A Highly Parallelized Architecture for a DQDB Node

Lütfullah Kuşdemir Semih Bilgen

Dept. of Electrical and Electronics Engineering
Middle East Technical University, 06531 Ankara

Abstract

This paper proposes a higly parallelized architecture for a DQDB MAC layer protocol processor, which is expected to improve the speed of execution of the protocol significantly to match the Gbps transmission rates. The architecture is based on pipelines of service blocks operating in parallel. Also the receive and transmit functions are separated and parallelized. Thus, basically, the isochronous, connection-oriented and MAC services are defined as three parallel blocks, each consisting of parallel receive and send functions. Fundamental to the architecture are multiport global receive and send memories which allow manipulation of separate data units in parallel without having to physically transfer data between different service stages. Each service block is divided into parallel subblocks, thereby exploiting the possibility of parallelism and pipelining as much as possible. Key-words: HSLAN, MAN, DQDB, Parallel Architectures.

1: Introduction

Local area networks with several Mbps data rates are well known today. To match the several Gbps data rates made possible by fiber optic links, the layers, especially the Physical and Medium Access Control (MAC) layer protocols should achieve better performances. Protocol processing speeds present a significant bottleneck in utilizing the speeds afforded by optical transmission technologies. To achieve better performance there exist several possibilities, which can generally be used to obtain better performance within a computer system:

- faster chip technologies,
- parallel algorithms,
- optimizing compilers,
- new architectures.

In the case of communication protocols for high speed networks, among several possibilities, suggested solutions are:

- new streamlined protocol architectures, sometimes referred to as light weight protocols, e.g VMTP [1], XTP/PE [2,3],
- optimized implementation of new and traditional protocols using special hardware, e.g. translating protocol specifications to VLSI design [4], PSi [5],
- implementation of those protocols on multiprocessor architectures, e.g. 1 Gbps Implementation of Physical Layer protocols [6].

In the literature, the multiprocessor implementation of OSI layers has been proposed, e.g. Multiprocossor architecture of the lower layers of a transport protocol system [7]. For the medium access control (MAC) protocol layer, specialized hardware (and thus expensive) solutions are proposed as well. This paper describes another approach; using general purpose processors in a multiprocessor environment to implement the medium access control protocol to achieve a significant performance gain. Although the focus is on the MAC layer of IEEE 802.6 DQDB, the proposed technique is being applied to other protocol specifications like FDDI-I, and FDDI-II . In the end, a flexible medium access control protocol processor with a highly layered architecture, suitable for VLSI implementation will be obtained.

This paper is organized as follows: First, an overview of the DQDB standard is presented. This includes summaries of the access protocol and DQDB services, as well as a brief description of the functional architecture of a generic DQDB node. Then, we focus on parallelism in communication architectures, and finally the proposed highly parallel DQDB MAC layer protocol processor architecture is presented.

2: An overview of the DQDB standard

2.1: The access protocol

As the IEEE 802.6 standard describes, the DQDB subnetwork is a distributed multi-access network that supports integrated communications [8]. In particular connectionless data transfer, connection-oriented data transfer and isochronous communications are the

0-8186-3670-X/93 $03.00

supported services which can flexibly share the total bandwidth. The DQDB subnetwork consists of two unidirectional buses and a multiplicity of nodes along them. The buses support communication in opposite directions allowing full duplex communication between any pair of nodes. The head nodes continuously supply fixed length time slots down to the buses. Nodes access the time slots via a global distributed queueing algorithm. The operation of the algorithm is based on a Busy/Idle bit and a Request Field in the header of each time slot. In particular, a node requests a time slot on one of the buses ("downstream" toward an intended receiver) by setting a request bit on the opposite bus to inform "upstream" nodes that an additional segment of information is queued for access. Nodes continuously monitor the buses, counting the requests and the idle time slots that pass by, so that they always know how many time slots are reserved by "downstream" nodes. From this information, a node can determine its position in the distributed queue, and know when it has access to an idle time slot. The throughput of the DQDB subnetwork is increased if a time slot is used multiple times as it flows down the bus, known as "destination release of slots". In the same spirit, some "erasure nodes" [9,10] on the buses may be incorporated to erase the slots that have been marked as "read" so that they can be reused downstream for other transmissions. Studies have shown that the access mechanism is unfair [11]: a node's delay-throuhput performance is strongly dependent on its position along the dual bus. To improve fairness, a procedure referred to as Bandwidth Balancing Mechanism [8], whereby a node occasionally skips the use of empty slots, has been added to the standard. The Physical layer can be aligned with a number of different standarts, e.g. Synchronous Optical Network/ Synchronous Digital Hiararchy (SONET/SDH).

2.2: DQDB layer service definition

The functional architecture for a generic DQDB node is shown in Figure 1 [8]. It consists of two layers: the Physical Layer and the DQDB layer. The latter uses the services of the former to provide a number of different services. The DQDB layer services are as follows:

1.The MAC service provided to the LLC sublayer.

2.Isochronous Service, provided to an Isochronous Service User. This service supports the transfer of isochronous service octets with a constant interarrival time over an isochronous connection.

3.A connection-oriented data service that supports the transfer of data over virtual channels. This service is asynchronous because there is no guarantee of constant interarrival time for data units.

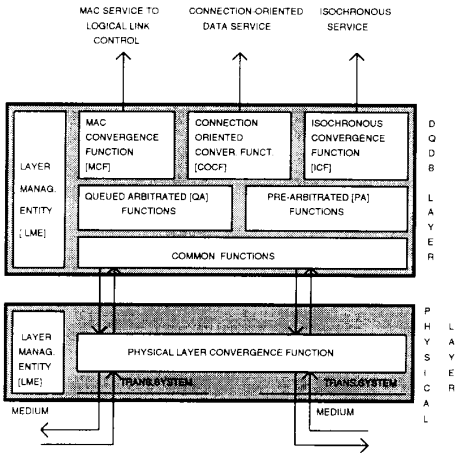

Figure 1. DQDB Node Functional Architecture

For detailed description of the services, the reader is referred to [8]. Realizing three services, common functions, queued arbitrated (QA) functions, pre-arbitrated (PA) functions and convergence functions as well as the DQDB layer management entity (LME) are essential blocks of a generic DQDB layer processor. The standard will not be described here, but detailed functional realization will be presented for a parallel implementation, below.

3: Parallelism in communication architectures

Communication architectures are typically structured and described in the form of hierarchical protocol layers as exemplified by the OSI reference model. Specifically, the OSI reference model exhibits parallelism in a number of places: between protocol layers, within individual protocol layers, and finally within the entire communication architecture [12]. The approach has been taken to improve the performance of the IEEE 802.6 MAC protocol, namely to exploit the inherent parallelism in its architecture.

Parallelism can be achieved by vertical and horizontal subdivisions [7]. Vertical subdivision results in a pipeline structure that performs overlapped computation to exploit temporal parallelism. An acceptable "speed up" is only achievable, if all pipeline stages will need almost the same processing time. Otherwise the utilization of the processors differ and some processors will be idle for a certain time interval. In addition to the vertical subdivision, the sublayer can be subdivided horizontally into two parts; namely send and receive parts. The resulting architecture is based on two pipelines: a send pipeline and a receive pipeline which almost work

independently from each other. To reduce the overhead caused by data moving from one stage of the pipeline to the other a "global data memory" is necessary. In order to decrease the number of accesses to the global data memory which otherwise would be a performance bottleneck of the system, there must be a global separate data memory for the receive part as well as for the send part.

4: Highly parallel DQDB node functional description

The DQDB Layer uses the services of the Physical Layer, to provide its own services to the higher layer. The functional architecture of a parallel DQDB layer subsystem of a node is shown in Figure 2. The proposed architecture aims to exploit the inherent parallelism within the original generic architecture. The transmit and receive functions are separated horizontally. The three service access points are for the isochronous service, connection oriented service and connectionless service. The service blocks are functionally organized as separate blocks as well as the separate receive and transmit parts, in the horizontal direction. Vertically, the receive and transmit parts of each service block consist of a number of pipeline stages or subblocks with comparable operational complexity. Specifically, the connection-oriented and connectionless service blocks consist of six stages in the transmit pipeline and seven stages in the receive pipeline (See Figure 5). The isochronous service block consists of two stages in both the receive and transmit pipelines (See Figure 3).

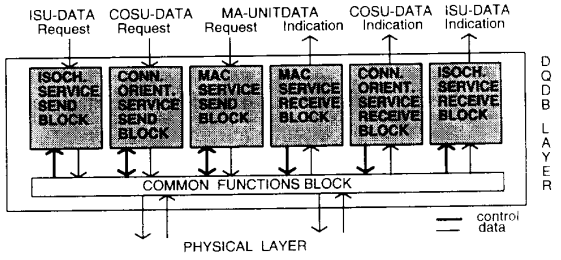

Figure 2. Functional Architecture of parallel DQDB Layer Subsystem of a Node

4.1: Isochronous service transmit and receive blocks

The isochronous service provided by the DQDB layer supports the transfer of an isochronous service octet from one DQDB node to one or more peer DQDB nodes. Referring to Fig. 3, the send and receive blocks for isochronous service are separated. The isochronous convergence functions block and pre-arbitrated transmit functions block are again horizontally organized as separate send and receive parts. The send and receive parts of the isochronous service block have their own local memories.

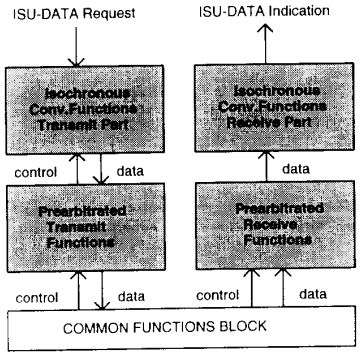

Figure 3. Isochronous Service Transmit and Receive Blocks

The isochronous service octet is received from the isochronous service user (ISU) at the source node as the isochronous Service Data Unit (ISDU) in an ISU-DATA request. Isochronous convergence functions transmit block places it in its FIFO buffer in the order received from the ISU. When PA Functions Block Transmit part requests, it passes the current octet to be sent. The PA Transmit Function block examines the VCI in the PA segment header of PA slots received on Bus A or Bus B, and determines if there is any offset value for this PA segment payload that are currently associated with transmission for an isochronous service octet from the ICF transmit block. In the case of a match, transmits the service octet to the common function block.

The PA receive function block examines the VCI in the PA segment header of PA slots received on any bus and determines if any offset value is involved. If so, it passes the octet to the ICF receive functions block.

4.2: Connectionless data service transmit and receive parts

The provision of MAC service to LLC consists of the segmentation of MAC Service Data Unit (MSDU) at the source into fixed length units and the transfer of these units to the destination, which reassembles them into the MSDU. Figure 4 shows the basic architecture of the connectionless data service block. Horizontally, it is divided into transmit and receive parts with their own

memories. It mainly consists of the MAC Convergence Functions (MCF) and Queued Arbitrated (QA) Access Functions subblocks. Figure 5 expands the connectionless data service block into its details.

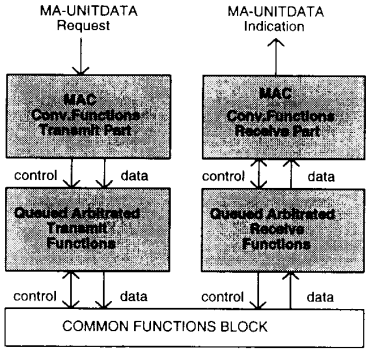

Figure 4. Connectionless Data Service Block

In the vertical direction, the send and receive functions are functionally layered to have a send and a receive pipeline. The Global Block Memory (GBM) is a shared, fast multiport memory, added to reduce the overhead caused by moving data from one stage to the other. In order to decrease the number of accesses to the GBM , it is organized as separate global data memories for the receive part as well as for the send part. The boxes at the sides of the memory block represent the subfunctional units, each would be implemented either by a processor or by a dedicated hardware. The only passed parameters between subfunctional units are the control parameters and pointers to the data residing in GBM. There exists no transfer of whole user service data units, which otherwise would be a performance bottleneck of the system. The following sections will describe the details of the functional parts, referring to the DQDB layer protocol data unit formats in [8].

4.2.1: Transmit part: Connectionless Data Service Transmit part consists of a transmit pipeline and its global transmit memory. The transmit pipeline can be considered functionally as two cascaded pipelines; the MAC Convergence Functions Block, and Queued Arbitration Functions Block. The following sections, hierarchically describe the task of each processor and the interaction between processors.

4.2.1.1: MAC convergence function (MCF) block: This section specifies the functions performed by the MCF block upon receipt of a single MAC Service Data Unit (MSDU). MCF Block consists of three pipelined subblocks.

♦ **Initial MAC protocol data unit (IMPDU) creator:** Upon receipt of MA_UNITDATA request from the LLC sublayer, it extracts the necessary data to create an IMPDU. The details of coding the fields and subfields in an IMPDU are given in the following paragraphs.

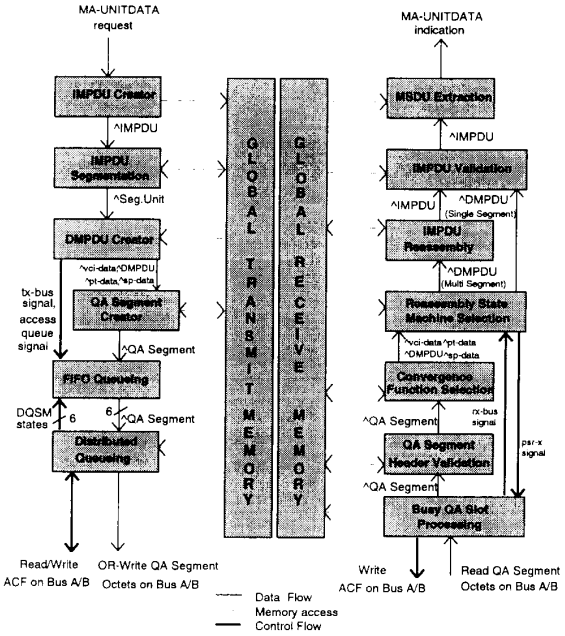

Figure 5. Detailed Connectionless Data Service Block

To create the Common Protocol Data Unit (PDU) Header, the IMPDU creator first codes the Reserved Field to zero, then selects a BEtag value and codes it. Finally BASize field is coded to the number of octets contained in the MAC Convergence Protocol Header, the header extension field, the MSDU, the PAD field and the CRC32 field (if present).

To create the MAC Convergence Protocol (MCP) Header, the IMPDU creator codes the destination address parameter, and the source address parameter received in the MA_UNITDATA request into DA field and SA field, respectively. Codes the Protocol Identification (PI) field to 1 (i.e LLC) and codes the PAD Length (PL) field to ((INFO+PAD)mod4). The priority parameter passed in MA_UNITDATA request is coded into QOS_Delay subfield while QOS_Loss subfield to zero. According to the CRC332_GEN_CONTROL flag, it codes the CRC32 Indicator Bit (CIB). Finally the Header Extension Length (HEL) subfield is coded and the Bridging field is coded to zero.

By concatenating the above two headers, the IMPDU header is obtained. The IMPDU creator extracts the data parameter (MSDU) from the received MA_UNITDATA request and codes it to INFO field up to and including

9,188 octets for PI=1 (LLC). Creates PAD field with "zero" octets. If required, the CRC32 field is filled in then. Finally the Common PDU Trailer which is the exact replica of the Common PDU Header is appended to complete the IMPDU creation.

IMPDU Creator then passes the pointer to IMPDU in GBM to the IMPDU Segmentation Block.

♦ **IMPDU segmentation block:** It gets the pointer to the IMPDU in GBM from IMPDU creation block and divides it into one or more segmentation units of eact 44 octets long, and stores them into global transmit memory while passing the associated pointers to the DMPDU Creation Block. In fact, the segmentation is purely logical and it involves generating pointers only.

♦ **Derived MAC protocol data unit (DMPDU) creator:** The DMPDU Creator accepts the pointer to the segmentation unit, asserted by the IMPDU Segmentation Block and by prepending a header and by appending a trailer, it constructs a DMPDU in the pointed location in GBM.Eventually passes the pointer information to the Queued Arbitration Function Block together with the other necessary control information such as;VCI_DATA Pointer, PT_DATA Pointer and SP_DATA Pointer, ACCESS_QUEUE_SIGNAL and TX_BUS_SIGNAL,

♦ **Transmit interactions between MCF block and QA block:** The interaction between the MCF block and Queued Arbitrated Functions block for the transfer of a DMPDU as the payload of a QA segment is independent of the type of the DMPDU, but is dependent of the parameters received in the MA_UNITDATA request.

(1) The TX_BUS_SIGNAL control shall be asserted by the DMPDU creator to the same value for every DMPDU derived from the same IMPDU. This signal indicates on which bus or buses each DMPDU should be sent. DMPDU creator checks the just generated, first (BOM) DMPDU in which the destination address parameter resides to determine the appropriate value for TX_BUS_SIGNAL (BUSA, BUSB, BOTH).

(2) The ACCESS_QUEUE_SIGNAL control shall be asserted by the DMPDU creator to the same value for every DMPDU of the same IMPDU and indicates in which priority level access queue the QA segment should be placed. The values are 0,1 or 2.

(3) VCI_DATA (virtual circuit identifier), PT_DATA (Payload Type) and SP_DATA (Segment Priority) shall be prepared by the DMPDU creator and the pointers to them shall be asserted to the QA Functions Block which would then use to prepare the QA segment header.

(4) The pointer to DMPDU as the payload of QA segment shall be passed to the QA Functions block aswell.

4.2.1.2: Queued arbitrated (QA) functions block

♦ **QA segment creator:** Upon receipt of the pointers associated with QA segment payload from the MCF pipeline block, the QA Segment Creator shall add the protocol control information by coding the VCI field, the Payload_Type field, and the Segment_Priority field of the QA segment payload and then calculate and code the Segment_Header_Check_Sequence field of QA segment header which is finally prepended to the QA segment payload to create a QA segment.

♦ **FIFO queue:** In the proposed standard it is declared that there can be a maximum of six QA segments queued within the Distributed Queue for access at each node, one for each combination of TX_BUS_SIGNAL and ACCESS_QUEUE_SIGNAL. However FIFO queueing block implements 6 local FIFO queues of QA segments for each combination of bus and access queue priority level so that it can accept multiple QA segments. The QA segment at the head of a FIFO queue is placed in the Distributed Queue for that combination of bus and access queue priority level if the associated Distributed Queue State Machine (DQSM) is in the IDLE state. In other words; the FIFO Queueing block accepts pointers to the QA segments and according to the bus and access queue control signals asserted by the MCF block (DMPDU creator) and queues them in its local queues. It also gets the DQ_State_Machine control information from DQ block. If the state is IDLE then asserts the pointer of the QA segment at the head of the queue to the Distributed Queue.

♦ **Distributed queue:** The DQ performs the medium access control procedure for the write access of QA segments into empty QA slots. The DQ has six DQSMs operating: request counters, countdown counters, local request queue counters for each combination of bus and access queue priority level. DQ also operates a bandwidth balancing counter for each bus. DQ gets the pointer to QA segment and directs it to the associated Distributed Queue State Machine if its in IDLE state. The DQ Function needs to be able to read the BUSY bit and REQUEST field of all slots passing on both buses. When it determines that a QA segment is permitted to gain access to an empty QA slot on either bus then it OR-writes the QA segment octets with the octets of the segment field of the QA slot as they pass along the bus.

4.2.2: Receive part

4.2.2.1: QA receive functions block: This section specifies the functions performed by the QA functions block upon receipt of an QA slot.

♦ **Busy QA slot processing:** When the common functions block receives an ACF with the BUSY bit set to one and the Slot_Type bit set to zero it then delivers a copy of the octets of the QA segment to the Busy QA Slot Processing block. This block collects the QA segment octets received from a busy QA slot on Bus x (x=A or B)

and writes them into the GBM while passing the pointer to them to the QA Segment Header Validation Block.

♦ **QA segment header validation:** Upon receipt of a QA segment, this block validates the correctness of the QA segment header using the Header Check Sequence (HCS). Then,

(1) If the HCS is not valid, then it may perform error correction on the QA segment header or it may be discarded.

(2) If the HCS is valid, or if error corrected, then the pointer to QA segment is passed to the Convergence Function Selection Block.

♦ **Convergence function selection block:** On the completion of the QA segment header verification, this block shall extract the VCI value in the QA segment header. It shall then examine the VCI value to determine whether it is currently associated with the MAC Convergence Function block at this node. If not, the QA segment is discarded. If so, the pointer to the QA segment payload is passed to the MCF block.

4.2.2.2: MCF receive functions: The process of reassembly is used to reconstruct an IMPDU from the segmentation units contained in DMPDUs whose pointers are received by the MCF block.

♦ **Reassembly state machine selection (RSMS):** When the RSMS block receives the pointer of a DMPDU from the QA functions block at a connectionless VCI which the MCF block is programmed to receive, then it validates the correctness of the DMPDU using the Payload_CRC Subfield in the DMPDU trailer. Then,

(1) If the DMPDU is valid and the destination address (DA) field of the MCP header matches an individual MSAP address which the node is programmed to receive, then RSMS Block shall perform the following operations, depending upon the value in the Segment_Type Subfield of the DMPDU header.

A) Segment_Type=BOM,COM, or EOM. Then it forwards the relavant pointer to the reassembly State Machine associated with the VCI/MID pair.

B) Segment_Type=SSM. Then it means that the DMPDU contains a complete IMPDU i.e. a single segment message and does not need to be forwarded to a reassembly process, but its pointer to the IMPDU validation block.

If the DMPDU is destined only to this node, RSMS block shall assert the PSR_x_SIGNAL (Previous Segment Received on Bus x) to the Busy QA Slot Processing Block in the QA Receive Functions Block which then will be used by QA Slot Processing Block to set the PSR bit in the ACF of the next QA slot passing on Bus x.

(2) If the DMPDU is not valid, it shall be discarded by not asserting the PSR_x_SIGNAL to QA Receive Block.

♦ **Reassembly of the IMPDU:** A reassembly process is activated when a pointer to a BOM DMPDU is received from the RSMS Block. The reassembly process is associated with the value of VCI_DATA for the BOM DMPDU and the MID value in the BOM DMPDU header. The process extracts the BOM segmentation unit. The IMPDU to be reassembled in this block, as a default, consists of multiple DMPDUs with the same value of VCI_DATA and the same MID value in each COM DMPDU. The value of the Sequence Number should be larger than that of the previous DMPDU bye one (modulo 16). The reassembly process logically extracts the segmentation units and appends it to the previously received BOM segmentation unit and COM segmentation unit. Appending the EOM segmentation unit stops the reassembly process.

The reassembly process definitely depends on the implementation of the data structures in the global receive memory. For example it could be just constructing a linked list of previous pointers.

The pointer to the reassembled IMPDU is then passed to the IMPDU validation block.

♦ **IMPDU validation block:** Each completely received IMPDU either fully contained in an SSM DMPDU or from a completed reassambly process is validated by performing the following operations:

(1) The value in the Length Field of the Common PDU trailer is compared with the number of octets in the IMPDU. A mismatch shall cause the block to discard the IMPDU.

(2) The value in the BEtag Field of the Common PDU header is compared with the value in the trailer. A mismatch again discards the IMPDU.

(3) If the CRC32 Indicator Bit (CIB) is set to one, and the CRC32_CHECK Flag is set to ON, then CRC32 field is used to validate the IMPDU. A mismatch discards the IMPDU.

(4) The Header Extension Length Field of the MCP header is checked to see if it is valid. A value outside the valid range of 0 to 5, inclusive, shall cause the block to discard the IMPDU.

If all four operations are successful, then the pointer to IMPDU is passed to the MSDU Extraction Block.

♦ **MSDU extraction block:** The MAC Service Data Unit Extraction Function Block takes the pointer to a validated IMPDU and performs the following functions to create the parameters passed in the MA_UNITDATA indication:

(1) Use the value in the DA field of the MCP header to create the destination_address parameter.

(2) Use the value in the SA field of the MCP header to create the source_address parameter.

(3) Use the value in the QOS_DELAY subfield of the MCP header to create the priority parameter.

(4) Extract the MSDU contained in the INFO field to create the data parameter.

The MSDU Extraction Function Block then generates an MA_UNITDATA indication primitive to the LLC sublayer. Note that the MA_UNITDATA indication primitives shall be generated in the same order as the block receives the pointers to IMPDUs from IMPDU Validation Function Block.

4.3: Connection oriented service transmit and receive blocks

The connection oriented data service provides a virtual channel between a pair of that service users. The DQDB Layer supports the transfer of data for the connection oriented data service user from one DQDB node to one or more peer DQDB nodes. Fig.6 summarizes the architecture for the transmit and receive blocks.

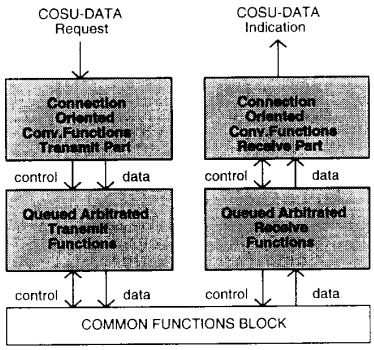

Figure 6. Connection Oriented Service Transmit and Receive Blocks

In the horizontal direction, transmit and receive functions are separately organized with their own memories. The Connection Oriented Transmit and Receive Function Blocks use the same segmentation and reassembly procedures as those defined for the connectionless data service. The internal organizations are almost the same with that of MAC Convergence functions transmit and receive blocks. The architecture indeed, consists of two pipelines with their own memories just as in the case of connectionless data service parallel architecture. So, the details of the convergence functions block and QA functions block will not be shown again. Although the architecture is similar, the functions of some of the fields of the common PDU header and the trailer differ for COCF. Also the QA segment payloads generated by the COCF and MCF have the identical formats, but the use of, for example, the Payload_Length Field differs for data transfers by the COCF, as partial filling of BOM and COM segmentation units is allowed.

5: Expected performance of the architecture

Access to multiport memory is the decisive factor in determining the overall performance. For the receive and send multiport memories, a time division scheme [12] will be used as shown in Figure 7. The proposed architecture for multiport memory is simple enough to have very short internal cycle time and indeed it offers a simple solution as regards the access control, the number of busses, number of latches etc. Ordinary RAM chips will be used in the RAM block.

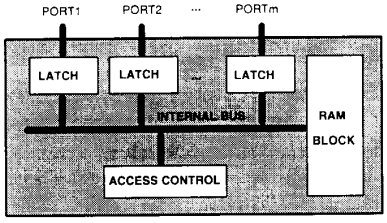

Figure 7. Multiport memory implementation architecture

The internal bus connecting the RAM block and the latches should have a high transfer rate. To achieve this a 64-bit internal bus will be used. With the BICMOS 4Kx4 bit SRAM (which are available in the market) having an access time of 3 ns and with 2 ns BICMOS latches, the multiport memory can be constructed to offer an access cycle time of 15 ns [13]. (This number is based on an internal cycle time of 4.5 ns for the multiport memory and assuming an 8B10B encoding scheme.)

For the processing units, a simple and high performance architecture is needed. Todays RISC processors (MIPS R4400, MC88110, etc.) are simple and fast enough (machine cycle times down to 8 ns.) [14, 15] to cope with the high transfer rates at the ports of multiport memory and execute their concurrent tasks.

Based on the assumptions of a general purpose RISC processor with 100 MIPS processing capacity and parallel DMA capability with multiple communication ports inter connecting processing blocks and a multiport memory unit with the specifications given above, we have shown [16] that the processing overhead per slot turns out to be less than 424 ns which guarantees at least 1 Gbps data rate.

6. Conclusion

The proposed architecture is currently being analyzed and verified. Effectiveness of the proposed pipeline

structure in terms of comparability of the time taken by each stage for completion of processing is among the major concerns. Also crucial are processor speeds and multiport memory access times. With current off-the-shelf components, at least 1 Gbps data rate can be sustained. A general simulation-based evaluation of the performance depending on processor speeds, memory characteristics and traffic load mixes and volumes is currently undertaken.

A drawback of the architecture is that its implementation requires expensive memory chips for the multiport memory. But it has several advantages over the conventional designs. These include an extremely high throughput and improved modularity.

Modularity brings in the following advantages.

- Within the architecture, it is easy to make changes to the software of individual sublayers with minimal inpact on the neighbouring sublayers.
- The many identical processors permit the use of common software and hardware development platforms.
- For principally economic reasons, general purpose processors are aimed to be used. However hard-wired state machines, microprogrammed state machines can also be used at any sublayer.

Speedup is mainly brought about by the pipelines in the service blocks. Parallelism of the six and seven stages in the transmit and receive pipelines, respectively, of the connectionless and connection-oriented service blocks, and of the two stages in each of the isochronous service pipelines constitute the main advantage of the proposed architecture. As the connection-oriented, connectionless and isochronous service blocks are to be implemented in parallel, optimum performance will result when network load is evenly distributed among the three types of service.

References

[1] Cheriton, D.R., "VMTP: Versatile Message Transaction Protocol - Protocol Specification", Network Working Group, Requests for Comments, RFC 1045, February 1988.

[2] Chesson, G., et.al. " XTP Protocol Definition, Revision 3.1", Protocol Engines Incorporated, 28. 3.88 Sept. 1987, pp. 70-77

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

[4] Krishnakumar, A.S., et.al. "Translation of Formal Protocol Specifications to VLSI Design", in Proc IFIP Workshop on Protocol Specification, Testing and Verification VII, 1987, pp.375-385.

[5] Abu-Amara, H., et.al "PSi: A Silicon Compiler for Very Fast Protocol Processing", Proc. IFIP Workshop on Protocols for High Speed Networks,1989, pp.181-195

[6] Cristensen, S.S., et.al. "A Flexible 1 Gbit/s Implementation of Physical Layer Protocols", Proc. IFIP Workshop on Protocols for High Speed Networks II, 1991, pp.173-199.

[7] Zitterbart, M., "High-Speed Protocol Implementations Based on a Multiprocessor Architecture", Proc. IFIP Workshop on High Speed Protocols, 1989 pp.151-163.

[8] DQDB Subnetwork of a MAN, IEEE Std.802-6-1990

[9] Rodrigues,M.A., "Erasure node: Performance Improvements for the IEEE 802.6 MAN", IEEE Infocom 1990, pp.636-643.

[10] Garret, M.W., Li, S.-Q., "A Study of Slot Reuse in Dual Bus Multiple Access Networks", IEEE Journal on Selected Areas in Communications Feb. 1991, Vol.9, No.2.,pp.248-256.

[11] Conti, M.,et.al. "A Methodological Approach to an Extensive Analysis of DQDB Performance and Fairness", IEEE J. on SAC. Jan. 1991, Vol.9, No.1.,pp.76-87.

[12] Giarrizzo,D.,et.al. "High Speed Parallel Protocol Implementation", Proc. IFIP Workshop on High Speed Protocols, 1989 pp. 165-180.

[13] "Product Focus: Ultra-Fast SRAMs", Computer Design, June 1993, pp.101-106.

[14] Diefendorff K., Allen M., "Organization of the Motorola 88110 Superscalar RISC Microprocessor", IEEE MICRO, April 1992, pp.40-63.

[15] Mirapuri S.,et.al. "The MIPS R4000 Processor", IEEE MICRO, April 1992, pp.10-22.

[16] Kuşdemir L., Bilgen S. "Time Overhead of DQDB MAC Layer Processing Based on a Parallel Architecture", Technical Report, METU EEE 93, To be published.

L. Kusdemir is currently in the Computer Science Dept., Eastern Mediterranean University, G. Magusa, N. Cyprus