Hardware-Based TCP Processor for Gigabit Ethernet

Tomohisa Uchida, *Member, IEEE*

***Abstract*—Transmission Control Protocol (TCP) and Ethernet have been widely used in readout systems. These protocols are *de facto* standards and have been implemented on standard operating systems. However, some small devices, e.g., front-end devices and detectors, are not capable of employing these protocols because of hardware size limitations. This paper describes a TCP processor for Gigabit Ethernet with a circuit size suitable for implementing on a single field programmable gate array. The only peripheral device required is a single Ethernet physical layer device. The hardware was implemented and its TCP throughput was measured. The throughputs in both directions simultaneously were at the upper limits of Gigabit Ethernet. A mechanism for slow control over User Datagram Protocol (UDP) is also provided. The processor described here allows adoption of TCP/Ethernet in small devices that have hardware size limitations.**

***Index Terms*—Ethernet, FPGA, TCP/IP.**

I. Introduction

ETHERNET [1] and TCP/IP [2]–[4] have been widely used and have been implemented in a variety of commodity products (Ethernet is a trademark of Xerox Corporation). Their ratio of performance to cost is high, which allows cost-effective network construction. Ethernet and TCP/IP have been widely used in current readout systems. Recently, Gigabit Ethernet has also been used; however, software tuning is required for data transfer at high rates over Gigabit Ethernet [5]. TCP/IP is a *de facto* network protocol standard and is implemented in standard operating systems (OS).

Many backend systems have been designed as distributed systems using TCP/IP and Ethernet (TCP/Ethernet). An experimental modular front-end system was recently designed with TCP/Ethernet [6], [7]. However, TCP/Ethernet has not been widely used in front-end systems. Simple processing of large amounts of data at high rates is required for devices in such systems. Many systems employ a bus specified in hardware. These are monolithic one-box systems, such as VME, PCI, CAMAC, etc., and have low flexibility. This is a disadvantage in designing a distributed system. Use of TCP/Ethernet as an extension bus instead of such a hardware bus would make it possible to design a highly flexible system. However, this has proven difficult as these devices cannot process TCP at high rates. Generally, TCP is processed in software, and the throughput limit of TCP is determined by the processing speed of the software. Powerful hardware is required for processing at high rates. It is therefore difficult to process TCP at high rates using a CPU that can be embedded in a single chip without peripheral devices, such as processor systems built on a Field Programmable Gate Array (FPGA) or an Application-Specific Integrated Circuit (ASIC). A hardware-based TCP processor, called SiTCP, has been developed to overcome this constraint. As the hardware can be built on one chip, it is possible for detectors to employ TCP/Ethernet. These detectors can be built as modules with a common interface.

SiTCP has been adopted by a number of groups. One example is the Super Kamiokande experiment, involving the development of a next-generation readout system to be installed some time in 2008 [6], [7]. The SiTCP in this project is optimized for Fast Ethernet [1]. A front-end electronics integrated neutron image-detector using a gas electron multiplier (GEM) has been developed, which has an Ethernet port through which it transfers data [8]. SiTCP for Gigabit Ethernet will be employed by the HyperSuprime-Cam project [9] for reading data from large Charge-Coupled Devices (CCDs). The readout module is currently under development.

II. SiTCP

SiTCP is a hardware-based TCP processor for devices limited by hardware size, such as front-end devices or detectors. The processor has the following features: small circuit size, high-speed data transfer with TCP, a Hardware Description Language (HDL), simple external interfaces, and remote bus access.

SiTCP can be implemented on a single chip. The standard communication protocols are large and complex, because the terminal must be able to communicate with its partners in various networks and situations. However, as detector systems use well managed and closed networks, many management protocols are not required in such systems, and the protocol set can therefore be reduced. SiTCP adopts the minimum protocol set required by a PC for communication by using standard OS socket functions without special tuning. SiTCP can process only one TCP connection. By reducing the protocol set, the circuit size is sufficiently small to allow its implementation on a single chip. For example, implementation using an FPGA does not require an external memory device. TCP processing is non-blocking, i.e., it is designed with pipeline-based circuits, and receiving and transmitting are processed at the same time. SiTCP processes TCP, IP, and Ethernet protocols. A user circuit should process a protocol over TCP. Users can optimize the protocol for their applications.

Simple external interfaces are employed in these systems. Gigabit-Ethernet media independent interface (GMII) specified by IEEE802.3 [1] is used as the interface between an Ethernet physical layer device (PHY) and a media access controller (MAC). Many PHY products have GMII. The user interface adopts first-in, first-out (FIFO) memory, and allows simple interface design to an external circuit. The processor is designed under the assumption that communications partners will be intelligent devices, such as PCs. In many cases, PCs run

Manuscript received November 15, 2007; revised February 12, 2008.

The author is with the University of Tokyo, Bunkyo-ku, Tokyo, Japan (e-mail: uchida@hep.phys.s.u-tokyo.ac.jp).

Digital Object Identifier 10.1109/TNS.2008.920264

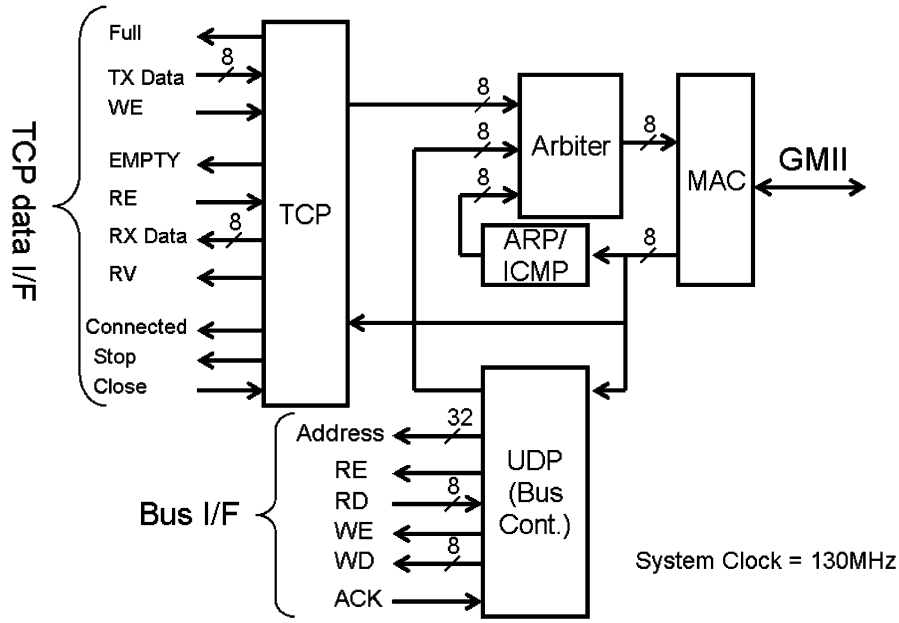

Fig. 1. Block diagram of SiTCP.

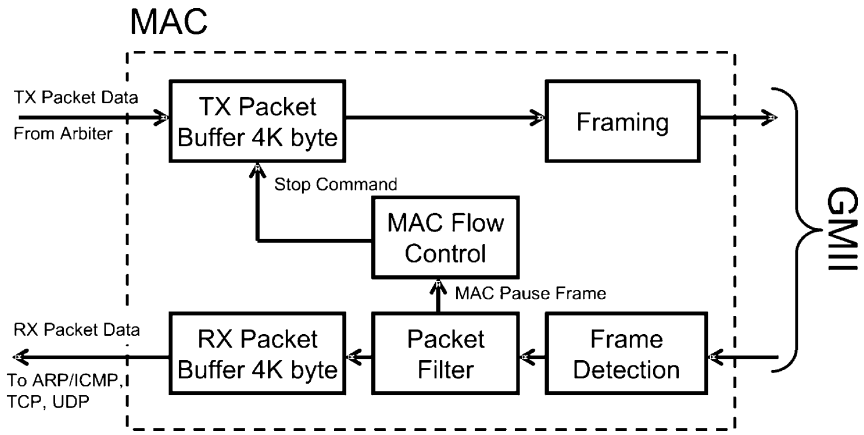

Fig. 2. Block diagram of MAC block.

management and control programs. A remote control method is required in many cases, for example, to configure a module or download firmware of an FPGA using a peripheral circuit designed by users. SiTCP has a method—called Remote Bus Control Protocol (RBCP) in this paper—that uses User Datagram Protocol (UDP) [10] packets and is controlled by a PC connected *via* Ethernet.

Fig. 1 shows a block diagram of SiTCP. The main data-paths are shown. SiTCP consists of MAC, TCP, ARP/ICMP, UDP, and Arbiter blocks. The MAC block processes the Ethernet layer and converts data to/from MII signals. The TCP block is the main block and processes TCP. This block controls the data interface for an external circuit. The ARP/ICMP block processes management packets—Address Resolution Protocol (ARP) [11] and Internet Control Message Protocol (ICMP) [12]. The UDP block processes RBCP and controls the external bus. Finally, the Arbiter block arbitrates and selects among transmission sources.

A. MAC Block

SiTCP supports only full-duplex mode, as half-duplex mode is not suitable for high-speed data transfer. There are no commercially available Gigabit devices that support only half-duplex mode. Jumbo frames, which require large-size buffers, are not supported. These features contribute to reduction of the circuit size. Although some modern FPGAs have a built-in MAC, they were not adopted here to avoid dependency on a specific FPGA family. Different MACs have different control methods and a register map in general. Thus, the circuit is strongly dependent on an FPGA using a built-in MAC.

Fig. 2 shows a block diagram of the MAC block. A transmitted packet is written to the TX packet buffer by the Arbiter block. The packet is incomplete as an Ethernet frame. The packet is read by the Framing circuit retaining the inter-frame gaps [1]. The Preamble and Frame Check Sequence (FCS) [1] are added to the packet, and it is then converted to a complete frame, which is transmitted through the GMII.

A received frame is transferred to the Frame Detection circuit through the GMII. The circuit checks for inter-frame gaps, searches for the start of the frame with the preamble, and extracts the Ethernet frame from the bit sequences. The packet filter circuit checks the FCS, then filters by destination addresses. Filtered packets that have broadcast or SiTCP

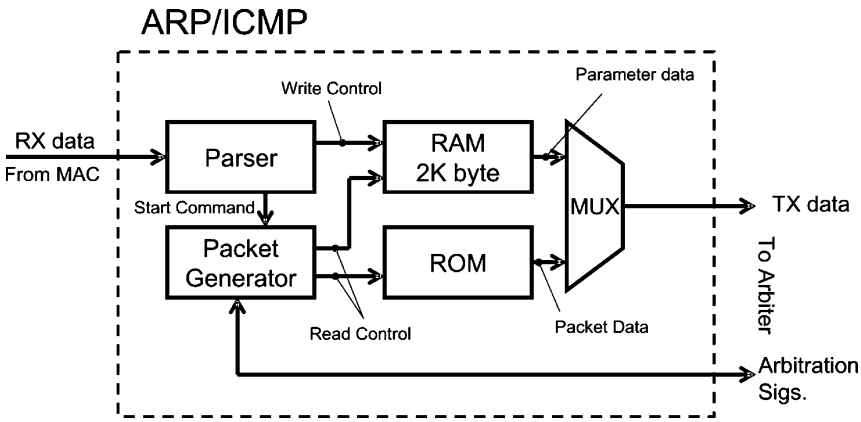

Fig. 3. Block diagram of ARP/ICMP block.

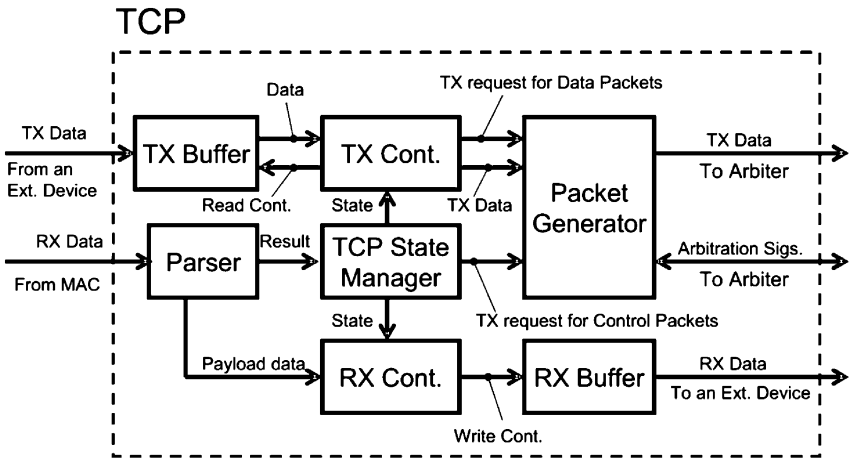

Fig. 4. Block diagram of TCP block.

address are written to the receive buffer except MAC pause frames [1]. A tag that indicates its protocol—TCP, UDP, ARP, or ICMP—is then added to the start of a frame. A MAC pause packet is transferred to the MAC control circuit that controls reading of the Tx Buffer circuit as specified in IEEE802.3 [1]. The Rx buffer starts to read automatically after writing has finished. There is no flow control in this circuit. The TCP block is a non-blocking circuit. In other blocks, a frame is discarded by each block when it cannot be processed.

B. ARP/ICMP Block

This block processes ARP request and ICMP echo-request packets. Other management packets are discarded. This block generates and transmits reply packets for these. ARP requests are essential because they are required by the PC—generally IP sending and forwarding devices—to resolve the IP address. ICMP echo replay is not essential. However, these packets are useful to check the path from the PC to the SiTCP. This is known as a PING command.

Fig. 3 shows a block diagram of the ARP/ICMP block. The Parser circuit processes only packets that have a tag of ARP or ICMP, and other packets are discarded. ARP and ICMP packets are analyzed. If the packet is an ARP request or ICMP echo request, it is written to RAM. The Parser requests generation of a reply packet, and the Packet generator requests transmission to the Arbiter block. After receiving the acknowledge signal for the request, the circuit begins to generate the packet from data in RAM and the fixed values of protocol headers stored in ROM. The packet is then transferred to the Arbiter block.

C. TCP Block

This is the main block of SiTCP and processes TCP acting as a server or client. Fig. 4 shows a block diagram of the TCP block. Table I shows user interface signals. The parser circuit processes only packets that have a TCP tag and a destination port number set beforehand. These packets are analyzed. The parameters of TCP, e.g., Sequential Number (SN#), Acknowledge Number (ACK#), Window Size, Flags, and Source Port Number [4], are extracted and transferred to the TCP state manager circuit that manages the TCP connection and requests transmission of management packets that have SYN, FIN, and RST flags [4]. Circuits in this block except the Parser are controlled by the manager with a TCP state. A reduced TCP state machine is adopted. Simultaneous opening and closing is not supported. However, simultaneous opening does not occur in the client-server model, but simultaneous closing can be avoided by system design, e.g., a predefined terminal always finishes before another.

Fig. 5 shows a state diagram of the TCP state manager drawn from the hardware view, and is different from a standard state

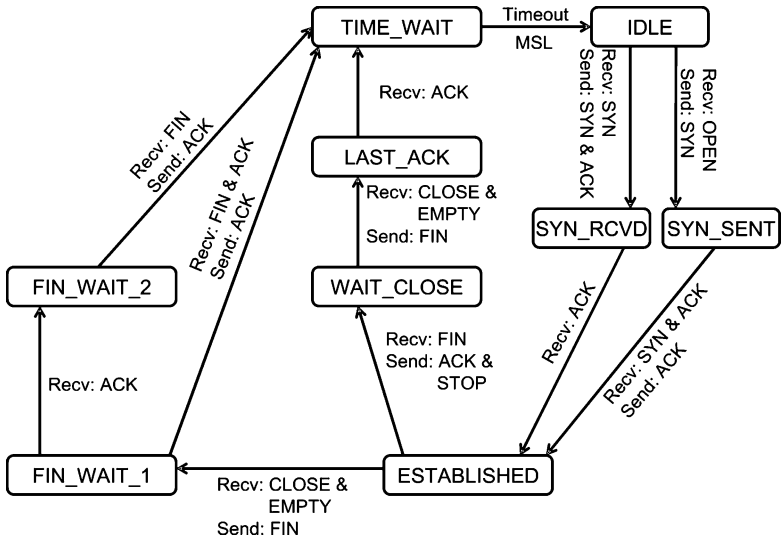

Fig. 5. State diagram of the TCP manager.

TABLE I
USER INTERFACE SIGNALS

Category	Signals	Description
Access Control	CNCTD	TCP connected
	OPEN	Request to open a connection
	STOP	Request to stop writing
	CLOSE	Request to close a connection
TX buffer control	FULL	Full flag
	WE	Write enable
	TXDATA	Write data, 8-bit width
RX buffer control	EMPTY	Empty flag
	RE	Read enable
	RV	Read data valid
	RXDATA	Read data, 8-bit width

diagram of TCP found in textbooks (for example [4]). Transition criteria are shown as "Recv" These transitions occur by received packets or signals. "Recv: ACK" means that the manager received the TCP packet with the ACK flag. Actions are represented as "Send". OPEN, CLOSE, and STOP are signals of the user interface. EMPTY is an empty flag signal of the transmission buffer. Only normal transitions are represented and abnormal transitions are omitted. If abnormal conditions are detected, the manager sends an RST packet and the state goes to TIME_WAIT from all states. This is followed by a wait for the Maximum Segment Lifetime (MSL) [4] and it then goes to IDLE, which is the Initial state. Timeout criteria are summarized in Table II.

The TX controller will be active in ESTABLISHED and WAIT_CLOSE. There is a transmission buffer in this circuit, and its size is selectable as 8, 16, 32, or 64 Kbytes. There is a FIFO memory buffer for an external circuit. The external circuit can write data to the buffer when CNCTD is active, STOP is inactive, and FULL is inactive. If FULL is asserted, the external circuit should stop writing within 8 clocks. SiTCP uses a buffering algorithm; when data length is over the Maximum Segment Size (MSS) [4] or there is no writing over a period of 4 ms, the controller requests transmission of data to the Packet Generator circuit. Use of this function is selectable. The data length of transmission is calculated by

TABLE II
CONFIGURATION PARAMETERS OF TCP

Parameters	Description
WAIT_PORT_NUMBER	Waiting port number
TX_BUF_SIZE	Transmission buffer size of TCP Selectable 8, 16, 32, 64 Kbytes
RX_BUF_SIZE	Receiving buffer size of TCP Selectable 0, 8, 16, 32, 64Kbytes
TX_BUF_ALGRSM	Transmission buffering algorithm, ON/OFF
KEEP_ALIVE	Send keep-alive packets, ON/OFF
KEEP_INTRVL_EMPTY	Interval time to sent keep-alive packets when the Tx buffer is empty
KEEP_INTRVL_FILL	Interval time to sent keep-alive packets when the Tx buffer is empty
TOUT_OPEN	Timeout for opening a connection
TOUT_ESTBLSH	Timeout for a connection established
TOUT_RETRNS	Timeout for a retransmission
FAST_RTRNS	Fast retransmission algorithm, ON/OFF
MSS	Initial Maximum-Segment-Size. The MSS used in communication is determined by exchanging TCP options. (see [4])
MSL	Maximum Segment Lifetime. Equivalent to the minimum interval time between connections

$\text{Length} = \text{MIN}\,(\text{LastSN\#} - \text{Last ACK\#}, \text{MSS})$; here Last SN# is the last sent SN#, Last ACK# is the last received ACK#, and MSS is the max segment size. Data corresponding to the SN# are transmitted. If the difference between the sent SN# and the last received ACK# is equal to the last received window size, the transmission is stopped and the system waits to receive a new ACK#. If a timeout occurs upon receiving no ACK, the retransmission process is started. SiTCP also supports fast-retransmission. If three duplicate ACK# are received, the process is also started. The retransmission process sets SN# to the last ACK#. The circuit transmits one packet, and then waits for a new ACK packet. If the ACK packet is received, the transmission process is restarted.

The RX controller will be active in ESTABLISHED, FIN_WAIT_1, and FIN_WAIT_2. There is a buffer in this circuit, and its size is selectable as 0, 8, 16, 32, or 64 Kbytes. This

TABLE III
BUS INTERFACE SIGNALS

Signals	Description
ADDRESS	32-bit width address
RD	8-bit width read data
WD	8-bit width write data
RE	Read enable
WE	Write enable
ACK	Access acknowledge from target bus devices. In read accesses, valid read data.

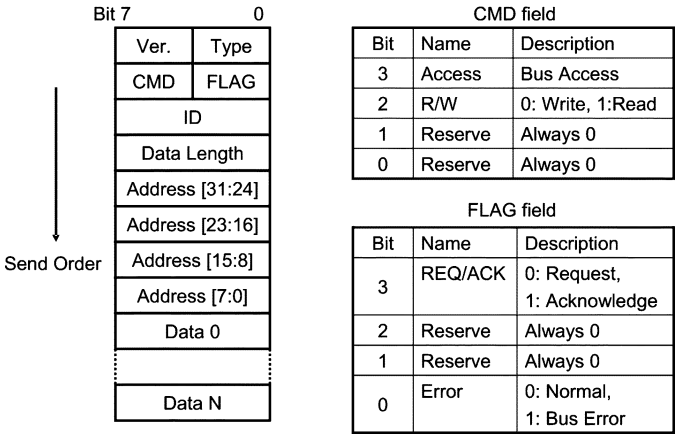

Fig. 6. Packet format of RBCP.

is a FIFO memory buffer for an external circuit. The external circuit can read data from the buffer when CNCTD is active and EMPTY is inactive. The external circuit reads data with RE, and then read data, RD, and read valid, RV, are asserted. The length that can be written in the buffer is the value of the sending window. When a capacity of 0 bytes is selected, the window size is fixed to 64 Kbytes and RE is not used. RD and RV are asserted automatically.

The packet generator generates TCP packets on requests from the TCP state manager, the TX controller, and the RX controller.

Configurable parameters of TCP are summarized in Table II. Users can design the parameters to be fixed or variable, because these are defined in signal lines. The variable design can be realized using an additional circuit for parameter registers and the bus control functions described in the next section.

D. UDP Block

UDP is used to control the bus from a remote terminal. RBCP encapsulated in UDP packets was originally defined for this purpose. RBCP accesses the bus with a request-acknowledgement method. A PC as an initiator sends a request packet (RREQ). The SiTCP accesses the bus and in turn replays an acknowledgement packet (RACK) to the PC. As UDP is not a reliable protocol, packets are sometimes lost. The initiator program manages this loss and should have a timeout function to detect such losses. The bus signals are summarized in Table III. All signals are synchronized to the system clock.

Fig. 6 shows the RBCP packet format. Ver. and Type fields are reserved for future use. The value is 0xFF in the current version. The CMD field indicates command/action for target/initiator in RREQ/RACK, and the FLAG field indicates the results of an access. A bus-error occurs on bus timer expiry due to a no response from bus-target devices. The ID field is used for identifying the RACK corresponding to a given RREQ. The RACK has the same ID number as the RREQ. The LENGTH field in RREQ indicates the data length to access. In the RACK, the field is the data length accessed normally. The ADDRESS field indicates the start bus-address. DATA fields are read data or write data. RREQs for read-accesses do not have this field. DATA0 are data corresponding to the start address. The fields in RACKs are the normal data accessed.

Fig. 7 shows a block diagram of the UDP block. The parser filters UDP packets according to their tags, and transfers a packet with a port number set beforehand in the UDP header to the RBCP parser. This block processes RBCP packets one by one. When the Data Buffer is occupied by a packet, receiving packets are discarded. The RBCP parser analyzes the packet and extracts parameters for bus accesses. The circuit requests an access to the Direct Memory Access Controller (DMAC) with the parameters. The DMAC transfers data from/to a bus-target to/from a data buffer. After the end of accesses, the DMAC requests transmission from the Packet Generator. The generator generates a RACK and transfers it to the Arbiter block.

E. Arbiter Block

This block arbitrates the transmitters. Arbitration signals are of the handshake type. The priority of arbitration is from the highest, ARP/ICMP, UDP, TCP block. Checksums in the protocol headers of IP, TCP, and UDP are calculated in this block.

III. IMPLEMENTATION

To evaluate SiTCP, it was implemented on an FPGA using a Xilinx ML403 as an FPGA test board [13]. Fig. 8 shows the block diagram of the test circuit. The main parts were PHY (Marvell Alasaka 88E1111), FPGA (Xilinx XC4VFX12-10C), and the RS232C transceiver. The test circuit consisted of 6 blocks. SiTCP, Test data generator, Test data checker, and Rate monitor blocks were implemented in the FPGA, and SiTCP worked in client or server mode. Its TCP buffer sizes were 32 Kbytes for TX and 0 bytes for RX. The generator block generated incremented sequence numbers with a width of 32 bits. The checker block checked receiving data. As this block was non-blocking, the TCP buffer for receiving was set to zero bytes. When an error was found, the block turned on an LED. The Rate Monitor sampled the last received ACK# and the last sent ACK# at intervals of 200 ms, and the sampled numbers were sent to a PC through RS232C for performance measurements.

Table IV shows the implementation results of the FPGA generated Xilinx ISE8.2i. The number of logic-resources used—Number of Slices (Used) in this figure, where a Slice consists of two 4-input look-up tables and two flip-flops, and is nearly equivalent to two Logic Elements (LE) in ALTERA devices [14]—was 3111 Slices (28%). This size is sufficient to implement user circuits with SiTCP on one FPGA.

IV. MEASUREMENT AND RESULTS

Transfer performance was confirmed with the test circuit. Fig. 9 shows the experimental setup. The SiTCP and PC2, which

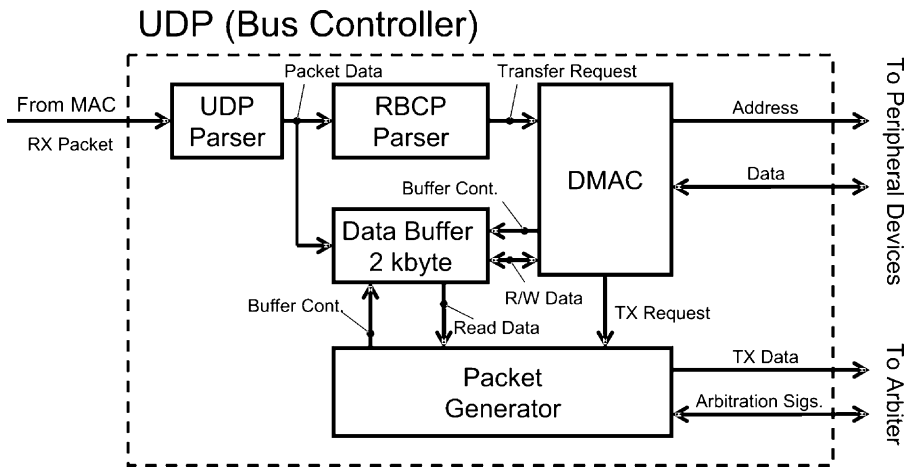

Fig. 7. Block diagram of UDP block.

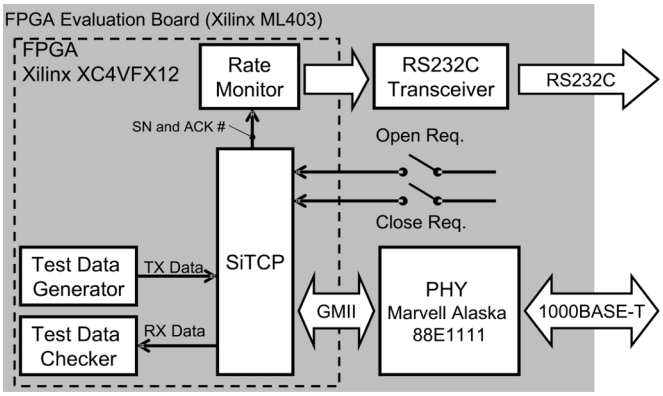

Fig. 8. Block diagram of test circuit.

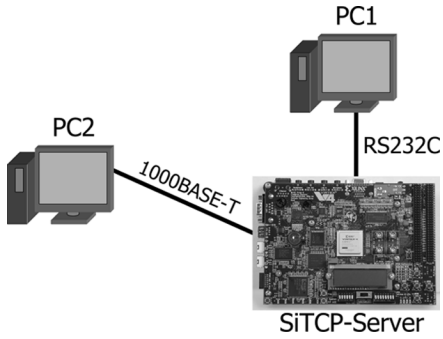

Fig. 9. Measurement setup between PCs.

TABLE IV
FPGA IMPLEMENTATION RESULTS

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	3,111	10,944	28%
DCM autocalibration logic	14	3,111	1%
Number of 4 input LUTs	2,556	10,944	23%
DCM autocalibration logic	8	2,556	1%
Logic Distribution			
Number of occupied Slices	2,285	5,472	41%
Number of Slices containing only related logic	2,285	2,285	100%
Number of Slices containing unrelated logic	0	2,285	0%
Total Number 4 input LUTs	3,015	10,944	27%
Number used as logic	2,556		
Number used as a route-thru	320		
Number used as 16x1 RAMs	8		
Number used as Shift registers	131		
Number of bonded IOBs	39	320	12%
Number of BUFG/BUFGCTRLs	6	32	18%
Number used as BUFGs	6		
Number used as BUFGCTRLs	0		
Number of FIFO16/RAMB16s	22	36	61%
Number used as FIFO16s	0		
Number used as RAMB16s	22		
Number of DCM_ADVs	2	4	50%

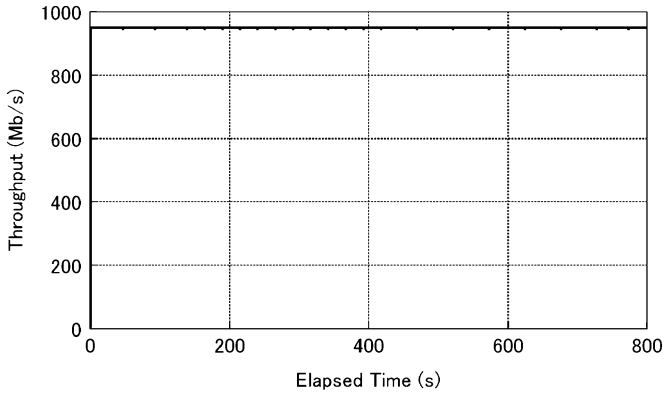

Fig. 10. TCP throughput from SiTCP to the PC as a function of elapsed time.

were connected directly with a crossover cable, acted as a TCP server and a TCP client, respectively. Forwarding devices were not used to avoid any influence on performance. PC2 was a DELL PowerEdge SC1430 Dual Core Xeon 5120 (4 MB L2 Cache, 1.86 GHz, 1066 MHz FSB) running Scientific Linux CERN SLC release 4.5. PC1 receives ACK# from the Rate Monitor circuit. A simple program, which only reads the socket and checks the receiving data, was used with no parameter tuning. The MSS in this measurement is 1460 bytes, or 1518 bytes length in an Ethernet frame.

Fig. 10 shows the throughput as a function of elapsed time. The vertical axis shows throughput of user data and the horizontal axis shows elapsed time from establishment of the connection. The throughput was calculated from ACK# received from the rate-monitor every 200 ms. Fig. 11 shows the distribution of throughputs. SiTCP transfers data to PC2 at 949 Mbps.

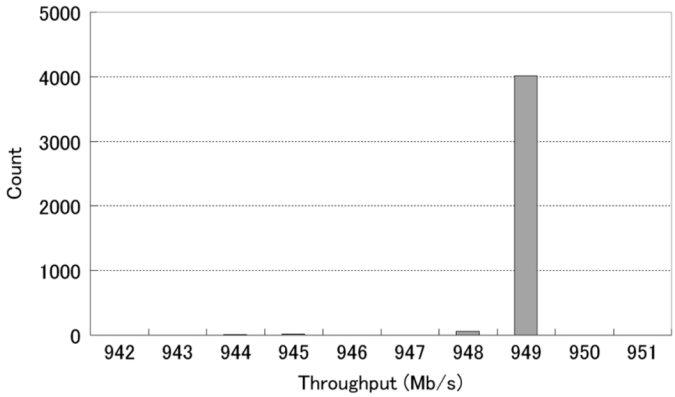

Fig. 11. TCP throughput distribution from SiTCP to the PC as average throughput every 200 ms.

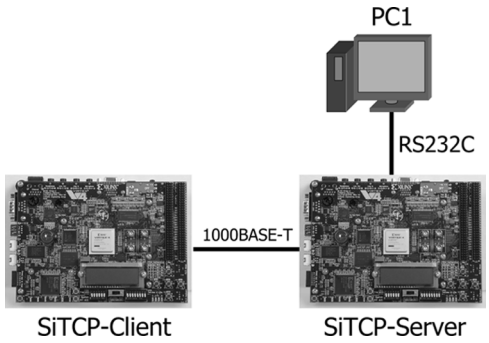

Fig. 12. Measurement setup between SiTCPs.

Throughput was measured in both directions using this setup at the same time, and the results were about 60% of 949 Mbps. This was thought to be due to the technique used in coding. However, studying the coding technique was not the aim of this study.

To measure throughput in both directions at the same time, two SiTCPs were used. The setup is shown in Fig. 12. Constant throughput of 949 Mbps was measured in both detections simultaneously, corresponding to the limit of TCP using Gigabit Ethernet calculated with overheads, such as protocol headers.

V. Conclusion

A hardware-based TCP processor (SiTCP) was developed, which requires only one external device—an Ethernet PHY device. No other devices are required. Its circuit size, about 3000 slices, is small enough to allow implementation together with user circuits on a single FPGA. Throughput of user data is 949 Mbps in both directions, which is the limit of TCP using Gigabit Ethernet.

SiTCP is enabling small devices to adopt TCP/Ethernet. SiTCP will contribute to modular and distributed system design.

References

[1] *Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, IEEE Standard 802.3, 2003.

[2] "Transmission control protocol," Internet Engineering Task Force RFC793, Sep. 1981.

[3] "Internet protocol DARPA Internet program protocol specification," Internet Engineering Task Force RFC791, Sep. 1981.

[4] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Reading, MA: Addison-Wesley, 1994.

[5] M. Kozlovszky *et al.*, "Analysis of STCP and TCP based communication in high speed clusters," in *Proc. NIMA*, 2006, vol. 559, pp. 85–89.

[6] H. Nishino *et al.*, "Development of new data acquisition electronics for the large water Cherenkov detector," in *Proc. IEEE NSS*, 2006, pp. 124–127.

[7] H. Nishino *et al.*, "The new front-end electronics for the super-Kamiokande experiment," in *Proc. IEEE NSS*, 2007, pp. 127–132.

[8] S. Uno *et al.*, "Development of Neutron Gaseous detector with GEM," in *Proc. IEEE NSS*, 2007, pp. 4623–4626.

[9] S. Miyazaki *et al.*, "HyperSuprime: Project overview," in *Proc. SPIE*, 2006, pp. 9–16.

[10] "User datagram protocol," Internet Engineering Task Force RFC768, Aug. 1980.

[11] "An ethernet address resolution protocol or converting network protocol addresses to 48.bit ethernet address for transmission on ethernet hardware," Internet Engineering Task Force RFC826, Nov. 1982.

[12] Internet control message protocol DARPA Internet program protocol specification Internet Engineering Task Force RFC792, Sep. 1981.

[13] *ML401/ML402/ML403 Evaluation Platform User Guide*, XILINX Inc., UG080(v2.2), Nov. 2005.

[14] *Stratix Device Handbook: Stratix Architecture*, XILINX Inc., UG080(v2.2), Jul. 2005.