same application adj data	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/07/05 10:54
L13	0	protocol near3 process$3 near5 (nic network adj interface adj card) same application adj data	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/07/05 10:55
L14	0	protocol near3 process$3 near5 (nic network adj interface adj card) same only near3 data	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/07/05 10:55
L15	11	protocol near3 process$3 near5 (nic network near3 card inic) same only near3 data	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/07/05 10:55
L16	1	15 and 2	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/07/05 11:02
L17	2	"20040064578"	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/07/05 11:02
S1	2684	(session same transport same network) same layer	US-PGPUB; USPAT	OR	OFF	2007/07/05 10:47

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 588

EAST Search History

S2	2560249	@ad<"19971014"	US-PGPUB; USPAT	OR	OFF	2007/07/05 10:48
S3	426	S1 and S2	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:25
S4	210	S1 same without	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:26
S5	37	S4 and S2	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:43
S6	125206	(intelligent smart) (nic network adj interface adj card)	US-PGPUB; USPAT	OR	OFF	2007/07/05 10:51
S7	108	(intelligent smart) adj (nic network adj interface adj card)	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:45
S8	3	S7 and S2	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:43
S9	1802	(nic network adj interface adj card) with processing	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:44
S10	2	S9 same S1	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:49
S11	1	"20060092934" and session	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:50
S12	1	"6,591,302".pn.	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:52
S13	47	alacritech.as.	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:52
S14	46	S13 and session	US-PGPUB; USPAT	OR	OFF	2006/06/28 14:57
S15	1	"20020091844".pn.	US-PGPUB; USPAT	OR	OFF	2006/06/28 15:45
S16	9	offload$3 same S1	US-PGPUB; USPAT	OR	OFF	2006/06/28 15:47
S17	0	S16 and S2	US-PGPUB; USPAT	OR	OFF	2006/06/28 15:47
S18	0	neurauter-g.xa. and S13	US-PGPUB; USPAT	OR	OFF	2006/06/28 15:47
S19	112	(intelligent smart) adj ((nic) (network adj interface adj card)(network adj card))	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:47
S20	2560249	@ad<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:45
S21	4	S19 and S20	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:45
S22	108	(intelligent smart) adj (nic network adj interface adj card)	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:46

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 589

EAST Search History

S23	3	S22 and S20	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:45
S24	1	S21 not S23	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:46
S25	144	offload$3 near10 ((nic) (network adj interface adj card)(network adj card))	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:48
S26	1	S25 and S20	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:47
S27	3262	bypass$3 near4 (network transport)	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:47
S28	18	S27 same ((nic) (network adj interface adj card)(network adj card))	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:54
S29	0	S28 and S20	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:54
S30	1568	709/250.ccls.	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:54
S31	336	S30 and S20	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:54
S32	76	S31 and network and transport	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:54
S33	17	S32 and ((nic) (network adj interface adj card)(network adj card))	US-PGPUB; USPAT	OR	OFF	2006/06/29 10:56
S34	10	S33 and session	US-PGPUB; USPAT	OR	OFF	2006/06/29 11:10
S35	1	"09970124"	US-PGPUB; USPAT	OR	OFF	2006/06/29 11:11
S36	1	"20020091844".pn. and (network same transport)	US-PGPUB; USPAT	OR	OFF	2006/06/29 11:13
S37	56	craft-peter$.inv.	US-PGPUB; USPAT	OR	OFF	2006/06/29 11:12
S38	53	S37 and (network same transport)	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:46

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 590

EAST Search History

S39	113	("20010004354"|"20010013059"|"20010014892"|"20010014954"|"20010025315"|"20010048681"|"20010053148"|"4336538"|"4589063"|"4991133"|"5056058"|"5058110"|"5097442"|"5163131"|"5212778"|"5280477"|"5289580"|"5303344"|"5412782"|"5418912"|"5448566"|"5485579"|"5506966"|"5511169"|"5524250"|"5535375"|"5548730"|"5566170"|"5588121"|"5590328"|"5592622"|"5598410"|"5619650"|"5629933"|"5634099"|"5634127"|"5642482"|"5664114"|"5671355"|"5678060"|"5682534"|"5692130"|"5699317"|"5701434"|"5701516"|"5727142"|"5749095"|"5751715"|"5752078"|"5758084"|"5758089"|"5758186"|"5758194"|"5771349"|"5778013"|"5790804"|"5794061"|"5802258"|"5802580"|"5809328"|"5812775"|"5815646"|"5872919"|"5878225"|"5898713"|"5913028"|"5930830"|"5931918"|"5935205"|"5937169"|"5941969"|"5941972"|"5950203"|"5970804"|"5991299"|"5996024"|"6005849"|"6009478"|"6016513"|"6021446"|"6021507"|"6026452"|"6034963"|"6044438"|"6047323"|"6047356"|"6049528"|"6057863"|"6061368"|"6065096"|"6067569"|"6070200"|"6101555"|"6141705"|"6145017"|"6157955"|"6172980"|"6173333"|"6202105"|"6226680"|"6246683"|"6247060"|"6279051"|"6298403"|"6345301"|"6356951"|"6389468"|"6427169"|"6434651"|"6449656"|"6453360"|"6490631"|"6591310").PN.	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:46
S40	1809	(nic network adj interface adj card) with processing	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:46
S41	10	S40 and S39	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:46
S42	2691	(session same transport same network) same layer	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:46
S43	12	S42 and S39	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:47
S44	2560249	@ad<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:47
S45	8	S43 and S44	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:56
S46	76	S39 and S44	US-PGPUB; USPAT	OR	OFF	2006/06/29 13:56

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 591

EAST Search History

S47	13	("6226680" "6389479" "6434620" "6470415" "6247060" "6427173" "6427171" "6807581" "6334153" "6757746" "6687758" "6658480" "6393487").pn.	US-PGPUB; USPAT	OR	OFF	2006/06/30 14:05
S48	14	("6226680" "6389479" "6434620" "6470415" "6247060" "6427173" "6427171" "6807581" "6334153" "6757746" "6687758" "6658480" "6393487" "7076568" "7042898" "6393487").pn.	US-PGPUB; USPAT	OR	OFF	2006/06/30 14:07
S49	2560281	@ad<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:14
S50	2560281	S49	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:14
S51	1581	709/250.ccls.	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:15
S52	1938	709/230.ccls.	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:15
S53	26732	header$1 with (send$3 transmit$5)	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:16
S54	5919	S49 and S53	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:16
S55	405	S53 and S52	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:16
S56	53	S55 and S49	US-PGPUB; USPAT	OR	OFF	2006/07/13 14:18
S57	40914	((media medium) adj access adj layer$1) mac	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:23
S58	947603	network transport	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:23
S59	603407	network	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:23
S60	443079	transport	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:24
S61	10455	S57 and S59 and S60	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:24
S62	1335	S57 same S59 same S60 same layer$1	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:24
S63	2560281	@ad<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:24
S64	92	S62 and S63	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:24
S65	4	S64 and 709/250.ccls.	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:25

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 592

EAST Search History

S66	1	offload$3 and S64	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:25
S67	16	offload$3 same S62	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:25
S68	0	S67 and S63	US-PGPUB; USPAT	OR	OFF	2006/07/13 15:25
S69	129196	(intelligent smart) (nic network adj interface adj card)	US-PGPUB; USPAT	OR	OFF	2006/08/17 12:50
S70	6580839	@ad<"19971014" @pd<"19971014" @rlad<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/08/17 12:50
S71	190	single adj (nic network adj interface adj card)	US-PGPUB; USPAT	OR	OFF	2006/12/28 16:36
S72	1	"5920566".pn.	US-PGPUB; USPAT	OR	OFF	2006/12/28 16:36
S73	1	"20040064578".pn.	US-PGPUB; USPAT	OR	OFF	2006/12/28 16:38
S74	2560534	@ad<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:06
S75	18	S71 and S74	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:00
S76	2	S71 same process$3 adj network	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:00
S77	0	S75 and S76	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:00
S78	7885843	@ad<"20020306" @rlad<"20020306" @pd<"20020306"	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:03
S79	139053	(intelligent smart) (nic network adj interface adj card)	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:03
S80	90582	S79 and S78	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:04
S81	118	(intelligent smart) adj (nic network adj interface adj card)	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:04
S82	81	S81 and S78	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:04
S83	2981	(session same transport same network) same layer	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:04
S84	40	S82 and S83	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:04
S85	6583981	@ad<"19971014" @rlad<"19971014" @pd<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:06
S86	8	S81 and "154"	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:06

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 593

EAST Search History

S87	4	S81 and S85	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:14

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 594

EAST Search History

S88	165	("20010004354"|"20010004354"|"20010013059"|"20010014892"|"20010014954"|"20010025315"|"20010025315"|"20010048681"|"20010053148"|"20020073223"|"20030066011"|"20030110344"|"20030165160"|"20040054814"|"20040153578"|"20040213290"|"4366538"|"4589063"|"4700185"|"4991133"|"5056058"|"5058110"|"5097442"|"5163131"|"5212778"|"5280477"|"5289580"|"5303344"|"5412782"|"5418912"|"5448566"|"5485455"|"5485460"|"5485579"|"5506966"|"5511169"|"5517668"|"5524250"|"5535375"|"5548730"|"5566170"|"5574919"|"5588121"|"5590328"|"5592622"|"5598410"|"5619650"|"5629933"|"5633780"|"5634099"|"5634127"|"5642482"|"5664114"|"5671355"|"5678060"|"5682534"|"5692130"|"5699317"|"5699350"|"5701434"|"5701516"|"5727142"|"5742765"|"5749095"|"5751715"|"5752078"|"5758084"|"5758089"|"5758186"|"5758194"|"5768618"|"5771349"|"5778013"|"5778419"|"5790804"|"5794061"|"5802258"|"5802580"|"5809328"|"5812775"|"5815646"|"5828835"|"5848293"|"5872919"|"5878225"|"5892903"|"5892903"|"5898713"|"5913028"|"5920566"|"5930830"|"5931918"|"5935205"|"5937169"|"5941969"|"5941972"|"5950203"|"5970804"|"5987022"|"5991299"|"5996013"|"5996024"|"6005849"|"6009478"|"6016513"|"6021446"|"6021507"|"6026452"|"6034963"|"6038562"|"6041058"|"6044438"|"6047323"|"6047356"|"6049528"|"6057863"|"6061368"|"6065096"|"6067569"|"6070200"|"6078564"|"6078733"|"6097734"|"6101555"|"6115615"|"6122670"|"6141701"|"6141705"|"6145017"|"6157955"|"6172980"|"6173333"|"6181705"|"6202105"|"6223242"|"6226680"|"6246683"|"6247060"|"6247169"|"6279051"|"6289023"|"6298403"|"6334153"|"6345301"|"6345302"|"6356951"|"6385647"|"6389468"|"6389479"|"6421742"|"6434651"|"6449656"|"6453360"|"6473425"|"6480489"|"6487202"|"6487654"|"6490631"|"6526446"|"6591310"|"6650640"|"6681364"|"6697868"|"6765901"|"6807581"|"6912522"|"6941386"|"6965941").PN.	US-PGPUB; USPAT	OR	OFF	2006/12/28 17:14

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 595

EAST Search History

S89	0	"5717691".pn. and S88	US-PGPUB; USPAT	OR	OFF	2006/12/28 19:10
S90	12	DIGHE-RAJIV$.inv.	US-PGPUB; USPAT	OR	OFF	2006/12/28 19:10
S91	26008	"12" and network near2 card	US-PGPUB; USPAT	OR	OFF	2006/12/28 19:10
S92	2	S90 and network near2 card	US-PGPUB; USPAT	OR	OFF	2006/12/28 19:11
S93	36	("5717691").URPN.	USPAT	OR	OFF	2006/12/28 19:11
S94	6583981	@ad<"19971014" @rlad<"19971014" @pd<"19971014"	US-PGPUB; USPAT	OR	OFF	2006/12/28 19:11
S95	11	S93 and S94	USPAT	OR	OFF	2006/12/28 19:12
S96	6	("5548587" | "5555244" | "5594732" | "5617539" | "5623495" | "5623605").PN.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/12/29 13:03
S97	27	dennison-j$.xa.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/12/29 13:04
S98	0	bilgrami-a$.xa.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/12/29 13:04
S99	0	bilgrami.xa.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/12/29 13:04
S100	0	"5717691".pn. and simultan$5	US-PGPUB; USPAT; USOCR	OR	OFF	2006/12/29 15:49
S101	0	"5717691".pn. and simultaneous$5	US-PGPUB; USPAT; USOCR	OR	OFF	2006/12/29 15:49
S102	1	"5717691".pn. and "same time"	US-PGPUB; USPAT; USOCR	OR	OFF	2006/12/29 15:49

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 596

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

NOTICE OF ALLOWANCE AND FEE(S) DUE

24501 7590 07/30/2007

MARK A LAUER
6601 KOLL CENTER PARKWAY
SUITE 245
PLEASANTON, CA 94566

EXAMINER	
DENNISON, JERRY B	
ART UNIT	PAPER NUMBER
2143	

DATE MAILED: 07/30/2007

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/260,878	09/27/2002	Laurence B. Boucher	ALA-006E	9902

TITLE OF INVENTION: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	$1400	$300	$0	$1700	10/30/2007

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. PROSECUTION ON THE MERITS IS CLOSED. THIS NOTICE OF ALLOWANCE IS NOT A GRANT OF PATENT RIGHTS. THIS APPLICATION IS SUBJECT TO WITHDRAWAL FROM ISSUE AT THE INITIATIVE OF THE OFFICE OR UPON PETITION BY THE APPLICANT. SEE 37 CFR 1.313 AND MPEP 1308.

THE ISSUE FEE AND PUBLICATION FEE (IF REQUIRED) MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED. SEE 35 U.S.C. 151. THE ISSUE FEE DUE INDICATED ABOVE DOES NOT REFLECT A CREDIT FOR ANY PREVIOUSLY PAID ISSUE FEE IN THIS APPLICATION. IF AN ISSUE FEE HAS PREVIOUSLY BEEN PAID IN THIS APPLICATION (AS SHOWN ABOVE), THE RETURN OF PART B OF THIS FORM WILL BE CONSIDERED A REQUEST TO REAPPLY THE PREVIOUSLY PAID ISSUE FEE TOWARD THE ISSUE FEE NOW DUE.

HOW TO REPLY TO THIS NOTICE:

I. Review the SMALL ENTITY status shown above.

If the SMALL ENTITY is shown as YES, verify your current SMALL ENTITY status:

A. If the status is the same, pay the TOTAL FEE(S) DUE shown above.

B. If the status above is to be removed, check box 5b on Part B - Fee(s) Transmittal and pay the PUBLICATION FEE (if required) and twice the amount of the ISSUE FEE shown above, or

If the SMALL ENTITY is shown as NO:

A. Pay TOTAL FEE(S) DUE shown above, or

B. If applicant claimed SMALL ENTITY status before, or is now claiming SMALL ENTITY status, check box 5a on Part B - Fee(s) Transmittal and pay the PUBLICATION FEE (if required) and 1/2 the ISSUE FEE shown above.

II. PART B - FEE(S) TRANSMITTAL, or its equivalent, must be completed and returned to the United States Patent and Trademark Office (USPTO) with your ISSUE FEE and PUBLICATION FEE (if required). If you are charging the fee(s) to your deposit account, section "4b" of Part B - Fee(s) Transmittal should be completed and an extra copy of the form should be submitted. If an equivalent of Part B is filed, a request to reapply a previously paid issue fee must be clearly made, and delays in processing may occur due to the difficulty in recognizing the paper as an equivalent of Part B.

III. All communications regarding this application must give the application number. Please direct all communications prior to issuance to Mail Stop ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Utility patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees. It is patentee's responsibility to ensure timely payment of maintenance fees when due.

PTOL-85 (Rev. 07/07) Approved for use through 07/31/2007.

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail **Mail Stop ISSUE FEE**
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
or Fax (571)-273-2885

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

24501 7590 07/30/2007

MARK A LAUER
6601 KOLL CENTER PARKWAY
SUITE 245
PLEASANTON, CA 94566

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission
I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO (571) 273-2885, on the date indicated below.

(Depositor's name)
(Signature)
(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/260,878	09/27/2002	Laurence B. Boucher	ALA-006E	9902

TITLE OF INVENTION: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	$1400	$300	$0	$1700	10/30/2007

EXAMINER	ART UNIT	CLASS-SUBCLASS
DENNISON, JERRY B	2143	709-230000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).
☐ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.
☐ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list
(1) the names of up to 3 registered patent attorneys or agents OR, alternatively, 1 _____
(2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed. 2 _____ 3 _____

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document has been filed for recordation as set forth in 37 CFR 3.11. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE (B) RESIDENCE: (CITY and STATE OR COUNTRY)

Please check the appropriate assignee category or categories (will not be printed on the patent) : ☐ Individual ☐ Corporation or other private group entity ☐ Government

4a. The following fee(s) are submitted:
☐ Issue Fee
☐ Publication Fee (No small entity discount permitted)
☐ Advance Order - # of Copies _____

4b. Payment of Fee(s): **(Please first reapply any previously paid issue fee shown above)**
☐ A check is enclosed.
☐ Payment by credit card. Form PTO-2038 is attached.
☐ The Director is hereby authorized to charge the required fee(s), any deficiency, or credit any overpayment, to Deposit Account Number _____ (enclose an extra copy of this form).

5. **Change in Entity Status** (from status indicated above)
☐ a. Applicant claims SMALL ENTITY status. See 37 CFR 1.27. ☐ b. Applicant is no longer claiming SMALL ENTITY status. See 37 CFR 1.27(g)(2).

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

Authorized Signature _____ Date _____

Typed or printed name _____ Registration No. _____

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

CAVIUM-1002
Cavium, Inc. v. Alacritech, Inc.
Page 598

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/260,878	09/27/2002	Laurence B. Boucher	ALA-006E	9902

24501 7590 07/30/2007

MARK A LAUER
6601 KOLL CENTER PARKWAY
SUITE 245
PLEASANTON, CA 94566

EXAMINER	
DENNISON, JERRY B	
ART UNIT	PAPER NUMBER
2143	

DATE MAILED: 07/30/2007

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The Patent Term Adjustment to date is 789 day(s). If the issue fee is paid on the date that is three months after the mailing date of this notice and the patent issues on the Tuesday before the date that is 28 weeks (six and a half months) after the mailing date of this notice, the Patent Term Adjustment will be 789 day(s).

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) WEB site (http://pair.uspto.gov).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (571)-272-7702. Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at 1-(888)-786-0101 or (571)-272-4200.

PTOL-85 (Rev. 07/07) Approved for use through 07/31/2007.

Notice of Allowability	Application No.	Applicant(s)	
	10/260,878	BOUCHER ET AL.	
	Examiner	Art Unit	
	J. Bret Dennison	2143	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to *4/19/2007*.

2. ☒ The allowed claim(s) is/are *1-24*.

3. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
   a) ☐ All b) ☐ Some* c) ☐ None of the:
      1. ☐ Certified copies of the priority documents have been received.
      2. ☐ Certified copies of the priority documents have been received in Application No. _____ .
      3. ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

   * Certified copies not received: _____.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application.
THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.

4. ☐ A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.

5. ☐ CORRECTED DRAWINGS (as "replacement sheets") must be submitted.
   (a) ☐ including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached
      1) ☐ hereto or 2) ☐ to Paper No./Mail Date _____.
   (b) ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No./Mail Date _____.

   Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet. Replacement sheet(s) should be labeled as such in the header according to 37 CFR 1.121(d).

6. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1. ☐ Notice of References Cited (PTO-892)
2. ☐ Notice of Draftperson's Patent Drawing Review (PTO-948)
3. ☒ Information Disclosure Statements (PTO/SB/08), Paper No./Mail Date _____
4. ☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material
5. ☐ Notice of Informal Patent Application
6. ☐ Interview Summary (PTO-413), Paper No./Mail Date _____ .
7. ☐ Examiner's Amendment/Comment
8. ☒ Examiner's Statement of Reasons for Allowance
9. ☐ Other _____.

DAVID WILEY
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

DETAILED ACTION

Allowable Subject Matter

Claims 1-24 are allowed in view of the Applicant's arguments (Response filed 4/19/2007) and the cited prior art of record. The independent claims recite an intelligent network interface card that processes packet headers and sorts the packets into two types and passes along only the application data to memory, which, in addition to the rest of the claim limitations, are distinguished from the prior art. For support, see Instant Specification (p7-8, 10-11).

Any comments considered necessary by applicant must be submitted no later than the payment of the issue fee and, to avoid processing delays, should preferably accompany the issue fee. Such submissions should be clearly labeled "Comments on Statement of Reasons for Allowance."

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to J. Bret Dennison whose telephone number is 571-272-3910. The examiner can normally be reached on Monday-Thursday 9am-5:30pm Eastern.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, David Wiley can be reached on 571-272-3923. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

JBD

DAVID WILEY
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE
JUL 15 2003
PATENT & TRADEMARK OFFICE

RECEIVED
JUL 17 2003
Technology Center 2100

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJL	N	United States Patent Application No. 08/964,304, by Napolitano, et al., entitled "File Array Storage Architecture", filed 11/04/97.
EJL	O	"File System Design For An NFS File Server Appliance", Article by D. Hitz, et al., 13 pages. 7/17/03
EJL	P	Adaptec Press Release article entitled "Adaptec Announces EtherStorage Technology", 2 pages, May 4, 2000, printed 6/14/00.
EJL	Q	Adaptec article entitled "EtherStorage Frequently Asked Questions", 5 pages, printed 7/19/00.

Examiner		Date Considered	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence Boucher, et al.
	Group Art Unit: 2154
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: Unknown
	Attorney Docket No.: ALA-006E

OIPE
JUL 15 2003
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

*Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date, If Appropriate
EJK	A	5,058,110	10/15/91	Beach et al.	370	85.6	
EJK	B	6,021,446	02/01/00	Gentry, Jr.	709	303	
EJK	C	6,356,951	03/12/02	Gentry, Jr.	709	250	
EJK	D	6,389,468	05/14/02	Muller et al.	709	226	
EJK	E	6,427,169	07/30/02	Elzur	709	224	
EJK	F	6,434,651	08/13/02	Gentry, Jr.	710	260	
EJK	G	6,449,656	09/10/02	Elzur et al.	709	236	
EJK	H	6,453,360	09/17/02	Muller et al.	709	250	
	I						
	J						
	K						

RECEIVED
JUL 17 2003
Technology Center 2100

Foreign Patent Documents

							Translation	
		Document Number	Date	Country	Class	Subclass	Yes	No
	L							
	M							

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EJK	N	Article from Rice University, Department of Computer Science entitled "Lazy Receiver Processing (LRP): A New Network Subsystem Architecture for Server Systems", by Peter Druschel and Gaurav Banga, 15 pages. 7/17/03
EJK	O	Internet RFC/STD/FYI/BCP Archives article with heading "RFC2140" entitled "TCP Control Block Interdependence", web address http://www.faqs.org/rfcs/rfc2140.html, 9 pages, printed 9/20/02.
	P	
	Q	

Examiner	Date Considered
E. Kim	1/13/2006

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

Issue Classification	Application/Control No.	Applicant(s)/Patent under Reexamination	
	10/260,878	BOUCHER ET AL.	
	Examiner	Art Unit	
	J. Bret Dennison	2143	

ISSUE CLASSIFICATION

ORIGINAL		CROSS REFERENCE(S)	
CLASS	SUBCLASS	CLASS	SUBCLASS (ONE SUBCLASS PER BLOCK)
709	250		
INTERNATIONAL CLASSIFICATION			
g 0 6 f	15/16		
	/		
	/		
	/		
	/		

(Assistant Examiner) (Date)

(Legal Instruments Examiner) (Date)

DAVID WILEY
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100
(Primary Examiner) (Date)

Total Claims Allowed: 24

O.G. Print Claim(s)	O.G. Print Fig.
1	2

☒ Claims renumbered in the same order as presented by applicant ☐ CPA ☐ T.D. ☐ R.1.47

Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original
	1		31		61		91		121		151		181
	2		32		62		92		122		152		182
	3		33		63		93		123		153		183
	4		34		64		94		124		154		184
	5		35		65		95		125		155		185
	6		36		66		96		126		156		186
	7		37		67		97		127		157		187
	8		38		68		98		128		158		188
	9		39		69		99		129		159		189
	10		40		70		100		130		160		190
	11		41		71		101		131		161		191
	12		42		72		102		132		162		192
	13		43		73		103		133		163		193
	14		44		74		104		134		164		194
	15		45		75		105		135		165		195
	16		46		76		106		136		166		196
	17		47		77		107		137		167		197
	18		48		78		108		138		168		198
	19		49		79		109		139		169		199
	20		50		80		110		140		170		200
	21		51		81		111		141		171		201
	22		52		82		112		142		172		202
	23		53		83		113		143		173		203
	24		54		84		114		144		174		204
	25		55		85		115		145		175		205
	26		56		86		116		146		176		206
	27		57		87		117		147		177		207
	28		58		88		118		148		178		208
	29		59		89		119		149		179		209
	30		60		90		120		150		180		210

Index of Claims	Application/Control No.	Applicant(s)/Patent Under Reexamination
	10260878	BOUCHER ET AL.
	Examiner	Art Unit
	Dennison, J. Bret	2143

✓	Rejected	-	Cancelled	N	Non-Elected	A	Appeal
=	Allowed	÷	Restricted	I	Interference	O	Objected

☐ Claims renumbered in the same order as presented by applicant ☐ CPA ☐ T.D. ☐ R.1.47

CLAIM		DATE								
Final	Original	07/05/2007								
	1	=								
	2	=								
	3	=								
	4	=								
	5	=								
	6	=								
	7	=								
	8	=								
	9	=								
	10	=								
	11	=								
	12	=								
	13	=								
	14	=								
	15	=								
	16	=								
	17	=								
	18	=								
	19	=								
	20	=								
	21	=								
	22	=								
	23	=								
	24	=								

Search Notes	Application/Control No.	Applicant(s)/Patent under Reexamination
	10/260,878	BOUCHER ET AL.
	Examiner	Art Unit
	J. Bret Dennison	2143

SEARCHED			
Class	Subclass	Date	Examiner
709	250	7/5/2007	JBD

INTERFERENCE SEARCHED			
Class	Subclass	Date	Examiner
709	**250**	7/5/2007	**JBD**

SEARCH NOTES (INCLUDING SEARCH STRATEGY)		
	DATE	EXMR
Performed Inventor search in PALM	1/13/2006	EJK
Searched EAST Databases See attached Search History	1/16/2006	EJK
Conducted PLUS search	1/6/2006	EJK
Spoke with John Follansbee	12/29/2006	JBD
David Wiley - allowable subj matter East Search Notes Inventor Search Double Patenting search on related cases.	7/5/2007	JBD

OIPE IAP6
AUG 06 2007
PATENT & TRADEMARK OFFICE

2143

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of Laurence B. Boucher et al.		Ser. No:	10/260,878
Filing Date:	September 27, 2002	Examiner:	J. Bret Dennison
Atty. Docket No:	ALA-006E	GAU:	2143

For: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

August 2, 2007

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

8th Supplemental Information Disclosure Statement

Sir:

Pursuant to 37 C.F.R. §§ 1.56, 1.97 and 1.98, applicants bring thirty-seven reference documents to the Examiner's attention. Included is a three-page form PTO-1449 listing the thirty-seven U.S. Patent reference documents. Copies of the thirty-seven U.S. Patent reference documents are not enclosed.

Citation of these documents shall not be construed as an admission that the documents are prior art with respect to the instant invention, a representation that a search has been made, or an admission that the information cited herein is, or is considered to be, material to patentability as defined in 37 C.F.R. § 1.56(b).

CERTIFICATE OF MAILING
I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on August 2, 2007.

Date: 8-2-07

Mark Lauer

Respectfully submitted,

Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence B. Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J. B. Dennison
	Attorney Docket No.: ALA-006E

OIPE AUG 06 2007 PATENT & TRADEMARK OFFICE

U.S. Patent Documents

Examiner Initial		Document	Date	Name	Class	Subclass	Filing Date
	1	4,485,455	November 27, 1984	Gary W. Boone et al	364	900	
	2	4,485,460	November 27, 1984	Mark A. Stambaugh	365	203	
	3	4,700,185	October 13, 1987	Thomas J. Balph et al	340	825.5	
	4	5,566,170	October 15, 1996	Bakke et al.	370	60	
	5	5,574,919	November 12, 1996	Arun N. Netravali et al	395	561	
	6	5,699,350	December 16, 1997	Andrew J. Kraslavsky	370	254	
	7	5,768,618	June 16, 1998	Gene R. Erickson et al	395	829	
	8	5,828,835	October 27, 1998	Mark S. Isfeld et al	395	200.3	
	9	5,987,022	November 16, 1999	Robert L. Geiger et al	370	349	
	10	5,996,013	November 30, 1999	Gary Scott Delp et al	709	226	
	11	6,041,381	March 21, 2000	Geoffrey B. Hoese	710	129	
	12	6,111,673	August 29, 2000	Gee-Kung Chang et al	359	123	
	13	6,141,701	October 31, 2000	Mark M. Whitney	710	5	
	14	6,181,705	February 24, 2004	San-Hong Kim	370	392	
	15	6,324,649	November 27, 2001	Kevin W. Eyres et al	713	202	
	16	6,343,360	January 29, 2002	David Feinleib	713	1	

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence B. Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J. B. Dennison
	Attorney Docket No.: ALA-006E

	17	6,370,599	April 9, 2002	Sanjay Anand et al	710	15	
	18	6,421,753	July 16, 2002	Geoffrey B. Hoese et al	710	129	
	19	6,427,173	July 30, 2002	Laurence B. Boucher et al	709	238	
	20	6,473,425	October 29, 2002	Gilles Bellaton et al	370	392	
	21	6,487,202	November 26, 2002	Daniel E. Klausmeier et al	370	395	
	22	6,502,144	December 31, 2002	Jean-Paul Accarie	710	8	
	23	6,523,119	February 18, 2003	Dominique Vincent Pavlin et al	713	192	
	24	6,570,884	May 27, 2003	Glenn William Connery et al	370	419	
	25	6,648,611	November 18, 2003	David M. Morse et al	417	310	
	26	6,657,757	December 2, 2003	Gee-Kung Chang et al	359	124	
	27	6,658,480	December 2, 2003	Laurence B. Boucher et al	709	239	
	28	6,678,283	January 13, 2004	Yakov Teplitsky	370	463	
	29	6,765,901	July 20, 2004	Michael Ward Johnson et al	370	352	
	30	6,842,896	January 11, 2005	Mark E. Redding et al	717	172	
	31	6,965,941	November 15, 2005	Laurence B. Boucher et al	709	230	

Examiner	Date Considered
*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.	

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence B. Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J. B. Dennison
	Attorney Docket No.: ALA-006E

Published Applications

Examiner Initial		Document	Date	Name	Class	Subclass	Filing Date
	32	2002/0073223	June 13, 2002	B. Scott Darnell et al	709	232	
	33	2002/0112175	August 15, 2002	Makofka et al	713	200	
	34	2003/0110344	June 1, 2003	Szezepanek et al.	711	100	
	35	2004/0059926	March 25, 2004	Angelo, et al.	713	168	
	36	2004/0213290	October 1, 2004	Johnson et al.	370	469	
	37	2004/0246974	December 9, 2004	Gyugyi et al.	370	395.31	

Examiner	Date Considered
*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.	

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

BIBDATASHEET

Bib Data Sheet

CONFIRMATION NO. 9902

SERIAL NUMBER	FILING OR 371(c) DATE	CLASS	GROUP ART UNIT	ATTORNEY DOCKET NO.
10/260,878	09/27/2002 RULE	709	2143	ALA-006E

APPLICANTS

Laurence B. Boucher, Saratoga, CA;
Stephen E. J. Blightman, San Jose, CA;
Peter K. Craft, San Francisco, CA;
David A. Higgen, Saratoga, CA;
Clive M. Philbrick, San Jose, CA;
Daryl D. Starr, Milpitas, CA;

** CONTINUING DATA **********

** FOREIGN APPLICATIONS **********

IF REQUIRED, FOREIGN FILING LICENSE GRANTED
** 11/01/2002

		STATE OR COUNTRY	SHEETS DRAWING	TOTAL CLAIMS	INDEPENDENT CLAIMS
Foreign Priority claimed	☐ yes ☑ no	CA	89	24	3
35 USC 119 (a-d) conditions met	☐ yes ☐ no ☐ Met after Allowance				
Verified and Acknowledged	Examiner's Signature Initials				

ADDRESS
24501

TITLE

FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

FILING FEE RECEIVED		
812	FEES: Authority has been given in Paper No. _____ to charge/credit DEPOSIT ACCOUNT No. _____ for following:	☐ All Fees ☐ 1.16 Fees (Filing) ☐ 1.17 Fees (Processing Ext. of time) ☐ 1.18 Fees (Issue) ☐ Other _____ ☐ Credit

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/260,878	09/27/2002	Laurence B. Boucher	ALA-006E	9902

24501 7590 09/07/2007

MARK A LAUER
6601 KOLL CENTER PARKWAY
SUITE 245
PLEASANTON, CA 94566

EXAMINER	
DENNISON, JERRY B	
ART UNIT	**PAPER NUMBER**
2143	

MAIL DATE	DELIVERY MODE
09/07/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

UNITED STATES DEPARTMENT OF COMMERCE
U.S. Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450

APPLICATION NO./ CONTROL NO.	FILING DATE	FIRST NAMED INVENTOR / PATENT IN REEXAMINATION	ATTORNEY DOCKET NO.
10260878			

EXAMINER	
ART UNIT	PAPER
	[redacted]

DATE MAILED:

NOTICE OF NON-COMPLIANT INFORMATION DISCLOSURE STATEMENT

An Information Disclosure Statement (IDS) filed 9-6-07 in the above-identified application fails to meet the requirements of 37 CFR 1.97(d) for the reason(s) specified below. Accordingly, the IDS will be placed in the file, but the information referred to therein has not been considered.

The IDS is not compliant with 37 CFR 1.97(d) because:

☛ The IDS lacks a statement as specified in 37 CFR 1.97(e).

☛ The IDS lacks the fee set forth in 37 CFR 1.17(p).

☐ The IDS was filed after the issue fee was paid. Applicant may wish to consider filing a petition to withdraw the application from issue under 37 CFR 1.313(c) to have the IDS considered. See MPEP 1308.

TC
initials

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail Mail Stop ISSUE FEE
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
or Fax (571)-273-2885

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

24501 7590 07/30/2007

MARK A LAUER
6601 KOLL CENTER PARKWAY
SUITE 245
PLEASANTON, CA 94566

10/17/2007 SSESHE2 00000005 10260878

01 FC:1504 300.00 OP
02 FC:1506 1400.00 OP

OIPE
OCT 16 2007
PATENT & TRADEMARK OFFICE

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission
I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO (571) 273-2885, on the date indicated below.

Mark Lauer	(Depositor's name)
	(Signature)
10-12-07	(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/260,878	09/27/2002	Laurence B. Boucher	ALA-006E	9902

TITLE OF INVENTION: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	$1400	$300	$0	$1700	10/30/2007

EXAMINER	ART UNIT	CLASS-SUBCLASS
DENNISON, JERRY B	2143	709-230000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).
- ☐ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.
- ☐ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list
(1) the names of up to 3 registered patent attorneys or agents OR, alternatively,
(2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1 Mark Lauer
2 Silicon Edge Law Group LLP
3

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document has been filed for recordation as set forth in 37 CFR 3.11. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE: Alacritech, Inc.

(B) RESIDENCE: (CITY and STATE OR COUNTRY): San Jose, California

Please check the appropriate assignee category or categories (will not be printed on the patent): ☐ Individual ☒ Corporation or other private group entity ☐ Government

4a. The following fee(s) are submitted:
- ☒ Issue Fee
- ☒ Publication Fee (No small entity discount permitted)
- ☐ Advance Order - # of Copies _____

4b. Payment of Fee(s): **(Please first reapply any previously paid issue fee shown above)**
- ☒ A check is enclosed.
- ☐ Payment by credit card. Form PTO-2038 is attached.
- ☐ The Director is hereby authorized to charge the required fee(s), any deficiency, or credit any overpayment, to Deposit Account Number _____ (enclose an extra copy of this form).

5. **Change in Entity Status** (from status indicated above)
- ☐ a. Applicant claims SMALL ENTITY status. See 37 CFR 1.27.
- ☐ b. Applicant is no longer claiming SMALL ENTITY status. See 37 CFR 1.27(g)(2).

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

Authorized Signature _____ Date 10-12-07

Typed or printed name Mark Lauer Registration No. 36,578

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

OIPE
OCT 16 2007
PATENT & TRADEMARK OFFICE

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of Laurence B. Boucher, et al.		Ser. No:	10/260,878
Filing Date:	September 27, 2002	Examiner:	J. Bret Dennison
Atty. Docket No:	ALA-006E	GAU:	2154

For: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

October 12, 2007

MS Issue Fee
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Comments on Statement of Reasons for Allowance

Sir:

Applicants respectfully disagree with the statement, in a Notice of Allowability dated January 16, 2007, that: "The independent claims recite an intelligent network interface card that processes packet headers and sorts the packets into two types and passes, along only the application data to memory..." Applicants note that the claims define methods, not devices, and nowhere in the claims is "an intelligent network interface card" recited. Should the Examiner somehow disagree with this statement, he is respectfully requested to withdraw this application from allowance and refund the accompanying Issue Fee.

Respectfully submitted,

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the U.S. Postal Service with sufficient postage for first class mail in an envelope addressed to: the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on October 12, 2007.

Date: 10-12-07

Mark Lauer

Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

Document Code: IMIS

Notice of Fee Due

Date: 10-17-07

Application Number: 10/200,878

A fee is due for the attached document for the reason indicated below. Please check the application for the appropriate authorization to charge a deposit account. If an authorization is present, please charge the appropriate fee*. If an authorization is not present, notify the application of the fee deficiency.

***If the fee due is for any of the filing fees, check for authorization to charge the surcharge. If authorization is present, charge the surcharge for late payment of the filing fees as well.**

- ☑ Insufficient payment by check or money order.
- ☐ Insufficient funds in deposit account _____.
- ☐ Insufficient payment by credit card.
- ☐ Declined credit card.
- ☑ No authorization to charge a deposit account.

Fee code(s) to be applied:	1501	1440
	1506	1450
Amount in holding fee code:	1622	
	2622	
	1999	
Total remaining due from applicant:		40

RAM Operator_____

Rev. 4/20/06

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

MARK A LAUER
6601 KOLL CENTER PARKWAY
SUITE 245
PLEASANTON, CA 94566

Mail Date: 10/25/07
Application Number: 10/260878

NOTICE TO PAY BALANCE OF ISSUE FEE

The issue fee payment filed on 10/12/07 has been received. Although the fee paid in the Notice of Allowance was paid, new patent fees went into effect on September 30, 2007 after the mailing date of the Notice. See the Sections 801 and 803 of the Consolidated Appropriations Act, 2005 (H.R. 4818). As stated in Section 803 of the Act, "the provisions of this title shall take effect on the date of enactment of this Act. . . the provisions of section 801 shall apply to all patents, whenever granted, and to all patent applications pending on or filed after the effective date [September 30,2007]. Because the issue fee was paid on or after September 30, 2007, the new issue fee was due instead of the amount specified in the Notice of Allowance.[1]

In accordance with 37 CFR 1.18, applicant is given a time period of **THREE (3) MONTHS** from the mailing date of this notice during which to pay the **BALANCE DUE** indicated below. The balance due is the difference between the issue fee required on the date that the correct issue fee is paid and the amount that was previously paid. This three-month time period may not be extended. If the balance due is not paid before the expiration of the three-month period, the application will become abandoned (if not issued) or the patent will lapse (if issued) at the termination of the three-month period.

App. Type	Column A Issue Fee Req. large entity / small entity		Column B Issue Fee PAID	Balance Due. Col. A minus Col. B
UTILITY or REISSUE	$1,440.00/ $720.00	$	1400.00	$ 40.00
DESIGN	$820.00 / $410.00	$		$
PLANT	$1,130.00 / $565.00	$		$

Krystal Paige
Office: 703-308-9250 x139
Fax: 571-270-9937

A copy of this notice MUST be returned with payment.

CERTIFICATE OF MAILING

I hereby certify that this notice and the required additional fee are being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to Mail Stop Issue Fee, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on the date indicated below.

Printed Name: _____ Signature: _____

Registration Number, if any_____ Date: _____

[1] Applicants should check the current fee schedule posted on the USPTO Internet web site at: http://www.uspto.gov/main/howtofees.htm before paying the balance due in order to ensure that the correct issue fee is paid.

UNITED STATES PATENT AND TRADEMARK OFFICE

OIPE
NOV 28 2007
PATENT & TRADEMARK OFFICE
IAP35

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

MARK A LAUER
6601 KOLL CENTER PARKWAY
SUITE 245
PLEASANTON, CA 94566

Mail Date: 10/25/07
Application Number: 10/260878

NOTICE TO PAY BALANCE OF ISSUE FEE

The issue fee payment filed on 10/12/07 has been received. Although the fee paid in the Notice of Allowance was paid, new patent fees went into effect on September 30, 2007 after the mailing date of the Notice. See the Sections 801 and 803 of the Consolidated Appropriations Act, 2005 (H.R. 4818). As stated in Section 803 of the Act, "the provisions of this title shall take effect on the date of enactment of this Act. . . the provisions of section 801 shall apply to all patents, whenever granted, and to all patent applications pending on or filed after the effective date [September 30,2007]. Because the issue fee was paid on or after September 30, 2007, the new issue fee was due instead of the amount specified in the Notice of Allowance.[1]

In accordance with 37 CFR 1.18, applicant is given a time period of **THREE (3) MONTHS** from the mailing date of this notice during which to pay the **BALANCE DUE** indicated below. The balance due is the difference between the issue fee required on the date that the correct issue fee is paid and the amount that was previously paid. This three-month time period may not be extended. If the balance due is not paid before the expiration of the three-month period, the application will become abandoned (if not issued) or the patent will lapse (if issued) at the termination of the three-month period.

App. Type	Column A Issue Fee Req. large entity / small entity		Column B Issue Fee PAID	Balance Due. Col. A minus Col. B
UTILITY or REISSUE	$1,440.00/ $720.00	$	1400.00	$ 40.00
DESIGN	$820.00 / $410.00	$		$
PLANT	$1,130.00 / $565.00	$		$

Krystal Paige
Office: 703-308-9250 x139
Fax: 571-270-9937

A copy of this notice MUST be returned with payment.

CERTIFICATE OF MAILING

I hereby certify that this notice and the required additional fee are being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to Mail Stop Issue Fee, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on the date indicated below.

Printed Name: Mark Lauer Signature:

Registration Number, if any 36578 Date: 11-21-07

[1] Applicants should check the current fee schedule posted on the USPTO Internet web site at: http://www.uspto.gov/main/howtofees.htm before paying the balance due in order to ensure that the correct issue fee is paid.

12/05/2007 GTEKLE1 00000005 10260878
01 FC:1501 1400.00 OP
Adjustment date: 11/28/2007 FHETEKI
10/17/2007 SSESHE2 00000005 10260878
02 FC:1506 -1400.00 OP

EAST Search History

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L2	73	("4485455" "4485460" "4700185" "5566170" "5574919" "5699350" "5768618" "5828835" "5987022" "5996013" "6041381" "6111673" "6141701" "6181705" "6324649" "6343360 6370599" "6421753" "6427173" "6473425" "6487202" "6502144" "6523119" "6570884" "6648611" "6657757" "6658480" "6678283" "6765901" "6842896" "6965941" "20020073223" "20020112175" "20030110344" "2004059926" "20040213290" "20040246974").pn.	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/12/20 09:26
L3	34009650	@ad<"20020927" @rlad<"20020927" @pd<"20020927"	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/12/20 09:26
L4	60	1 and 2	US-PGPUB; USPAT; USOCR; FPRS; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/12/20 09:26

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/260,878	09/27/2002	Laurence B. Boucher	ALA-006E	9902

24501 7590 01/14/2008

MARK A LAUER
6601 KOLL CENTER PARKWAY
SUITE 245
PLEASANTON, CA 94566

EXAMINER	
DENNISON, JERRY B	
ART UNIT	**PAPER NUMBER**
2143	

MAIL DATE	DELIVERY MODE
01/14/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

UNITED STATES DEPARTMENT OF COMMERCE
U.S. Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450

APPLICATION NO./ CONTROL NO.	FILING DATE	FIRST NAMED INVENTOR / PATENT IN REEXAMINATION	ATTORNEY DOCKET NO.
10260878	9/27/02	BOUCHER ET AL.	ALA-006E

MARK A LAUER
6601 KOLL CENTER PARKWAY
SUITE 245
PLEASANTON, CA 94566

EXAMINER
J. Bret Dennison

ART UNIT	PAPER
2143	20071220

DATE MAILED:

Please find below and/or attached an Office communication concerning this application or proceeding.

Commissioner for Patents

PTO-90C (Rev.04-03)

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence B. Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J. B. Dennison
	Attorney Docket No.: ALA-006E

OIPE
AUG 06 2007
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

Examiner Initial		Document	Date	Name	Class	Subclass	Filing Date
JR	1	4,485,455	November 27, 1984	Gary W. Boone et al	364	900	
	2	4,485,460	November 27, 1984	Mark A. Stambaugh	365	203	
	3	4,700,185	October 13, 1987	Thomas J. Balph et al	340	825.5	
	4	5,566,170	October 15, 1996	Bakke et al.	370	60	
	5	5,574,919	November 12, 1996	Arun N. Netravali et al	395	561	
	6	5,699,350	December 16, 1997	Andrew J. Kraslavsky	370	254	
	7	5,768,618	June 16, 1998	Gene R. Erickson et al	395	829	
	8	5,828,835	October 27, 1998	Mark S. Isfeld et al	395	200.3	
	9	5,987,022	November 16, 1999	Robert L. Geiger et al	370	349	
	10	5,996,013	November 30, 1999	Gary Scott Delp et al	709	226	
	11	6,041,381	March 21, 2000	Geoffrey B. Hoese	710	129	
	12	6,111,673	August 29, 2000	Gee-Kung Chang et al	359	123	
	13	6,141,701	October 31, 2000	Mark M. Whitney	710	5	
	14	6,181,705	February 24, 2004	San-Hong Kim	370	392	
	15	6,324,649	November 27, 2001	Kevin W. Eyres et al	713	202	
JR	16	6,343,360	January 29, 2002	David Feinleib	713	1	

Examiner JR	Date Considered 12/20/07

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence B. Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J. B. Dennison
	Attorney Docket No.: ALA-006E

	17	6,370,599	April 9, 2002	Sanjay Anand et al	710	15	
	18	6,421,753	July 16, 2002	Geoffrey B. Hoese et al	710	129	
	19	6,427,173	July 30, 2002	Laurence B. Boucher et al	709	238	
	20	6,473,425	October 29, 2002	Gilles Bellaton et al	370	392	
	21	6,487,202	November 26, 2002	Daniel E. Klausmeier et al	370	395	
	22	6,502,144	December 31, 2002	Jean-Paul Accarie	710	8	
	23	6,523,119	February 18, 2003	Dominique Vincent Pavlin et al	713	192	
	24	6,570,884	May 27, 2003	Glenn William Connery et al	370	419	
	25	6,648,611	November 18, 2003	David M. Morse et al	417	310	
	26	6,657,757	December 2, 2003	Gee-Kung Chang et al	359	124	
	27	6,658,480	December 2, 2003	Laurence B. Boucher et al	709	239	
	28	6,678,283	January 13, 2004	Yakov Teplitsky	370	463	
	29	6,765,901	July 20, 2004	Michael Ward Johnson et al	370	352	
	30	6,842,896	January 11, 2005	Mark E. Redding et al	717	172	
	31	6,965,941	November 15, 2005	Laurence B. Boucher et al	709	230	

Examiner	Date Considered 12/20/07

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence B. Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J. B. Dennison
	Attorney Docket No.: ALA-006E

Published Applications

Examiner Initial		Document	Date	Name	Class	Subclass	Filing Date
	32	2002/0073223	June 13, 2002	B. Scott Darnell et al	709	232	
	33	2002/0112175	August 15, 2002	Makofka et al	713	200	
	34	2003/0110344	June 1, 2003	Szezepanek et al.	711	100	
	35	2004/0059926	March 25, 2004	Angelo, et al.	713	168	
	36	2004/0213290	October 1, 2004	Johnson et al.	370	469	
	37	2004/0246974	December 9, 2004	Gyugyi et al.	370	395.31	

Examiner	Date Considered 12/20/07

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

BIBDATASHEET

Bib Data Sheet

CONFIRMATION NO. 9902

SERIAL NUMBER	FILING OR 371(c) DATE	CLASS	GROUP ART UNIT	ATTORNEY DOCKET NO.
10/260,878	09/27/2002 RULE	709	2143	ALA-006E

APPLICANTS

Laurence B. Boucher, Saratoga, CA;
Stephen E. J. Blightman, San Jose, CA;
Peter K. Craft, San Francisco, CA;
David A. Higgen, Saratoga, CA;
Clive M. Philbrick, San Jose, CA;
Daryl D. Starr, Milpitas, CA;

** CONTINUING DATA **********

** FOREIGN APPLICATIONS **********

IF REQUIRED, FOREIGN FILING LICENSE GRANTED
** 11/01/2002

Foreign Priority claimed	STATE OR COUNTRY	SHEETS DRAWING	TOTAL CLAIMS	INDEPENDENT CLAIMS
☐ yes ☑ no	CA	89	24	3
35 USC 119 (a-d) conditions met ☐ yes ☐ no ☐ Met after Allowance				
Verified and Acknowledged Examiner's Signature Initials				

ADDRESS
24501

TITLE
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

FILING FEE RECEIVED	FEES: Authority has been given in Paper No. _____ to charge/credit DEPOSIT ACCOUNT No. _____ for following:	☐ All Fees ☐ 1.16 Fees (Filing) ☐ 1.17 Fees (Processing Ext. of time) ☐ 1.18 Fees (Issue) ☐ Other _____ ☐ Credit
812		

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	ISSUE DATE	PATENT NO.	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/260,878	02/26/2008	7337241	ALA-006E	9902

24501 7590 02/06/2008

MARK A LAUER
6601 KOLL CENTER PARKWAY
SUITE 245
PLEASANTON, CA 94566

ISSUE NOTIFICATION

The projected patent number and issue date are specified above.

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The Patent Term Adjustment is 879 day(s). Any patent to issue from the above-identified application will include an indication of the adjustment on the front page.

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) WEB site (http://pair.uspto.gov).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (571)-272-7702. Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at (571)-272-4200.

APPLICANT(s) (Please see PAIR WEB site http://pair.uspto.gov for additional applicants):

Laurence B. Boucher, Saratoga, CA;
Stephen E. J. Blightman, San Jose, CA;
Peter K. Craft, San Francisco, CA;
David A. Higgen, Saratoga, CA;
Clive M. Philbrick, San Jose, CA;
Daryl D. Starr, Milpitas, CA;

OIPE AP14
FEB 11 2008
PATENT & TRADEMARK OFFICE

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of Laurence B. Boucher et al.		Ser. No:	10/260,878
Filing Date:	September 27, 2002	Examiner:	J. Bret Dennison
Atty. Docket No:	ALA-006E	GAU:	2143

For: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

February 8, 2008

MS AF
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

9th Supplemental Information Disclosure Statement

Sir:

Pursuant to 37 C.F.R. §§ 1.56, 1.97(i) and 1.98, applicants bring two U.S. Patent documents to the Examiner's attention. Copies of the two U.S. Patent documents are not enclosed.

Citation of these documents shall not be construed as an admission that the documents are prior art with respect to the instant invention, a representation that a search has been made, or an admission that the information cited herein is, or is considered to be, material to patentability as defined in 37 C.F.R. § 1.56(b).

Respectfully submitted,

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with sufficient postage in the US Postal Service as first class mail in an envelope addressed to: MS AF, Commissioner for Patents, P.O. Box 1450,Alexandria, VA 22313-1450, February 8, 2008.

Date: 2-8-08

Mark Lauer

Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence B. Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J. B. Dennison
	Attorney Docket No.: ALA-006E

PTO/SB/08A
FEB 11 2008
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

Examiner Initial		Document	Date	Name	Class	Subclass	Filing Date
	1	5,774,660	6/30/98	Brendel et al.	395	200.31	
	2	5,809,527	9/15/98	Cooper et al.	711	133	
	3						
	4						
	5						
	6						
	7						
	8						
	9						
	10						
	11						
	12						
	13						
	14						
	15						
	16						

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

OIPE
FEB 19 2008
PATENT & TRADEMARK OFFICE

AF 2143

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of Laurence B. Boucher et al.		Ser. No:	10/260,878
Filing Date:	September 27, 2002	Examiner:	J. Bret Dennison
Atty. Docket No:	ALA-006E	GAU:	2143

For: FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION

February 15, 2008

MS AF
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

9th Supplemental Information Disclosure Statement

Sir:

Pursuant to 37 C.F.R. §§ 1.56, 1.97(i) and 1.98, applicants bring twenty-five U.S. Patent documents to the Examiner's attention. Copies of the twenty-five U.S. Patent documents are not enclosed.

Citation of these documents shall not be construed as an admission that the documents are prior art with respect to the instant invention, a representation that a search has been made, or an admission that the information cited herein is, or is considered to be, material to patentability as defined in 37 C.F.R. § 1.56(b).

Respectfully submitted,

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with sufficient postage in the US Postal Service as first class mail in an envelope addressed to: MS AF, Commissioner for Patents, P.O. Box 1450,Alexandria, VA 22313-1450, February 15, 2008.

Date: 2-15-08

Mark Lauer

Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 484-9295
Fax: (925) 484-9291

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence B. Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J. B. Dennison
	Attorney Docket No.: ALA-006E

PTO/SB/08A (10-01)
Approved for use through 07/31/2006. OMB 0651-0031
FEB 19 2008
PATENT & TRADEMARK OFFICE

U.S. Patent Documents

Examiner Initial		Document	Date	Name	Class	Subclass	Filing Date
	1	6,393,487	May 21, 2002	Boucher, et al.	709	238	
	2	6,427,171	July 30, 2002	Craft, et al.	709	230	
	3	6,434,620	August 13, 2002	Boucher, et al.	709	230	
	4	6,470,415	October 22, 2002	Starr, et al.	711	104	
	5	6,591,302	July 8, 2003	Boucher, et al.	709	230	
	6	6,687,758	February 3, 2004	Craft, et al.	709	250	
	7	6,751,665	June 15, 2004	Philbrick, et al.	709	224	
	8	6,757,746	June 29, 2004	Boucher, et al.	709	250	
	9	6,938,092	August 30, 2005	Burns	709	230	
	10	6,996,070	February 7, 2006	Starr, et al.	370	252	
	11	7,042,898	May 9, 2006	Blightman, et al.	370	463	
	12	7,076,568	July 11, 2006	Philbrick, et al.	709	250	
	13	7,089,326	August 8, 2006	Boucher, et al.	709	242	
	14	7,093,099	August 15, 2006	Bodas, et al.	711	206	
	15	7,124,205	October 17, 2006	Craft, et al.	709	250	
	16	7,133,940	November 7, 2006	Blightman, et al.	710	22	

Examiner	Date Considered

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.

U.S. Department of Commerce, Patent and Trademark Office	Application No.: 10/260,878
INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Filing date: September 27, 2002
	Inventors: Laurence B. Boucher, et al.
	Group Art Unit: 2143
FAST-PATH APPARATUS FOR RECEIVING DATA CORRESPONDING TO A TCP CONNECTION	Examiner name: J. B. Dennison
	Attorney Docket No.: ALA-006E

	17	7,167,926	January 23, 2007	Boucher, et al.	709	250	
	18	7,167,927	January 23, 2007	Philbrick, et al.	709	250	
	19	7,174,393	February 6, 2007	Boucher, et al.	709	250	
	20	7,185,266	February 27, 2007	Blightman, et al.	714	776	
	21	7,191,241	March 13, 2007	Boucher, et al.	709	230	
	22	7,191,318	March 13, 2007	Tripathy, et al.	712	225	
	23	7,237,036	June 26, 2007	Boucher, et al.	709	245	
	24	7,254,696	August 7, 2007	Mittal, et al.	712	210	
	25	7,284,070	October 16, 2007	Boucher, et al.	709	250	

Examiner	Date Considered
*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with your communication to applicant.	

AO 120 (Rev. 08/10)

TO:	Mail Stop 8 Director of the U.S. Patent and Trademark Office P.O. Box 1450 Alexandria, VA 22313-1450	REPORT ON THE FILING OR DETERMINATION OF AN ACTION REGARDING A PATENT OR TRADEMARK

In Compliance with 35 U.S.C. § 290 and/or 15 U.S.C. § 1116 you are hereby advised that a court action has been filed in the U.S. District Court Eastern District of Texas (Marshall Division) on the following

☐ Trademarks or ☑ Patents. (☐ the patent action involves 35 U.S.C. § 292.):

DOCKET NO.	DATE FILED	U.S. DISTRICT COURT
2:16-cv-692	6/30/2016	Eastern District of Texas (Marshall Division)

PLAINTIFF	DEFENDANT
ALACRITECH, INC.	WISTRON CORPORATION, et al

	PATENT OR TRADEMARK NO.	DATE OF PATENT OR TRADEMARK	HOLDER OF PATENT OR TRADEMARK
1	7,124,205	10/17/2006	Alacritech, Inc.
2	7,237,036	6/26/2007	Alacritech, Inc.
3	7,337,241	2/26/2008	Alacritech, Inc.
4	7,673,072	3/2/2010	Alacritech, Inc.
5	8,131,880	3/6/2012	Alacritech, Inc.

In the above—entitled case, the following patent(s)/ trademark(s) have been included:

DATE INCLUDED	INCLUDED BY ☐ Amendment ☐ Answer ☐ Cross Bill ☐ Other Pleading	
PATENT OR TRADEMARK NO.	DATE OF PATENT OR TRADEMARK	HOLDER OF PATENT OR TRADEMARK
1		
2		
3		
4		
5		

In the above—entitled case, the following decision has been rendered or judgement issued:

DECISION/JUDGEMENT

CLERK	(BY) DEPUTY CLERK	DATE

Copy 1—Upon initiation of action, mail this copy to Director Copy 3—Upon termination of action, mail this copy to Director
Copy 2—Upon filing document adding patent(s), mail this copy to Director Copy 4—Case file copy

AO 120 (Rev. 08/10)

TO:	REPORT ON THE FILING OR DETERMINATION OF AN ACTION REGARDING A PATENT OR TRADEMARK
Mail Stop 8 Director of the U.S. Patent and Trademark Office P.O. Box 1450 Alexandria, VA 22313-1450	

In Compliance with 35 U.S.C. § 290 and/or 15 U.S.C. § 1116 you are hereby advised that a court action has been filed in the U.S. District Court Eastern District of Texas (Marshall Division) on the following

☐ Trademarks or ☑ Patents. (☐ the patent action involves 35 U.S.C. § 292.):

DOCKET NO.	DATE FILED	U.S. DISTRICT COURT
2:16-cv-692	6/30/2016	Eastern District of Texas (Marshall Division)

PLAINTIFF	DEFENDANT
ALACRITECH, INC.	WISTRON CORPORATION, et al

	PATENT OR TRADEMARK NO.	DATE OF PATENT OR TRADEMARK	HOLDER OF PATENT OR TRADEMARK
1	8,805,948	8/12/2014	Alacritech, Inc.
2	9,055,104	6/9/2015	Alacritech, Inc.
3			
4			
5			

In the above—entitled case, the following patent(s)/ trademark(s) have been included:

DATE INCLUDED	INCLUDED BY ☐ Amendment ☐ Answer ☐ Cross Bill ☐ Other Pleading	
PATENT OR TRADEMARK NO.	DATE OF PATENT OR TRADEMARK	HOLDER OF PATENT OR TRADEMARK
1		
2		
3		
4		
5		

In the above—entitled case, the following decision has been rendered or judgement issued:

DECISION/JUDGEMENT

CLERK	(BY) DEPUTY CLERK	DATE

Copy 1—Upon initiation of action, mail this copy to Director Copy 3—Upon termination of action, mail this copy to Director
Copy 2—Upon filing document adding patent(s), mail this copy to Director Copy 4—Case file copy

AO 120 (Rev. 08/10)

TO:	Mail Stop 8 Director of the U.S. Patent and Trademark Office P.O. Box 1450 Alexandria, VA 22313-1450	REPORT ON THE FILING OR DETERMINATION OF AN ACTION REGARDING A PATENT OR TRADEMARK

In Compliance with 35 U.S.C. § 290 and/or 15 U.S.C. § 1116 you are hereby advised that a court action has been filed in the U.S. District Court Eastern District of Texas (Marshall Division) on the following

☐ Trademarks or ☑ Patents. (☐ the patent action involves 35 U.S.C. § 292.):

DOCKET NO.	DATE FILED	U.S. DISTRICT COURT
2:16-cv-693	6/30/2016	Eastern District of Texas (Marshall Division)

PLAINTIFF	DEFENDANT
ALACRITECH, INC.	CENTURYLINK, INC.

	PATENT OR TRADEMARK NO.	DATE OF PATENT OR TRADEMARK	HOLDER OF PATENT OR TRADEMARK
1	7,124,205	10/17/2006	Alacritech, Inc.
2	7,237,036	6/26/2007	Alacritech, Inc.
3	7,337,241	2/26/2008	Alacritech, Inc.
4	7,673,072	3/2/2010	Alacritech, Inc.
5	7,945,699	5/17/2011	Alacritech, Inc.

In the above—entitled case, the following patent(s)/ trademark(s) have been included:

DATE INCLUDED	INCLUDED BY ☐ Amendment ☐ Answer ☐ Cross Bill ☐ Other Pleading	
PATENT OR TRADEMARK NO.	DATE OF PATENT OR TRADEMARK	HOLDER OF PATENT OR TRADEMARK
1		
2		
3		
4		
5		

In the above—entitled case, the following decision has been rendered or judgement issued:

DECISION/JUDGEMENT

CLERK	(BY) DEPUTY CLERK	DATE

Copy 1—Upon initiation of action, mail this copy to Director Copy 3—Upon termination of action, mail this copy to Director
Copy 2—Upon filing document adding patent(s), mail this copy to Director Copy 4—Case file copy

AO 120 (Rev. 08/10)

TO:	REPORT ON THE FILING OR DETERMINATION OF AN ACTION REGARDING A PATENT OR TRADEMARK
Mail Stop 8 Director of the U.S. Patent and Trademark Office P.O. Box 1450 Alexandria, VA 22313-1450	

In Compliance with 35 U.S.C. § 290 and/or 15 U.S.C. § 1116 you are hereby advised that a court action has been filed in the U.S. District Court Eastern District of Texas (Marshall Division) on the following

☐ Trademarks or ☑ Patents. (☐ the patent action involves 35 U.S.C. § 292.):

DOCKET NO.	DATE FILED	U.S. DISTRICT COURT
2:16-cv-693	6/30/2016	Eastern District of Texas (Marshall Division)

PLAINTIFF	DEFENDANT
ALACRITECH, INC.	CENTURYLINK, INC.

	PATENT OR TRADEMARK NO.	DATE OF PATENT OR TRADEMARK	HOLDER OF PATENT OR TRADEMARK
1	8,131,880	3/6/2012	Alacritech, Inc.
2	8,805,948	8/12/2014	Alacritech, Inc.
3	9,055,104	6/9/2015	Alacritech, Inc.
4			
5			

In the above—entitled case, the following patent(s)/ trademark(s) have been included:

DATE INCLUDED	INCLUDED BY ☐ Amendment ☐ Answer ☐ Cross Bill ☐ Other Pleading	
PATENT OR TRADEMARK NO.	DATE OF PATENT OR TRADEMARK	HOLDER OF PATENT OR TRADEMARK
1		
2		
3		
4		
5		

In the above—entitled case, the following decision has been rendered or judgement issued:

DECISION/JUDGEMENT

CLERK	(BY) DEPUTY CLERK	DATE

Copy 1—Upon initiation of action, mail this copy to Director Copy 3—Upon termination of action, mail this copy to Director
Copy 2—Upon filing document adding patent(s), mail this copy to Director Copy 4—Case file copy