RC 20630 (91333) 11/12/96
Computer Science/Mathematics 13 pages

# Research Report

## Scalable Protocol Engine for High-Bandwidth Communications

Christos J. Georgiou, Chung-Sheng Li
IBM Research Division
T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

# SCALABLE PROTOCOL ENGINE FOR HIGH-BANDWIDTH COMMUNICATIONS

*Christos J. Georgiou and Chung-Sheng Li*

IBM T. J. Watson Research Center, P. O. Box 704
Yorktown Heights, NY 10598

(914) 945-4131, (914) 784-6661
e-mail: {geor,csli}@watson.ibm.com

**Abstract**

In this paper, we examine the criteria of designing a protocol engine for high-bandwidth communications, and derive the location of the boundary between dedicated hardware and a generic general-purpose processor as a function of the required performance. The design and implementation of a Fibre Channel protocol engine using both customized CMOS and embedded processors is used as an example to illustrate these design considerations.

## 1. Introduction

The explosive growth of multimedia and Internet/Intranet applications in the past few years has brought about a significant increase in bandwidth demand for both local and wide area networks. This trend is further accelerated by rapid advances in fiber-optic and broadband switching technologies, which have made possible the development of several gigabit or multi-gigabit data communication protocols. These protocols have the potential to provide a very high-bandwidth data transfer capability between peripherals (disk or disk array) and the host, among clusters of workstations, and in local area and metropolitan area networks. Already, several such protocols have been defined or are in the process of being drafted:

- Fibre Channel (and Fibre Channel Arbitrated Loop) [1,2], which is an ANSI standard, has already been defined for transmission speeds of up to 1.0625 Gb/s and is in the process of being extended to 4 Gb/s;
- Gigabit Ethernet [3], currently being proposed with transmission speeds of up to 1 Gb/s;
- ATM over SONET, which has been defined in the ATM forum for bit rates up to 622 Mb/s [4]. The SONET protocol, however, has already been defined for OC-48 (2.4 Gb/s), while proposals for OC-192 (10 Gb/s) are currently being actively discussed [5].

The market penetration of these protocols depends heavily on the successful realization of cost-effective implementations. It is essential that such implementations can be realized through the use of widely available technologies, such as CMOS, so that they can benefit from low production costs and fast advances in manufacturing processes.

In general, a protocol engine that executes the required functions for high-bandwidth transmission could be realized using a fully customized logic, a programmable VLSI engine, or a general-purpose processor. A fully customized approach usually allows for the maximal exploitation of the potential parallelism inherent in the protocol processing but may require significant development cost. By contrast, general-purpose procesors may not provide the same levels of performance, but offer better flexibility and can easily accommodate protocol revisions. A third possibility would be to devise a programmable VLSI chip set, capable of a wide range of protocols. Yet, another possibility would be to integrate some of the customized protocol processing functions with a programmable VLSI or a general-purpose processor.

Recently, a trend is developing similar to the latter approach, whereby signal processing capabilities are integrated with general-purpose processors, such as the Multimedia Extensions (MMX) in Intel's Pentium and Pentium Pro architectures, and similar functions in other microprocessors architectures. The intent of this approach is to take advantage of the potential speedup offered by the dedicated processing elements for certain multimedia signal processing functions, while maintaining the flexibility of a general purpose processor. It is also conceivable that in the future some of the functions required for protocol processing could be integrated with a general purpose processor, resulting in similar benefits.

With the increases in both communication bandwidth and processing capability of end nodes, network I/O has become a bottleneck. As a result, the design and implementation of high-performance network interfaces has been the subject of many studies, as found in a number of recent publications. In [6], a performance model for the FDDI network interface is analyzed. Techniques for partitioning the functions between the host processor and the network interface are also investigated. In [7], an experimental system is proposed which provides media processing on the network interface card. For continous media transmissions such as voice, audio, video and animation, the protocol processing results in a high overhead. Furthermore, there is often a bandwidth mismatch between the network and the host system bus. This proposal relieves the system bus from the overhead of transporting the media traffic between the network interface and the media specific processors. In [8], the design and implementation of a parallel protocol engine for the logical link control (LLC) function of 802.2 is described. The performance of the four processor version of the implementation can satisfy the demands of networks operating in the 100 Mb/s range. Finally, the design of a flexible programmable VLSI protocol engine, PROVE, is described in [9]. This protocol engine can process protocols such as LLC Class 2 or LAPD up to 50,000 packets/second.

Nevertheless, there is still a lack of study on the impact of network interface design for gigabit networks, which may require a network interface with a processing capability in excess of 10 million packets/second.

In this paper, we examine the criteria for designing a protocol engine suitable for gigabit/second bit rates, and derive the location of the boundary between dedicated hardware and generic processor as a function of the required performance. The design and implementation of a Fibre Channel protocol engine, which requires a processing capability of up to 10 million packets/second, is used as an example to illustrate these design considerations. The highlights of this implementation, consisting of both customized CMOS VLSI and general-purpose processors, include:

- concurrent protocol processing of both inbound/outbound data streams at the full link data rate specified by the protocol,

- high-speed realization of the line coding/decoding and CRC encoding/decoding using both parallel and pipelined techniques,
- concurrent finite state machine for both media access protocol processing and end-to-end protocol processing.

The customized VLSI design has been specified and verified using the VHDL hardware description language, and has been translated into approximately 20K gates using 0.5 μ m CMOS technology.

The remainder of this paper is organized as follows: Section 2 presents various considerations for the design of protocol engines for high-bandwidth communications, Section 3 gives the architecture and design of a Fibre Channel protocol engine, Section 4 discusses the performance evaluation and verification of such a design, and the paper concludes with a brief summary in Section 5.

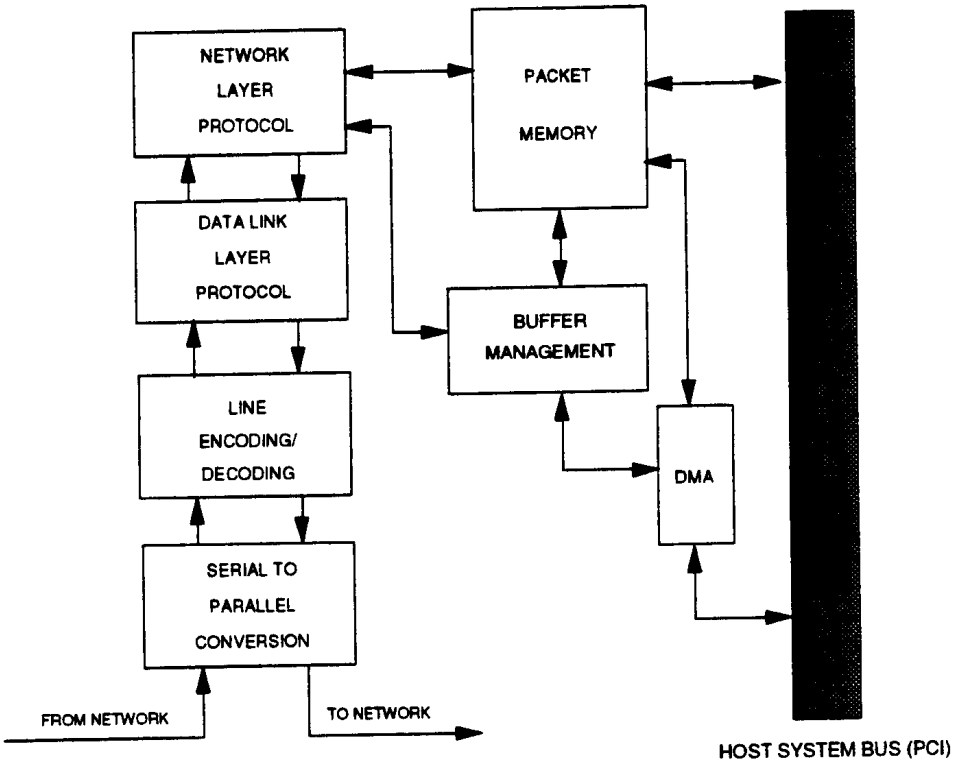


**Figure 1:** System block diagram of a protocol engine

## 2. Protocol Engine Design Considerations

Most of the current and likely future high-speed data links are serial links. In managing the links, the protocol engine is responsible for (a) receiving the serial data into the host memory, and (b) transmitting the data from the host memory to the serial link. Since protocol processing usually requires significant amounts of computation that can be a burden to the host processor, it is desirable to offload the protocol functions to a dedicated subsystem. The functionality of such a protocol engine, as shown in Fig. 1, usually includes the following:

- Serial-to-parallel conversion, for performing the conversion between parallel and serial data; Byte and word synchronization, for monitoring the incoming bit streams and detecting the occurrrence of out-of-sync conditions;

- Line encoding/decoding, for ensuring that there is a sufficient number of data transitions so that the timing recovery circuits can work properly;

- Data Data link layer protocol engine (or media access protocol engine), for performing the media accessing function (as required, for example, to implement CSMA in the 802.x protocol, and the abitrated loop function, FC-AL, in Fibre Channel);

- Network layer protocol engine (including possibly CRC generation/checking), for performing end-to-end flow control, congestion control, and header processing.

- Packet memory, for storing the inbound packets before they are processed, and outbound packets before they are transmitted.

- Buffer management, for performing buffer allocation upon the arrival of packets and buffer deallocation once the packets are delivered to the host.

- Direct Memory Access, which performs data transfer between the packet memory and the host memory without the intervention of the host processor.

Some of these functions, such as serial-to-parallel conversion and line encoding/decoding, can only be implemented in hardware in order to be able to operate at gigabit data rates or above. On the other hand, header processing, end-to-end flow control, and in-sequence delivery can be implemented either in software or hardware. The optimal partitioning of the tasks between dedicated hardware and general-purpose processor, either as a stand alone unit or as an embedded processor, is dictated by the performance and cost requirements of the system. As previously pointed out, a programmable processor allows better flexibility, lower cost, and can easily accommodate protocol revisions, or even multiple protocols. However, its capacity to exploit the maximal inherent parallelism provided by the protocol functions is fairly limited.

From a processing point of view, a protocol's operations can be divided into three categories:

- Byte/Word-level datapath operations: such as CRC generation and checking as well as line encoding/decoding in which several processing steps are involved for every byte (or word) received by the protocol engine;

- Frame (Packet)-level datapath operations: such as header processing and memory management in which the operations are invoked only at the beginning or the end of each received frame (or packet);

- Control path operations: such as the initialization of a connection and the house cleaning operation when a connection is terminated. These operations are invoked only at the beginning and the end of a connection, which usually lasts for more than one packet or frame.

In examining the requirements imposed on the protocol engine by the above operations, we begin with memory bandwidth. We assume that the link rate is $B$ and the packet length is $L$, where a packet consists of a header with fixed length $H$ and data with variable length $D$. $D$ can vary from zero (as in the case of control packets) to a maximum length of $D_{max}$, that is defined by the specific protocol. As a result, the peak packet rate is $B/H$, which occurs when control packets arrive contiguously. As an example, the peak packet rate for Fibre Channel is 4 million packets/second, if the minimum-length packets (each consisting of a 36-byte header) are sent back-to-back at a gigabit data rate. Because only the data portion of a packet needs to be delivered to the host memory, the maximum data throughput to the host memory is $D_{max} * B /(H+D_{max})$. Similarly, the data bandwidth out of the host memory to the protocol subsystem needs to be identical to the above, in order to support full duplex communication. As a result, the total host memory bandwidth required to support the transmission and reception is $2D_{max} * B /(H+D_{max})$. Using Fibre Channel as an example, the host memory bandwidth required to sustain packets transmitted at a gigabit data rate (both inbound and outbound) is approximately 200 MB/s. Currently, only the 66 MHz PCI bus, which has a peak throughput of 266 MB/s, can support data transfer at such a rate.

Although the protocol processing can always share the memory with the host, a staging buffer is usually required to avoid the temporary unavailability of the system bus. When the packet memory is shared between the communication subsystem and the host, the data portion of the packet is sent to the memory as soon as it arrives. However, the protocol engine notifies the system only when a complete upper layer protocol (e.g., IP or SCSI) data unit is received.

It is also possible to provide sufficient memory in the communication subsystem to temporarily store the incoming packets, so that a data transfer to the host occurs only when a complete upper layer protocol data unit is received. In this case, the total amount of memory bandwidth required for the dedicated memory in the protocol engine is $4D_{max} * B /(H+D_{max})$, since the buffer memory on the communication subsystem has to support the data to be read in and out simultaneously.

After a packet (such as a Fibre Channel frame or an ATM cell) is received, pointer manipulations are required for memory allocation. Similarly, pointer manipulations are required to deallocate memory when the packet is transmitted to the host. All of these operations occur on a per-packet basis, for each packet. Variable-length packets impose an additional challenge by potentially requiring multiple pointer manipulations. In general, both inbound and outbound packets would require pointer manipulation if they share the same packet memory. Using the notation defined earlier, the maximal rate of pointer manipulation for inbound or outbound traffic is $B/H$, which is the maximal rate at which packets can arrive on the inbound or outbound link. If the pointer structure is implemented via a linked list, each memory allocation requires one register read to obtain the address of the current head-of-list block, a memory read to read the address of next available memory block, and a register write to place the new address in the head-of-list register. Similarly, memory deallocation requires one register read to read the tail-of-list location, a memory write to link the newly available memory block to the linked list, and a register write to place the address of the newly available memory block in the tail-of-list register. Therefore, the maximum number of memory accesses required for memory management purposes is $4B/H$, assuming symmetric inbound and outbound traffic, and the total number of processing steps involved is $12B/H$. For Fibre Channel, this implies that the processing power required for memory management alone is approximately 50 MIPS (million instructions per second).

Header processing usually requires interpreting the individual bits in the header and accessing context information corresponding to those currently opened connection (for those protocols that are

connection-oriented). If the protocol permits out-of-order delivery (such as Fibre Channel), additional processing is required to maintain a scoreboard in order to determine whether a segment of continuous streams of packets have been received. Since multiple connections can be opened simultaneously, the protocol engine is required to provide context switching capability when packets from different connections are intermixed. Context switching may impose a performance penalty when the processor, due to limited on-chip memory, would have to access an external memory for retrieving the context information.

Finally, there are multiple protocol states that are required to be processed continously. These states can be implemented by using dedicated finite state machines or by sharing the same general-purpose processor. Because of the sequential nature of the instruction execution of general-purpose processors, there may a latency associated with the processing of each state. This latency can be the limiting factor to how fast incoming frames can be processed.

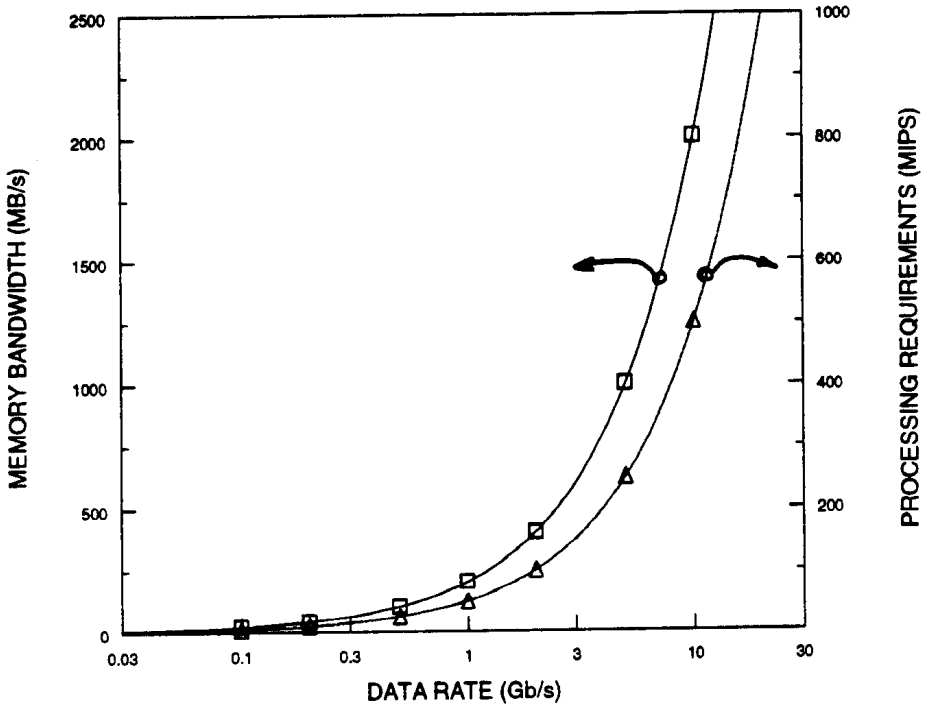


**Figure 2:** Memory bandwidth and processing requirements as a function of link bit rate

Figure 2 shows a logarithmic plot of the memory bandwidth and processing steps required as a function of bit rate. It can be seen that the memory bandwidth and number of procedures required for protocol processing increase linearly with the bit rate. As a result, it is desirable that the architecture of the protocol engine be scalable both with respect to the link rate and packet rate.

## 3. Protocol Engine Architecture

In this section, we propose the structure for a scalable high-bandwidth protocol engine. In order to illustrate our design approach, we use as an example the Fibre Channel, because it is a

functionally rich protocol that requires many considerations to be taken into account, such as

- packets of variable length,
- out-of-order packet arrival,
- maintaining multiple logical connections, with packets from any connection can arrive simultaneously,
- an arbitrated loop layer which requires an additional media access protocol.

To facilitate the reading of the discussion that follows, we give a brief overview of the protocol levels defined by Fibre Channel:

- FC-0 defines the physical intreface and electrical characteristics of the transmission medium including connectors, bit rates, jitter, and distance without repeater.
- FC-1 defines the transmission protocol which specifies the serial encoding, decoding, byte and word synchronization, and associated error control method.
- FC-2 defines the signaling protocol which specifies the rules and mechanisms needed to transfer blocks of data from one end to another.
- FC-3 defines a set of services which are common across multiple ports of the same node.
- FC-4 defines the channel protocol or mapping between the lower levels of Fibre Channel (i.e., FC-3) and upper level protocols such as IP, SCSI, etc.

The overall structure of the protocol engine is shown in Fig. 3, with several time critical functions such as line encoding/decoding, CRC processing, and part of header processing implemented on a custom-made integrated circuit, while the rest of the functions are realized using a programmable processor. In order for the architecture to be scalable, the following function modules have been carefully partitioned and pipelined:

- 8B/10B encoding/decoding
- CRC generation/checking
- concurrent header protocol processing and memory management.

Furthermore, the communication subsystem is partitioned into three different clock domains:

- Receiver, in which the clock is synchronized to the receiving data stream.
- Internal, in which the clock is set at the fastest available rate allowed by the design.
- Transmitter, in which the clock rate is used to determine the transmitting data stream.

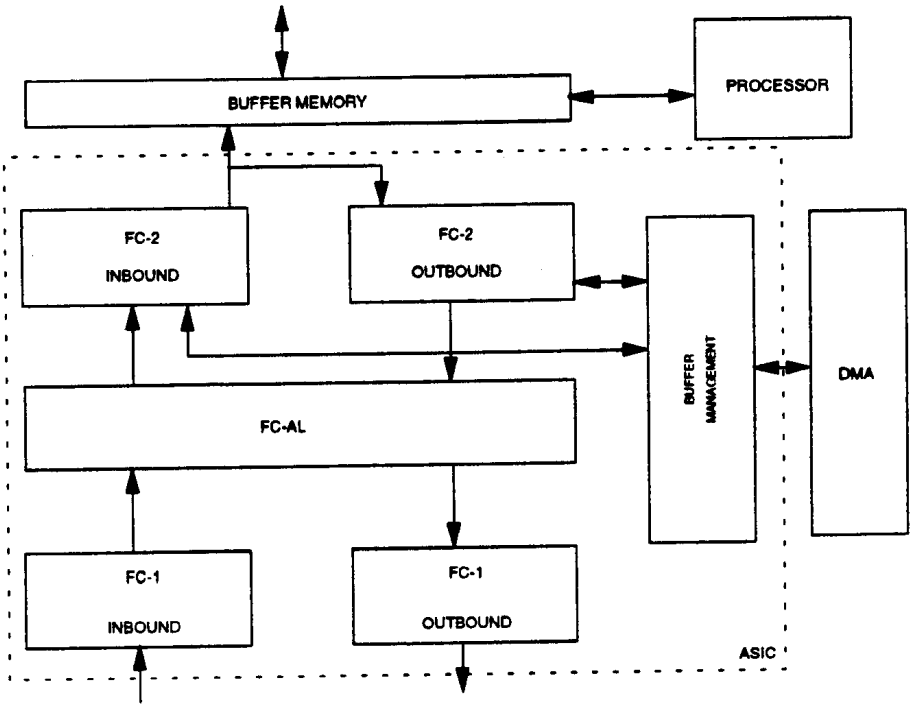


**Figure 3:** Architecture for the Fibre Channel protocol engine

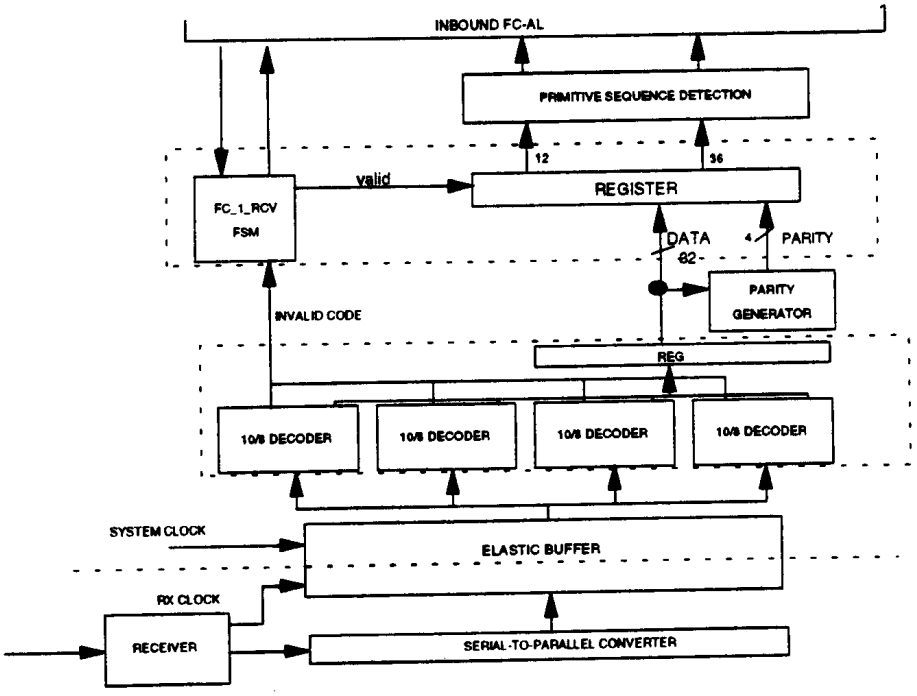


**Figure 4:** Inbound processing block diagram of FC-1

The inbound processing of FC-1 is shown in Fig. 4. The high-speed serial data that arrives at the receiver is deserialized into byte-parallel format before it is sent to the protocol engine. The protocol engine further converts the data stream from byte- to word-parallel format and puts it into an elastic buffer using the receiver clock extracted from the incoming data. Subsequently, the word-parallel data is synchronized to the internal clock when it is read out of the elastic buffer. A two-word elastic buffer is used for the clock domain conversion between receiving and internal clocks. The data is then decoded by a four-byte parallel 8B/10B decoder. A finite state machine monitors the incoming data for possible out-of-sync conditions, which are derived from the error conditions indicated by the 8B/10B decoder. When an out-of-sync condition has occurred, the finite state machine searches through the incoming data stream for a sync word (e.g., the IDLE word in Fibre Channel) to reestablish synchronization.

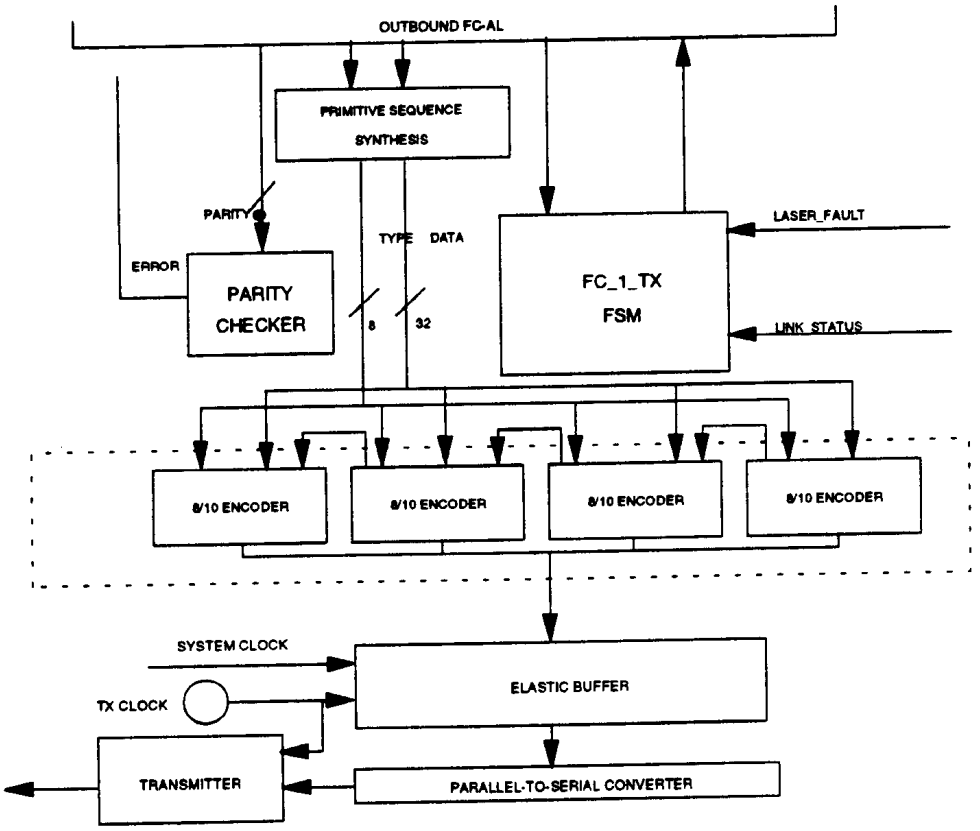


**Figure 5:** Outbound processing block diagram for FC-1

The outbound data is encoded by the parallel 8B/10B encoder circuit, as shown in Fig. 5, before passing through the elastic buffer for clock domain translation. Similar to the inbound case, a two-word elastic buffer is used in the outbound path for clock domain translation. Note that the transmitter clock is generated locally. The same clock is used to fetch the data from the elastic buffer and convert it into serial format. Also note that a "primitive sequence synthesis" combinational logic circuit is used in the outbound path for synthesizing special characters which

conform to the link layer protocol specified by the Fibre Channel.

The incoming data stream is converted into two separate streams, a data stream and a control stream, at the primitive sequence detection module. This is necessary in Fibre Channel as there are data delimiters (such as start of frame, end of frame, idle, and receiver ready) and a special control word transmitted between packets that are represented by special symbols in the 8B/10B code. The data delimiters cannot be converted to regular data but, instead, are converted into control signals within the protocol engine, resulting in a separate control path. Similarly, the data on the control path are used to synthesize the data delimiters on the outbound path.

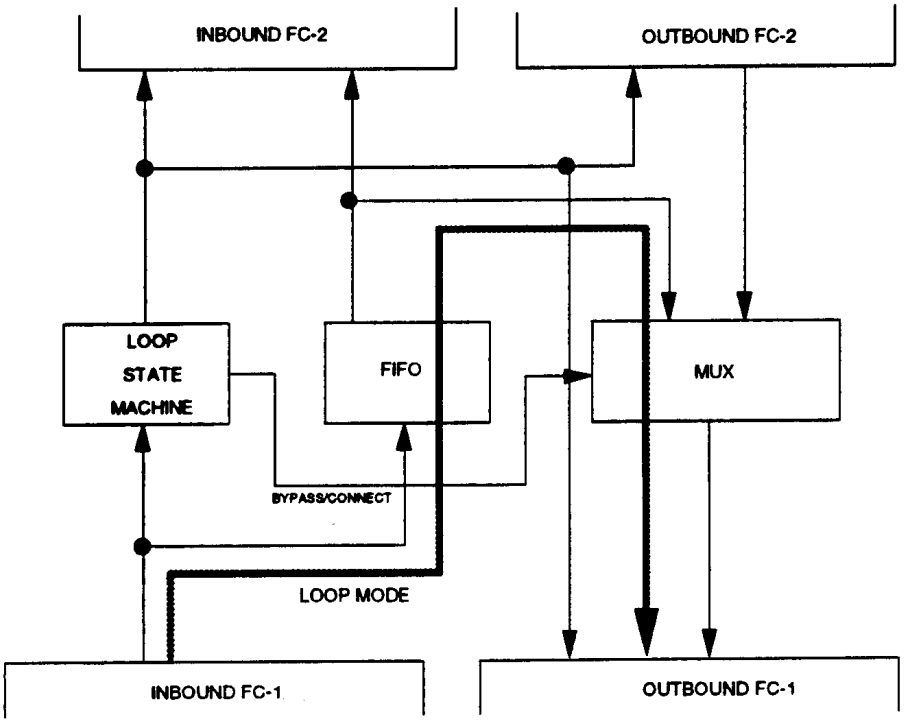


**Figure 6:**. Processing block diagram for FC-AL

The implementation of the protocol layer that provides media accessing control, as required by the Arbitrated Loop function, is shown in Fig. 6. The associated finite state machine must monitor the incoming data delimiter continuously for an opportunity to acquire the loop and, thus, be able to establish a connection. Before the loop is acquired, the FC-1 layer must route all frames received from the inbound link back to the outbound link. The routing path is indicated in the figure, in which the incoming data are stored into a FIFO. The output of the FIFO is routed to the outbound of the FC-AL during loop mode. When there is a frame in the host waiting for transmission, the finite state machine must announce its intention to transmit to the entire loop by sending out a special delimiter. When no other nodes intend to transmit, the delimiter will arrive back to the node. A connection can then be opened and the upper layer can start the data transmission. (When there are multiple nodes waiting for transmission in the loop, a predetermined priority resolution protocol is used to decide which node can trasmit next.)

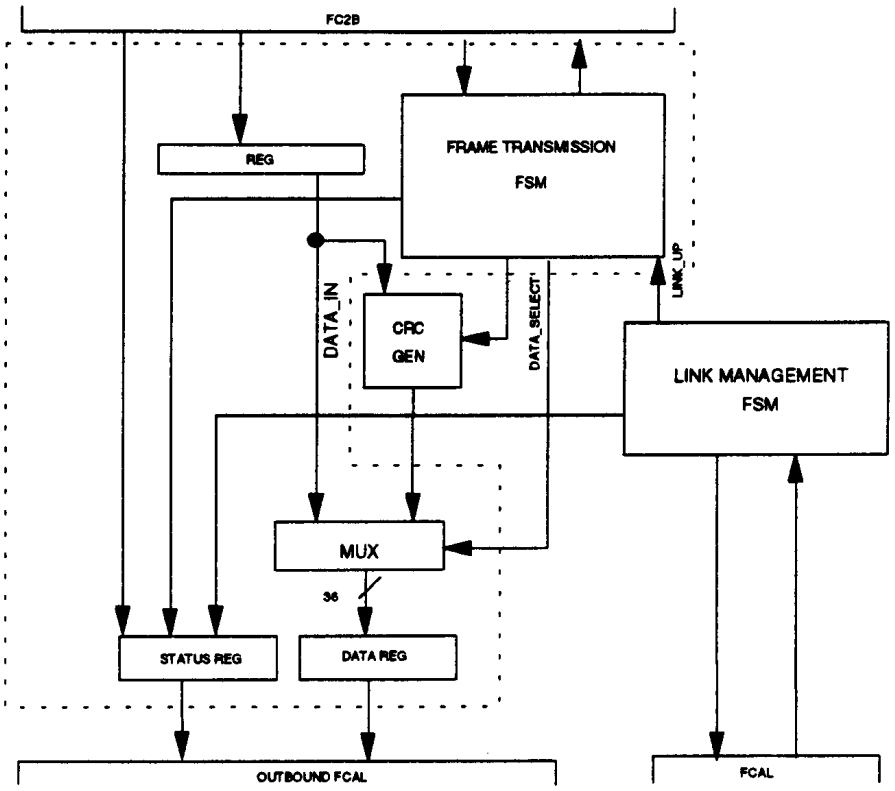


**Figure 7:** Outbound processing block diagram for FC2

The outbound part of FC-2 is shown in Fig. 7. The most important feature of this module is its parallel CRC generator. The flow control mechanism is also maintained in this module. Fibre Channel uses a back pressure mechanism based on buffer credits to regulate the traffic flow. The buffer credit of the receiver is maintained at its upstream node. A buffer credit counter of an upstram node is incremented each time a packet is sent to the downstream node. This process is suspended when the maximum buffer credit is reached. At that point, the upstream node has to wait for the arrival of the receiver ready signal before the credit counter can be decremented and a new packet be sent.

On the inbound side, as shown in Fig. 8, the CRC generated by the incoming data is checked against the CRC code carried by the packet. Multiple finite state machines are constructed to facilitate the capturing and processsing of the header information and to verify the validity of the frame (e.g., whether the maximum frame length is violated, whether an alias is used for the destination address, whether the start of frame format is consistent with the end of frame format, etc.). An independent link management finite state machine continously monitor the inbound link and is engaged in the link recovery protocol once the malfunction of the link is detected.

The buffer management, which is shown in Fig. 3, manipulates the pointers upon packet arrival and packet consumed events for inbound traffic. It consists of two independent producer and consumer finite state machines which are synchronized through a semaphore when they have to access information simultaneously.

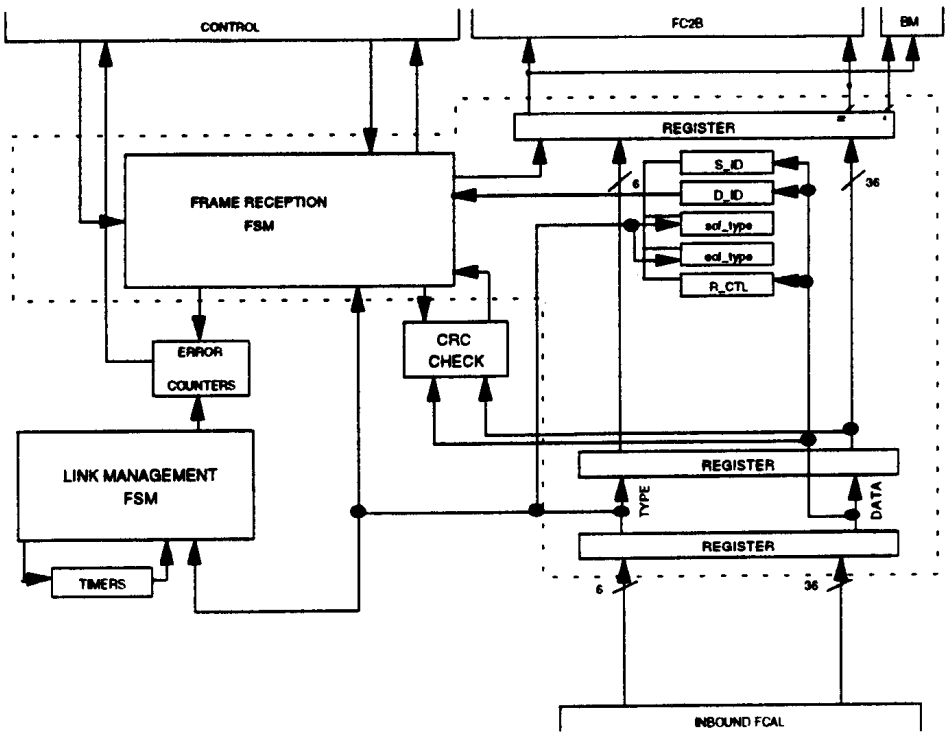


**Figure 8:** Inbound processing block diagram for FC-2

## 4. Performance Evaluation

The performance of the proposed structure is measured in terms of latency and throughput. Total latency is defined as the elapsed time from the first bit of a packet arriving at the network interface to the last bit of the packet leaving the packet memory. The latency of each individual component in the system can be defined in a similar way. For inbound traffic, the latency in the FC-1 layer is approximately three system cycles: two introduced by the elastic buffer and an additional cycle used by the register in the parallel line decoding. Two system cycles are introduced by the FIFO in the FC-AL layer. Three more system cycles are introduced in the FC-2 layer. Therefore, a total of 8 cycles are introduced when a packet is received from the receiver to the packet memory. For outbound traffic, the latency in the FC-2 layer is two system cycles, while the latency introduced in FC-1 is an additional three cycles, for a total of five system cycles. If the system cycle time is 20 ns, the inbound delay before a packet is put into packet memory is 160 ns, while the outbound delay is 100 ns, from retrieving a packet to its actual transmission.

Throughput is measured in terms of the bits that are processed by the protocol engine, and heavily depends on the distribution of the packet length. Longer packet length improves the throughput due to its smaller overhead in header transmission and processing. The throughput of the proposed design is identical to the link rate. When the proposed design is operated at full duplex mode, the total throughput can exceed 400 MB/s, while the maximum packet throughput rate can exceed 20 million packets/second.

The accuracy and timing of the architecture as described above was verified through VHDL simulation. A 0.5 μm CMOS technology was assumed for synthesizing the architecture. The

maximal internal cycle time achieved was 20 ns, indicating that a maximal throughput of 2 Gb/s can be supported. The delay from the serial input to the packet memory is less than 200 ns, when the arbitrated loop is opened.

## 5. Summary

In this paper, we have proposed and evaluated a scalable architecture for implementing multi-gigabit protocol engines. The architecture utilizes a combination of custom-made VLSI circuitry and a general-purpose processor, such as the Intel 960 or the IBM PowerPC 403. Time critical operations such as line coding/decoding, CRC generation/checking, context-independent header processing, and buffer management are implemented in the customized VLSI part. These designs are nevertheless scalable and can be cascaded to further increase throughput. Some of the packet level processing such as context-dependent header processing are performed by the general-purpose processor. As processing power increases, more and more functions can be included in the general-purpose processor. The throughput of this architecture is shown to be adequate for the operations and bit rates currently specified by Fibre Channel. Future CMOS technology advances will have the potential to further improve the raw throughput.

## References


[1] ANSI Standard, "Fiber Channel: Physical and Signaling Interface (FC-PH) (Working draft, Rev. 4.3), X3T11 Project 755D," *American National Standards Institute,* June 1994.

[2] M. W. Sachs and A. Varma, "Fibre Channel and Related Standards," *IEEE Communication Magazine*, Aug. 1996.

[3] S. Lawson, "Vendors to show pre-standard Gigabit Ethernet technologies," *Infoworld*, vol. 18, no. 38, Sept. 16, 1996.

[4] M. De Prycker, "Asynchronous Transfer Mode," 3rd edition, Prentice Hall, 1995.

[5] T. Radick, "More power to you OC-192 transport terminals," *Telephony*, vol. 229, no. 6, Aug. 1995.

[6] K. K. Ramakrishnan, "Performance Considerations in Designing Network Interfaces," *IEEE Journal of Selected Areas in Communications*, vol. 11, no. 2., Feb. 1993.

[7] G. Blair, A. Cambell, G. Coulson, F. Garcia, D. Hutchison, A. Scott, and D. Shepherd, "A Network Interface Unit to Support Continuous Media," *IEEE Journal of Selected Areas in Communications*, vol. 11, no. 2, Feb. 1993.

[8] M. Kaiserwerth, "The Parallel Protocol Engine," *IEEE/ACM Transaction on Networks*, vol. 1, no. 6, Dec. 1993.

[9] A. S. Krishnakumar, W. C. Fischer, and K. Sabnani, "The Programmable Protocol VLSI Engine (PROVE)," *IEEE Transactions on Communications*, vol. 42, no. 8, Aug. 1994.

Copies may be requested from:

IBM Thomas J. Watson Research Center
Publications Office, 16-220
Post Office Box 218
Yorktown Heights, NY 10598