

A Single-Chip Storage LSI for Home Networks

Han-Kyu Lim and Deog-Kyoon Jeong, Seoul National University

KyungTae Kim and JunMo Park, XIMETA, Inc.

Han-gyoo Kim, Hongik University

ABSTRACT

Network direct attached storage (NDAS) is a network storage architecture that allows direct attachment of existing ATA/ATAPI devices to Ethernet without a separate server. Unlike other architectures such as NAS, SAN, and USB mass storage, no server computer intervenes between the storage and the client hosts. We describe an NDAS disk controller (NDC) amenable to low-cost single-chip implementation that processes a simplified L3/L4 protocol and converts commands between ATA/ATAPI and Ethernet, while the remaining complex tasks are performed by remote hosts. Unlike NAS architectures that use TCP/IP, NDAS uses a TCP-like lean protocol that lends itself well to high-performance hardware realization. Thanks to the simple NDAS architecture and protocol, an NDC implemented on a single 4 mm × 4 mm chip in 0.18 μm CMOS technology achieves a maximum throughput of 55 Mbytes/s on Gigabit Ethernet, which is comparable to that of a high-performance disk locally attached to a host computer.

INTRODUCTION

Today, digital multimedia devices such as set-top boxes, DVD players, and HDTVs have become very popular; thus, large storage for high-capacity multimedia data has played an important role even in the home. The exploration of a suitable efficient architecture for high-capacity storage targeted at home applications is underway. Although many high-capacity storage devices such as hard disk drives and DVD drives are widely used in a PC environment, mere high capacity is not enough in today's home environments.

Advances in network technology have expanded the convenience and efficiency of the network from the PC environment into the home; the era of the home network is coming. To keep pace with this trend, storage devices also should adapt to the home network. True advantages of the

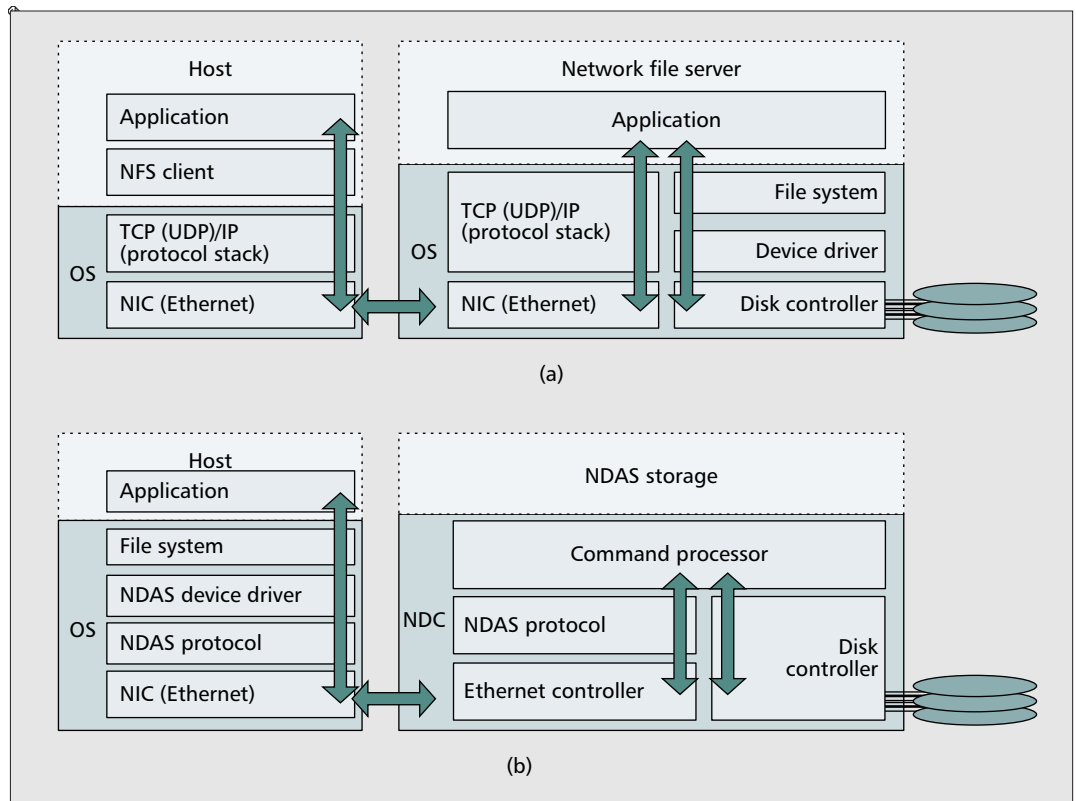
home network are realized when storage devices dedicated solely to one application are offered for sharing among other systems for many different applications through the network. However, it is difficult to share storage efficiently and conveniently because many existing storage devices are targeted for use solely inside application devices.

For example, to play an HDTV program on a computer with no HDTV tuner card, the program is first demodulated in a set-top box and saved in the hard disk drive inside the set-top box. Then it must be copied to the disk in the computer to be played on the computer display. Copying is very inconvenient since it is time consuming and wastes storage capacity with duplicated contents. The network storage addresses this problem by allowing access to its contents symmetrically by participating application devices. The set-top box would directly save the recorded program in the network storage, and it would be directly accessed and played on the computer for viewing. Therefore, the network interface to such storage and application devices needs to be adequately designed for efficient use by multimedia devices at home.

For successful home network storage devices, two requirements, cost and performance, should be satisfied. Consumer electronics devices are sold at very low prices. Furthermore, to cope with rapid growth of available contents, additional devices for expanding capacity must be supplemented frequently. Therefore, low cost is a primary requirement for home network storage. Also, it should offer enough performance to support multimedia applications requiring high bandwidth. In our opinion, the performance of network storage must be limited only by the performance of the storage itself. That is, the network must perform like an I/O, and the storage should be able to yield performance comparable to that of locally connected storage.

Although there have been many network storage devices, they are not well suited to applications for home networks. While in some research [1, 2] network file servers have been

An application on the host makes an access to the disks in the server through Network NFS or CIFS protocols. Since the server is exposed to the user network, storage resource can be shared by many hosts in the user network.



■ **Figure 1.** Storage system architectures: a) traditional network storage system; b) NDAS system.

designed using an embedded processor for the purpose of reducing implementation cost, they suffered from the bottleneck imposed by the software processing of critical protocols. A universal serial bus (USB) mass storage [3], which offers high performance at low cost, is a candidate for home network storage. However, it is very hard for USB mass storage to be shared among multiple devices because of the inherent limitation that only one host is allowed in every USB network. To overcome this limitation, USB mass storage should be attached to a server, like a PC, and the server should play the role of file server. However, in some environments where only multimedia devices are used without any PC, this configuration is impossible. Besides, the server, turned on all the time, consumes too much power, which is unacceptable in home environments.

In order to satisfy the requirements of home network storage, we propose network direct attached storage (NDAS) [4], a network storage architecture. It offers convenience and efficiency like a network file server, but is designed to achieve high performance at low cost like USB mass storage. That is, NDAS has advantages of both file server and USB mass storage. As its name shows, NDAS allows multiple devices to access storage through the network without any server intervention.

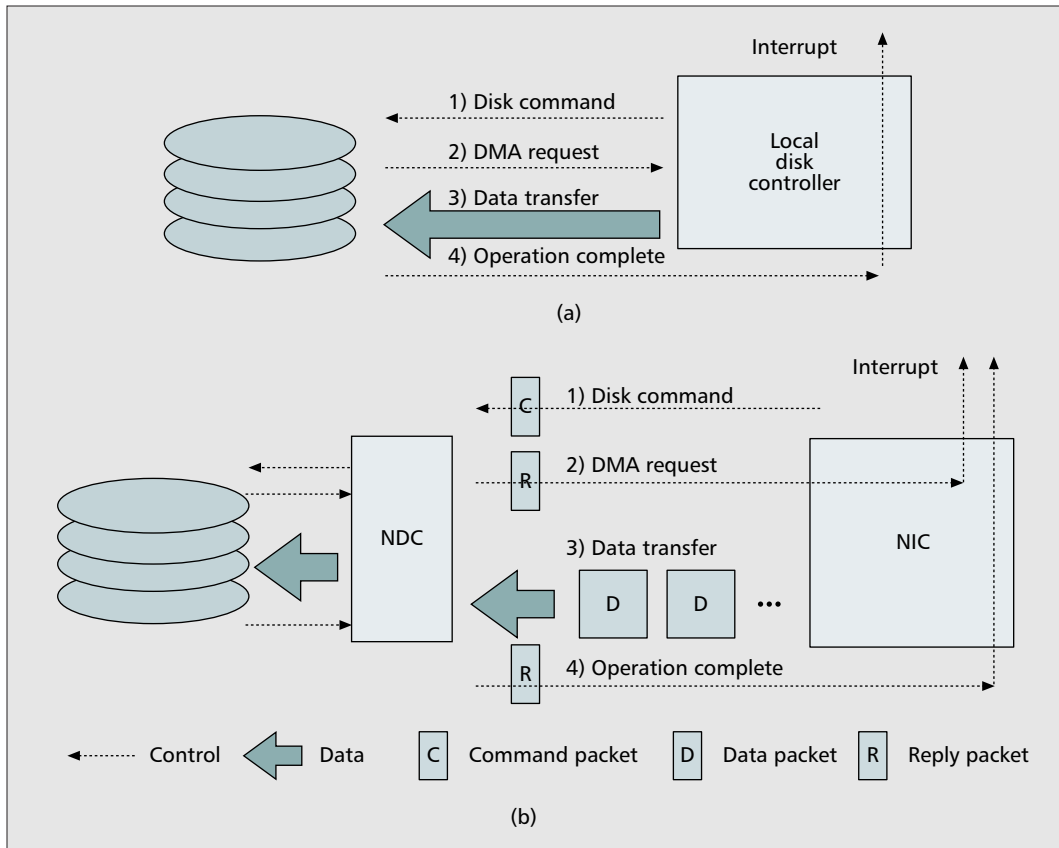
In this article we describe an NDAS disk controller (NDC) enabling commodity ATA/ATAPI devices to be incorporated in the NDAS architecture. Its architecture and implementation are optimized for low cost and high performance. The NDC large-scale integration (LSI) chip inte-

grates all components, except the physical layer chip, in a die area of only 16 mm² in 0.18 μm complementary metal oxide semiconductor (CMOS) technology. Its die area is small enough for the NDC to be integrated with an Ethernet physical layer chip and a hard disk driver controller system on a chip (SoC) for more cost savings. The present NDAS storage is realized with only two chips, the NDC LSI chip and an Ethernet physical layer chip. NDAS has been successfully demonstrated and shows performance of 55 Mbytes/s, comparable to a disk locally and directly attached to a computer.

The organization of this article follows. The NDAS system architecture is described, and performance analysis and optimization of the NDC LSI are presented. The architecture and design of the NDC LSI are explained. The performance evaluation of the NDC LSI follows, and finally, the article is summarized with conclusions.

NDAS SYSTEM ARCHITECTURE

Examining the traditional network storage architecture shown in Fig. 1a helps to understand our NDAS architecture. A server consists of disks for data placement, a processor, and an external interface to the user network. An application on the host accesses the disks in the server through network file system (NFS) [5] or common Internet file system (CIFS) [6] protocols. Since the server is exposed to the user network, storage resources can be shared by many hosts in the user network. This architecture uses a file-oriented delivery method; therefore, the server must run a file system in the operating system (OS) to accommodate the processing of meta-



■ **Figure 2.** Operation analysis: a) local disk operation process; b) first-cut NDC operation process.

Although the file system offload might make NDAS look like a derivative of SAN, NDAS has a distinct advantage over SAN. While the storage in the SAN cannot be directly shared by multiple hosts, in the NDAS it is possible.

data that contains information about file location, size, and so on. In this architecture all requests invoke the file system operation, so the performance of the processor limits that of the server.

Figure 1b shows the NDAS system architecture. In this architecture, the NDAS, which is composed of an NDC and storage, plays the role of server in the traditional network storage architecture, but it does not execute any file system on itself. To be more specific, NDAS directly operates on a block, raw material from which files are formed, like storage area network (SAN) [7] architecture. Because the file system is offloaded from the NDC and moved to remote hosts, the NDC's tasks are simple. Although it is possible to implement the NDC functions in software on a low-performance embedded processor, we chose to build an application-specific integrated circuit (ASIC) for higher performance and lower cost.

Although the file system offload might make NDAS look like a derivative of SAN, NDAS has a distinct advantage over SAN. While the storage in a SAN cannot be directly shared by multiple hosts, in the NDAS it is possible. In the SAN architecture, applications on remote hosts can access the SAN only through file servers directly attached to the SAN. However, in the NDAS architecture, the applications can directly access the NDAS with no server intervention. This is why we named this architecture NDAS.

The operation of NDAS is described as follows. Once an application on the remote host requests a file access through a system call, the

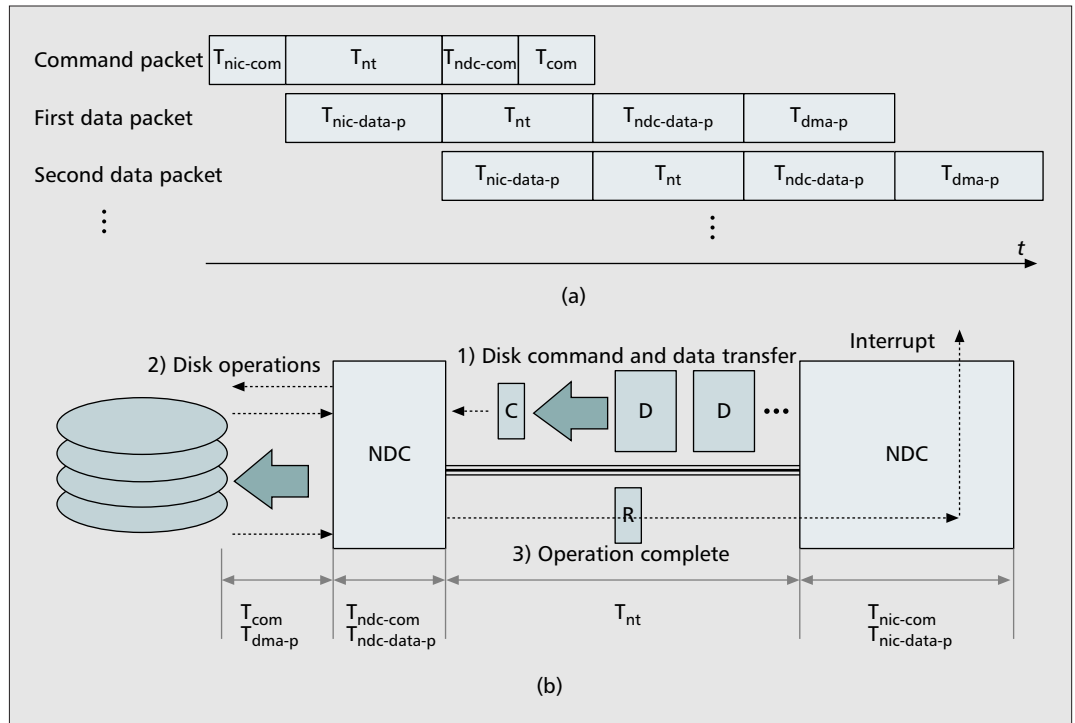
file request is passed to the file system running on the same host. The file system fragments the request into many primitive disk I/O requests and sends its requests to an NDAS device driver. The NDAS device driver delivers the disk I/O requests to the NDC through an NDAS protocol. The NDC operates in the same manner as a disk controller attached to a system except that it has an interface to the network.

PERFORMANCE ANALYSIS AND OPTIMIZATION

We would like to find out what design constraints are required to achieve performance comparable to a local disk controller. In the initial design stage, we attempt to quantitatively analyze and calculate performance of the NDC and compare that with that of the local disk controller. The widely used performance metrics are latency, throughput, and bandwidth. In our work throughput and bandwidth have the same meaning because the NDC has enough buffer to handle the same maximum transferred data as the local disk controller does. As an ATA/ATAPI device should perform each command serially, reducing latency means increasing bandwidth. Therefore, in our article high performance means low latency and high bandwidth.

Figure 2a illustrates the operation process of the local disk controller. The process consists of four steps: the disk controller issues a command via programmable input/output (PIO) writes and waits for a direct memory access (DMA) request

Because the first-cut NDC simply translates signals of the local disk controller to network packets without considering characteristics of network, network delays and processing overheads are imposed at each transaction and they increase the operation time.



■ **Figure 3.** Pipelined operations: a) overlapped packet processing time; b) NDC operation process.

from the disk. Upon receipt of the request, the controller performs the DMA operation. Finally, the disk acknowledges completion of the DMA operation to the controller through an interrupt. The first-cut design of the NDC is started by mapping signals and data of the local disk controller to packets one to one. Its operation process is illustrated in Fig. 2b. A command is issued via a command packet delivered to the NDC through a network interface card (NIC) and a network medium; then the device driver waits for a DMA request. Upon receiving a reply packet containing the DMA request, the device driver starts to transfer data packets. After the DMA operation is done, the NDC sends a reply packet to acknowledge the completion. Because the first-cut NDC simply translates signals of the local disk controller to network packets without considering characteristics of the network, network delays and processing overheads are imposed at each transaction and increase the operation time.

To improve the performance, the effects of network delay and processing overhead should be minimized. As an improvement on the first-cut NDC, as shown in Fig. 3, where each pipelined process executes in parallel, all network delays and processing overheads are hidden by overlapped processes. The device driver starts an operation by requesting the NIC to send a command packet. The requested command packet experiences network delay (T_{nt}) and processing overheads, which consist of the NIC processing time ($T_{nic-com}$), NDC processing time ($T_{ndc-com}$), and command processing time (T_{com}). However, because the device driver transfers data packets without waiting for the DMA request packet, the following data packet reduces the overheads and network delay to $T_{nic-com}$. Thanks to the next data packet, each data

packet except the last one also cuts down its operation time, so the network delay (T_{nt}), NIC processing time ($T_{nic-data-p}$), NDC processing time ($T_{ndc-data-p}$), and DMA processing time (T_{dma-p}) are reduced to only $T_{nic-data-p}$.

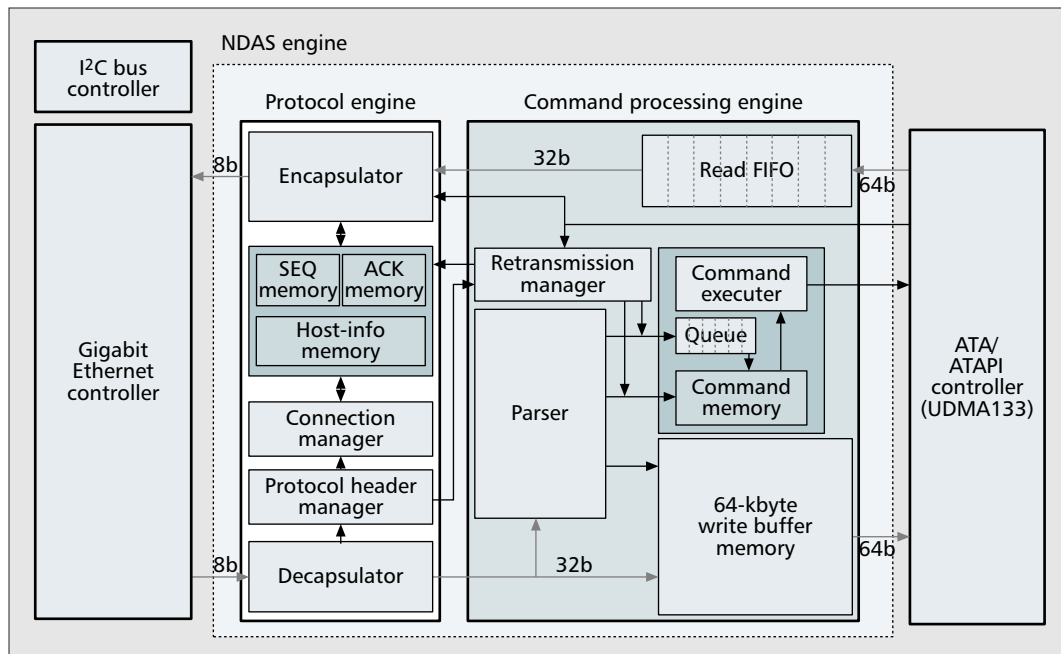
As the NIC and local disk controller have similar operation times, the operation time of the NDC is approximated to

$$T_{ldc} + T_{ndc-data-p} + 2T_{nt}$$

where T_{ldc} is the operation time of the local disk controller and $T_{ndc-data-p}$ is one from the last data packet.

Although $2T_{nt}$ is given by a network environment and cannot be optimized, it is negligible since it is small relative to T_{ldc} . In a typical home network, where at most one or two network switches are used with cable lengths of tens of meters, the network delay is only several microseconds, while the disk operation time is several milliseconds. Therefore, the latency of the NDC, $T_{ndc-data-p}$, is the one of the most important design constraints affecting latency. We expect that if we could limit the latency of the NDC within tens of microseconds, the NDC would eventually offer performance equal to that of the local disk controller.

In summary, the NDC must satisfy two design objectives: pipelined operation and low latency. For the former, the NDC processes data in units of packets instead of blocks using so-called per-packet operation. On receiving a command packet, it starts DMA operation even if any portion of a data packet is not received; then the DMA operation is partially executed whenever a data packet arrives. To support the per-packet operation, each block must concurrently operate because, while processing a previously received data packet, a subsequent data packet may arrive at the NDC. For the implementation of each



■ **Figure 4.** A block diagram of the NDC LSI.

The protocol layer of the NDC should be well designed to process the header at wire speed to pass payload to the upper layer without buffering. The NDC is designed to meet these constraints with a minimum amount of hardware.

block, a hardwired design is preferred over a processor-based design because it offers faster speed with lower cost. To reduce latency, the NDC should minimize buffering of the received packet. In a general layered network architecture, each protocol layer buffers the whole packet until identification and verification of the protocol header are done. This would inevitably increase latency. Therefore, the protocol layer of the NDC should be well designed to process the header at wire speed to pass the payload to the upper layer without buffering. The NDC is designed to meet these constraints with a minimum amount of hardware.

ARCHITECTURE AND DESIGN OF THE NDC

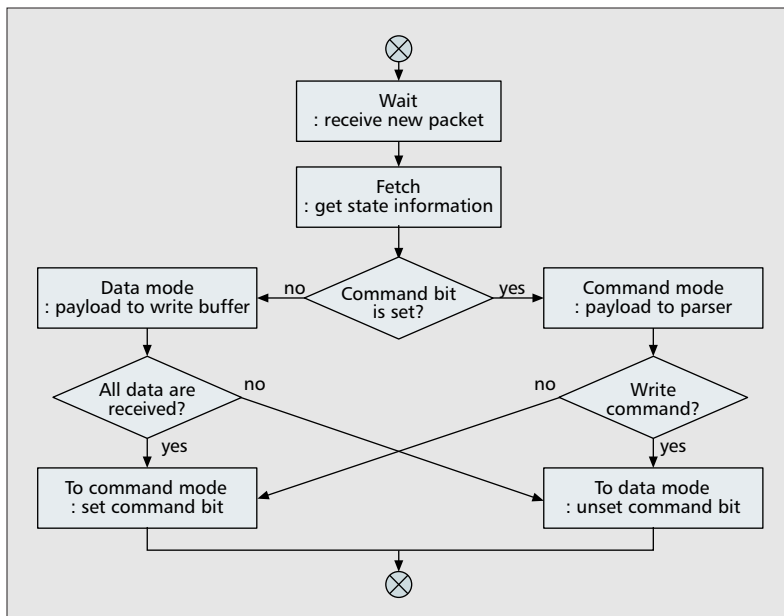
NDC LSI BLOCK DIAGRAM

The architecture of the NDC is shown in Fig. 4. The NDC consists of a Gigabit Ethernet controller, a protocol engine, a command processing engine, a 2-kbyte read first-in first-out (FIFO) buffer, a 64-kbyte write buffer, an ATA/ATAPI controller, and an inter-IC (I²C) bus controller. The protocol engine keeps information on remote hosts, manages connections, and performs encapsulation and decapsulation of the protocol header. In response to a requested command, the command processing engine manages the read FIFO, which stores data read from the disk, and the write buffer, which holds data to be written to the disk. The ATA/ATAPI controller is designed to support up to 133 Mbytes/s bandwidth to cope with 1 Gb/s network bandwidth. The I²C bus controller reads the MAC address, option parameters, and security parameters, which include a login password and an encryption key from a serial electrically erasable programmable read-only memory (EEPROM), and initializes other blocks.

The operation of the NDC starts with a request for connection establishment from a host; then the protocol engine grants the request. After establishment, the host requests a disk I/O command through a command packet. After the header of the received packet is decapsulated in the protocol engine, the payload of the received packet is passed to the command processing engine as a disk command. The command processing engine executes the I/O command on the ATA/ATAPI controller. When the operation is completed, the command processing engine sends a reply packet through the protocol engine.

THE PROTOCOL ENGINE

As shown in Fig. 1b, the NDAS architecture does not use TCP/IP, the most commonly used protocol, on the network, but instead a newly defined NDAS protocol we call LeanTCP. Although using the TCP/IP stack offers many advantages, it is not suitable for the NDC because hardware implementation of the TCP/IP stack is impractical, and software implementation [8] requires an extremely high-performance processor to process packets at the gigabit line rate. Although attempts [9] at implementing TCP/IP in hardware have been successful in server or router applications, their enormous hardware cost is prohibitive for a home network. By streamlining TCP/IP, we defined LeanTCP, which provides reliable services like TCP/IP, but is much lighter. Frame reordering and flow control of TCP are useful only for routers in WANs and so are removed. Since only the medium access control (MAC) address is used to deliver packets in layer 2 switching of LANs, the IP address is not included. Because the network layer is bypassed, complex processes and configurations (e.g., an address resolution protocol, ARP, process and IP address management) can be avoided. On the other hand, it has a drawback of not being routable in WANs. However,



■ Figure 5. Flowchart of the command processing engine.

we believe this limitation is not important in a home network environment.

The protocol engine is a hardware implementation of the LeanTCP stack. It is in charge of performing connection establishment and tear-down, checking validity of incoming packets, retransmitting lost packets, and managing information of remote hosts. The protocol engine consists of a decapsulator, an encapsulator, a connection manager, an information memory block, and a protocol header manager. When the protocol engine receives a packet, the decapsulator strips the protocol header from the received packet. The stripped header is analyzed and identified by the protocol header manager and verified by the connection manager using information saved in the information memory block, which consists of a sequence number memory, an acknowledge number memory, and a host information memory. After verification, the payload of the received packet is passed to the command processing engine with a connection identifier detected by the protocol header manager. When the operation of the command processing engine is completed, the encapsulator builds packets with the data from the command processing engine and transfers them.

The protocol header manager and connection manager were carefully designed to process the received header at wire speed for the purpose of reducing latency. If the protocol engine cannot process the header at wire speed, while waiting for completion the payload of the packet must be buffered to allow more processing time, which will increase latency. They complete operations before receiving the payload from the Ethernet controller; then the payload is directly passed to the command processing engine without buffering in the protocol engine.

THE COMMAND PROCESSING ENGINE

The command processing engine, composed of a command queue, a write buffer, a read FIFO, a

parser, a command executor, and a retransmission manager, executes disk I/O commands requested by remote hosts. Figure 5 shows operation processes of the command processing engine. Before processing the payload, the command processing engine reads state information from the state memory with the address, which is a connection identifier, passed from the protocol engine. The state information indicates the present operation mode: command or data. Initially, the command processing engine is in command mode. In this mode, the command processing engine treats all received payloads as commands, so it sends all payloads to the parser for analysis. After parsing, the commands are queued in the command queue and wait until the command executor is idle. Receipt of a write command changes state from command mode to data mode. In data mode, all subsequent payloads are saved in the write buffer, and after completion the state returns to command mode.

To support multiple write operations concurrently, the command processing engine offers semaphores implemented in the command executor. By serializing write commands, they prevent packet dropping caused by contention of write commands. In addition, they are used by the host software to protect the critical section used to maintain the coherency of caches located in each host.

As previously mentioned, LeanTCP is a connection-oriented protocol providing reliable service. Generally a protocol stack is implemented in the layered architecture where each layer is not aware of the status of other layers; a protocol retransmits lost packets while an application does not know about loss of packets. When implemented in the layered architecture, the protocol engine retransmits lost packets while the command processing engine does nothing. The layered architecture makes implementation easy, but requires a vast amount of memory. If the protocol engine accords a general layered architecture, it should hold all transmitted packets to prepare for packet loss until receiving an acknowledgment packet. Therefore, memory of over tens of megabytes, the maximum transfer size of the requested data times the maximum number of supporting hosts, is required.

To reduce the memory requirement to an acceptable range, we can design the command processing engine to postpone the issue of the next command until the protocol engine receives the acknowledgment packet of the last packet. Thus, the size of required memory becomes tens of kilobytes, the maximum transfer size of the requested data. Although we can reduce the memory requirement this way, waiting time between commands results in excessive performance degradation when multiple hosts simultaneously access the NDC. Especially when a packet is lost, the performance degradation becomes serious because the lost packet halts all commands in the command queue until a retransmission timer, hundreds of milliseconds, expires.

In a new scheme, *command-based retransmission*, the protocol engine does not save any transmitted packet; instead, the command processing engine saves the last executed command.

When packets are lost, the command processing engine again queues the saved command into the command queue, and the protocol engine retransmits the lost packets using data retrieved from the disk. Required memory size is reduced to tens of bytes, the command size times the number of maximum supporting hosts. The moment a packet is lost, command-based retransmission experiences seriously increased operation time by an additional disk operation, which is slower than memory by six orders of magnitude. However, since the possibility of a packet being lost is very low in a LAN environment and the operation time of the disk is less than a retransmission timeout value of hundreds of milliseconds, the performance degradation is negligible, as the experiment results show in the next section.

The NDC chip has been fabricated in a 0.18 μm CMOS technology. It consists of 130K logic gates and 69 kbytes of SRAM with a die area of 16 mm^2 including I/O cells. It operates at 125 MHz with power consumption of 266 mW, which is low enough to allow power over Ethernet (PoE) to supply power through a specialized Ethernet connector. A chip photomicrograph of the NDC LSI is shown in Fig. 6.

PERFORMANCE EVALUATION

In order to benchmark the NDC performance, we built three evaluation environments. One was for benchmarking the performance of the NDC itself without the effect of network delay. For comparison, the performance of three other devices, a local disk controller, a USB 2.0 mass storage controller, and a 1 Gb/s NAS implemented in an embedded processor, were also included. To minimize the network delay effect, the NDC and NAS were directly connected to the host with no network switch in between. The other was to examine how network delay affected NDC performance. The experiment was performed as the number of network switches placed between the host and the NDC was scaled up. A Pentium 4 2.4 GHz desktop computer with 512 Mbytes of main memory and an Intel PRO/1000 MT Desktop Adapter were used in experiments, and the performance was measured by running a benchmark [10] on Windows2000 Professional.

Figure 7 shows the performances of the NDC. Since NDAS was targeted for the home network, all the experiment results were based on using a desktop NIC. The measured performance of the NDC was comparable to that of the local disk controller; the sequential read bandwidth is 55 Mbytes/s, sequential write bandwidth 49 Mbytes/s, random read bandwidth 7 Mbytes/s, random write bandwidth 11 Mbytes/s, and average access time 7 ms. Each performance metric was equal to that of the local disk controller except the sequential write bandwidth, which was 10.9 percent less than that of the local disk controller. However, the degradation was gone when we used a server NIC instead of a desktop NIC. Generally, the network device for a desktop may not be optimized for write performance. Although we considered only the case of the desktop NIC, the NDC overwhelmed other

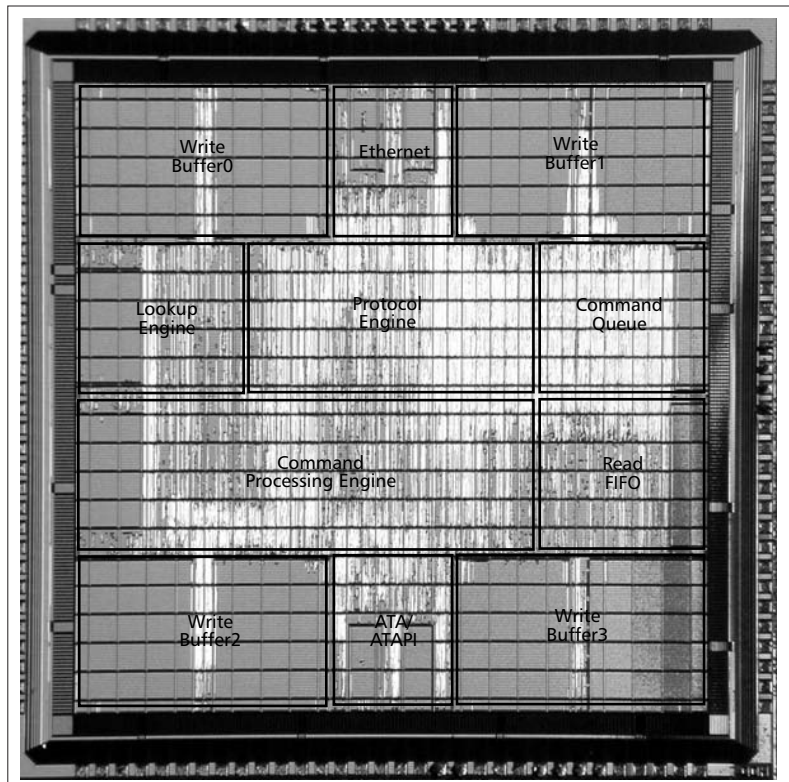


Figure 6. Die photomicrograph.

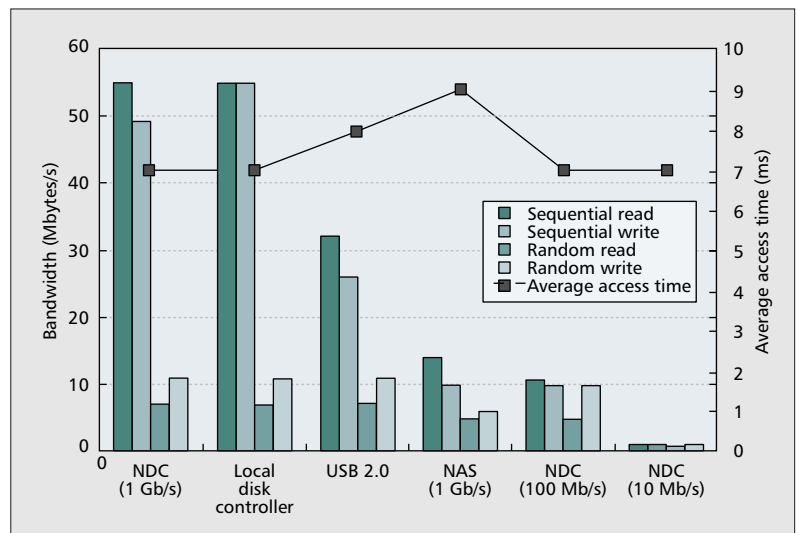


Figure 7. Performance comparison.

devices. One thing to note was that even if NAS had a network interface of 1 Gb/s, it exhibited very low performance, only 25.6 percent that of the local disk controller, indicating that the processor performance was the bottleneck in NAS. Figure 7 also includes the performance of the 100 and 10 Mb/s NDC, showing that the performance bottleneck is on the network, where the NDAS cannot enjoy its maximum benefit. However, we believe that Gigabit Ethernet will be deployed rapidly because of its low cost overhead, just two or three times over 100 Mb/s Ethernet.

In the second experiment, until the number

We can confirm that the NDC will yield performance comparable to the local disk controller in real home network environments, where only a few hops of network switches are expected between a host and a disk.

of network switches reached three, the added network delay does not significantly affect the performance. With three or four intervening network switches, the bandwidth of the sequential read pattern was reduced by only 1.8 percent, while the bandwidths of other patterns and the average access time underwent no degradation. The measured results match the analysis done at the design stage. Therefore, we can confirm that the NDC will yield performance comparable to the local disk controller in real home network environments, where only a few hops of network switches are expected between a host and a disk.

CONCLUSION

In this article a low-cost high-performance NDC LSI for home network applications is presented. This chip incorporates a Gigabit Ethernet controller and an ATA/ATAPI controller supporting a peak bandwidth of 133 Mbytes/s in order to achieve good enough performance to satisfy the bandwidth requirements of multimedia data. It is implemented in full hardware in order to lower power consumption, reduce die area, and achieve high performance. The LSI performs the sustained performance of 55 Mbytes/s, equaling that of a local disk controller.

REFERENCES

- [1] A. Tomita *et al.*, "A Scalable, Cost-Effective, and Flexible Disk System Using High-Performance Embedded-Processors," *Proc. 2000 Int'l. Conf. Parallel Processing*, pp. 317–26.
- [2] M. Anderson *et al.*, "Serverless Network File Systems," *15th SOSF*, Dec. 1995, pp. 109–26.
- [3] USB Implementers Forum, Inc., "Universal Serial Bus Mass Storage Class Bulk-Only Transport," 1999.
- [4] XIMETA, Inc., "What Is NDAS," <http://www.ximeta.com>
- [5] S. Shepler *et al.*, "NFS Version 4 Protocol," RFC 3010, Dec. 2000.
- [6] C. R. Hertel "Implementing CIFS," <http://ubiqx.org>
- [7] T. Clark, *Designing Storage Area Networks*, Addison-Wesley, 1999.

- [8] Y. Hoskote *et al.*, "A TCP Offload Accelerator for 10 Gb/s Ethernet in 90-nm CMOS," *IEEE J. Solid-State Circuits*, Nov. 2003.
- [9] D. V. Schuehler and J. W. Lockwood, "TCP Splitter: a TCP/IP Flow Monitor in Reconfigurable Hardware," *IEEE Micro*, Jan.–Feb. 2003.
- [10] SiSoftware, "Sandra 2004," <http://www.sisoftware.co.uk>

BIOGRAPHIES

HAN-KYU LIM (limbear@isd1.snu.ac.kr) received B.S. and M.S. degrees in electrical engineering and computer science from Seoul National University, Korea, in 2000 and 2002, respectively, where he is currently working toward a Ph.D. degree. His research interests are in the area of storage systems, network systems, and VLSI systems.

KYUNG-TAE KIM (ktkim@ximeta.com) received B.S. and M.S. degrees in computer engineering from Korea Institute of Technology, Daejeon, and Hongik University, Seoul, Korea, in 1991 and 1996, respectively. He is currently working with XiMeta, Inc., where he is developing NDAS communication protocols, device drivers, and NDAS file system software.

JUN-MO PARK (jmpark@ximeta.com) received B.S. and M.S. degrees in computer engineering from Hongik University in 1998 and 2000, respectively. He is currently working with XiMeta, Inc., where he is developing NDAS device drivers and distributed NDAS file system.

DEOG-KYOON JEONG (dkjeong@isd1.snu.ac.kr) is currently a professor at the School of Electrical Engineering, Seoul National University. He was director of the Embedded Systems Research Center at Seoul National University until the end of 2003. His main research interests include high-speed I/O circuits, VLSI systems design, high-performance network systems, and high-frequency RF circuits. He has B.S. and M.S. degrees in electronics engineering from Seoul National University, and a Ph.D. degree in electrical engineering and computer sciences from the University of California at Berkeley.

HAN-GYOO KIM (hkim@cs.hongik.ac.kr) is currently a professor in the Computer Engineering Department, Hongik University. He founded XiMeta, Inc., Irvine, California. His main research interests include computer networks, distributed operating systems, and high-speed network I/O systems. He received B.S. and M.S. degrees in engineering from Seoul National University, in 1981 and 1984, respectively, and a Ph.D. degree in computer sciences from the University of California at Berkeley in 1994.