Protocol Stack-based Telecom Emulator

Takahiro Murooka and Toshiaki Miyazaki

NTT Network Innovation Laboratories
Yokosuka, Kanagawa, 239-0847, Japan
Tel: +81-468-59-3572 Fax: +81-468-55-1604
e-mail: {murooka, miyazaki}@exa.onlab.ntt.co.jp

Abstract

The paper describes the concept and implementation of a telecom emulator that features both reconfigurability and high-speed processing. The emulator can be easily transmuted into any telecom system as a real node. It has two innovative system design concepts. The first is to divide the specification into simplified processes based on the open system interconnection (OSI) reference model. The second is the use of a sophisticated hardmacro and its software-callable driver. We implemented a prototype system called ATTRACTOR and applied it to some telecom applications. The applications were able to be implemented in a short design time and were operated in real computer network environments.

1. Introduction

Today's telecommunication (telecom) nodes require continuous enhancement to support various kinds of new multimedia services, and effective enhancements in a timely manner is the key to surviving the time-to-market race. However, the development of new telecom nodes requires a long design period that involves mainly verification time. This is because very long data streams are used in the verification process to ensure functionality and reliability, and verification is done with software-based logic simulators in general. If we had an emulation system that could be operated on a real network, we could effectively shorten the verification time. To create such a design environment, we examined the architecture of a telecom emulation system.

Field programmable gate arrays (FPGAs) based emulators [1, 2] give us some hints as to how flexibility can be improved. They are widely used in designing application specific ICs (ASICs). Although their operation clock rate is around 1-MHz, which is much faster than software-based simulators, they as yet cannot be applied to real telecom data processing, which generally requires clock rates of at least 20-MHz.

A telecom emulator has recently been developed [3] that uses a performance-oriented FPGA called PROTEUS [4] and has a novel system architecture that can handle telecom data at a real clock rate. The telecom application is implemented entirely on the FPGAs without the need for any software. This is because the emulator targets only the lower protocol layer's telecom data, which have simple formats and structures. Hence, it does not have to manipulate complexly structured telecom data, such as the internet protocol (IP) and internet transmission control protocol (TCP), which are often manipulated with software running on a micro-processors (MPUs) or digital signal processors (DSPs).

Our goal is to construct a telecom emulator that is both flexible and fast and has a easy-to-use application design environment. The key ideas to achieving these goals are a protocol-structure-oriented system architecture and a formalized hardware/software interface.

The rest of this paper is organized as follows. Section 2 discusses our concept in detail. Section 3 describes implemented system and design environment. Section 4 gives test results for telecom applications. Section 5 concludes this paper.

2. System concept

2.1. System model

Telecom applications are often designed based on the open system interconnection (OSI) reference model [5]. The model consists of seven layers and is depicted in Fig. 1(A).

The telecom data is processed layer by layer, starting with layer one, and each process works independently. In general, processes in layer one and two

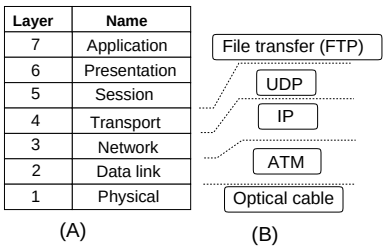

Figure. 1. Open system interconnection (OSI) reference model (A) and an implemented application structure (B).

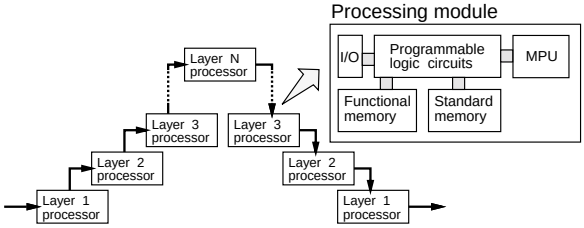

Figure. 2. The basic structure of a high-performance telecom emulator.

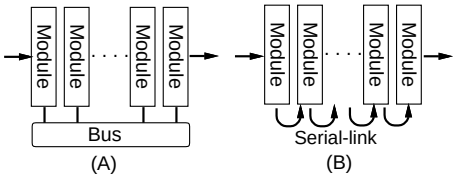

Figure. 3. Inter-module communications: (A) bus-based method, (B) cable-connection-based method.

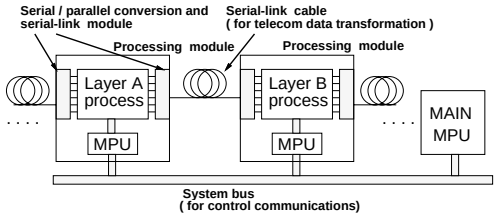

Figure. 4. The final model of our telecom emulation system.

are implemented as hardware, while processes in the upper layers are implemented as software. This is because layer one and two have simple data processing algorithms that require a rigorous timing specification. On the other hand, upper layers have to adopt various kinds of internetworking protocols. Hence, their processes were implemented as software in the conventional nodes.

Figure 1(B) shows the corresponding application for the file transfer program (FTP) on asynchronous transfer mode (ATM) networks. A physical signal is terminated in layer one, and the payload data is transferred to the ATM layer. The ATM layer terminates data as ATM adaptation layer (AAL layer) data and transfers it to the IP layer. It evaluates the network address of the data and transfers the data to the user diagram protocol layer (UDP layer), where data is generated by the FTP of the source computer. This application reads the data frame from the UDP layer. The whole process is very complex, but each individual process is simple. From this independently layered process structure, we came up with a novel concept for creating a high-performance telecom emulator.

Figure 2 illustrates the basic concept behind our telecom emulator. The processing modules consist of I/O port, programmable logic circuits, MPU and memories and handle a protocol layer's data with implemented hardware and software. Programmable logic circuits are used for the hardware part of the implemented process. The MPU executes tight hardware-related routines and house-keeping tasks. Each memory is used as a table with an implemented hardware function. The processing modules have a homogeneous architecture.

In our model, a performance bottleneck may occur in the inter-module communication. To remedy this situation, we adopted two kinds of inter-module communication mechanisms, as shown in Fig. 3.

Figure 3 (A) shows the bus-based method, which is good for highly flexible connections and joint memory access communication. Unfortunately, several modules must share the bus, which limits overall performance. Hence, this method can not ensure adequate data transformation performance all the time.

Figure 3 (B) shows the cable-connection-based method, which is based on point-to-point serial-links. While this method does not have any performance problems, the connections are not very flexible. However, dynamic flexibility is not required in our emulator because the implemented protocol structure is not changed while it is running. Therefore, we adopted method (B) for inter-module data connection of our telecom emulator.

Figure 4 illustrates the final model of our telecom emulator. Two kinds of communication interfaces can be used to connect any two modules, even if their layer processes are different. One is a serial-link that connects two modules directly. The other is a system bus. The system bus is used to send a control signal from/to modules and to communicate with the main

MPU, which controls the entire system. In other words, each heterogeneous layer process can be encapsulated by a common communication interface that allows us to access any process in the same way without worrying about the differences between the processes.

This module arrangement allows us to freely alter the telecom system's functionalities by choosing the layer process combination that best fits a given application without sacrificing system performance.

2.2. Design environment

Target telecom applications have to manipulate higher OSI protocol layers, which are, in general, implemented using only software. We used the processing module to accelerate these layers. Each layer process is divided into a hardware part and a software part that are implemented in the processing module.

To implement them effectively, we needed a hardware/software interface routine that could provide a simple hardware control mechanism. These software routines are called device drivers. Device driver development requires knowledge of both hardware and software, and is, in general, very difficult. Our design environment concept is to provide a simple, effective method of designing the device drivers.

The core idea of this concept is to make a set of system elements that includes coarse-grain hardmacros for implementation, a behavioral description for simulation and a hardmacro device driver description for the software. This set is called the "virtual part".

The coarse-grain hardmacro is a large-scale high-functionality module that can manipulate processes in one layer and can be loaded into the programmable logic circuits of the layer's processing module (Fig. 2). The hardmacros are designed so that they satisfy the performance requirements of the implemented functions. Hence, the designer can be freed from a lot of performance-oriented design work.

The device drivers provide formal access for all hardmacros. The programmer can design a piece of software without worrying about access differences between hardmacros. This concept gives us an effective environment with which to realize a shorter design turn around time.

3. Implementation

3.1. System hardware

We developed a prototype telecom emulator and its dedicated design environment around the concept just outlined. The prototype system is called ATTRACTOR and operates on an ATM network with a 156-Mbps optical connection (OC-3). An overview of ATTRACTOR is illustrated in Fig. 5.

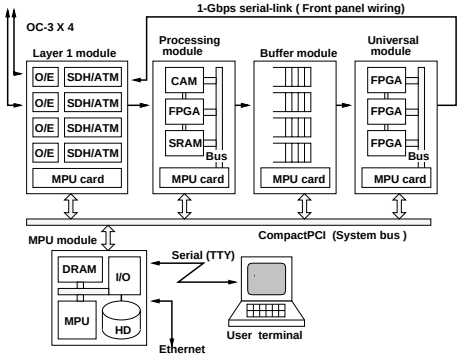

Figure. 5. An overview of ATTRACTOR.

The system consists of heterogeneous modules, including a layer one termination module, protocol processing module, data buffering module, universal reconfigurable logic module, and a main MPU module. Each module, except for the MPU, has an MPU-card as the on-board controller for executing house-keeping tasks and hardware-related routines of the application software. All logic circuits, except for functions realized by dedicated parts, are implemented as FPGAs. We used a custom-made FPGA called PROTEUS-Lite [8] for the universal module and commercial FPGAs for the other modules.

Ideally, we should use the same kind of processing modules as in the system model shown in Fig. 2. However, this model is difficult to implement. So we designed some dedicated modules, which still inherit the basic structure of the processing module, i.e, a MPU card and inter-module communication mechanism.

The layer one module is dedicated to OC-3 termination. The processing module consists of content addressable memories (CAMs), static RAMs (SRAMs) and FPGAs for protocol processing. The buffer module consists of highly functional FIFOs that are capable of ATM cell storage and AAL frame termination. The universal module consists of many FPGAs and can be used as a data analyzer. The main MPU module utilizes a high-performance MPU, a large-scale memory, a hard disk, and several kinds of I/Os. It executes the user interface and the house-keeping tasks for the whole system, as well as the complex parts of the user's application software.

There are two different ways to interchange data/informations among modules: the CompactPCI [6] bus and the 1-Gbps wide-band serial-link connection. The telecom data is transfered between any two

modules using serial-links, which are in the form of the coax-cables connections on the front panel of each module. A user can freely combine any two modules by using the serial-links. This serial-link mechanism increases the flexibility of each module connection. The CompactPCI bus corresponds to the system bus in Fig. 5 and is mainly used for interconnections between the main MPU and MPU cards on other modules. A photograph of the prototype system is shown in Fig. 6.

3.2. Operating environments

The MPU cards and the main MPU execute their own operating environment on a real-time operating system called VxWorks [7]. The system control software of all applications and ATTRACTOR's own operating environment are executed using VxWorks. The operating tasks depend on the module's structure but the following functions are implemented on all MPU cards.

Command Interpretation reads a commands from the CompactPCI bus and then invokes the appropriate routine.

Module Configuration allows the module to set itself up. This includes down-loading the configuration data into FPGAs and setting the initial data of the SRAMs and CAMs on the module. The application software is then loaded into memory and invoked. This function is invoked whenever ATTRACTOR is started.

Status Report gives the current status of the module. The values of the status registers can be read from the main MPU.

The configuration data of the FPGAs, memory initialization data and application software are stored in a hard disk attached to the main MPU module. The MPU card of each module can access the disk through the system bus.

In additional, we adopt a World-Wide-Web (WWW)-based graphic user interface. A hyper text transfer protocol (HTTP) server runs on the main MPU module and provides all user interface menus.

3.3. Design environment

A system design environment based on the our design concept is illustrated in Fig. 7. ATTRACTOR's design environment is composed of a hardware design environment, a software design environment, and a "virtual parts" library as a design database.

The designer specifies the telecom application by referring to the OSI model. Thus the specification can be easily allocated to each protocol layer and implemented in each module. Most of the specification's processes are implemented with hardware and require control software. The layer processing module editor uses the hardmacros in the virtual parts library for the hardware part. This provides a simple design environment for generating the hardware description language (HDL) descriptions. The FPGA CAD translates HDL descriptions into the FPGA's configuration data. The software part is described using the generated hardware/software interface routines and is compiled into executable data. The processed data are stored in the hard disk which is equipped on the main MPU as the layer processor configurations. The configurations are installed in the modules and executed.

The "virtual parts" library is the most unique feature of our design environment. It contains frequently used hardware designs for the protocol layer processing

Figure. 6. A photograph of the prototype ATTRACTOR.

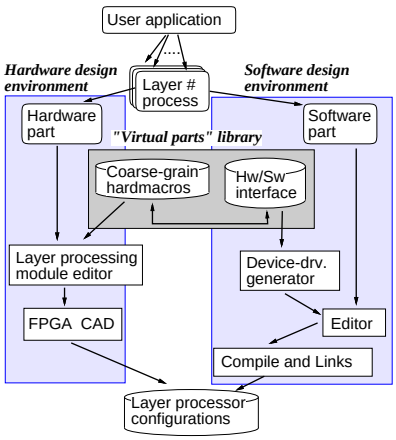

Figure. 7. Design environment for ATTRACTOR.

Table 1: Implemented circuits performance and grain size of hardmacro.

Circuits	Grain size of hardmacro		
	not used	Fine	Coarse
Circuit A	104	72	49
Circuit B	151	58	45

Critical path delay (nsec)

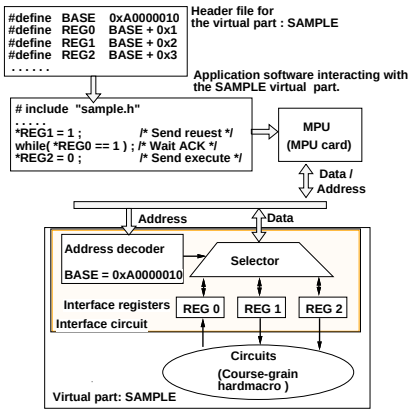

Figure. 8. Interaction between the virtual part and the application software.

modules. Each virtual part consists of a coarse-grain hardmacro and software interface routines. The coarse-grain hardmacro is a large-scale logic circuit netlist. The functionality of the hardmacro is the same as that for an OSI protocol layer like AAL termination or IP layer handling.

Table 1 shows the relationship between the grain size and the implemented circuits performance in case of PROTEUS-Lite. The circuits that have coarse-grain hardmacro can work faster than the others and thus satisfy our requirements.

Figure 8 illustrates how we can use the virtual parts with the software. Each virtual part communicates through the MPU interface circuit with the software running on the MPU card. All interface registers in the MPU interface circuits are mapped onto MPU-accessible memory addresses. The device driver generator generates a header file (C language), which includes interface register address definitions as static values for each virtual part. During program development, a programmer can use nicknames, defined in the header file, instead of the raw addresses. The base address of each circuit in the header file uses the address map of each module, which prevents address congestion. Thus, the designer can be freed from the address management work.

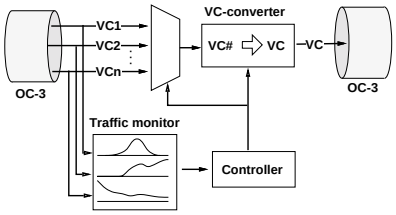

Figure. 9. A simplified block diagram of the dynamic VC switch function.

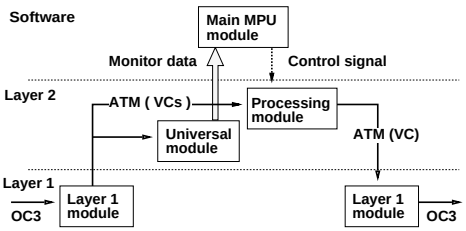

Figure. 10. A block diagram of the implemented dynamic VC switch.

4. Application implementations

We implemented several applications on the ATTRACTOR to evaluate our concepts. Two of them are described here. The first one is a layer two application called "Dynamic VC (Virtual Channel) switch" that manipulates ATM cell header information. The second is a layer three application called "IP router" that manipulates IP header information.

4.1. Dynamic VC switch

Figure 9 is a simplified block diagram of the dynamic VC switch. The incoming ATM cell traffic load, which can be identified by ATM header information, is continuously monitored for each VC. The VC that has the largest load is automatically connected to the outgoing VC via the VC-converter.

The implemented result is shown in Fig. 10. The OC-3 is terminated with the layer one module. The universal module includes the traffic monitor function in layer two. The VC-converter function is done in the processing module. The controller is implemented as software running on the main MPU module. The software reads the monitored data from the universal module through the system bus and then analyzes it to make the next decision.

To demonstrate this function, we sent video data over two different VCs at different traffic load through ATTRACTOR. The implemented system could switch

quickly from the higher traffic load VC to the lower one whenever the higher traffic load path was cut or overloaded.

4.2. IP router

Figure 11 is a functional block diagram of the IP router, which manipulates IP header information encapsulated in the ATM cells.

The incoming ATM cells are first terminated as an AAL frame that contains an IP datagram, and then, the IP address is extracted from the frame. The destination VC is decided by looking it up on the routing table, which contains many pairs of incoming IP addresses and destination VCs. The AAL frame is converted to ATM cells using the destination VC.

Figure 12 is a block diagram of the implemented IP router. The OC-3 line was terminated with the layer one module as in the first case. We used the buffer module as the AAL terminator in layer two because the AAL termination needs a high-performance FIFO, which is a pre-implemented function of the buffer module. Hence, we used a buffer module for layer two termination. The AAL frame consists of an IP datagram and trailer data, which are located at a fixed position on the frame. Thus, we can use the buffer module to extract the IP diagram from the AAL frame.

The buffer module sends an IP datagram to a processing module that has been loaded the IP address extracting function in the FPGAs and the routing table data in the CAMs. These are layer three functions. The IP diagram is encapsulated in AAL frame again and is converted to ATM cells using the destination VC information in the processing module. The ATM cells are sent by OC-3 line through the other layer-one module. The system can also be operated as a node in a real ATM-LAN environment.

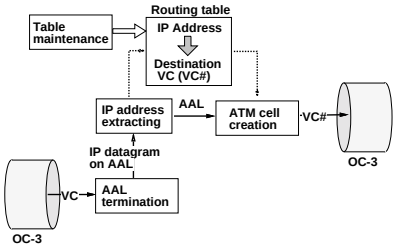

Figure. 11. A simplified block diagram of the IP router application.

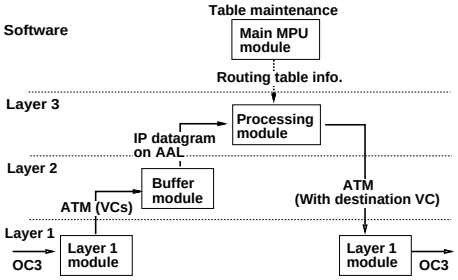

Figure. 12. An implemented block diagram of the IP router.

5. Conclusion

We developed a telecom emulator that is able to operate in a real network. The system is based on the open system interconnection reference model in which many different layer processes are packed into individual modules that are connected to each other with wide-band serial-links. It can be applied to different telecom-applications by using homogeneous processing modules, which are implemented in different OSI layers. We demonstrated our prototype system's flexibility and high-speed processing capability with some real network applications.

In the evaluation, we used simple applications that could be categorized as being layer two or three. We are currently implementing a higher layer application as a further test of the usefulness of our concept.

References

[1] Aptix Corp. Aptix System Explorer, Brochure, (http://www.aptix.com).

[2] Quickturn Systems, Inc. System Realizer M3000/M250, Brochure, (http://www.qcktrn.com).

[3] K. Hayashi, et al. A Novel Approach to Real-Time Verification of Transport System Design. *Proc. of 7th IEEE International Workshop on Rapid System Prototyping* , pp. 5–9, 1996.

[4] N. Ohta, et al. PROTEUS: Programmable Hardware for Telecommunication Systems, *Proc. IEEE International Conference on Computer Design (ICCD)*, pp. 178–183, 1994.

[5] Organization International Normalization(ISO) Information technology – Open System Interconnection – Basic Reference Model: The Basic Model ISO/IEC 7498-1, 1994.

[6] CompactPCI Specification Short Form: PICMG 2.0 R2.1, PCI Industrial Computer Manufactures Group, Sep. 1997, (http://www.picmg.com).

[7] Wind River Systems, Inc., VxWorks Programmer's Guide 5.2., Manual, Mar. 1995.

[8] T. Miyazaki, et al. CAD-oriented FPGA and Dedicated CAD System for Telecommunications. In *Proc. of Field-Programmable Logic (FPL)*, 1997.