A Switch Architecture for Real-Time Multimedia Communications

Gerard J.M. Smit, Paul J.M. Havinga
University of Twente, dept. Computer Science
P.O. Box 217, 7500 AE Enschede, the Netherlands
e-mail: {smit, havinga}@cs.utwente.nl

Abstract

In this paper we present a switch that can be used to transfer multimedia type of traffic. The switch provides a guaranteed throughput and a bounded latency. We focus on the design of a prototype Switching Element using the new technology opportunities being offered today. The architecture meets the multimedia requirements but still has a low complexity and needs a minimum amount of hardware.

A main item of this paper will be the background of the architectural design decisions made. These include the interconnection topology, buffer organization, routing and scheduling.

The implementation of the switching fabric with FPGAs, allows us to experiment with switching mode, routing strategy and scheduling policy in a multimedia environment. The switching elements are interconnected in a Kautz topology. Kautz graphs have interesting properties such as: a small diameter, the degree is independent of the network size, the network is fault-tolerant and has a simple routing algorithm.

1. Introduction

In the past decade we have seen a revolution in VLSI technology. The transmission and switching technologies have rapidly advanced to higher speeds; it is now technically feasible to build networks with gigabit throughput. The Field Programmable Gate Arrays (FPGA) technology allows gate arrays to be reprogrammed an unlimited number of times, and also has on-chip static memory (the X4000 family of Xilinx). Using programmable elements gives us the opertunity to learn from experiences on prototypes and to adapt the architecture.

We are currently building a prototype network using these off-the-shelf technologies. Our goal is to build a local area network that supports multimedia applications. These applications require not just high transmission speeds, but also small end-to-end latency with little variation, a guaranteed throughput, graceful degradation under heavy workloads, and performance that is both fair and predictable. Low-latency services are necessary for voice and video transfer, process control, remote sensing etc. Data for these services is usually worthless if it does not arrive in time. A video sequence, for instance, must be retrieved at a high and constant rate; frames retrieved too late are no longer useful and can be ignored. An additional problem is the synchronization of different media. These applications require real-time *and* reliable communications where certain strict deadlines must be met.

The bandwidth of many existing networks is by far not enough for distributed multimedia applications. Their throughput and latency is becoming a bottleneck in demanding real-time applications. One reason is that these networks use a single shared path for their communication.

Point-to-point networks are organized as star shaped networks, in which all stations are connected by dedicated links to a central switching fabric. A connection between two stations is established through the switching fabric. The main advantages [9] of point-to-point networks are that they offer an aggregate network bandwidth that can be much larger than the throughput of a single link; and have a high availability by allowing multiple paths in the switching fabric. The interfaces in the stations can be simple and low cost (most of the complexities are contained within the switching fabric) and the design is relatively independent of the technology of the physical layer of the links. If there are multiple paths in the switching fabric, the connections will experience a higher availability.

In this paper[1] we present a network that can be used to transfer multimedia traffic with the above requirements. We focus on the design of a prototype switching fabric using the new technology opportunities being offered today. The

1. Recent results obtained about the comminication infrastructure in the Pegasus research project [7] are presented in this paper. The Pegasus Project is a project of the Universities of Twente and Cambridge, supported by the European Communities Esprit Programme through BRA project 6586. It is partially supported by the Cambridge Olivetti Research Laboratory and a grant from Digital Equipment Corporation.

architecture we are proposing meets the multimedia requirements but still has a low complexity and a small number of chips. A minimal cost is required as we expect that high speed networking will not only be used for the high-end workstations, but also for stand-alone network devices such as cameras and displays. The switching fabric must be able to adapt easily to other requirements and situations. Once prototyping has been completed an implementation of such an architecture in custom VLSI easily scales to larger switches and faster links.

2 Architectural design issues

A wide range of communication types and primitives should be supported in a distributed multimedia system. In these systems workstations are connected to a various number of services such as: high-performance file servers, communication servers (gateways to Wide-Area-Networks), servers for manipulation of voice video and animation, etc. Our network typically provides communication facilities within a building or campus. In each building or floor there are one or more switching fabrics (often called hubs). We have chosen a *star* shaped network, in which all stations are connected by dedicated links to a central switching fabric. A connection between two stations is established through the switching fabric.

Each switching fabric can connect about 100 workstations that are located in a distance of up to 250 meters from the fabric. The switching fabrics are interconnected through high speed links. All stations communicate via ATM only. In this paper we mainly discuss the communication within a switching fabric.

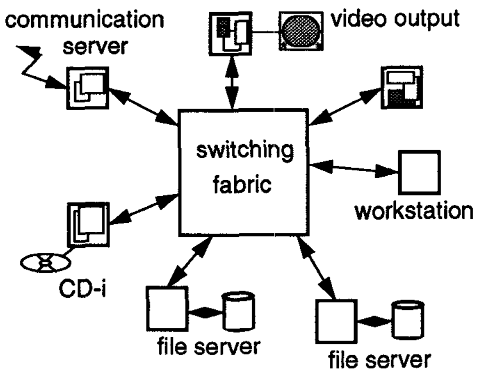

Figure 1: Global architecture of a distributed multimedia system.

In this section we will give the background of the architectural design decisions made. These include the interconnection topology, buffer organization, routing and scheduling.

2.1 Interconnection topology

This section describes the network topology used to interconnect the switching elements inside the switching fabric. The interconnection topology must be able to support the stringent demands of multimedia communications. The topology we are seeking for is a multistage network of very simple switching elements for several reasons. First of all the fabric must be able to connect a various number of nodes (up to 100 nodes). Therefore it is by far not possible to use a shared medium as interconnect. To provide a low latency the topology must have a low diameter. Furthermore, since we want to build networks of arbitrarily large size using (VLSI) components as nodes, we need to have a *fixed* number of connections per node. Therefore the degree of the graph should be fixed and independent of the number of nodes. A crossbar for example requires the number of connections to increase with the number of nodes.

A multistage network must be able to admit self-routing. This means that each switching element is able to switch (route) a received message autonomously by only using a self-routing label preceding the message. Messages placed on the network are automatically routed and delivered to the output destination. Since multistage networks suffer from internal blocking, special care has to be taken to provide a guaranteed performance. Misrouting of a cell to avoid blocking should be handled with care, since all cells must be kept in order.

The connectivity should be high to obtain a high availability. If there are multiple disjoint paths possible in the graph, the connections will perceive a higher availability and the performance degradation due to increased routing distances resulting from faults is low.

In our prototype switch we are using a Kautz topology because of its valuable properties [6]. For detailed definition and properties of Kautz networks we refer to [11]. The main advantages are:

- *Fixed degree:* only one type of switching element needs to be designed. Using this switch an arbitrary large network can be constructed by choosing an appropriate diameter k.
- *Low diameter and high connectivity:* due to these facts we were able to use a simple, circuit switching type of transfer mode. In our prototype network we use a directed Kautz network with diameter k=4 and degree 3. On this network with 108 nodes on average only 3 hops are required to reach a destination. In combination with virtual channels it allows us to apply bandwidth reservation for real-time traffic.
- *Fault tolerant routing:* directed Kautz graphs support a simple algorithm for generating the shortest route

(length ≤ k). It can be extended to generate a route (of length ≤ $k+2$) that survives at least $d-1$ node or link faults. This property can also be used to avoid congested nodes or links [5], [10].

- The network admits *self routing* of messages. In a Kautz digraph a straightforward generic route of length k can be found by simple concatenation of the source address and destination address. In the example directed graph[1] with d=3 and k=2 (figure 2) we find a route R = <1013>

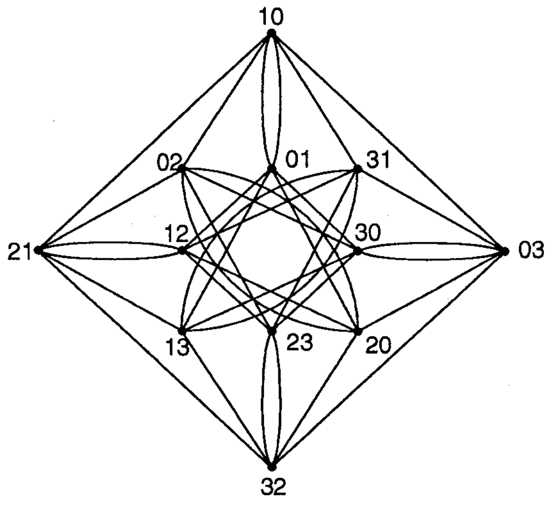

Figure 2: Example of a Kautz graph (K(3,2)).

from (10) to (13) via node (01). This route has length 2 (= k). In addition to that there are alternative routes: <10213> and <10313> of length 3.

- *Embedding of trees:* Kautz graphs can embed standard computation graphs such as: a ring and a d-ary tree. This last property can be used to implement multicast and broadcast efficiently in the network.

2.2 Buffer organization

One of the main decisions when designing a switch is how to organize buffering inside switches. Even in an internally non-blocking switch at some points buffers are required because a number of paths share common links. There exist many alternatives for organizing these buffers. The buffers may be placed at the switch inputs, the output, or they might be shared.

[8] gives a detailed discussion about the advantages and drawbacks of some typical multistage configurations. Generally, pure input queueing is easier to implement than pure output or sharing queueing in switch architectures. The latter architectures, however, provide optimal delay and throughput performance, whereas the former suffer severe throughput degradation due to the Head Of Line (HOL) blocking. If the first message in the queue is blocked, the physical channel is idle, because no other message is able to cross the switch to the required output link.

From a performance point of view a better approach is using parallel buffers. A switch may organise the input buffers by associating them with virtual channels. *Virtual networks*, implementing a number of virtual channels on one physical link, was first introduced as a technique to avoid deadlocks in networks. Dally [3] showed that virtual networks increase the connectivity of networks and have performance advantages as well.

The buffers of each virtual channel are allocated independently of the buffers of the other virtual channels. This added allocation flexibility increases channel utilization and thus throughput. A blocked message, even one that extends through several links (e.g. in case of worm-hole routing), blocks only one virtual channel per link and can

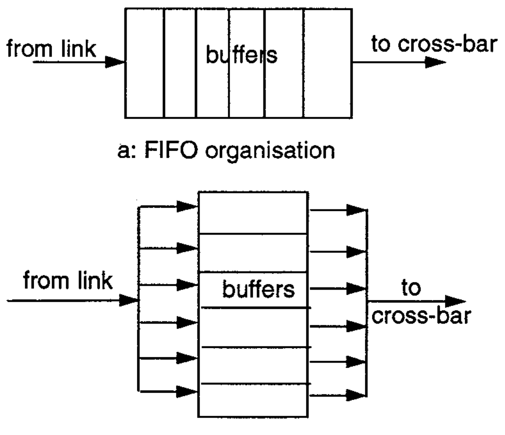

Figure 3: buffer organizations

be overtaken by messages in other virtual channels. The size of the buffer depends on the routing mechanism used.

2.3 Routing

In order to provide an arbitrary and fully dynamic connectivity in a static network of nodes, routing mechanisms must be implemented, which provide the propagation of data from node to node, based on addresses contained within a packet. This is an important issue in parallel and distributed systems because it is one of the key components that determines communication latency. Such a data routing mechanism must satisfy a number of functional require-

1. In a directed graph there is an arc from a vertex x to a vertex y if and only if the last $k-1$ letters of x are the same as the first $k-1$ letters of y. In other words the vertex $(x_1,...,x_k)$ is neighbor to d vertices $(x_2,...,x_k,z)$, where z is any letter from the alphabet different from x_k. So for example in figure 2 there is an arc from (10) to (03).

ments:

- The routing protocol must be *deadlock* free; no message may be stalled permanently.
- The routing resources (message buffers, link bandwidth) must be allocated in a *fair* way.
- Freedom of *live lock* situations; every message injected into the network must eventually be delivered.
- Freedom of *starvation*; every sender must be able to inject a message into the network.

Due to the little available buffer space inside the switching element a store-and-forward routing mechanisms seems not appropriate. Moreover, since we intend to use a parallel input queueing this mechanism would impose too stringent requirements on buffer space in each node since it must be able to store a number of cell packets.

Instead of storing a packet completely in a node and then transmitting it to the next node, *wormhole routing* operates by advancing the head of packet directly from incoming to outgoing channels. Only a few bytes (called flits) are buffered at each node. A *flit* is the smallest unit of information that a queue or a channel can accept or refuse. As soon as a node examines the header flit of a message, it selects the next channel on the route and begins forwarding flits down that channel. As flits are forwarded, the message spreads out across the links between the source and the destination. Wormhole routing requires a separate routing unit within each node.

There exist a number of deadlock free routing algorithms. Most algorithms are based on preventing cycles in the dependency graph [1]. However, this approach restricts the routing of packets: it reduces the number of paths that a packet may take. The method of nosy worms [15] is another deadlock free approach. In this method, messages start off by being buffered at their sending node. Each message attempts to establish a connection with the destination node by forming an unbroken path across intermittent nodes. If a block or a failure is encountered along the route the message gives up by recoiling back to the sender, thus avoiding deadlock. If there are alternative routes another node disjoint route can be tried. This leads to an adaptive routing mechanism that routes around congestions and faults. Once a connection is made the entire message is transferred from source to destination, where it is again buffered before it is off-loaded to the receiving system. In fact this is a kind of circuit switching.

Circuit switched routing is often used in combination with virtual channels. Virtual channels can be used to improve the performance, avoid communication deadlocks or to realize real-time communication.

Applications generating real-time data are able to reserve a circuit switched connection from source to destination. Less demanding applications, like file transfers, may release the connection after a message transfer is completed. In this way the the buffers are released and can be used for other connections.

A performance analysis of the routing mechanisms and transfer modes used in the prototype Rattlesnake switch is presented in [12].

2.4 Scheduling

In all switch architectures with buffering, a scheduling mechanism determines which buffer may transmit its data in a given time slot. The scheduler must be fair, and must not incur starvation. Basically there are two types of scheduling: static and dynamic scheduling [2]. In *static scheduling* the bandwidth allocation is stored in scheduling tables, that describe what communication takes place in each time slot. The main advantage of static scheduling is that is has a bounded latency. Therefore it can be used for real-time traffic. The disadvantage is that it is hard to set up the scheduling tables in all nodes on the path, and that for each reservation the scheduling tables have to be reorganized.

In *dynamic scheduling*, the bandwidth allocation is done distributed on the nodes along the path. A scheduling algorithm determines which input is connected to an output at a given time. The scheduling mechanism can be distributed over the switch. At first each input sends requests to the required output. Each output returns an acknowledgement indicating whether the message is blocked or can be forwarded. An input that receives a blocked status can try to transmit another message. Because there is only a limited amount of time to schedule it is not possible to achieve the highest possible link utilization.

Furthermore it is hard to make a fair scheduling with no starvation. Fair scheduling requires that each input flow (possibly from several input links) must be allocated an equal part of the output link bandwidth.

In the prototype switch, each flow (virtual channel) has a dedicated buffer, so it is possible to have a *scheduling table* associated with each output. For every active flow of the outgoing link there is an entry in this table. The scheduler uses this table to schedule the buffers of the input, such that each flow gets a fair share of the bandwidth.

3 The Rattlesnake Switch

In this section we will give some implementation details about the prototype Rattlesnake Switch. The switch consists of a number of *switching elements* interconnected via bi-directional links. A typical switching element has 3 input and 3 output links. Although the switching elements can be configurated in several network topologies such as: torus, mesh and deBruijn networks, we advocate a Kautz network topology [10] because of its valuable properties. Particularly, Kautz graphs have a small diameter, a fixed and a small degree (see section 2.1).

Each switching element is connected to a port controller (*Snake Control*). The port controller is connected via point-to-point links (> 100Mbit/s) to a (work)station or a server. It has local memory in which the ATM cells coming from and going to the serial link are buffered. The Snake Control takes care of the ATM adaptation and performs an advanced priority scheme. It is responsible for the implementation of the transfer mode and use of the Switching Elements [13]. It uses a Hybrid Time Division Multiplexing transfer mode [4] in which time is divided into frames. Each frame has a fixed maximum number of slots. Part of these slots are assigned to real-time services, and the rest to non real-time services. The real-time messages as well as the non real-time message have to be split up into small fragments with the size of a flit. These flits can be transmitted in the allocated slots; i.e. real-time flits in real-time slots and other flits in non real-time slots. In addition to that the non real-time traffic is allowed to seize any unused real-time slot.

A connection between two arbitrary stations is made via two or more switching elements in the switching fabric. A message generated by a source station travels through these switching elements to reach a destination station. The Switching elements forward messages from an input link to an output link, as directed by the destination address in each message header.

3.1 The Switching Element

In this section we present the switch architecture in more detail. A switch has 3 input sections and 3 output sections. The input and output sections are connected via a crossbar. The claim unit is the global administration controller that assigns virtual channels to connections.

The Snake Controller splits up ATM cells coming from the stations into flits and transmits them to its Switching Element. A *flit* consists in our case of 32 bits data and 8 bits identification. The identification consists of a 4 bits virtual channel number and 4 bits type.

We use a physical channel consisting of an 8-bit forward path and a 1-bit reverse path. Each flit is sent (internally and externally) over the forward path in five phases: one flit identification phase and four data phases. Simultaneously a 4-bit status of the current virtual channel is transmitted over the reverse path.

Figure 4 shows the internal organisation of an input link of a switching element. Each inlink contains 16 *parallel flit buffers*. These buffers are used to implement the 16 virtual

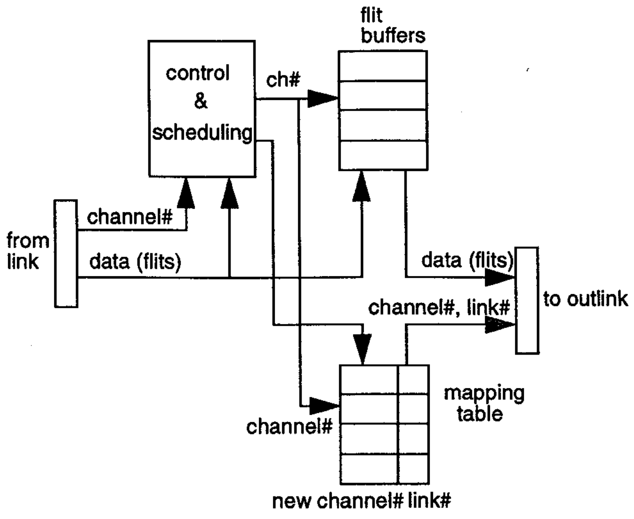

Figure 4: Structure of the input part of a link.

channels of a link. When establishing a connection from source to destination, each switch assigns a flit buffer (and thus a virtual channel) for this circuit. The routing information (i.e. the selected outlink) is stored in *mapping tables* associated with each inlink. Each entry in the table also contains the new virtual channel number that must be used for the connection with the next SE. This channel number is assigned by the claimer unit during the connection setup phase (see section 3.3). Each entry in the mapping table has only local significance and identifies the local virtual channel translation.

While a flit is being transferred over the link, information in the reverse direction is returned to indicate the status of the buffers of the receiving node (since they might not be emptied yet). Associated with each flit buffer is a *status buffer* of 4 bits. This status is transferred via the reverse path during the reception of a flit. The outlink section of the previous SE, will transfer this status to its inlink where it will be stored in the status buffer. With this mechanism it is possible to return small status messages not only between Switching Elements, but also from the receiving Snake Control.

3.2 Scheduling

In the first prototype we use a simple dynamic scheduler. This scheduler does not use the fair queueing mechanism as described in section 2.4. The input and output links negotiate about the scheduling of the next data transfer action while a data transfer is taking place.

Each inlink has its own scheduler that selects an input flit that will be forwarded to an output link. Associated with each flit buffer there is a register indicating whether the flit actually has data. The scheduler uses a round robin mechanism to select a flit buffer that contains data. It sends a request to the required output. If the output receives requests, it chooses one (round robin) to grant and sends an acknowledgement to the inlink.

It is possible that in one time slot more than one inlinks choose to send a flit to the same outlink section, even though there are other flits available for other outlinks. This drawback is circumvented because during the time of one flit transfer 4 scheduling actions can be performed. So all outgoing links can be requested.

Note that these iterations are only used to fill the gap of unused outputs. The inlink scheduler may only change the priority (the round robin scheduling) of a flit buffer during one flit cycle. Although this mechanism is not fair, it does not incur starvation. Every virtual channel of an input will receive a grant from an output.

3.3 Connection setup

The communication architecture should provide a guaranteed throughput and a bounded latency. Therefore it must be able to establish *hard real-time* connections (i.e. bandwidth reservation using real-time virtual channels).

We use a simple deadlock free routing algorithm (called *nosy worms* algorithm) similar to the one described in section 2.3. The algorithm attempts to establish a connection with the destination by allocating an unbroken path across intermittent nodes. If a failure or a blocking condition is encountered along the route the set-up message gives up by recoiling back to the sender; thus avoiding deadlock. Another route can be tried[1].

To find a path from inlet to outlet of the switching fabric we take advantage of the self-routing property of Kautz networks. The route to be followed is contained in a single string of digits, called the routing tag. The consecutive digits of the routing tag are interpreted stage-by-stage. Each Switching Element reads one control flit, sets up the mapping table, and sends the rest of the message to the required outlink. When an input section receives the first *claim* flit of a connection, it delivers this flit at the claim unit. The Claim Unit will assign a free virtual channel and update the mapping table of the inlink.

The path of a logical connection is set for the duration of a connection. This means that messages of a certain connection will always follow the same path through the switching fabric. When the input section receives a *release* flit, it notifies the Claim Unit, which will update the mapping tables. The inlink will transmit the release flit to the outlink to notify the other switches down stream.

When the Claim Unit could not find a free virtual channel for a certain link, it can reply with a 'route error' status. The input section stores this error code in the status buffer. On reception of the following flit of that channel, this error code is returned to signal that the connection request failed. This status ripples back to the source, which can decide to try another route.

4 Implementation

In our prototype the switch is implemented with Field Programmable Gate Array (FPGA) technology [16]. Xilinx's FPGA architecture is similar to other gate arrays, with an interior matrix of configurable logic blocks and a surrounding ring of I/O interface blocks.

This technology allows gate arrays to be reprogrammed an unlimited number of times. Therefore they are suited for designs in which the functions of the hardware needs adaptations in order to meet changing application requirements. The main reason for using FPGAs is *flexibility*. We use FPGAs as dynamic programmable units, whose function can be changed under program control [11].

We had to make a number of *a priori design decisions*, decisions that cannot be changed because they are fixed for instance due to circuit layout or interconnections structure. We have made a prototype printed circuit board containing 2 Xilinx PG4010 FPGAs to implement the Switching Element and the Snake Control, and two memories of 2 Mbytes connected to the Snake Control. A switch has 3 input and 3 output links with each 9 wires. These wires can be selected uni-directional or bi-directional. The interface with the node computers is provided using TAXI links [17]. (We use the Fairile transmission boards designed by University of Cambridge.)

This prototype board allows us to do experiments with and performance measurements of several transmission modes, bandwidth reservation, scheduling, routing for non real-time traffic, link protocols, multicast etc. We use VHDL as a design tool and a VHDL synthesizer from VIEWlogic [14] to generate the configuration code for the Xilinx chips. A limitation of the current FPGAs is that they have only a limited amount of on-chip memory. However, this does not affect wormhole routing, where only a few flits need to be stored in the switch. Wormhole routing can be implemented efficiently in FPGAs. If the switch is used in store-and-forward fashion, the off-chip memory is used. The available FPGA memory is sufficient to implement wormhole routing with a limited number of virtual channels.

1. The probability of a connection message reaching its destination depends primarily on the chances of finding an unbroken chain of channels. [12] gives simulation results showing that these chances are improved by using a low diameter network (e.g. Kautz), using alternative routes, and by using virtual channels.

5 Conclusion

In this paper we have presented the architecture of a high-performance, low-latency network suitable for hard real-time multimedia applications. For real-time services we use a connection oriented protocol. A modified nosy worms protocol is used for setting up a connection for real-time services. Once a connection is established it guarantees a bounded latency. To increase the probability of finding a connection from source to destination we exploit the node disjoint routes and the low diameter of the interconnection network. Above that we use virtual channels in order to improve the performance and to avoid communication deadlocks.

The switches use input buffering. For each virtual channel there is a buffer in the switch. The buffers are organized in a parallel fashion to avoid head of line blocking.

The network of the switching fabric is built up from switching elements interconnected in a Kautz topology. Kautz graphs have interesting properties such as: small diameter, the degree is independent of the network size, the network is fault-tolerant and has a simple routing algorithm.

A prototype of a switching element has been implemented with a standard FPGA. The design of the switching fabric with FPGAs, allows us to experiment with switching mode, routing strategy and scheduling policy in a multimedia environment.

References

[1] Adamo, J.M.: "Minimal, adaptive and deadlock-free routing for multiprocessors", Laboratoire de LIP-IMAG, Ecole normale superieure de Lyon, France, June 1991.

[2] Anderson T.E., Owicki S.S., Saxe J.B., Tacker C.P.: "High Speed Switch Scheduling for Local Area Networks", Proc. ACM ASPLOS V, pp 98-110, 1992.

[3] Dally W.J.: "Virtual-channel Flow Control", Proc. 17th ACM/IEEE Symposium on Computer Architecture, 1990, pp 60-67.

[4] Hui J.Y.: "Switching and traffic theory for integrated broadband networks.", Dordrecht, The Netherlands: Kluwer Academic Publishers, 1990.

[5] Imase M., Soneoka T., Okada K.: "A fault-tolerant processor interconnection network" (original in Japanese); translated in Systems and Computers in Japan, vol 17, no 8 pp 21-30, 1986.

[6] Kautz W.H.:"Bounds on directed (d,k) graphs. Theory of cellular logic networks and machines", AFCRL-68-0668 Final report, pp 20-28, 1968.

[7] Leslie I.M., McAuley D., Mullender S.J.: "Pegasus - operating system support for distributed multimedia systems (pegasus paper 92-2)", Memoranda Informatica 92-76, Twente University dept. Computer Science, 1992.

[8] Pattavina A.: "Nonblocking Architectures for ATM Switching", IEEE Communications Magazine, February 1993, pp 38-48.

[9] Schroeder M.D., Birrell A.D. et al.;"Autonet: a High-speed, Self-configuring Local Area Network Using Point-to-point Links", Digital Systems Research Center, Palo Alto, CA, April 1990.

[10] Smit G.J.M., Havinga P.J.M., Jansen P.G.: "An algorithm for generating node disjoint routes in Kautz digraphs', Proceedings Fifth International Parallel Processing Symposium, Anaheim, CA, 1991.

[11] Smit G.J.M., Havinga P.J.M., Jansen P.G.: "On the design of a reconfigurable network switch", Microprocessing and Microprogramming 34, pp 59-62, 1992

[12] Smit G.J.M., Havinga P.J.M.: "Performance analysis of routing algorithms for the Rattlesnake network", Proceedings MASCOTS '93, January 1993, pp 155-160

[13] Tibboel W. H.: "Virtual Lines, a deadlock free and real-time buffer allocation mechanism for an ATM-network", Ms Thesis, University of Twente, department of Computer Science, July 1993.

[14] "VHDL-Designer User's Guide", VIEWlogic Systems Inc, April 1990.

[15] Whobrey D.: "A communications chip for multiprocessors", Proc. CONPAR 88 pp 464-473, 1988.

[16] "The Programmable Logic Data Book", Xilinx Inc., 1993.

[17] Advanced Micro Devices: "AM79168/AM79169-275 TAXIchip Transmitter/Receiver", Publication 15765, February 1992.