Using Reconfigurable Logic to Implement an Active Network

Edson L. Horta, Sergio T.Kofuji
Laboratory of Integrated Systems - Department of Electronic Engineering
Escola Politécnica da Universidade de São Paulo, São Paulo, Brazil
Av. Prof. Luciano Gualberto, trav. 3, 158. CEP 05508-900 - São Paulo, SP, Brazil
Tel: (011) 818-5589 fax: (011) 211-4574. horta@lsi.usp.br, kofuji@lsi.usp.br

Abstract

This work describes how the technology of Reconfigurable Logic can be used to implement an Active Network. It presents the reconfigurable switch, a switch capable of changing its architecture according to special codes encapsulated in the message data traversing the network. It provides a hardware approach of an emerging area, Active Networking, that is focused mainly in software implementations.

Keywords: active networking, reconfigurable logic, switching.

Introduction

In order to implement a flexible communication system it is necessary to build a network capable of providing new features to the user in a fast way. But the standardization process to make these features available to the final user is very long and time consuming. Thinking on these problems a group of researchers proposed a new technology to transform the flexibility of a network system in reality: the Active Network [17]. It allows the user to modify the network features through the use of programming packets traversing the network. These packets will program the new features in the network as soon as they are developed.

The present state of active networking deals mainly with the software approach [7], that is, the packets act like a program that will run on each active node of the network. If the active nodes could change their hardware features in the same manner they change the software ones, they would have been much more flexible.

To change a hardware feature of a node it is necessary to use reconfigurable logic. This technology makes use of FPGA (Field Programmable Gate Arrays) that can be programmed "on the fly" by the user.

In the next sections it will be explained in details these two technologies and how to combine them in order to build a reconfigurable switch.

Active Networks

All the nodes of a network have a primary task: receive incoming packets from one point and send them to one or more points on the network. In order to send and receive data from one point to another, the users have to follow strict rules. Furthermore, these rules are not keeping in pace with the new advances in technology due to the fact that a new rule takes too much long to be issued as a new standard.

If the network had the capability of changing its properties without waiting for a new standard, it would have been much more flexible. The Active Network (AN) [18] had evolved from this necessity.

An Active Network is composed of active nodes, responsible by the modification of the network features through the use of special programs encapsulated in the message data. These programs can reconfigure the network "on the fly", i.e., at the same time data is traversing the network.

Among the advantages of Active Networks it can be found the facility of implementation of new protocols by the industry. As soon as they are developed they can be sent to the network and the active nodes can transform them in a new "standard", without having to pass through all the steps of a formal approbation.

One of the problems of the Active Network is the security. If one wants to block a network, it is dangerous to provide to any person the possibility of changing the features of the network by simply sending an encapsulated program through it.

All the implementations of Active Networks focus in a software approach, that is, the functionality of the network is programmed through the use of special codes within the message data [16]. The active nodes have only to interpret these codes and modify its features according to them.

The next section will show a new way of implementing these active nodes.

A Hardware Approach

As explained in the previous item, the actual implementations of the active nodes are being made by software. They are composed of fixed elements capable of interpret a specific language.

With the advent of reconfigurable logic, now it is possible to build an active node that can modify its hardware to implement a new feature.

This technology is based on a component called FPGA (Field Programmable Gate Array), that is based on SRAM elements. These elements can be reprogrammed an unlimited number of times. According to [9], the implementation of a system that uses reconfigurability can be done in two ways: Compile-Time and Run-Time Reconfiguration.

In Compile-Time Reconfiguration (CTR) the FPGA does not change its configuration during the application lifetime. Each application has its specific functions that are loaded when you start it. This way is very similar to the use of an ASIC to implement a system. The main difference between them is the reprogrammability of the FPGA, which allows its re-utilization. When using CTR, it will be necessary to develop discrete images (partitions) for the application. Then each one of them will be programmed into an FPGA. These images will behave as static hardware during all the life of the project.

Some examples of CTR systems are SPLASH I and II (a Reconfigurable Linear Logic Array) [3] [4] and PAM (Programmable Active Memories) [13] [14].

The Run-Time Reconfiguration (RTR) allows changing the functions of the system during the application execution. This approach introduces two new design problems: the partition of the system and the communication between them. When partitioning a system, one has to divide it into time-exclusive segments that have to run non-concurrently, enabling each one to be programmed independently in the FPGA. This partition must be designed according to a temporal partition, visualizing the concurrent parts of the circuit and grouping them in each of the configurations loaded by the FPGA. Another important aspect in the design of a RTR system is how each configuration scheme will communicate with each other. It will be necessary to preview some procedures to store the intermediate values, wait for the reconfiguration process and retrieve the values to continue the process.

Some examples of RTR systems are Imputer (Imaging Computer) [15] and RRAN 2 (run-time reconfigurable artificial neural network) [5].

The implementation of RTR can be done in two ways: global or local.

The global RTR reconfigures all the FPGA when it changes the configuration. In order to have an efficient partition of configuration, one has to go through an iterative process that divides the system into segments using as much as reconfigurable resources as possible. This is an iterative process because no one knows the physical size of a partition before it has been placed and routed. After the partition process it is necessary to project the interfaces between the configurations and the communication between them. Because all the FPGA is used during the reconfiguration, it is necessary to have an area to store the common data and each configuration has to treat this data correctly. One of the disadvantages of global RTR is the time spent to change the configuration data in the FPGA. The advantage of this method is its simplicity, making it possible with the use of available commercial tools.

The local RTR consists of programming only a part of the FPGA, reducing the reconfiguration time. The partition process of this implementation is based on a functional division of the system. The functional parts used in all the application lifetime are programmed only once, during the initialization process. Then, when the application needs a specific function, the part used to perform it is loaded in place of a part no more useful. To have the reconfigurable resources utilized in an efficient manner, it is necessary to eliminate all the idle circuit or reduce it to a minimum. One problem that comes with local RTR is concerned to the physical space utilization. Because the FPGA will support multiple configurations at a time, it is necessary to predict the available space for them, preventing from the multiple occupation of the same physical space during an application execution.

An FPGA must have a reconfiguration time very fast to be utilized as a RTR device and the ability of maintaining some of its internal states, when reconfigured.

Through the use of local RTR, it is possible to build a reconfigurable switch, that can be used as an active node.

The architecture of this new switch will be explained in the next section.

The Reconfigurable Switch

A switch is a set of input and output ports that interconnects users, other switches or other network elements. Its architecture can be seen in the figure 1. According to [2] it can be divided in SM (System Management), CAC (Connection Admission Control), IM (Input Modules), OM (Output Modules), and CSF (Cell Switch Fabric).

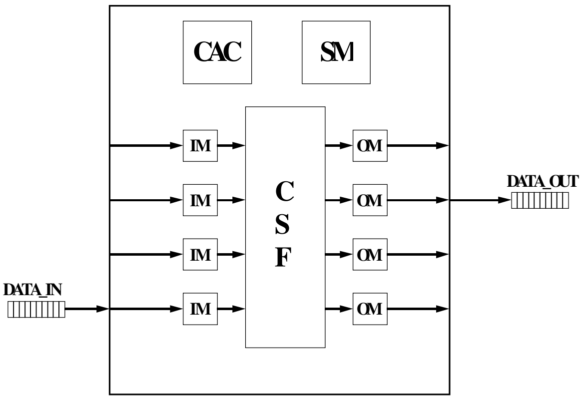

Figure 1 - A Switch

The System Management (SM) module is responsible for performing all the management and traffic control functions to ensure the correct and efficient operation of the switch. One can divide this function in six types:

- Fault management;
- Performance management;
- Configuration Management;
- Accounting management;
- Security management;
- Traffic management.

All these functions may be distributed among the other blocks of the switch. It depends on what kind of performance and the costs involved in order to achieve it.

The Connection Admission Control (CAC) module processes and interprets signaling information and decides about the connection admission or rejection. This block is responsible for the interaction with other users and other switches through signaling, and resource allocation decisions for requested connection.

The Input Modules (IM) treat the incoming cells and prepare them for routing trough the CSF (explained later). It is possible, also, to have some functions of CAC and/or SM integrated on the IMs. This integration yields a better performance, but it is more expensive to implement.

The Output Modules (OM) receives cells from the CSF and prepares them for physical transmission on the output links. This block is simpler than the IM and may have some functions of CAC and/or SM integrated on it.

The Cell Switch Fabric (CSF) routes the user data cells from input to output ports and possibly route signaling and management cells between others functional blocks in the switch. Besides this main task (routing), the CSF is responsible for others functions, such as:

- Cell buffering;
- Traffic concentration and multiplexing;
- Redundancy for fault tolerance;
- Multicasting or broadcasting;
- Cell scheduling based on delay priorities;
- Selective cell discarding based on loss priorities;

To transform the switch explained above in a reconfigurable one [6] it is necessary to include a Reconfigurable Control Block, a Configuration Memory block and make the CSF programmable (see figure 2).

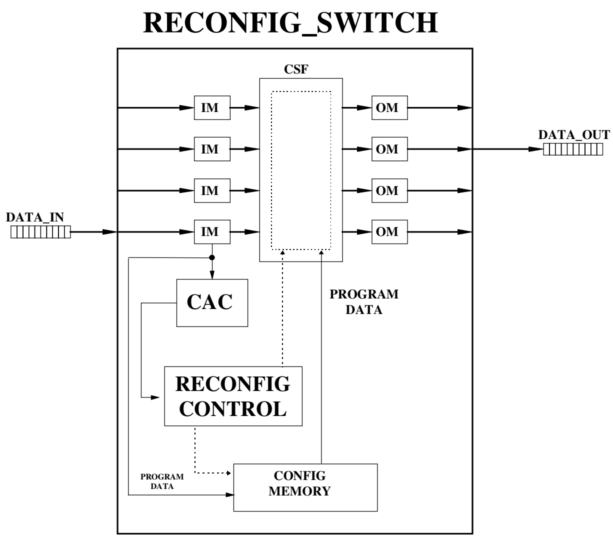

Figure 2 - Reconfigurable Switch

The CAC block will monitor the data entering the switch by each one of its input ports. When some special codes encapsulated in the data appear, the CAC sends them to the RECONFIG CONTROL.

In order to implement the reconfigurable switch, the project ATLAS I [11] will be used as a starting point. It is a single-chip ATM switch with credit flow control, described in Verilog. The source code will be modified to introduce the reconfigurability.

The main blocks, explained in [10], that will be redesigned are:

- RT'ng/Tran – Cell Header Processing, Routing and VP/VC Translator. This block will process the special codes encapsulated in the messages;
- Ctrl. Mgt. – Chip Control and Management. This block will have to reconfigure some portions of the chip, according to the special codes processed by the previous block;

The I/O ports will be reduced to four, in order to implement the whole switch in a FPGA with high density, such as APEX[1], from ALTERA or Virtex [19], from Xilinx.

At the time of this writing (January 2000), the modified architecture of the switch is under evaluation. The Verilog code is being simulated and, after these simulations, it will be processed by an FPGA development tool.

Conclusion

With the use of reconfigurable logic it was shown that is possible to build an active network composed of switches capable of changing their features "on the fly".

The main advantages of these elements are:

- more flexibility in the implementation of new protocols;
- reduced time-to-market of a new development;
- facility to supply new services in the network.

There are other important aspects related to this area that can be exploited, such as:

- description of new reconfigurable schemes to implement the switch;
- simulation of new reconfigurable architectures;
- comparisons between performance versus reconfiguration time;

Some applications of the hardware approach described in this article include:

- the implementation of a flux control for ABR multipoint traffic [12];
- the implementation of some MPI Collective Operations [8].

References

[1] ALTERA, Inc.: APEX 20K Programmable Logic Device Family, ALTERA, Inc., 1999

[2] T. M. Chen, and S. S. Liu, ATM Switching Systems, 1995, Norwood, MA: Artech House, Inc, 1995.

[3] M. Gokhale, Bill Holmes. Andrew Kopser, Dick Kunze, Dan Lopresti, Sara Lucas, Ron Minnich, Peter Olsen, SPLASH: A Reconfigurable Linear Logic Array, January 1994

[4] D. T. Hoang. Searching genetic databases on Splash 2. In D. A. Buell and K. L. Pocek, editors, Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, pages 185-191, Napa, CA, April 1993.

[5] J.D. Hadley, B. L. Hutchings, Designing a partially reconfigured system, in Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing, John Schewel, Editor, Proc. SPIE 2607, pp. 210-220 (1995).

[6] Edson L. Horta, Carlos A. P. S. Martins, Sérgio T. Kofuji, RECATS: A Reconfigurable ATM Switch Proposal, em Workshop em Arquiteturas Reconfiguráveis-IX Simpósio Brasileiro de Arquiteturas de Computadores-Processamento de Alto Desempenho, Campos do Jordão - Brasil, 1997

[7] John Hartman, Udi Manber, Larry Peterson and Todd Proebsting, Liquid Software: A New Paradigm for Networked Systems, Technical Report TR96-11, Dept of Computer Science, Univ. of Arizona, 1996

[8] Edson L. Horta, Sérgio Takeo Kofuji, Using a reconfigurable switch to improve MPI performance, 4th International Conference on Computer Science and Informatics, Research Triangle Park, NC, USA, October 1998

[9] B. L. Hutchings, M.J. Wirthlin. Implementation Approaches for Reconfigurable Logic Applications, In 5th International Workshop on Field Programmable Logic and Applications, pp 419-428, August 1995, Oxford, England.

[10] G. Kornaros, D. Pnevmatikatos, P. Vatsolaki, G. Kalokerinos, C. Xanthaki, D. Mavroidis, D. Serpanos, M. Katevenis Implementation of ATLAS I: a Single-Chip ATM Switch with Backpressure, in Proc. IEEE Hot Interconnects VI Symposium, Stanford, California, USA, Aug. 1998.

[11] M. Katevenis, D. Serpanos, P. Vatsolaki, ATLAS I: A General-Purpose, Single-Chip ATM Switch with Credit-Based Flow Control, in Proc. Hot Interconnects IV Symposium, Stanford Univ., CA, USA, Aug. 1996, pp. 63-73.

[12] Joo Yean Oh, Um algoritmo eficiente de controle de fluxo para tráfego ABR multiponto. São Paulo, 1998 65p. Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Eletrônica.

[13] Bertin Patrice, Roncin Didier, and Vuillemin Jean, Programmable Active Memories: a Performance Assessment, Research on Integrated Systems: Proceedings of the 1993 Symposium, G. Borriello and C. Ebeling, editors, pp. 88-102, MIT Press, 1993.

[14] Bertin Patrice and Touati Hervé. PAM Programming Environments: Practice and Experience, IEEE Workshop on FPGAs for Custom Computing Machines, D. A. Buell and K. L. Pocek, editors, Napa, CA, USA, 1994.

[15] D. Ross, O. Vellacott, and Turner. An FPGA-based hardware accelerator for image processing. In W. Moore and W. Luk, editors, More FPGAs: Proceedings of the 1993 International workshop on field-programmable logic and applications, pages 299-306, Oxford, England

[16] Jonathan Smith, David Farber, Carl A. Gunter, Scott Nettles, Mark Segal, William D. Sincoskie, David Feldmeier, and Scott Alexander SwitchWare: Towards a 21st Century Network Infrastructure, Jan. 1996

[17] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, A Survey of Active Network Research, IEEE Communications Magazine, Vol. 35, No. 1, pp80-86. January 1997.

[18] D. L. Tennenhouse, D. J. Wetherall, Towards an Active Network Architecture. Computer Communication Review, Vol. 26, No. 2, April 1996.

[19] XILINX Inc. : Virtex 2.5 Field Programmable Gate Arrays, Xilinx, Inc., 1999