14

A Reduced Operation Protocol Engine (ROPE) for a multiple-layer bypass architecture

Y.H. Thia (*)[1] and C.M. Woodside (**)

Newbridge Networks, Inc., Ottawa, Canada (*) and
Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada (**)

Abstract — The Reduced Operation Protocol Engine (ROPE) presented here offloads critical functions of a multiple-layer protocol stack, based on the "bypass concept" of a fast path for data transfer. The motivation for identifying this separate processing path is that it involves only a small subset of the complete protocol, which can then be implemented in hardware. Multiple-layer bypass also eliminates some inter-layer operations such as queue and buffer management, context switching and movement of data across layers, all of which are a significant overhead. ROPE is intended to support high-speed bulk data transfer. The paper describes the design of a ROPE chip for the OSI Session and Transport layer protocols, using VHDL. The design is practical in terms of chip complexity and area, using current gate array technology, and simulation shows that it can support a data rate approaching 1 gigabit per second, in a connection attached to an end-system.

Keyword codes: C.2.2, B.4.1
Keywords: Network Protocols, Data Communications Devices

1 Introduction

The advent of Fibre Optic technology, which offers high bandwidth and low bit error rates, has shifted the performance bottleneck from the communications channel to the communications processing in the end-points of the system [26]. Other trends such as improved quality-of-service guarantees will reinforce this effect. The heavy processing load is due to a combination of operating system overhead, protocol complexity, and per-octet processing on the data stream. To alleviate the end-system bottleneck one may consider new protocols [10], improved software implementation of existing protocols [5, 35], parallel processing techniques [14, 21, 38], special protocol structures [15, 30] and hardware assist [22] by offloading all or part of the protocol functions to an adaptor. This paper takes the latter approach.

The key problems associated with offboard processing include:

- Partitioning the functionality between the host and the adaptor is difficult and may easily lead to a complex additional protocol between the two parts, which may cancel out or offset the potential gain from offloading. For example, the buffer management task [36] may be offloaded, but this leaves the problem of control for accessing it within the full protocol logic.

[1] This research was done while Dr. Thia was at Carleton University

G. Neufield et al. (eds.), *Protocols for High Speed Networks IV*

- Non-protocol-specific processing is a large part of the total load, as shown in [35]. Examples include interrupt handling, context switching and data copying at layer boundaries in deeply layered protocol stacks.
- The choice of hardware for the adaptor depends on the complexity of the functions it supports. In [2, 22] where the transport protocol layer is offloaded or in [7] where the full protocol stack can be offloaded, general purpose microprocessors are used. Probably because of the complexity of existing protocols, VLSI [24] implementation above the data link layer has been disappointing so far. In [8], dedicated VLSI chips are used to support TCP checksums. Also, some newer lightweight transport protocols are specially designed for VLSI implementation [1, 3].
- There is a tradeoff between performance, flexibility and cost. If the key functions of the frequently executed portion of the protocol remain relatively stable, there can be significant advantage in providing hardware support for these functions leaving the other tasks in the host software for flexibility.
- As host processing speed continues to outpace memory bandwidth and as the network bandwidth approaches the processor memory bandwidth, it is important to keep data movement on the workstations down to the minimum [4, 9, 28].

This paper presents a feasibility study for a new approach to hardware assistance. It combines the relatively simple operations needed for data transfer across multiple layers and provides a hardware "fast path" for them, which will be efficient for bulk data transfer. It is based on the "protocol bypass concept" [37] which is a generalization of Jacobson's "Header Prediction" algorithm [20] for TCP/IP. Bypass solves the problems identified above, which may limit the use of offboard processing, by implementing an entire service through all layers for certain cases. This simplifies the interface between the host and the adaptor chip and minimizes their interaction, which is supported by an access test, some DMA processing and a simple command protocol. The chip design based on bypassing is called ROPE, for Reduced Operation Protocol Engine. The contribution of this paper is to define the host/chip interface and the chip operation, and to report on a VHDL-based feasibility study of the chip design. It appears to be feasible to support an end-system single-connection data rate approaching 1 Gbps.

The next section introduces the bypass concept, its architecture and implementation. Section 3 analyzes the key protocol processing overheads and discusses the requirements for a bypass VLSI implementation. Sections 4, 5 and 6 describe a design study of a ROPE chip using the industry standard hardware description language, VHDL, with conclusions in Section 7.

2 The Bypass Concept

A bypass adds an additional path for certain operations, with minimal changes to the original software. Conformance to the protocol is maintained by doing all the other operations through the normal "heavyweight" path. A bypass path can be provided for send, for receive, or for both together, and is compatible with other end-systems implemented without a bypass.

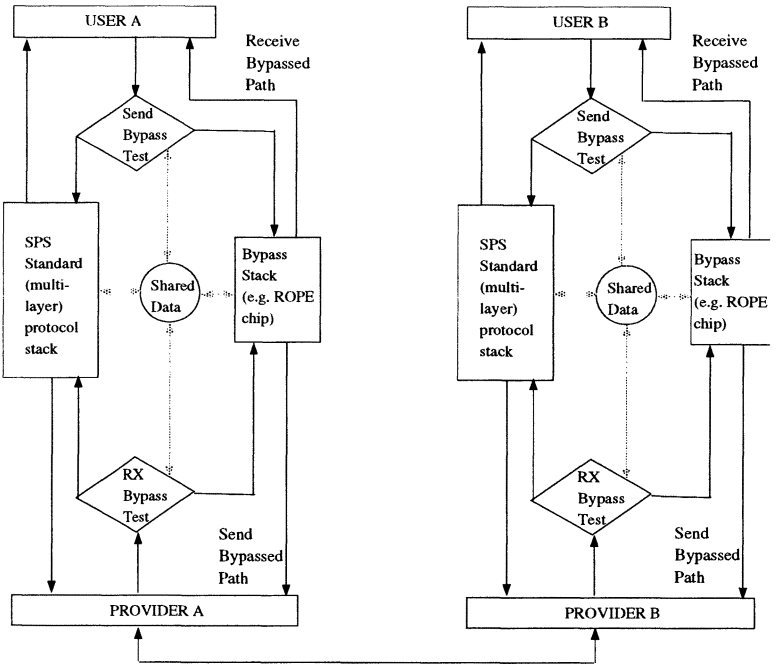


Figure 1 Bypass Architecture

2.1 Bypass Architecture

Figure 1 illustrates the architecture of a bypass implementation for any standard protocol. The standard protocol stack (SPS) is the processing path taken by all PDUs during a connection without the bypass. The SPS may refer to a single layer or to multiple adjoining layers of a layered protocol stack. The bypass has 4 key components:

- *Send Bypass* Test;
- *Receive Bypass* Test;
- *Bypass Stack*;
- *Shared data* for access by the two tests, the Bypass stack and the SPS.

The send bypass test identifies outgoing packets that are data packets in the data transfer phase. The receive bypass test matches the incoming PDU headers with a template that identifies the predicted bypassable headers. The bypass stack performs all the relevant protocol processing in the data transfer phase. The shared data are used to maintain state consistency between the SPS and the bypass stack, including window flow control parameters and connection identifiers. Whenever there is a change in the processing path between the SPS and the bypass stack, checks are performed to ensure that there are no outstanding packets in the current path, i.e. "no in-transit PDUs", before the change is made. A more detailed discussion on this is presented in another paper [33].

2.2 Efficient logic for the bypass test

The "no-in-transit PDU" test can often be avoided. At the beginning of data transfer on a new connection, it is automatically satisfied. It holds as long as no packet fails a bypass test, and it is sufficient to maintain a flag to indicate this. Once a packet fails, and goes to the SPS, then a full "no-in-transit PDU" test must be performed for each packet until the test succeeds, after which control can go back to the flag. Token management and synchronization points of the session layer [19, 18] which are mapped by equivalent application and presentation service primitives can be used as synchronization points within the bypass architecture, as it switches between the SPS and the bypass stack. Since they are only inserted periodically in bulk data transfer and can be controlled by the application process, these overheads are not excessive.

2.3 Multiple-layer bypass

A bypass for multiple layers instead of just one gives additional gains by avoiding:

- ☐ Overhead of encoding and decoding the interface control information passed between layers;
- ☐ Executing the full general protocol logic for the layers to decide how to manipulate the data;
- ☐ Queueing of data at layer boundaries.

The advantage is increased further in cases where some layers, like the network and application layers, have been further subdivided into sublayers.

A multiple-layer bypass path is a concatenation of processing procedures performed by the adjacent layers when they are simultaneously in the data transfer phase. Meanwhile, the separate layers in the SPS path handle the other phases.

In summary, the separation of the bypass path offers the following advantages:

- ☐ The processing path of data PDUs can be optimized;
- ☐ The number of possible PDU formats in the bypass path is reduced to data transfer PDUs;
- ☐ The finite state machine of the protocol is now reduced to only the "OPEN" state, for as long as processing remains in the bypass path. The state of the system does not change during the entire data transfer phase and the protocol processing is reduced to ensuring reliable transfer of data across the communications network.

3 Design Considerations for a Hardware Bypass

3.1 Factors affecting system performance

This section summarizes the major factors affecting throughput performance in a deeply layered stack on an end system. It follows the description by Heatley and Stokesberry [16].

Protocol procedures can be characterized as per-octet, per-packet or per-group-of-packets operations. The per-octet operations take an average time A per octet (e.g., checksum), and the per-packet procedures take time B per packet (e.g. address decoding and multiplexing). Per-group-of-packets operations include for example transmission of acknowledgments, whose frequency is implementation-dependent, and their timing will be aggregated and included in parameter B.

The throughput bound imposed by protocol processing alone, λ_{max} in octets per second, is then given by the equation:

$$\lambda_{max}(x) = \frac{x}{A.x + B.\lceil x/M \rceil} \tag{3.1.1}$$

where x is the size of the user message in octets and M is the maximum PDU size. In bulk data transfer, as x becomes large,

$$\lim_{x \to \infty} \lambda_{max}(x) = \frac{1}{A + B/M} \tag{3.1.2}$$

The protocol processing load on an end system is typically shared between the host and the network adaptor. As the raw data bit rate supported by optical networks approaches the main memory bandwidth of the end system, the cost of moving data and of per-octet processing limits the effective throughput presented to the application process, especially for bulk data transfer. The data portion of a PDU may be physically moved for the following reasons:

- Copying between the adaptor buffer and the host system memory;
- Crossing protection domains (address spaces) — e.g. at the user/kernel boundary. This problem becomes more pronounced in microkernels which treats a protocol task as a server process outside the kernel domain [11];
- Per-octet processing like presentation conversion and checksum routines.

Hardware implementation is particularly efficient for per-octet operations.

3.2 Requirements of a bypass VLSI implementation

The problems associated with separate or offboard processing, which were discussed in the introduction, are addressed by the VLSI design as follows:

- ☐ A clean separation of functionality requiring only a simple protocol to communicate between the host and adaptor is desired, and is provided by a bypass. Its particular set of functions are complete in themselves and have a focussed interface with the host software at the packet entry point. There is relatively infrequent switching between the SPS and the bypass stack;
- ☐ Reduced non-protocol-specific processing overhead. For example the processing of acknowledgment packets is dominated by interrupt handling, typically a few hundred instructions, rather than by the protocol processing itself. Our approach removes acknowledgment handling altogether from the host. Also, the bypass system can be extended to incorporate multiple-layer stacks and remove overhead that way;
- ☐ VLSI implementation complexity: only the data transfer functions are implemented.
- ☐ Tradeoff between performance, flexibility and cost. The host software processes non-data-transfer packets which are typically small but require more flexible and more complex processing. They are also only per-packet, so they have less impact overall. They can be efficiently handled by the host given the projected increase in host processing speed [17]. The operations implemented in VLSI are understood to have less need of flexibility.

Layer	*Procedure*	***Bypass Chip***	*Host*	*Per-Octet (A)*	*Per-Packet (B)*	*Per-Group-Of-Packets Aggregated to Per-Packet for bulk data transfer (B)*	*Remarks*
Presentation	Encoding	X		X			
	Encryption	X		X			
	Compression	X		X			
	Context Alteration		X			X	
Session	Synchronization Management		X			X	
	Token management		X			X	
Transport (Class 4)	Checksum (Optional)	X		X			
	Timer Management	X			X	X	Depends on Implementation
	Generation of ACK packets (Flow Control)	X				X	
	Resequencing	X			X		
All 3 layers	Header Construction	X			X		
	Header Decode	X			X		
	Buffer Management	X			X		Minimized (Simple scheme)
	Context Switching	X			X	X	Moved away from host OS
	Data Copying	With multiple-layer bypass, data Copying within layers is elminated.			X		Use of dual-ported memory and DMA..

Table 1 Bypassable versus Non-bypassable functions

- Remove operations that are inefficient on the host. It is important to look at both the processing requirements and data movement of the Application PDUs. The critical resource is often the host bus/memory path, and caching is used to increase its efficiency. However long traverses through the data for per-octet operations like checksum and encoding interfere with efficient caching, so it is particularly appropriate to offload them.

Table 1 identifies procedures which are strong candidates for implementation in the bypass chip, and those which are better handled by the host, during the data transfer phase. Besides these, the send bypass test is done on the host and the receive bypass test is done on the Network Interface Adaptor.

4 VLSI implementation of a Reduced Operation Protocol Engine (ROPE) chip using VHDL

The VHSIC Hardware Description Language (VHDL) [6, 27] was used to model and synthesize the chip. VHDL is an industry standard language which can be used to represent all levels of abstraction, from logic gates to the system level. By utilizing VHDL as a specification tool, it is possible to begin simulation and debugging of complex systems

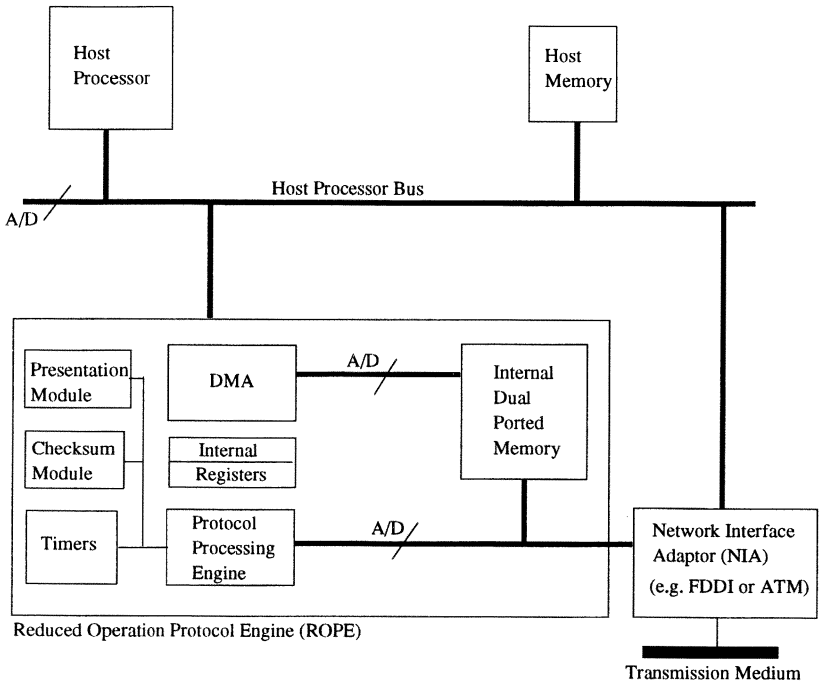


Figure 2 Block Diagram of VLSI bypass system

before details regarding the implementation are fully specified. It also offers the potential of an automatic path from the protocol specification to VLSI implementation, in which any modifications to the specification can be easily propagated to the gate level design.

4.2 Architectural Description

Figure 2 shows the block diagram of the system. The host processor and NIA components provide logical interfaces for simulation of behaviour, but they insert no timing delays. For modeling purposes they were described as being infinitely fast, either as a source or as a sink. This places the maximum stress on the ROPE chip.The architectural considerations involved in the chip design can be summarized as follows:

- Movement of data across the host bus interface are minimized by using an on-chip DMA for fast block data transfer to/from the host system memory.
- On-chip dual-ported memory is used, rather than the host memory, to avoid critical constraints on bus access latency and throughput.
- The control registers of the bypass chip are I/O mapped to the host processor. This enables the host processor to configure the bypass chip directly.

The presentation module shown in the Figure was allowed for in the data structures but was not fully designed.

4.3 First Design: Design Steps

Figure 3 shows the steps followed in this study. There were three stages, a behavioural model, a structural or RTL model, and a gate level design. These gave us two kinds of feasibility check, that the logic we specified will execute the protocol within the environment we envisage, and that the design is technically feasible, for instance in a reasonable chip area.

A VHDL behavioral model for the system was initially written and tested. It includes the bypass chip, the host processor with a simplified bus/memory architecture, and a vestigial high-speed network interface adapter which acts simply as an infinite source/sink for data packets. The first design implemented the Basic Combined Subset (BCS) of the session layer and the Transport protocol class 2 (TP2) protocols. Only the logic for the data transfer phase is included in this model, as it is during this phase that the bypass chip is active. A host processor submodel generates a simple test sequence that initiates bypass connections and a series of bypassable data packets. Non-bypassable packets are also inserted to simulate for example the arrival of a connection release PDU. The test sequence allowed us to verify the following:

- Window flow control logic.
- Generation of the header field including the sequence number.
- Generation of acknowledgment packets on receive.
- Synchronization between the SPS and the bypass stack.

Next, the behavioural model of the bypass chip was manually converted to a structural (RTL) model for synthesis, leaving the other components as behavioral constructs. A behavioral description has no implied architecture in its representation, while an RTL level description has a definite architecture and clocking scheme, and characterizes the system in terms of registers, switches (multiplexors), and operations. An initial assignment of operations into clock cycles is also made. These descriptions, like behavioral descriptions, are technology-independent (silicon library independent). In a VHDL RTL level description, not all of the functionality of VHDL can be used because some of the language features do not map into the RTL model. For example a WAIT FOR statement has no meaning in hardware, but is useful in behavioral simulation. Features that can be easily mapped to hardware include IF THEN ELSE statements and signal assignments. The same test sequence was again generated by the host processor, and its results were compared with the original model to ensure behavioral consistency.

The structural model was then passed through the SYNOPSYS synthesis tool [31] for gate level generation with the $0.8\,\mu m$ BiCMOS macro library from Texas Instruments [32]. The timing information generated by this process was back-annotated to the structural model to obtain the simulation throughput results presented in section 5. This was sufficient for our present purposes, to estimate the space complexity and timing of the chip, and the final step of generating a chip layout for fabrication and fault analysis was not performed. As the model was too large for the SYNOPSYS package we were using, it was divided into 3 submodels. The dual-ported SRAM was not synthesized as its gate counts and performance characteristics can be easily extracted from data books.

The second design, with additional functionality, is described in the next section.

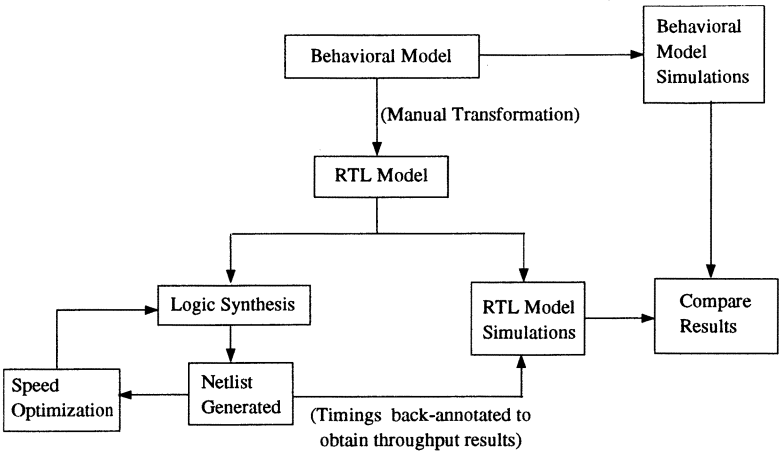


Figure 3 Design flow diagram

4.4 Behavioural description

The sequence of operation in the bypass system is summarized as follows:

1) Four high level procedures are made available to the host processor to control the bypass chip, namely: BYPASS_START, BYPASS_DMA, BYPASS_SYNC and BYPASS_RESTART. On receiving the first bypassable PDU, the BYPASS_START procedure is called. This procedure sets up a bypassable connection by sending information like its initial window flow control parameters and the DST_REF field to the bypass chip. These are stored in a process control block for the particular connection in fixed on-chip memory locations and are also accessible by the host (I/O mapped). The maximum number of connections allowed for simultaneous bypassing will be equal to the number of these control blocks allocated in the bypass chip (5 in this study — See figure 4).
2) For subsequent bypassable packets, the host processor initiates the BYPASS_DMA procedure which checks for free buffer space in the bypass chip and programs the DMA by sending the starting address pointer where the PDU is located, and its total length. The destination address is supplied by the bypass chip. Arbitration for the host processor bus between the host and DMA is provided by the DMAreq and DMAack lines. DMA transfers the PDU into the internal dual-ported SRAM (Static RAM). Buffers are pre-allocated in fixed sizes and are accessed by a simple round robin scheme using a set of buffer pointers.
3) The protocol engine polls the status field of a packet to check if there are any data to be processed. If the status is FILLED, protocol processing of the bypass stack can proceed. The dual-ported structure allows protocol processing to proceed concurrently with any DMA transfer to/from the host processor. A precomputed header template is used to construct the header field very quickly.

4) Processed in-sequence packets within the transmit window are passed to the network interface adapter, which in this study acted as an instantaneous sink.
5) Whenever the host processor encounters a switch in the processing path, i.e. from the bypass stack to the SPS, it will issue a BYPASS_SYNC procedure. This procedure will flush the bypass chip of any "in-transit PDU" for that particular connection and return any updated information, for example window control parameters, from the bypass chip to the host in order to maintain state consistency between the two paths. This may occur when it receives, for example, a connection release primitive or a session control primitive of the session layer (see section 6) during the data transfer phase.
6) Whenever the host processor wishes to re-enter the bypass path after a switch in the processing path, the host will issue a BYPASS_RESTART procedure which will pass only those data that were updated in the standard protocol stack, like window flow control parameters, back to the bypass chip. Parameters like the DST-REF field which is not changed for the duration of the connection need not be updated.

4.5 Second Design, including major procedures for Transport Class 4 (Implemented)

This section describes extensions to the first design, which only supports Session BCS and TP2 functionality, to include some common TP4 functionality. Procedures for checksum, retransmission on timeout and resequencing were implemented. Extensions to the Session layer functionality and procedures for presentation layer conversion were not implemented, but are also discussed in section 6.

4.5.1 OSI Checksum The transport protocol class 4 checksum algorithm [12] was implemented. It is often difficult to perform it on the fly at the sender end as the two-byte checksum field is placed in the variable part of the header. However, with the bypass system, the header structure and the position of the checksum field are known in advance. Also, calculation of the checksum can now be simplified further by precomputing the partial checksum of the header fields.

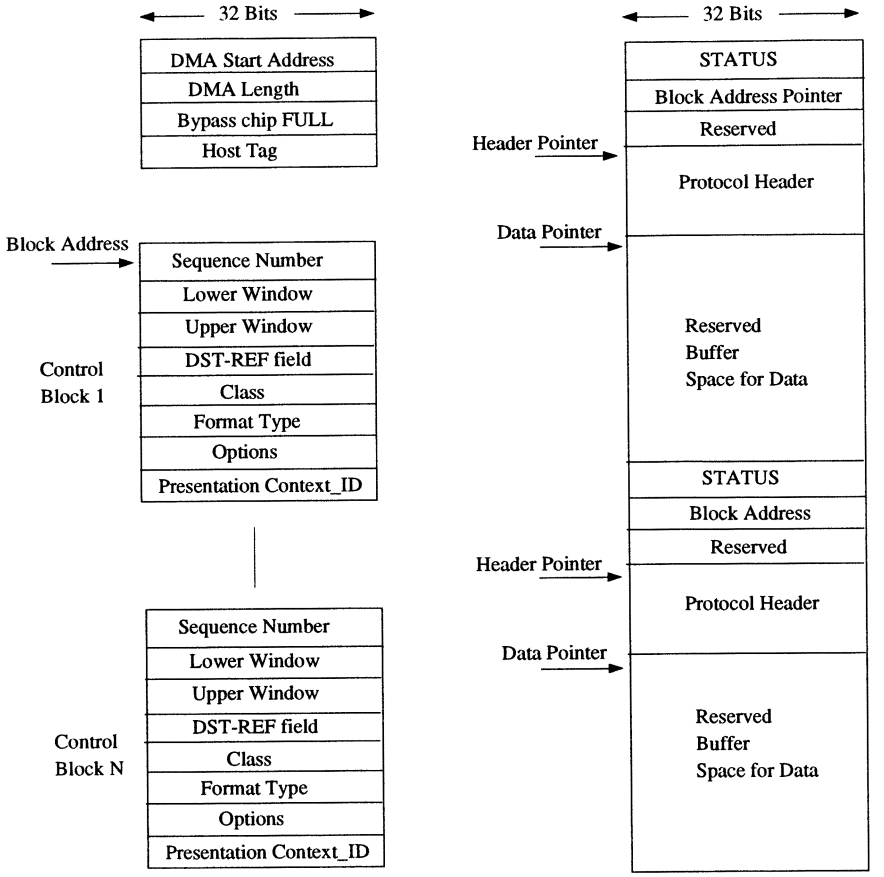
4.5.2 Timers

During the data transfer phase TP4 uses a Retransmission timer (T1), a Window timer (W) and an Inactivity timer (I). Only the retransmission timer with one interval per connection and the window timer were implemented here.

Timer management in software is an expensive process due to software interrupt handling overhead and update processing of the timer queue [34], but can be easily handled by VLSI implementation. On-chip timers are very efficient and can be executed concurrently with other protocol processes. The only overhead is in starting and stopping the timers. Once it is started, a timer is an autonomous process until an interrupt signal is activated. A separate area of on-chip memory is reserved to store the state information of the timer list.

4.5.3 Retransmission and Resequencing

At the receiver end, out-of-sequence PDUs outside the flow-control window will be discarded. Otherwise, a PDU is buffered for resequencing. Duplicate TPDUs can be detected



Host Tag : This tag is set on receipt of a host command, e.g BYPASS_START, BYPASS_DMA, BYPASS_SYNC or BYPASS_RESTART.

STATUS : Indicates the status of the buffer, e.g. EMPTY, FILLING, FILLED or CLOSED.

Figure 4 Organization of internal bypass chip memory

easily because the sequence number matches that of a previously received TPDU. At the sender end, if timer T1 expires, the transport entity can retransmit either the first TPDU, or all TPDUs (Go-back-N) waiting for acknowledgment. In this design the Go-back-N retransmission strategy was used. For a large window, the on-chip buffer may not be sufficient to hold the unacknowledged data packets for retransmission or to buffer data packets for resequencing, and slower external memory would be needed.

Procedures	*Combinational Area (Equivalent NAND2 gates)*	*Non Combinational Area (Equivalent NAND2 gates)*	*Total Area (Equivalent NAND2 gates)*	*Throughput performance with 1 Kbyte packet length (Mbps)*
Session (BCS)/ Transport Class 2	*6318*	*2929*	*9247*	*2,362.8*
Dual Ported SRAM (4 Kbyte)	*N/A*	*N/A*	*Approximately 41,984*	*N/A*
Session (BCS)/ Transport Class 4 with the addition of the Checksum Procedure with timer circuitries	*8334*	*4227*	*12561*	*313.3*

Table 2 Throughput Performance and gate count of bypass VLSI chip

5 Results

The timing information obtained from the netlist was back-annotated to the structural model to obtain throughput performance results. The operating parameters and assumptions made in this study are:

- A chip clock rate of 66 MHz.
- 1 Kbyte data packets
- Window size of 64. An acknowledgment packet is sent for every 20 packets received.
- The host bus/memory subsystem and network interface adapter were assumed to be infinite sinks/source of data packets.
- One thousand packets were processed for each iteration.
- 4 Kbyte of internal dual ported SRAM.

Table 2 shows the throughput performance of the ROPE chip. The throughput value includes the time taken to move the data packet out from the internal memory of the bypass chip to the network interface adapter, but not the data copy operation from the host memory. Hence the throughput result includes just one copy operation of the data packet. The total gate count for the bypass chip with Session (BCS), TP2 and 4 Kbyte internal dual ported SRAM is 51,231 equivalent NAND2 gates. With the additional TP4 procedures like checksum and timer circuitries, the total gate count increased to 54,545 gates. Texas Instruments offers a

gate array package with 112,000 usable gates (TGB1150). This leaves enough chip area for extra on-chip memory or hardwired presentation procedures. With no per-octet operations, i.e. just protocol processing of packet headers (TP2), the bypass stack could effectively achieve a throughput of 2.362 Gbps. This is consistent with results presented by Clark [5] which claims that TCP processing can achieve up to 800 Mbps (without checksum) on current RISC-based processors. With OSI checksum, the throughput performance drops to 313.3 Mbps. This is still more than an order of magnitude higher than an equivalent 19.2 Mbps optimized hand assembled software implementation of the OSI checksum algorithm on the VAX 8800, reported by Sklower [29].

6 Considerations for the Session and Presentation layers (not implemented)

The next steps would logically be to enhance the Session [18, 19] processing to the BSS (Basic Synchronized Subset) or BAS (Basic Activity Subset) level, or to add Presentation processing

In the BSS, synchronization points occur but the session services do not actually save session SDUs and do not themselves perform the recovery operations. These activities are the responsibility of the application layer or the application program. The session layer merely decrements the serial number back to the synchronization point and the user must apply it to determine where to begin recovery procedures. Hence these activities are best handled on the host processor and are not very suitable for bypassing. The benefit they would confer (if bypassed) would be to reduce the frequency of switching paths.

Presentation processing can definitely be bypassed. During the data transfer phase, it consists only of the Presentation data encoding/decoding functions. Substantial performance gains could result if the presentation conversions are simple and are used consistently, although the inflexibility of a hardware version is an evident weakness. One possible application of ROPE with hardwired presentation conversion is in video servers with the proposed encoding standards such as MPEG [13].

7 Summary

It can be concluded from this study that it is feasible to implement the bypass stack (at least for the transport and session layers) in VLSI and that the performance would be at least an order of magnitude higher than software protocol processing. The bypass system offloads the critical protocol functions and the associated non-protocol-specific functions onto a "Reduced Operation Protocol Engine" (ROPE). The gate count for the bypass chip can easily fit into a commercially available gate array Integrated Circuit. Per-octet operations are particularly efficient when performed on the chip. The host processor is relieved of a significant proportion of protocol processing and can concentrate on the application processing. The speed of communication processing in the host system can now match the transmission bandwidth of high-speed networks, e.g. ATM technology, thereby increasing the application-to-application throughput performance. (In an ATM system we assume that the segmentation

and reassembly or SAR operation would also be in hardware, since it is done frequently.) The host processor is also relieved of acknowledgment processing. An existing implementation of the OSI stack can be adapted for bypassing with only a small modification of the original software, thus providing an easy migration path for current systems.

The scope of functions included in a bypass may be narrowly defined, or more extended. A bypass does not include fast connection setup but also does not interfere with it. There is no segmentation/reassembly within the bypass path, but we do not see this as a major restriction, as research suggests that fragmentation of PDUs should be restricted only to the lower layers and should occur only once in the protocol stack [23]. The Segmentation and Reassembly sublayer of the ATM adaptation layer is a good place for such functions [25].

In the first design, the bypass chip with a 66 MHz clock can support a throughput rate of 2.3 Gbps (Session BCS and TP2) for 1 Kbyte TPDU packets using current $0.8\mu m$ BiCMOS technology. In the second design, extended to include the TP4 checksum, retransmission on timeout and resequencing procedures, the throughput decreased to 313.3 Mbps. Further advances in speed can be obtained in proportion to technology improvements, making the approach viable for some considerable time to come.

Acknowledgments

Bell-Northern Research provided the VHDL tools used in this study, and Dr. Simon Curry, Hemi Thakar, Dr. Parviz Yousefpour, Bernard Doray, Mike Majid and Mustapha Bourahla helped with their use. This research was supported by the Ontario government program of Centers of Excellence, through the Telecom Software Methods Project of TRIO, the Telecommunications Research Institute of Ontario.

References

[1] Balraj T.S. and Yemini Y., "Putting the Transport Layer on VLSI - the PROMPT protocol chip". IFIP, Stockholm, May 13-15, 1992.

[2] Beach B., "UltraNet: An Architecture for Gigabit Networking," in Proc. 15th Conference on Local Computer Networks, Minnesota Oct, 1990.

[3] Chesson G., "XTP/PE Design Considerations," in Proc. IFIP Workshop Protocols for High-Speed Networks, Zurich, May 9-11, pp. 27-33, 1989.

[4] Clark D. and Tennenhouse D., "Architectural Considerations for a New Generation of Protocols," in ACM SIGCOMM 1990.

[5] Clark D., Jacobson V., Romkey J., and Salwen H., "An analysis of TCP processing overhead," in IEEE Commum. Mag., vol. 27, pp. 23-29, June 1989.

[6] Coelho D.R., "The VHDL Handbook," Kluwer Academic Publishers, 1989.

[7] Cooper E.C, Steenkiste P.A., Sansom R.D. and Zill B.D., "Protocol Implementation on the Nectar Communication Processor," in ACM SIGCOMM'90, 1990.

[8] Dalton C., Watson G., Banks D., Calamvokis C., Edwards A. and Lumley J., "Afterburner," in IEEE Network July 1993.

[9] Davie B.S., "Architecture and Implementation of a High-Speed host Interface," in IEEE Journal on selected areas in communications, Vol. 11, No. 2, February 1993.

[10] Doeringer W.A., Dykeman D., Kaiserwerth M., Meister B., Rudin H., and Williamson R., "A survey of Light-Weight Transport protocols for High Speed Networks," in IEEE Trans. on Comm., vol. 38, No. 11, pp 2025-2039, Nov. 1990.

[11] Druschel P., Peterson L.L., "Fbufs: A High-Bandwidth Cross-Domain Transfer Facility", in the Proceedings of the 14th ACM Symposium on Operating Systems Principles, Dec 1993.

[12] Fletcher J.G., "An Arithmetic Checksum for Serial Transmissions" in IEEE Transactions on Communications, Vol. Com-30, No. 1, Jan 1982.

[13] Gall D.L., "MPEG," in Communications of the ACM, Apr. 1991.

[14] Giarrizzo D., Kaiserswerth M., Wicki T. and Williamson R., "High-Speed Parallel Protocol Implementation," in Proc. IFIP Workshop Protocols for High-Speed Networks, Zurich, May 9-11, 1989.

[15] Haas Z., "A Communication Architecture for High-Speed Networking," in IEEE INFOCOM, pp. 433–441, June 1990.

[16] Heatley S., Stokesberry D., "Analysis of Transport Measurements Over a Local Area Network," IEEE Communications Magazine, Jun 1989.

[17] Hennessy J.L and D.A. Patterson, "Computer Architecture: A quantitative approach," Palo Alto, California, Morgan Kaufmann Publishers 1991.

[18] Information processing systems — OSI Basic connection oriented session protocol specification, standard ISO-8327, 1987.

[19] Information processing systems — OSI Basic connection oriented session service definition, standard ISO-8326, 1987.

[20] Jacobson V., "4BSD TCP Header Prediction," in ACM SIGCOMM, Comp. Commun. Review, vol. 20, no. 2, pp. 13–16, Apr. 1990.

[21] Jain N., Schwartz M. and Bashkow T.R., "Transport Protocol Processing at Gbps rates," ACM SIGCOMM '90 Symp, Philadelphia, pp. 188-199, Sep. 24-27, 1990.

[22] Kanakia H. and Cheriton D., "The VMP network adapter board (NAB): High-performance network communication for multiprocessors," ACM SIGCOMM '88 Symp., Stanford, CA, pp. 175-187, Aug. 16-19, 1988.

[23] Kent C.A., Mogul J.C., "Fragmentation Considered Harmful," ACM SIGCOMM '87 Workshop Comp Commun Review, vol. 17, no. 5, Special Issue, Aug 1987.

[24] Khrishnakumar A.S., Sabnani K., "VLSI Implementations of Communication Protocols — A Survey," IEEE Journal on Selected Areas in Communications, vol. 7, no. 7, pp. 1082–1090, Sep 1989.

[25] Lyles J.B., Swinehart D.C., "The Emerging Gigabit Environment and the Role of Local ATM," in IEEE Communications Magazine, Apr 1992.

[26] Partridge C., "Gigabit Networking," in Addison Wesley Professional Computing Series, 1993.

[27] Perry D.L., "VHDL," McGraw-Hill Inc., 1991.

[28] Ramakrishnan K.K., "Performance Considerations in Desigining Network Interfaces". IEEE Journal on Selected Areas in Communications, Vol. 11, No. 2, Feb 1993. .

[29] Sklower K., "Improving the Efficiency of the OSI Checksum Calculation," in ACM SIGCOMM Computer Communications Review, Vol. 19, No. 5, pp. 32-43, Oct. 1989.

[30] Sterbenz J.P.G. and Parulkar G.M., "AXON: A High Speed Communications Architecture for Distributed Applications," in IEEE INFOCOM, Jun 1990.

[31] SYNOPSYS Design Analyzer™ Reference Manual, Version 2.0, May 1991.

[32] TGB 1000 Series 0.8um BiCMOS Gate Arrays Macro Library Summary, Application Specific Integrated Circuits, Texas Instruments, Dec. 1991.

[33] Thia Y.H., Woodside C.M., "High-Speed Protocol Bypass Algorithm with Window Flow Control", in Proc. of the 3rd IFIP International Workshop on Protocols for High-Speed Networks, Stockholm, May 13-15, 1992.

[34] Varghese G. and Lauck T., "Hashed and Hierarchical Timing Wheels: Data structures for the efficient implementation of a Timer facility," in Proc. of the 11th ACM Symp. on Operating System Principles, Nov. 1987.

[35] Watson R.W. and Mamrak S.M., "Gaining efficiency insmith transport services by appropriate design and implementation choices," in ACM trans. on Computer Systems, vol. 5, no. 2, pp. 97–120, May 1987.

[36] Woodside C.M. and Montealegre J.R., "The effect of buffering strategies on protocol execution performance," in IEEE Trans. Commun., vol. COM-37, pp.545-554, June 1989.

[37] Woodside C.M., Ravindran K. and Franks R.G., "The protocol bypass Concept for High Speed OSI Data transfer," in Proc. IFIP Workshop on Protocols for High-Speed Networks, Zurich, May 9-11, 1990.

[38] Zitterbart M., "High-Speed Protocol Implementations based on Multiprocessor-Architecture," in Proc. IFIP Workshop Protocols for High-Speed Networks, Zurich, May 9-11, 1989.