YARDS: FPGA/MPU Hybrid Architecture for Telecommunication Data Processing

Akihiro TSUTSUI and Toshiaki MIYAZAKI*
NTT Optical Network Systems Laboratories
1-1 Hikarinooka, Yokosuka, Kanagawa, 238-03 Japan
akihiro@exa.onlab.ntt.co.jp
*NTT System Electronics Laboratories
3-1 Morinosato Wakamiya, Atsugi, Kanagawa, 243-01 Japan
miyazaki@exa.onlab.ntt.co.jp

Abstract

This paper presents a novel system architecture applicable to high-performance and flexible transport data processing which includes complex protocol operation and a network control algorithm. We developed a new tightly coupled Field Programmable Gate Array (FPGA) and Micro-Processing Unit (MPU) system named Yet Another Re-Definable System (YARDS). It comprises three programmable devices which equate to high flexibility. These devices are the RISC-type MPU with memories, programmable inter-connection devices, and FPGAs. Using these, this system supports various styles of coupling between the FPGAs and the MPU which are suitable for constructing transport data processing. In this paper, two applications of the system in the telecommunications field are given. One is an Operation, Administration, and Management (OAM) cell operations on an Asynchronous Transfer Mode (ATM) network. The other is a dynamic reconfiguration protocol enables the update or change of the functions of the transport data processing system on-line. This is the first approach applying the FPGA/MPU hybrid system to the telecommunications field.

1 Introduction

Conventional implementations of telecommunication systems are achieved through fixed hardware since importance was placed on high-speed transport data processing rather than flexibility. However, today's enthusiasm for the inter-networking trend is forcing network systems to support a wide variety of communication protocols, even those are not yet fully developed. Thus, not only high-performance but also high-flexibility will be indispensable for future network systems [1].

The Asynchronous Transfer Mode (ATM) technique is one of the solutions to this problem It is suitable for multimedia data communication because it has the capability to handle several data bandwidths equally and flexibly. In addition, its hardware implementation is so simple that it is easy to construct high-throughput network systems [2]. However, these advantages are effective only when complete network systems and protocols are designed in the ' world of ATM '. For example, efficient harmonizing of ATM protocols and universal computer network protocols such as TCP/IP, which is in strong demand for recent inter-networking, is a hard task because the former is hardware oriented and the latter is based on sofware technology. The main problem is how to close the gap between them.

Moreover, when providing high quality multimedia telecommunication services, it is indispensable to achieve a high-performance and flexible network control and management [3]. Up to now, there has been no other choice but to construct these as sofware because they are complex and require frequent updating. Recently, however, it is considered that some part of the operation should be implemented as a hardware for achieving high speed data manipulation. Thus, in the future, lower layer telecommunication protocols will require more flexible implementation and higher ones will be forced to achieve higher performance. Therefore, an efficient fusion technique for hardware and sofware is becoming indispensable for building the next generation network systems.

In such a situation, we considered that a Field Programmable Gate Array (FPGA) is the key device of future telecommunication systems and have developed an original FPGA especially designed for high-speed telecommunication data processing [4][5]. Using this device, we also constructed a reconfigurable signal-transport system dedicated to real-time emulation of transport processing circuits [6]. This system is useful in implementing lower-layer transport operations. However, more higher-layer protocols and network management applications mentioned above include excessively complex logic operations which are not suitable for FPGAs. Micro-Processing Units (MPUs) and program logic implemented as sofware are indispensable to them.

On the other hand, there are reports on some hybrid systems which consist of tightly coupled FPGAs and MPUs [7][8]. In these systems, both FPGAs and MPUs cooperate with each other to execute operations. Generally, however, it is difficult to find an actual target application that maximizes the performance of the system. Because it is difficult to divide a given problem into two implementation styles: hardware and sofware. From our experience, we discovered an effective application for the system in the field of telecommunications. In a transport processing system, the lower layer protocols are suitable for hardware implementation and it is preferable to describe the higher layer protocols as sofware. It also requires a high-throughput to operate real-time data transmission and flexibility to support various protocols.

Thus, we developed a novel architecture for a system comprising tightly coupled FPGAs and MPUs. This architecture is suitable for implementing flexible and real-time transport data processing operations which are the main part of telecommunication systems.

This paper presents the new architecture of the hybrid system and its advantages. General ideas for achieving efficient coupling of sofware (MPU) and hardware (FPGA) are also mentioned.

Moreover, applications of the hybrid system in the telecommunications field and actual implementation for a few instances are shown. This is the first approach to applying the FPGA/MPU hybrid system to the telecommunications field.

2 Related Work

Some other systems which comprise an MPU and FPGAs have been proposed [7][8]. In those hybrid systems, some particular operations suitable to hardware implementation are performed by specially designed logic circuits on the FPGAs and the MPU works in conjunction with them. They communicate each other using MPU's local bus. Thus, the FPGAs are treated as co-processors or special peripheral devices of the MPU. Moreover, specially designed FPGAs dedicated to co-working with the MPU were developed [9].

Most of the target applications for these systems involve numerical calculation or digital signal processing such as video-CODEC. However, it seems difficult to find an effective application which derives the maximum performance from the system. One reason for this problem is that it is difficult to divide the target application into hardware and sofware. Moreover, data transmission between the MPU and FPGA becomes a bottleneck which prevents the system from attaining a high throughput. Thus using only a local bus-style connection, the bus congestion and overhead of the bus arbitration protocol reduce the merits of the system.

3 YARDS Architecture

3.1 Overview

We developed a new system architecture comprising tightly coupled FPGAs and an MPU named YARDS (Yet Another Re-Definable System). Figure 1 shows the basic architecture of YARDS. The main parts are an MPU, an FPGA array, programmable switching devices, and 2-port SRAMs. Figure 2 shows the system overview of YARDS. It comprises three cards: the main card, the MPU card, and the FPGA card. The main card contains devices for interconnecting the FPGA part and the MPU part of the system. The MPU card contains a RISC MPU with a BIOS-ROM. The FPGA card features a multiple FPGA array mounted on it. The MPU card and the FPGA card are designed as separated modules.

YARDS also has two external interfaces: VME-Bus I/F and direct I/O channel derived from the FPGA card. Using the VME-Bus, this system can be communicate with other YARDS or host computer systems. We utilize it for controlling and monitoring the system. The direct I/O channel enables a direct exchange of signals between other devices and the FPGA card.

The front view of YARDS is shown in Figure 3 and the back is shown in Figure 4.

3.2 YARDS Main Card

The main card contains interconnection elements for the FPGAs and the MPU. It comprises field programmable switching devices (I-Cube) that support various types of connections among its pins such as a one-directional or a bus. The local-bus signals and a few interrupt pins of the MPU and most I/O pins of the FPGAs are connected directly to these switching devices.
The 2-port SRAMs on the main card have various uses. These are connected directly to the switching devices. Configuring the connection pattern of the switches, those SRAMs are used as shared memories or buffers among the FPGAs and the MPU. Thus, using these programmable switching devices and 2-port SRAMs, various connection styles between the FPGAs and the MPU can be established.

The programming and control logic of the system are implemented on the main card. Those circuits are implemented using an FPGA, thus the system can support various types of FPGAs and MPUs by re-programming the FPGA.

YARDS has three clock generators except for the MPU's base clock (20 MHz). One of these provides a fixed clock speed of 19.44 MHz. This is the base clock for a typical telecommunication circuit which handles a 155-MHz Synchronous Digital Hierarchy (SDH) interface. Others are programmable clock generators.

3.3 FPGA Card and MPU Card

We wanted to try various types of FPGA or MPU devices because YARDS is an experimental system. Thus the FPGA part and the MPU part of the system are designed as separate daughter cards. They can be replaced with the cards that contain other devices. As for the MPU part, we have only a few choices owing to the constraint of the local bus compatibility. However, as for the FPGA part, there are many candidates. Actually, we have developed two types of FPGA cards which consist of different FPGA devices. We adopted Xilinx LCA (XC4010) and ALTERA MAX (MAX-9000) respectively. Figure 5 shows a block diagram of the LCA array card. It consists of four devices which are connected directly to each other on the card as shown in Figure 5. Their link topology on the card is a cascade. However, most of the I/O pins of the FPGAs can be connected to the programmable switching devices on the main card and some of them are connected to external I/O connectors. Using the switches, the topology of the links among the FPGAs can vary as shown in Figure 6. The MAX-9000 array card also has an architecture such as that shown in Figure 7. Furthermore, we are planning to develop a new type of FPGA card which contains our original FPGA device [4] using a new compact package version of it.

Generally, different kinds of FPGAs require different configuration methods. Therefore, the configuration and logic control of an FPGA must be changed for each device. This difference in the logic configuration can be absorbed by the programming control FPGA (LCA XC4005) mounted on the main card. All of the pins for configuration and control of the FPGAs on the FPGA card are connected to this programming control FPGA via the switches. By reprogramming a suitable configuration logic circuit on the programming control FPGA, we can program and control different types of FPGAs on the FPGA card.

Figure 8 shows a block diagram of the MPU card. We adopted a 32-bit RISC microprocessor (Hyperstone E-1) which has a simple architecture and is easy to use. The MPU card comprises the MPU and a BIOS-ROM. All of the local bus signal lines are connected to the main card bus via the connectors. This RISC microprocessor has five different interrupt signals. A few interrupt signal pins are connected to the FPGA card directly, the rest of the pins are connected to switching devices on the main card via the connectors.

3.4 Configuration

YARDS has great flexibility because almost all of its parts are programmable. Its programming procedure is considered complex and difficult. Therefore, an auto configuration procedure of YARDS is pre-implemented in the BIOS-ROM. After the MPU is booted up, the initial setup procedures are automatically performed. In the procedure, the switching devices and the FPGAs are configured first. Because they have physical level flexibility and may destroy the system by an error.

For run-time programming of YARDS, some useful library functions written in C language are provided for users. They can easily program and control the FPGA card and switching devices using this library.

4 Advantages

4.1 Flexible Interconnection between MPU and FPGA

Most conventional hybrid systems that comprise MPU and FPGAs employ a bus architecture [8]. They treat the FPGAs as a co-processors or I/O devices as shown in Figure 9. This interconnection style couples these devices tightly. However, there are some problems that prevent harmonious cooperation between them.

For example, immediate communication from FPGAs to an MPU is difficult to implement using this interconnection style. In addition, a local bus congestion problem caused by the communication between an MPU and FPGAs is considered. Therefore, using only the bus architecture, the applications of the system will be restricted.

YARDS supports three different styles of connection between FPGAs and MPU: a bus, a direct interrupt, and a 2-port SRAM channel. These connection styles can be established using programmable switching devices and are configured easily.
The bus style is the same as the conventional one. The direct interrupt links connect the FPGA and the interrupt signal pins of the MPU as shown in Figure 10. This connection style enables the FPGA to interrupt and control the behavior of the MPU. In a conventional FPGA/MPU hybrid system, the main device is an MPU. That is, the main instructions are implemented as sofware and executed by the MPU, and the FPGAs are considered as sub-devices. However, by using these direct interrupt links, the FPGA part is able to perform a leading role in the system. In such a system, the main instructions are implemented as a logic circuit on FPGAs,

and the MPU performs a supporting role. The MPU is always ready for an interrupt signal from the FPGAs. When the interrupt signal is sent, a corresponding subroutine is invoked by the MPU. This connection style aids in implementing some transport data processing protocols which have data-driven operations such as a frame synchronizer and transmission error handling.

4.2 2-Port SRAM Channel

Considering our main target applications, the transport data processing operations, it is expected that the data communication between the MPU and the FPGAs occurs very frequently and asynchronously. In such a case, cooperation between the MPU and FPGAs will cause local bus congestion and degrade the performance of both. Using only the bus architecture, an implementation style of our target system should be similar to Figure 11. The transport data stream is input into the FPGA directly. The FPGA executes some low layer protocol operations and transfers the results to the main memory. Then the MPU accesses and reads the results from memory while executing some high layer protocol operations. When those operations are finished, the FPGA accesses the main memory, derives the processed data from it and re-shapes the data as the output transport data stream. In general, a local bus of MPU is usually occupied by data transmission among the memories, the peripheral devices and the MPU itself. Therefore, these repetitive data transformations among the MPU, the FPGAs, and the memories should block the local bus.

Moreover, if bus connections are used, both the MPU and FPGAs must be synchronized with local bus timing and yield to arbitration protocols. For FPGA in particular, a bus interface circuit should be incorporated into it, however it would occupy a considerable area in the device.

The 2-port SRAM channel is one of the remarkable features of YARDS. As shown in Figure 10, the 2-port SRAMs on the main card can be configured as channel devices between FPGAs and the MPU. They perform asynchronous data transformation. This mechanism aids in implementing applications that should work at two or more different base clock speeds. Moreover, using this communication style, the MPU and the FPGAs can devote themselves to their respective tasks without influencing each other. For example, multi-layer protocol operation which includes both lower and higher layer protocol operations is one good application for this connection style. In such an operation, each protocol operation is executed in sequence for the same datagram. The overhead of a data copy operation from one memory to another is a considerable problem Using this type of connection, the overhead can be canceled because each device can share the same data in a 2-port SRAM and access it independently.

In addition, the 2-port SRAM devices on the main card are used not only for channel elements between FPGAs and the MPU but also for a normal memory device, an FIFO, or a STACK. Using switching devices, various kinds of storage can be built as shown in Figure 12. Furthermore, when the MPU accesses the SRAM through the FPGAs, the combination of these devices is considered as a kind of functional memory. In this case, the FPGA can be configured as pre or post data access processing. For example, implementing a hash function into the FPGA is useful for a table lookup operation.

4.3 General Suggestion

In general, for the MPU/FPGA hybrid system, frequent data exchange between the MPU and the FPGA (such as that shown in Figure 11) is counter-productive towards achieving maximum performance. This is because there are few applications which overcome the data transmission overhead.

Most MPUs do not have direct I/O ports which handle continuous data streams. This is a weak point of the device when it is applied to telecommunication data processing systems. On the other hand, FPGAs have many direct I/O ports and are suited to handle real-time operations for continuous data streams. Therefore, we employ FPGAs as an I/O pre or post processing element for continuous streams and link the MPU and the FPGAs with shared memory which can access each other independently. Consequently, frequent and asynchronous data transfer between these devices can be achieved with little overhead.

This style of implementation should be useful not only for telecom applications but also for real-time and continuous data operations.

5 Applications

5.1 Advantages in Telecommunications Field

We have developed several types of telecommunication systems using FPGAs [6]. From our experience, we discovered that a hybrid system comprising MPUs and FPGAs is useful in the telecommunications field.

Most of the telecommunications protocols can be easily divided into hardware and sofware implementations. As is generally known, telecommunications protocols form layers. The lower layer protocols which require real-time high-throughput data operations are implemented as dedicated hardware such as ASICs . On the other hand, the higher layer protocols and network management operations which include complex procedures are implemented as sofware and are executed by the MPU. Originally they were comprised of hardware parts and sofware parts. Thus, it is easy to find the splitting point.

5.2 ATM Network Termination Card

We also developed an ATM Network Termination (ANT) card as a network interface device for YARDS. Using ANT card, YARDS can be applied to several protocols and network management operations in ATM easily. Figure 13 shows an overview of the card. It is designed as a standard VME-Bus card and consists of the host processor card and ATM network interface card. The host processor card is a one-board type computer which contains a RISC microprocessor and its basic peripheral devices. The ATM network interface card is designed as a daughter card for the host processor card and comprises a 155-MHz optical interface, ATM physical layer device, and an ATM Segmentation And Re-assembly (SAR) layer device. The function of the card is the same as commercial ATM network interface cards for personal computers or workstations except that it can directly input and output ATM cells.

5.3 Application Examples

We selected OAM processing in the ATM network as one of the effective applications for YARDS because some of the operations are not completely standardized and require some degree of flexibility [10][11]. Moreover, improvement in the processing performance is most effective for achieving high quality telecommunication services [12].

The OAM processing controls and manages the status of the network. Figure 14 shows a typical model of the operations. In an ATM network, two types of ATM cells are used for data communication. One is a user cell which contains user data for telecommunication services. The other is an OAM cell which conveys control and management data for network nodes. Canonical OAM procedures are picking up OAM cells from the data stream and executing corresponding operations indicated in the cell. These basic operations are: **DISCARD** (take and ignore the OAM cell), **DROP** (take the OAM cell and invoke some operations instructed by the cell), **MONITOR** (observe the OAM cell, invoke some operations instructed by the cell) and **THROUGH** (the node does not execute any operation by the cell, only forwards the cell to other nodes).

Figure 15 shows the conventional implementation style of the OAM processing on a network node [13]. The present network node consists of a transport processing unit (hardware part) and a work station. The transport processing unit handles the main data stream and performs most low layer protocol operations. The work station mainly executes the control algorithm. These two modules are connected to a LAN such as Ethernet. When OAM cells are terminated and extracted by the transport processing unit, they are sent to the work station using Ethernet. The turn around time can exceed 100 to 500 ms. This overhead causes a significant delay in services. For example, path restoration at a network accident can take a few minutes in such a condition [13]. According to previous research, restoration should take 2 seconds at most to minimize the damage to multimedia data communication using real-time video or voice [12]. Thus, the bottleneck of the present system is the loose coupling between hardware and sofware part of the system.

Figure 16 shows the function block diagram of the OAM cell processing. It comprises three parts. The first is cell operation part-1, which treats the main data stream and performs transport data processing such as ATM cell termination and generation. The second is cell operation part-2, which treats only OAM cells extracted from the main data stream. This part examines the message type included in the OAM cell and determines the appropriate action for itself, i.e., discard, drop, monitor, or through. CRC-10 error detection is also performed in this part. The last is the algorithm part, which is implemented as a sofware program in a work station. Its main tasks are accessing the database of the network and determining the actual control method. This implementation is reasonable but there is a considerable gap between sofware and hardware and the communication delay between them is large in conventional systems. Consequently, we cannot achieve a quick response for controlling the network.

The OAM cell processing with our system is shown in Figure 17. Cell operation part-1 is implemented using ANT. Cell operation part-2 is implemented with an FPGA card on YARDS. Furthermore, a high level of performance is needed in this part. The algorithm part is implemented as sofware running on the MPU.

The proposed system architecture bridges the gap between hardware and sofware. The remarkable difference from the old architecture is the data sharing mechanism using the 2-port SRAM channel. As shown in Figure 17, the FPGAs write the data of the OAM cells into the dual-port SRAM data channel. At this time, the data needed for network control, which is encapsulated in the OAM cells, are arranged and transformed to permit easy MPU access. For each data block, the timing and information type are added as a tag. Thus the sofware running on the MPU can distinguish the data and invoke the network control algorithm immediately. The generation of the data structure by the FPGAs is the key point of the sofware-hardware bridging. The transformed data structure corresponds to the data type as directly defned by C language. From the sofware programmer's viewpoint, this information looks similar to a simple variable. Furthermore, FPGAs and the MPU can asynchronously access the SRAM any time and memory access time is quite fast, thus communication overhead is negligible. This process cycle requires only the execution time of the algorithm and so we can expect to achieve response within several hundreds of microseconds.

In addition, using run-time programmability of the system, we implemented a dynamic reconfiguration protocol which updates or changes the function of the system itself. The procedure of the protocol is shown in Figure 18. This protocol performs as a client-server model. In Figure 18, there are program distribution servers (P-Servers) and programmable clients (P-Clients). Based on the protocol, they exchange several requests and acknowledgments encapsulated in the OAM cells. During those handshake periods, the P-Client and the P-Server confirm their statuses with each other. Then, the configuration data, i.e. program is segmented, encapsulated in OAM cells, and sent from the P-Server to the P-Client. The P-Client re-assembles and restores the data. After the closing handshake to confirm that the data transformation has successfully terminated, the P-Client re-configures its own function with the data. These series of procedures can be performed without any interruption of the main transport processing task. This is because almost all transport data processing systems are constructed as "duplex systems" for reliability. Therefore, while one part of the system performs, the other part can be configured concurrently as shown in Figure 19. Using this configuration and protocol, remote maintenance and upgrade for already installed systems which are remotely located from the operation center can be performed. Moreover, dynamic and adaptive protocol reconfiguration can be achieved on the system as well as effective data transportation, according to the circumstances. We have implemented this dynamic reconfiguration protocol on YARDS and verified that the configuration process is completed.

6 Conclusion

We developed a novel architecture for a high-performance and flexible transport data processing system. It comprises tightly coupled FPGAs and an MPU. This system also has programmable switching devices and 2-prot SRAMs. It supports various types of connection styles between FPGAs and the MPU. We implemented practical applications for OAM processing in an ATM network and examined its behavior and performance. This type of system proved to be useful for implementing transport data processing operations which require both high-throughput and complex algorithms. It is expected to become a key architecture in the telecommunications systems of the future.

References

[1] W. Fischer, *et al.*, "Data Communications Using ATM: Architectures, Protocols, and Resource Management," IEEE Communications Magazine, pp. 24-33, Aug. 1994.
[2] A. Iwata, *et al.*, "ATM Connection and Traffic Management Schemes for Multimedia Internetworking," Communication of the ACM, Vol. 38, No .2, pp. 73-89, Feb. 1995.
[3] W. Fischer, *et al.*, "Data Communications Using ATM: Architectures, Protocols, and Resource Management," IEEE Communications Magazine, pp. 24-33, Aug. 1994.
[4] A. Tsutsui, *et al.*, "Special Purpose FPGA for High-speed Digital Telecommunication Systems," Proc. ICCD'95, pp. 486-491, Oct. 1995.
[5] A. Tsutsui and T. Miyazaki, "An Efficient Design Environment and Algorithm for Transport Processing FPGA," Proc. VLSI'95, pp. 791-798, 1995
[6] K. Hayashi, *et al.*, "Reconfigurable Real-Time Signal Transport Systemusing CustomFPGAs," Proc. IEEE FCCM'95, pp. 68-75, Apr, 1995.
[7] S. Casselman, *et al.*, "Creation of Hardware Objects in a Reconfigurable Computer," Proc. FPL'95, pp. 119-128, Aug. 1995.
[8] S. Guccione, "List of FPGA-based Computing Machine," URL=http://www.io.com/~guccione/HW_list.html
[9] S. Churcher, *et al.*, "XC6200 FastMap Processor Interface," Proc. FPL'95, pp. 36-43, Aug. 1995.
[10] R. Kawamura, *et al.*, "Fast VP-Bandwidth Management with Distributed Control in ATM Networks," IEICE TRANS COM., Vol. E77-B, No. 1, pp. 5-14, Jan. 1994.
[11] The ATM Forum Technical Committee, "Traffic Management Specification Version 4," ATM Forum/95-0013R6, Jun. 1995.
[12] "Digital Cross-Connect Systems in Transport Network Survivability," Bellcore SPECIAL REPORT, SR-NWT-002514, No. 1, Jan. 1993.
[13] R. Kawamura, *et al.*, "Implementation of Self-Healing Functions in ATM Networks Based on Virtual Path Concept," Proc. of INFOCOM'95, Apr. 1995.

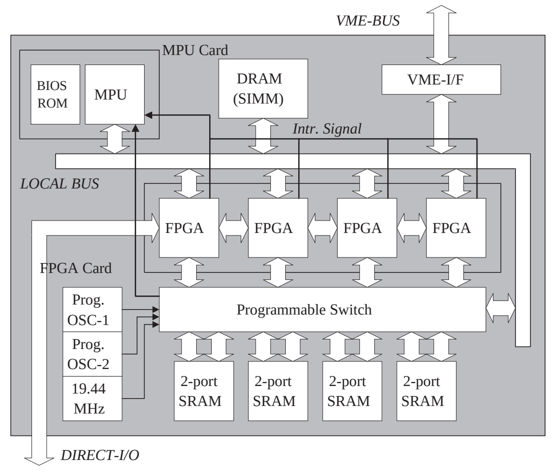

Figure 1. Basic Architecture of YARDS

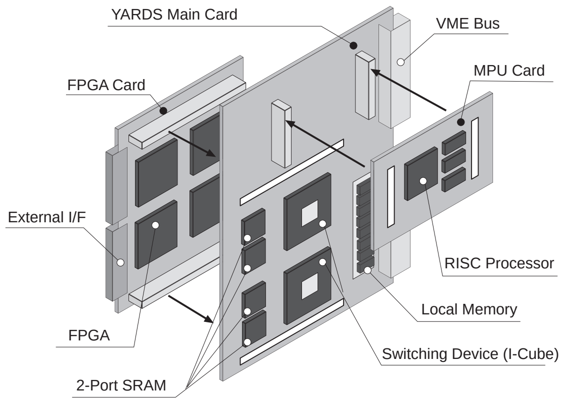

Figure 2. System Overview of YARDS

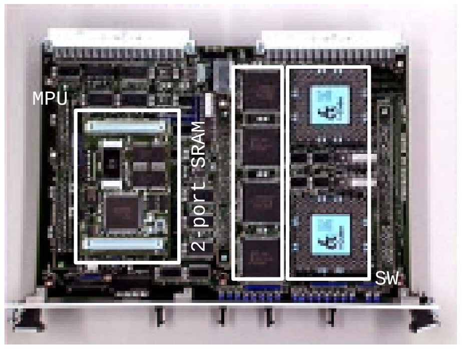

Figure 3. Picture of YARDS (Front)

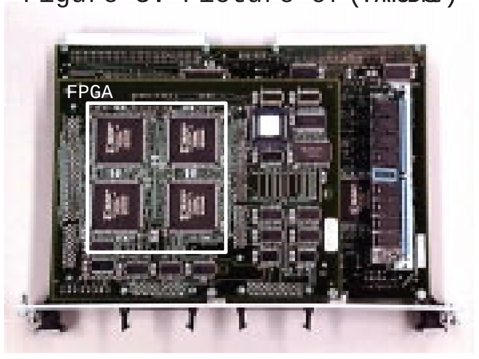

Figure 4. Picture of YARDS (Back)

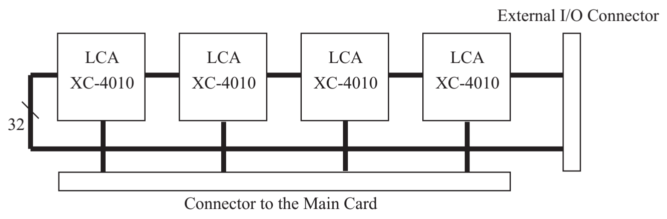

Figure 5. FPGA Card (using LCA)

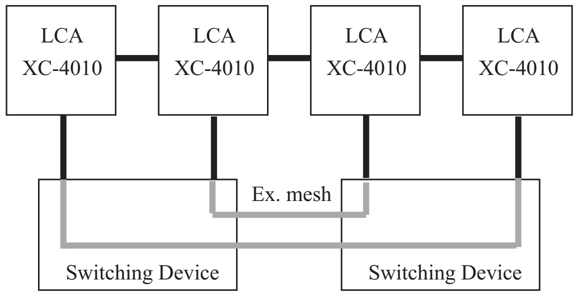

Figure 6. Example of a Link Topology (Using LCA)

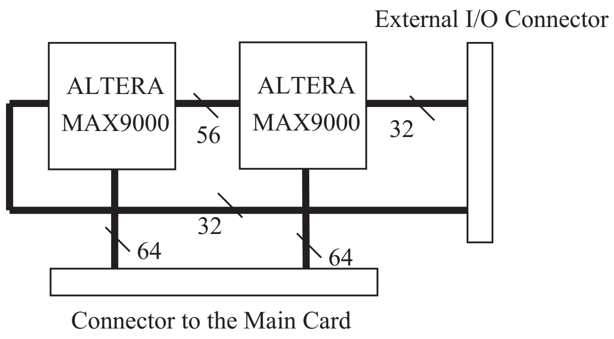

Figure 7. FPGA Card (using MAX9000)

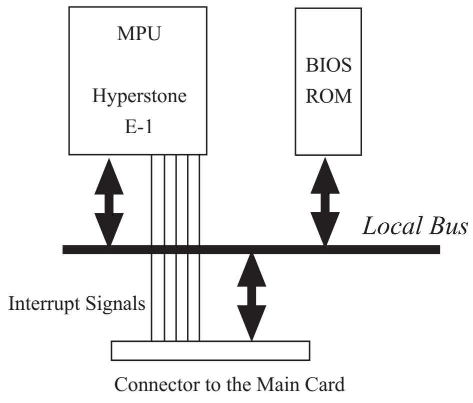

Figure 8. MPU Card

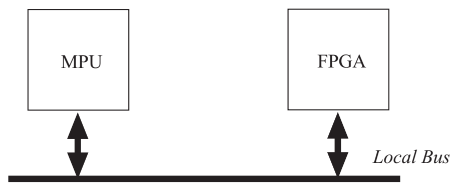

Figure 9. Interconnection using Local Bus

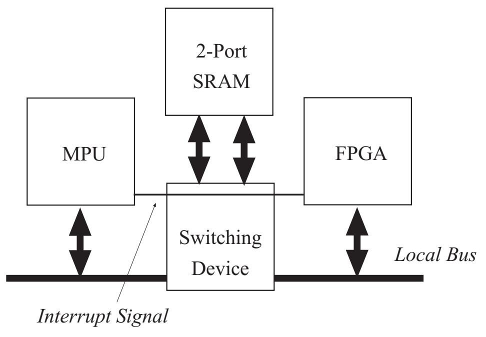

Figure 10. New Interconnection Styles

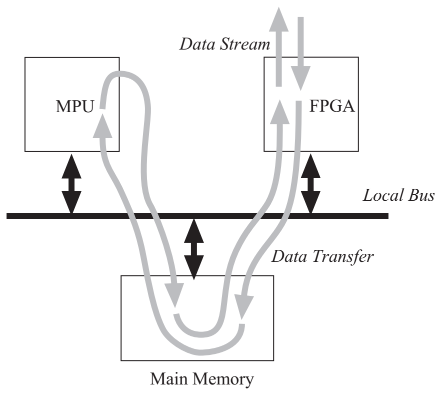

Figure 11. Typical Implementation Style using Local Bus

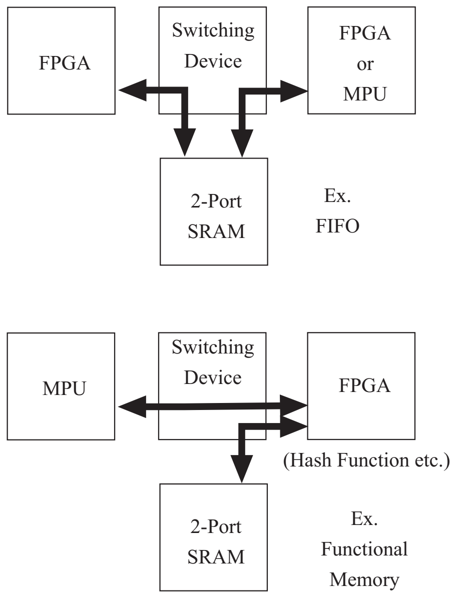

Figure 12. Memory Configuration

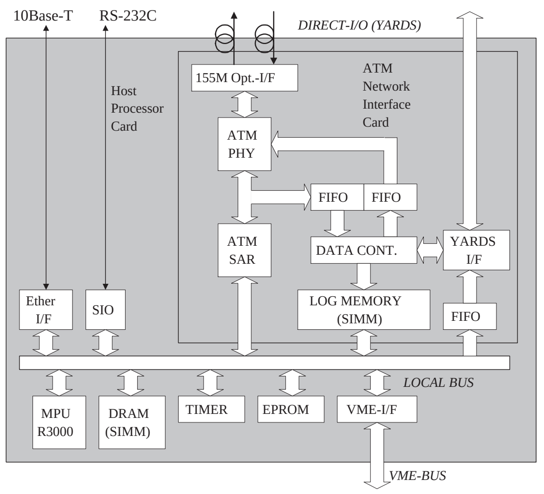

Figure 13. Basic Architecture of ANT

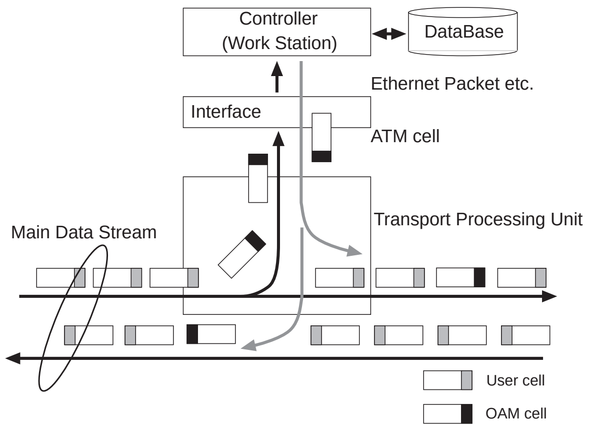

Figure 14. OAM cell Processing Model

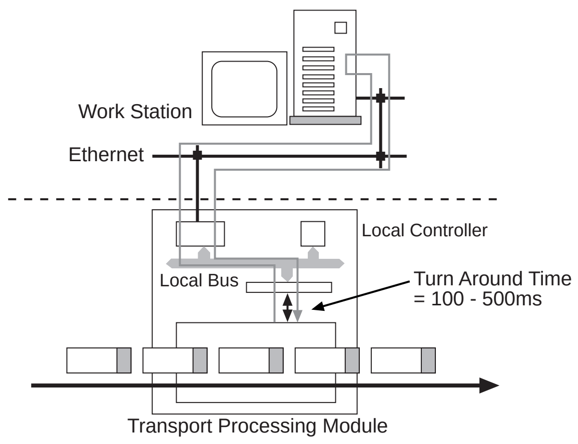

Figure 15. Conventional Implementation of OAM cel
processing

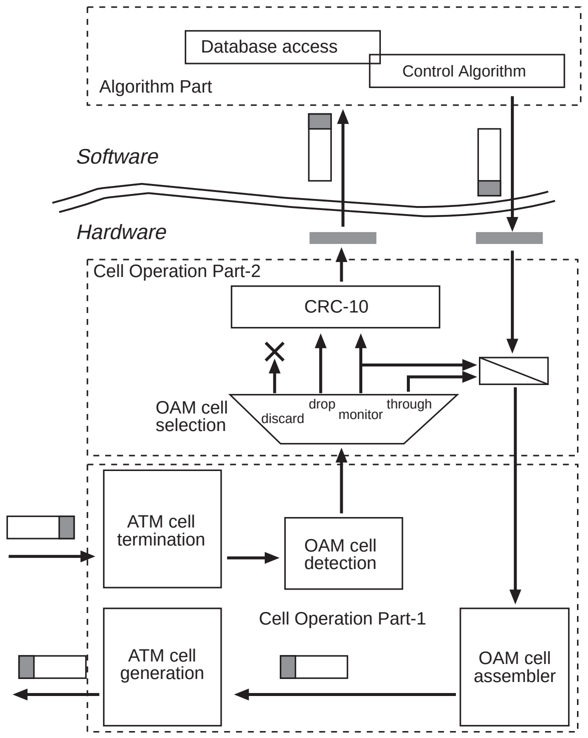

Figure 16. Functional Block Diagram of O
processing

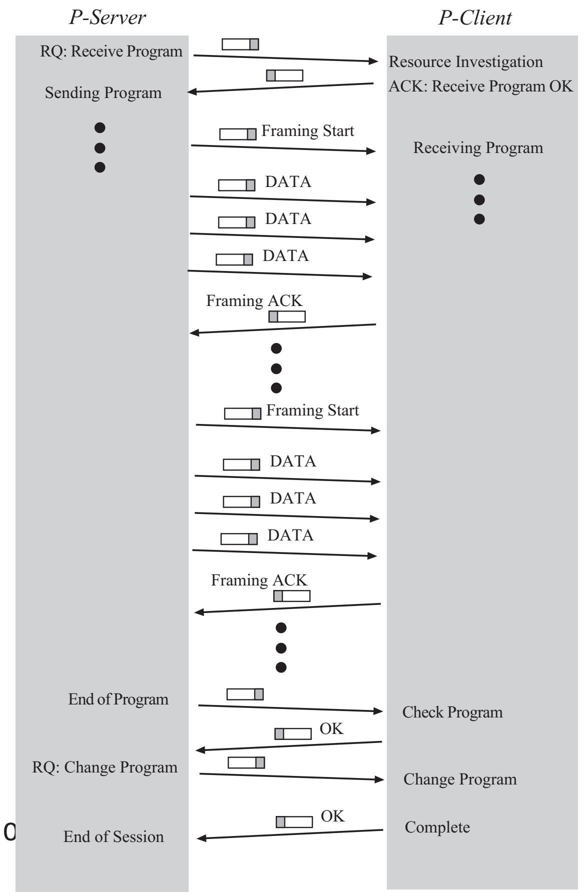

Figure 18. Dynamic Reconfiguration Protocol

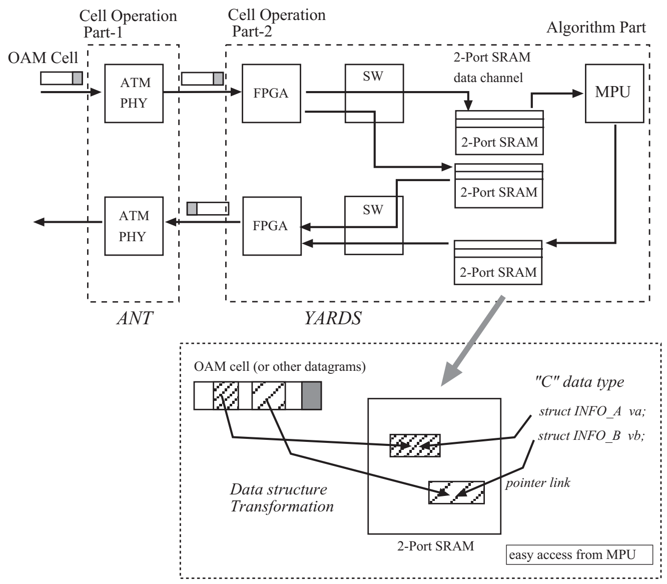

Figure 17. Implementation of OAM cell processing
on YARDS

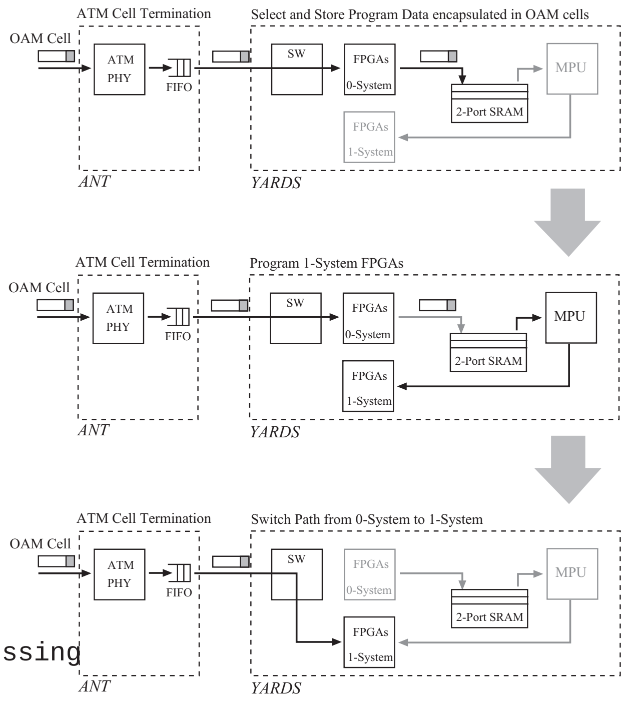

Figure 19. Implementation of the Dynamic Reconfigurat
Protocol on YARDS