Washington University in St. Louis
Washington University Open Scholarship

All Computer Science and Engineering Research | Computer Science and Engineering

Report Number: WUCS-88-15

1988-06-30

Advanced Communications Systems

Jonathan S. Turner

The Advanced Communications Systems Project is concerned with new communication technologies that can support a wide range of different communication applications in the context of large public networks. Communication networks in common use today have been tailored to specific applications and while they perform their assigned functions well, they are difficult to adapt to new uses. There currently are no general purpose networks, rather there are telephone networks, low-speed data network and cable television networks. As new communication applications proliferate, it becomes clear that in the long term, a more flexible communication infrastructure will be needed. The Integrated Service... **Read complete abstract on page 2.**

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Recommended Citation
Turner, Jonathan S., "Advanced Communications Systems" Report Number: WUCS-88-15 (1988). *All Computer Science and Engineering Research.*
https://openscholarship.wustl.edu/cse_research/772

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

This technical report is available at Washington University Open Scholarship: https://openscholarship.wustl.edu/cse_research/772

Advanced Communications Systems

Jonathan S. Turner

Complete Abstract:

The Advanced Communications Systems Project is concerned with new communication technologies that can support a wide range of different communication applications in the context of large public networks. Communication networks in common use today have been tailored to specific applications and while they perform their assigned functions well, they are difficult to adapt to new uses. There currently are no general purpose networks, rather there are telephone networks, low-speed data network and cable television networks. As new communication applications proliferate, it becomes clear that in the long term, a more flexible communication infrastructure will be needed. The Integrated Service Digital Network concept provides a first step in that direction. We are concerned with the next generation of systems that will ultimately succeed ISDN. The main focus of the effort in the ACS project is a particular switching technology we call broadcast packet switching. The key attributes of this technology are (1) the ability to support connections of any data rate from a few bits per second to over 100 Mb/s, (2) the ability to support flexible multi-point connections suitable for entertainment video, LAN interconnection and voice/video teleconferencing, (3) the ability to efficiently support bursty information sources, (4) the ability to upgrade network performance incrementally as technology improves and (5) the separation of information transport functions from applications-dependent functions so as to provide maximum flexibility for future services.

ADVANCED COMMUNICATIONS SYSTEMS

Jonathan S. Turner

WUCS-88-15

July 1, 1987-June 30, 1988

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

with Mark A. Franklin, Guru Parulkar, Pierre Costa, Makoto Imase, Riccardo Melen, Akira Arutaki, Shahid Akhtar, Neil Barrett, Victor Griswold, Mark Hunter, Scott Johnson, Shabbir Khakoo, Tony Mazraani, George Robbert, James Sterbenz, Einir Valdimarsson and Bernard Waxman.

Acknowledgements

The Advanced Communications Systems Project operates within the Computer and Communications Research Center, an inter-departmental research laboratory in the School of Engineering and Applied Science at Washington University. The Center's research program seeks an appropriate balance between theoretical and practical issues and has attracted considerable interest world-wide. Program sponsors interact with the Center through exchange of information and personnel.

The ACS project began on January 1, 1986. Our current sponsors are

National Science Foundation (grant DCI 8600947)
Bell Communications Research
Bell Northern Research
Italtel SIT
Nippon Electric Corporation

We thank all our sponsors for their collaboration and support. Special thanks go to Gil Devey and Steve Wolff at NSF, Eric Nussbaum and Neil Haller of Bell Communications Research, Al Winterbauer and Dan Stevenson of BNR, Maurizio Dècina and Anna Robrock of Italtel and Akihiro Kitamura and Takehiko Yamaguchi of NEC. We also thank Washington University for providing an excellent environment in which to carry out this work, in particular Dean James McKelvey and CS Department Chairman Jerry Cox for all their support and encouragement.

Research Objectives

The Advanced Communications Systems Project is concerned with new communication technologies that can support a wide range of different communication applications in the context of of large public networks. Communication networks in common use today have been tailored to specific applications and while they perform their assigned functions well, they are difficult to adapt to new uses. There currently are no general purpose networks, rather there are telephone networks, low-speed data networks and cable television networks. As new communication applications proliferate, it becomes clear that in the long term, a more flexible communication infrastructure will be needed. The Integrated Services Digital Network concept provides a first step in that direction. We are concerned with the next generation of systems that will ultimately succeed ISDN.

The main focus of the effort in the ACS project is a particular switching technology we call broadcast packet switching. The key attributes of this technology are (1) the ability to support connections of any data rate from a few bits per second to over 100 Mb/s, (2) the ability to support flexible multi-point connections suitable for entertainment video, LAN interconnection and voice/video teleconferencing, (3) the ability to efficiently support bursty information sources, (4) the ability to upgrade network performance incrementally as technology improves and (5) the separation of information transport functions from application-dependent functions so as to provide maximum flexibility for future services.

Contents

List of Figures

1. Summary of Progress

The research program of the ACS project can be divided into four major areas: (1) switching system architecture and performance, (2) connection management, (3) network control problems, such as routing and buffer/bandwidth management and (4) design of communications applications in the context of broadcast packet networks. In the area of switching system architecture, we have continued work on design and implementation of a prototype switching system. In support of this prototyping effort we have also been developing tools to aid in the design of the custom integrated circuits to be used in the prototype. We have also been exploring architectures that support link speeds of a few Gb/s and have been studied the performance of a wide class of different packet switching fabrics. We have made substantial progress on connection management, including the design and implementation of preliminary software, and we are continuing to make steady progress in the area of network control. Our work on application design is currently limited to an initial study of the issues associated with packetized video, focussing especially on the effect of packet transport on the design of video coding methods. We have also begun work on a framework for internetworking of diverse communication subnets.

We have been active in publishing our results on broadcast packet switching. Papers have been presented at several conferences and revised versions have appeared or are scheduled to appear in leading journals; several theses have been completed or will be shortly; one patent has been awarded and an application for a patent on a hardware implementation of a buffer management system has recently been filed. (See Figures 1.1,1.2 for details.) Our work has generated a great deal of interest throughout the world, and appears to be having an influence on the research programs at several major industrial laboratories. We find this impact of our work particularly gratifying and expect to see it continue as our research program develops.

The following subsections summarize the progress we have made in several specific areas during the past year and outline our plans for the coming year. More detailed accounts of each of these topics appear in later sections.

Published Papers

"Fluid Flow Loading Analysis of Packet Switching Networks," by Jonathan Turner. *Proceedings of the International Teletraffic Congress*, June 1988. Also, submitted to *Computer Networks and ISDN Systems.*

"Distributed Protocols for Access Arbitration in Tree Structured Communication Channels," by Riccardo Melen and Jonathan Turner. *Proceedings of ICC 88*, June 1988. Also, submitted to *IEEE Transactions on Communications.*

"Design of a Broadcast Packet Switching Network," by Jonathan S. Turner, to appear in *IEEE Transactions on Communications*, June 1988.

"Broadcast Packet Switching Network," by Jonathan S. Turner, U.S. Patent #4,724,907, March 1988.

"Performance of a Broadcast Packet Switch," by Richard Bubenik and Jonathan Turner. *Proceedings of ICC 87*, pp. 1118–1122, 6/87. Also, to appear in *IEEE Transactions on Communications.*

"The Challenge of Multipoint Communication," by Jonathan S. Turner, *Proceedings of the ITC Seminar on Traffic Engineering for ISDN Design and Planning*, 5/87.

Invited Lectures

Digital Equipment Corporation, Littleton, MA (3/88)
Telenet Inc., Reston, VA (8/87)
Southwestern Bell Telephone, St. Louis, MO (8/87)

Tutorial on "Integrated Networks for Diverse Applications," at *Globecom 88* and UCLA Extension Short Course (2/88).

Program committee for *Computer Networking Symposium*, April 1988. Guest editor for special issue of *IEEE Journal on Selected Areas in Communications* on broadband packet communications

Filed patent application on buffer management system for multipoint packet networks (3/88).

Figure 1.1: Publications and Related Activities

"A Circuit Generator for Synchronous Streams Processors," by George Robbert, Washington University Computer Science Department, MS thesis, expected completion May 1988.

"Improved Search Algorithms for Video Codecs," by Shabbir Khakoo, Washington University Electrical Engineering Department, MS thesis, expected completion June 1988.

"Congestion Control in Fast Packet Networks," by Shahid Akhtar, Washington University Electrical Engineering Department, MS thesis, November 1987.

"Worst-case Performance of Rayward-Smith's Steiner Tree Heuristic," by Makoto Imase and Bernard Waxman, WUCS-88-13.

"Towards a Framework for High Speed Communication in a Heterogeneous Networking Environment," by Guru Parulkar and Jonathan Turner, WUCS-88-7.

"Buffer Management System," by Jonathan Turner, WUCS-88-6.

"Design of a VLSI Packet Switch Element," by James Sterbenz, WUCS-88-5.

"Probable Performance of Steiner Tree Algorithms," by Bernard Waxman, WUCS-88-4.

"Nonblocking Multirate Networks," by Riccardo Melen and Jonathan Turner, WUCS-88-2.

"Distributed Protocols for Access Arbitration in Tree Structured Communication Channels," by Riccardo Melen and Jonathan Turner, WUCS-87-17.

"Fluid Flow Loading Analysis of Packet Switching Networks," by Jonathan Turner, WUCS-87-16.

Figure 1.2: Theses and Technical Reports

Switch Architecture and Hardware Design

The most novel aspect of our research program is its focus on networks supporting flexible multipoint communication. Any switching system supporting multipoint communication must be able to connect any subset of its incoming channels to any subset of its outgoing channels. This is in contrast to point-to-point switching systems which need only connect input-output pairs.

Recently we have been considering some variants on the architectures that we have been focussing on up to now. One variant we've been considering are architectures in which, within a single switching fabric, packets belonging to a given virtual connection are constrained to follow the same path through the system. This approach has the advantage that packets belonging to a given virtual connection are guaranteed to reach the destination in the proper order. On the other hand, it has the drawback that the load cannot be distributed as evenly within

the switching system and the connection establishment process is somewhat more complicated. However, since several research groups have been exploring systems of this sort (and are constructing prototypes), it is important to understand these systems better. In addition, these systems turn out to generalize classical circuit switching systems in a natural way, leading to some interesting questions in performance analysis, which are discussed further below.

The other architectural variant we've considered is the design of switching systems that incorporate very wide data paths using a bit-sliced approach rather than the word-serial approach we have been using up to now. The bit-sliced architecture, while requiring more complicated control, allows more economical implementation of very large switches, and equally important allows much wider data paths than are feasible with the word-serial architectures, and hence supports higher link speeds. We have quantified the complexity advantages of bit-sliced structures, developed practical solutions to the control problems, and developed a fairly detailed paper design of a system that can support up to 65 thousand fiber optic links operating at speeds of 1.6 Gb/s using current CMOS technology with 100 Mb/s clock speeds and 32 bit wide data paths. It appears likely that with near term technology improvements, this approach can be extended to handle speeds up to about 10 Gb/s.

Work on a laboratory prototype of our switching system has been progressing well. Four integrated circuits implementing the packet switch element and broadcast translation circuits have recently been received back from fabrication and are being tested now. The one chip that has been completely tested is functionally correct, but the yield and speed of the chips received was disappointingly low. We have not yet determined the reasons for the poor performance; we expect to have a better understanding of this problem when the other chips have been tested.

At the same time, we are proceeding with design of several other chips. We have nearly completed a second version of the packet switch element design. This design incorporates eight bit wide data paths instead of four and we have made some architectural modifications in order to achieve higher clock speeds. We have also completed and fully simulated the design of a general purpose packet buffer, which will be used within the packet processor circuit. It also incorporates eight bit data paths and its layout has been accomplished with a circuit generation program written for that purpose. The use of a circuit generator allows us to quickly layout different versions of the packet buffer that vary in size, data path width, etc. Both of these chips (the new packet switch element and the packet buffer) will be submitted for fabrication before the end of May.

We have also completed an initial version of a more ambitious circuit generation program that can be used to quickly layout a large class of similar circuits

required within the packet processor, broadcast translation circuit and other common subsystems. This program allows the user to specify a component of a system in a functional notation similar to a conventional programming language. It then translates this specification into a circuit satisfying the specification. This program has been written by George Robbert as part of his masters thesis research [77] and is now being applied to layout several of the major components within the packet processor.

Performance of Packet Switching Fabrics

During the past year we have sought to extend our understanding of the performance of packet switching fabrics in general, with of course a special focus on the broadcast packet switch. Where our previous efforts have centered on simulation, our recent work has been primarily analytical. One important result has been the development of a systematic method for analyzing the effects of different traffic patterns on the loading of internal links within a packet switching fabric [94]. This method allows us to make statements about the worst-case loading of a variety of different switch fabrics and has led to several new results quantifying the effect of distribution stages on switch fabric performance. One result shows that a k stage routing network requires an additional $k-1$ distribution stages in order to avoid overloading of internal links. Another shows that just two distribution stages dramatically improve the worst-case performance of copy networks. Other results concern the effect of the number of ports per node on worst-case loading. Of special interest is the observation that the worst-case performance of copy networks deteriorates as the number of ports per node increases.

More recently we have generalized the classical theory of non-blocking networks to networks in which internal links can multiplex multiple connections, with each connection consuming an arbitrary fraction of the link's capacity (subject of course to the constraint that the sum of the connection loads is no more than the link's capacity) [60]. This is relevant to the design of large switching systems constructed from multiple switch modules. It is also important for switch fabrics which route all packets of a given connection along the same path, such as the systems under development at CSELT and Bell Telephone Manufacturing. Our results include an analysis of the amount of expansion required to make Clos, Cantor and Benes networks strictly non-blocking and rearrangeably non-blocking. We have also devised two novel rearrangeably nonblocking multicast networks within this context that are of considerable interest.

Our general approach has important theoretical implications as it greatly extends the classical theory and opens up a new avenue of investigation. It is also of considerable practical importance for certain classes of switching systems. One of

our next steps will be to consider analyses of blocking probability for networks of this type.

We have also done some simulation studies quantifying the likelihood of packets getting out of sequence when passing through a broadcast packet switch fabric. The results from these and planned further studies will be used to help in the design of mechanisms to recover proper sequencing.

Connection Management

Connection management refers to the collection of algorithms used to create and maintain multipoint connections in a broadcast packet network. A multipoint connection is intended to be a flexible mechanism that can support a wide variety of different applications. In our last progress report, we described our approach for specifying general multipoint connections as well as an architecture for a connection management software system.

In the past year, we have developed an initial set of protocols based on that architecture and have implemented them in the form of a software simulation that allows us to configure an arbitrary network, then set up and modify multipoint connections in that network. Our implementation of multipoint connections includes a general transaction mechanism for sequencing concurrent changes to a connection. The software was implemented first on a VAX 11/750 and has since been ported to a Sun workstation environment; in this new context, we are developing a graphical user interface to allow simpler specification of network configurations as well as better observation and control of connections in progress. We expect this graphical interface to form the basis of some graphical network management tools that we hope to develop in the coming year. The simulation has proved very useful in testing out our ideas on multipoint connection management protocols. Based on experience obtained to date, we are now refining these protocols to make them simpler and more consistent at both the network access level and the internal network level. In the coming year, we will implement these refinements and add the lower level software required to control the protoype system under development.

Multipoint Routing

The objective of the routing problem is to determine a set of network resources (primarily trunk bandwidth) sufficient to support communication among a specified set of users. Networks supporting multipoint communication channels of arbitrary bandwidth raise a variety of new issues for routing algorithms. We have primarily

studied the formulation of the routing problem in which we seek to identify a shortest subtree within the network that contains the endpoints to be joined by a given connection and has sufficient bandwidth for the connection [98]. This formulation leads to a Steiner tree problem, which is known to be NP-complete.

We have experimentally evaluated several approximation algorithms for the Steiner tree problem. In the previous report, we described results for the so-called *minimum-spanning tree heuristic* (MST) and a dynamic greedy algorithm. In the last year, we have also evaluated an algorithm proposed by Rayward-Smith; we have recently completed an analysis of this algorithm's worst-case performance and shown that it is no better than that of MST [42]. On the other hand, our analysis has suggested a variation on Rayward-Smith that we expect may have better worst-case performance; also, its average case performance (based on exerimental evaluation) is somewhat better than MST, although both are very good. We have also evaluated a weighted version of the greedy algorithm for the dynamic version of the routing problem and have found that for appropriate choice of the weights, this algorithm gives better average performance than the simple greedy algorithm; more significantly, the weighted algorithm is less subject to pathological behavior than the simple one.

Our research plans include continued evaluation of these algorithms and others. We have begun to study the average case performance of these algorithms analytically, in order to obtain greater insight into the factors limiting their performance [99].

Buffer and Bandwidth Management

A principal advantage of packet switched networks is their ability to dynamically allocate bandwidth to the users who need it at a particular instant. Since networks are subject to rapid statistical variations in demand, care must be taken to ensure acceptable performance under conditions of peak loading. In conventional packet networks, feedback-oriented *congestion control mechanisms* are used to detect local overloads and control their impact by slowing down the rate at which traffic enters the network. These techniques are impractical in very high speed networks which instead must rely on buffer and bandwidth management techniques that seek to prevent potentially damaging overloads from occurring in the first place.

A prerequisite to the development of effective buffer and bandwidth management methods is an understanding of the impact that bursty sources have on queueing in the network. We have developed a model that allows us to more accurately predict the effect of a collection of bursty sources on a finite queue [2]. Using this model, we can numerically determine the probability of packet loss when a collection of bursty sources shares a finite queue. These results are used to associate

an *effective bandwidth* for a given traffic source. The effective bandwidth is used in an operational network context for making routing and bandwidth allocation decisions. We have found that for sources with peak and average bandwidths of more than a few percent of link bandwidth, the effective bandwidth is quite sensitive to how bursty the connection is, but for lower values, it is fairly insensitive. One implication of these studies is that to achieve effective bandwidths substantially lower than peak for bursty, high speed sources we must either increase link speeds, buffer sizes or both.

For point-to-point connections, the network can monitor bandwidth use at the access point using what we call a *traffic valve* to prevent users from sending traffic at a higher rate than allowed for their connection. For multipoint connections with several transmitters, additional complications arise, since the control of entering traffic provided by the traffic valves at the edge of the network allows excess traffic on internal links of multipoint connections. We have designed a mechanism to control this kind of overload, which in effect allocates link buffer space in direct proportion to bandwidth allocations, and discards packets belonging to connections that exceed their share. This mechanism (which is the subject of a recent patent application [95]), in combination with others we have developed, allows a general solution to the problem of multipoint congestion control.

We have also considered a different approach to multipoint congestion control, in which the network actively controls the number of simultaneous transmitters in a multipoint connection, rather than limiting itself to the protection of its internal resources [59]. This kind of access arbitration could be more attractive to users, as it regulates the flow of traffic on a channel in a more consistent fashion. We have developed two general approaches to access arbitration, and several specific algorithms.

Packet Video

Packetized transport of video signals raises a variety of important issues that we are beginning to explore. One major effect of packet transport on video coding is to eliminate the constraint of a constant bandwidth channel that currently drives most work in video coding. A variety of techniques including transform coding, motion compensation, differential coding and adaptive quantization are currently used to reduce the required bandwidth for video signals. Existing systems use buffering and variable rate coding, with the objective of achieving minimum image distortion for a given, fixed channel bandwidth. In the context of packet transport, we can exchange the objective function we seek to optimize with the constraint. That is, we code to achieve minimum bandwidth subject to a given constraint

on distortion. This approach allows the bandwidth to vary across a wide range, achieving low average bandwidths and high picture quality.

Packetized transport also raises the issue of picture quality in the presence of packet loss. Common video coding methods rely heavily on state information that can become inconsistent when data is lost. The impact of lost packets can be reduced by interpolation schemes, in which a given block of information is split across multiple packets, allowing partial recovery of lost information. We expect that the use of such methods in combination with low rate transmission of complete state information can maintain high picture quality in the face of substantial packet loss rates and we are studying such methods to assess their potential.

Historically, video coding methods have been used primarily to produce moderate quality video for conference applications. With high speed packet networks it may also be advantageous to apply video coding methods to very high resolution signals; the objective becomes not bandwidth reduction but higher resolution. In the last year, we have studied hybrid coding algorithms employing transform coding, motion compensation and adaptive quantization. We have discovered that the commonly used search algorithms for motion compensation perform poorly in the presence of moderate to high motion. While they work adequately in video conferencing situations (which typically involve very little motion), they do poorly in more general contexts. We have developed a new class of *signature-based* search algorithms, which compute a concise signature for each position in the search space and match the current sub-block against each signature. We have evaluated one set of algorithms in this class and have found it increases the effective compression by a factor of three or four during rapid motion [50].

High Speed Internetworking

In our earlier work, we have concentrated on high speed networking in the context of a homogeneous environment. This is also typical of the approach taken by other groups working on high speed communication systems, but is in some ways unrealistic as it fails to explicitly take into account the diversity of existing and future networks, and the resulting need for inter-operation among separately administered and/or technologically dissimilar networks. In the last six months we have begun work on a framework for allowing diverse networks to inter-operate, while supporting both very high speed applications and multipoint communication [67]. This framework follows the general approach to interworking adopted in the ARPA internet protocols, but extends it in several respects. First it adds a connection-oriented transport service at the internet level, that can support applications with demanding performance requirements. Second it includes a more general addressing scheme, to allow interworking among diverse subnets. Third,

it provides a framework for parametric description of subnet capabilities and connection requirements, allowing the routing of connections through subnets with appropriate capabilities in an application-independent fashion.

A connection-oriented transport service is important for several reasons. Perhaps the most obvious is performance. Connection-oriented systems separate the more complex control operations from data transfer, allowing simple and fast hardware implementations of the data transfer. Connection-oriented networks are also attractive because they allow the network to make explicit resource allocation decisions when connections are established, and this in turn makes it possible to offer far more predictable performance than in connectionless networks. Finally, connection-oriented networks offer more generally useful methods of multipoint communication than are possible in truly connectionless networks.

In our work we envision ineroperation among a much wider class of networks than envisioned by the current internet model. In particular, we would like to support inter-operation between high speed packet networks, the current ARPA internet, X.25 networks and the public telephone network. Addressing is a key issue in allowing this level of diversity. We have proposed an addressing scheme that would accommodate such diversity without requiring that the individual subnetworks abandon their native addressing mechanisms.

Given the variety of capabilities of the subnetworks included in an extended internet, we feel it is essential that the internet protocol include mechanisms for describing the capabilities of subnetworks, so that routing decisions can be guided by this information. For example, when selecting a route for a connection requiring a bandwidth of 1 Mb/s it is essential that the route not traverse subnetworks incapable of supporting that bandwidth. Similarly connections requiring low packet loss rates should not be routed through subnets that lose packets frequently.

Administrivia

In the past year, our research team has grown by about 20%. Perhaps the most important addition has been that of a new faculty member, Guru Parulkar who joined the Computer Science Department in September 1987, after completing his PhD at the University of Delaware. Dr. Parulkar's thesis research focussed on the design and analysis of highly reliable local area networks based on flooding protocols. We expect him to be an important collaborator for the ACS project. We now have three faculty members involved in the project, one full-time staff person, one visiting research associate and ten graduate students. Additional faculty are being recruited for next fall in both the Computer Science and Electrical Engineering departments.

Our funding picture is fairly healthy. In addition to the support we receive from the National Science Foundation, we have funding from four industrial sponsors, the newest being Bell Northern Research. Currently, our funding is evenly divided between NSF and the industrial sponsors. In addition to the direct grant support, NSF provides access to MOSIS, their silicon fabrication service which we are using heavily in our prototyping effort. In the last year we have benefitted from a change in the policy of the Washington University School of Engineering and Applied Science; the school now pays the tuition of graduate students on research assistantships rather than requiring the research grant to pay that portion. This change has allowed us to increase our graduate student stipends which have been low with respect to other schools with which we compete. We have also used some of the funds made available by this change to improve our base of computing equipment. While the project's funding situation is in fairly good shape at the moment, we anticipate that additional funding will be required if we are to achieve all our major goals. Our current NSF grant runs through June 1989. We are now starting to think about a new research proposal for submission this fall that would allow us to continue our work beyond that point.

As mentioned above, the project supports three professors, Jonathan S. Turner, Guru Parulkar and Mark Franklin (part-time) as well as nine graduate research assistants (see Figure 1.3). We have one additional student (Akira Arutaki) who is supported by NEC. Shahid Akhtar graduated with an MS degree last fall and is now working at Bell Northern Research. Mark Hunter, Shabbir Khakoo and George Robbert will all be graduating with MS degrees this spring. Shabbir is leaving to work for AT&T Bell Labs and George will be going to work in the communications division at Hewlett-Packard. All four of these students have made strong contributions to the project. We have several students who have joined the project in the past year. Neil Barrett and Einir Valdimarsson began last summer and have been working primarily on design of integrated circuits for the prototype switching system we are constructing. Tony Mazraani joined the project in January and has been concentrating initially on work in the same area. We also have one undergraduate student, Scott Johnson who started last fall and has been working on design of a graphical interface for the connection management software system. We are planning to take on three new graduate students for the coming year, to replace those who are graduating.

Riccardo Melen, who visited us for one year from CSELT, the Italian national telecommunications laboratory, returned home at the end of December. His year here was a very productive one; he co-authored two papers with Dr. Turner [59,60] and co-invented a novel multipoint switching fabric. In March, Dr. Makoto Imase of NTT joined us for a one year visit. He has begun working with Buddy Waxman on the multipoint routing problem and their collaboration has already resulted in the solution of an important open problem in this area; a paper describing this work

Name	Degree (exp. graduation date)	Research Area
Akira Arutaki	DSc (5/90)	switching architectures
Neil Barrett	MS (5/89)	communication circuit design
Victor Griswold	DSc (1/90)	connection management
Mark Hunter	MS (5/88)	connection management
Shabbir Khakoo	MS (5/88)	packet video
Tony Mazraani	MS (12/89)	communication circuit design
George Robbert	MS (5/88)	CAD tools
James Sterbenz	DSc (1/90)	communication circuit design
Bernard Waxman	DSc (1/89)	multipoint routing
Einir Valdimarsson	MS (5/89)	communication circuit design

Figure 1.3: Current Graduate Students

will be submitted for publication in the near future. We also have one professional staff engineer, Pierre Costa who joined the project last spring and is responsible for overall coordination of the hardware prototyping effort.

For administrative purposes, the ACS project operates within the Computer and Communications Research Center directed by Professor Mark Franklin. The Center has a central office suite housing professors Franklin and Turner, one technical staff person, plus seven graduate students, on the third floor of Bryan Hall, across from our main laboratory facility. This laboratory houses most of our computers, and a cluster of terminals and workstations for graduate student use and also serves as an informal meeting room. Last summer, we acquired additional office and laboratory space on the fifth floor of Bryan. Eight students and two additional staff members are located in this area. While we are in reasonably good shape with respect to space at the moment, there is little room for expansion and there is likely to be some crowding in the coming year. On the other hand, the Engineering School will break ground this summer for a new research building of approximately 50,000 square feet that will provide substantial new space for Electrical Engineering and Computer Science. This is part of the school's commitment to expand the EE and CS faculties by about 50% over the next several years, with corresponding growth in graduate enrollment, particularly at the doctoral level.

The Center's base of equipment includes a VAX 750, a MicroVax II/GPX and a Sun workstation environment including a 3/280 file server, a 3/150 which will interface to our prototype switching system, and six 3/50 diskless workstations. A Sun 3/60 color workstation has also been ordered. The Suns have been acquired over the last year and support a variety of activities, including software develop-

ment, VLSI design, simulation, general-purpose computing and word processing. The MicroVax is used primarily to support VLSI design work. The Center has also recently taken delivery of a 64 processor NCUBE parallel computer, which Professor Franklin will be using to support his research in the area of design automation. We also anticipate its possible use in our project. In addition we have about fifteen conventional terminals, a PC/AT, another VLSI design station, several printers, and assorted lab equipment including a Tektronix logic analyzer and IC tester.

We have been generally successful in expanding the Center's space and facilities to meet our needs. As we are not planning substantial additional growth in the immediate future, we feel reasonably comfortable with the current situation. On the other hand, space shortages may develop in the next year as the Computer Science and Electrical Engineering departments continue to expand their faculties. While the construction of the new building should provide ample space in the longer term, there will be an intermediate period of limited space that will have to managed carefully.

2. Switch Architecture Studies

Faculty	Jonathan Turner
Research Associate	Riccardo Melen

The architecture of high speed packet switching fabrics is of course central to the work of this project. While we are concentrating our efforts on a particular design [89], we continue to evaluate alternatives, in order to identify possible improvements. Recently, we have studied architectures that can support higher speed operation (link speeds of a few Gb/s) and be economically implemented even in fairly large configurations (thousands of links). These architectures differ from earlier ones in two respects. First they use a bit-sliced organization to allow much wider data paths and second, they constrain packets in a given connection to follow a single path. At the same time, they can fully support the same connection-oriented packet switching paradigm, including multipoint connections.

Packet switching fabrics employing parallel data paths can be organized in a couple different ways. One possibility is the so-called *word-serial* approach, in which all the bits in a given data path pass through the same physical components. Another is the *bit-sliced* approach, in which the components making up the switch fabric are "sliced" so that each bit of the data path passes through a different set of components. An example illustrating these two approaches is shown in Figure 2.1. In the figure on the left, each circle corresponds to a single integrated circuit, as do the rectangles on the right. Notice that each of these structures implements a 16 port switching fabric with 8 bit wide data paths and that the integrated circuits in both cases require 32 signal leads. However, the word-serial structure requires 32 chips while the bit-sliced structure requires just eight. For large systems, this advantage of bit-sliced structures becomes even more dramatic. Figure 2.2 plots the chip count per port for Benes networks with several choices of the data path width. N is the number of ports the switch fabric has, m is the data path width, *ws* stands for word-serial and *bs* for bit-sliced. Notice that for the bit-sliced organization, we can achieve data path widths of 32 at a cost of about five chips per port for switches with between 2,048 and 32,768 ports.

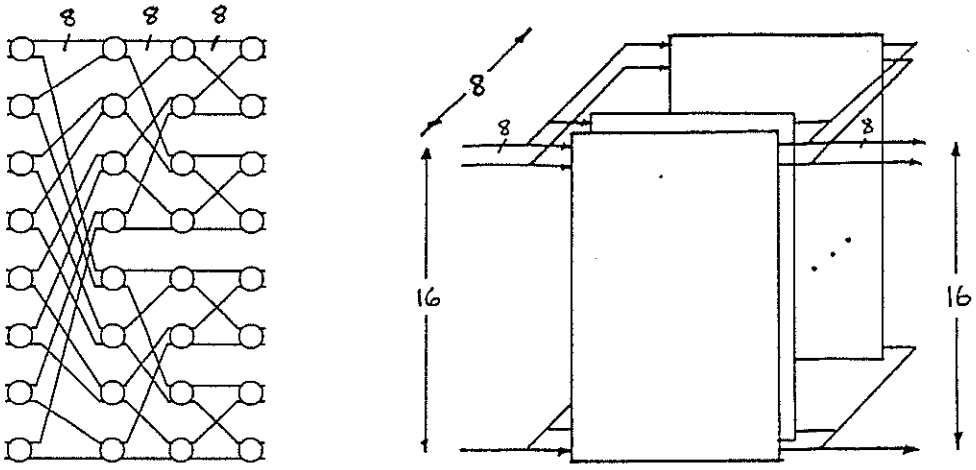

Figure 2.1: Word-Serial vs. Bit-Sliced Organization

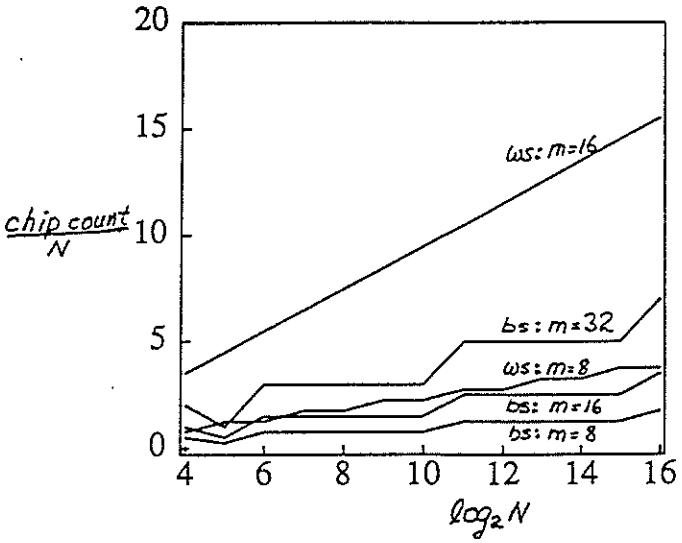

Figure 2.2: Complexity of Word-Serial and Bit-Sliced Benes Networks

Of course, on the other side, the word-serial organization is somewhat simpler to control. In the bit-sliced organization, each slice must make the same control decisions for the packets it receives. This can be done either by replicating the control information and sending it to each slice, or by having one slice decode the control information, make the appropriate decisions and communicate the results to the other slices. We examine the latter alternative and describe a practical design of a packet switching element that implements it.

The proposed switch element can be used to implement a buffered binary routing network with hardware flow control between adjacent switch elements to prevent internal buffers from overflowing [89]. Two different chip types are used; one

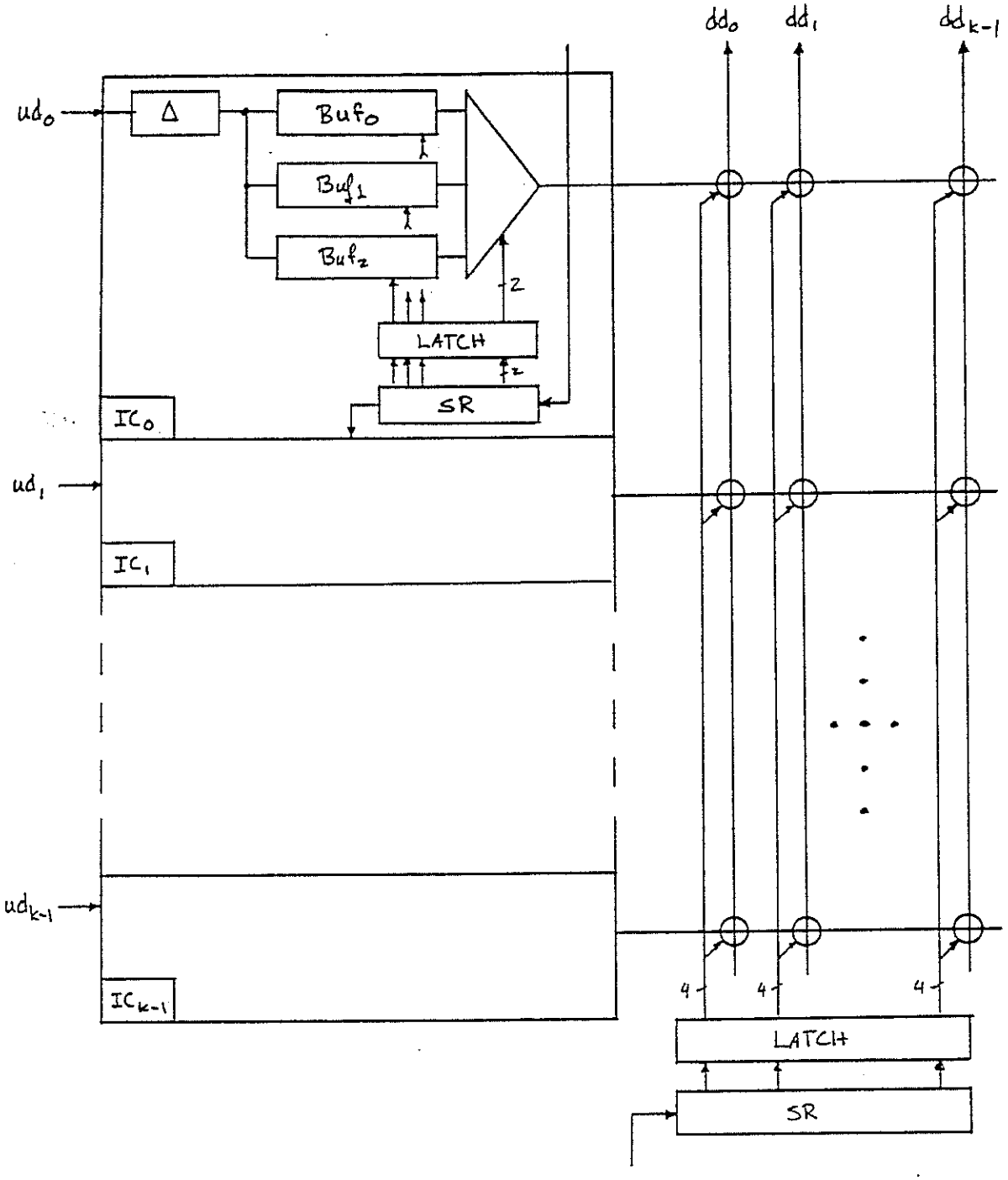

Figure 2.3: Data Slice

for control and one for data. Packets are formatted with all of the control information in a single slice to facilitate access by the control chips. Figure 2.3 shows the organization of a chip implementing the data slice. It has k data inputs and k outputs, where typically k might be 16 or 32. The chip contains one *Input Circuit* (IC) for each input. Each IC contains several buffers, each able to store a complete bit slice of a packet. The buffers can be implemented as shift registers; control is provided through a set of latches which enable/disable shifting and select which of the several shift registers to use for output. The latches are loaded through a shift register connected to an external pin, which is driven by the chip implementing the control slice. In addition to the ICs, the data slice contains a $k \times k$ crossbar switch matrix, which controls the transmission of data to the appropriate output. The control information for the crossbar also comes from the control slice. Note that the control is output driven, allowing a given input to be connected to several outputs.

The control slice is shown in Figure 2.4. This chip does not include any data storage; it merely monitors the bit slice containing the control information, makes the appropriate decisions and transmits these decisions to each of the data slices. The chip has an *Input Control Circuit* (ICC) for each of the k inputs. It also has a set of k *downstream grants* and k *upstream grants.* A downstream grant is asserted by one of a switch element's downstream neighbors if the neighbor is able to receive another packet. Similarly, the switch element asserts its upstream grant for each input that is able to receive another packet. The bit slice containing control information enters the chip on the *upstream data leads.* Each ICC shifts in the control information, latches it and decodes it. It is then stored in one of several control registers corresponding to the data buffers in which the packet data is stored. During a given operation cycle, each ICC requests access to one or more outputs. These requests are forwarded to an arbitration circuit that consists of a $k \times k$ array of arbitration elements (AE). Each ICC's request is either granted or denied by the arbitration element; the results of these decisions are then forwarded to the data slices. The sorting network between the ICCs and the arbitration circuit sorts the requests in priority order; priority is based primarily on the number of outputs required by a given ICC.

We can estimate the complexity of the control slice as follows; let x_1 be the complexity of an ICC, let x_2 be the complexity of a single sorting element in the sorting network and let x_3 be the complexity of an arbitration element. Then the complexity of the control slice is approximately

$$kx_1 + (k/4)(\log_2 k)(1 + \log_2 k) + k^2 x_3$$

If we estimate x_1 at 1,000 transistors, x_2 at 100 transistors and x_3 at 100 transistors we find that a 16 port control slice requires approximately 50,000 transistors and

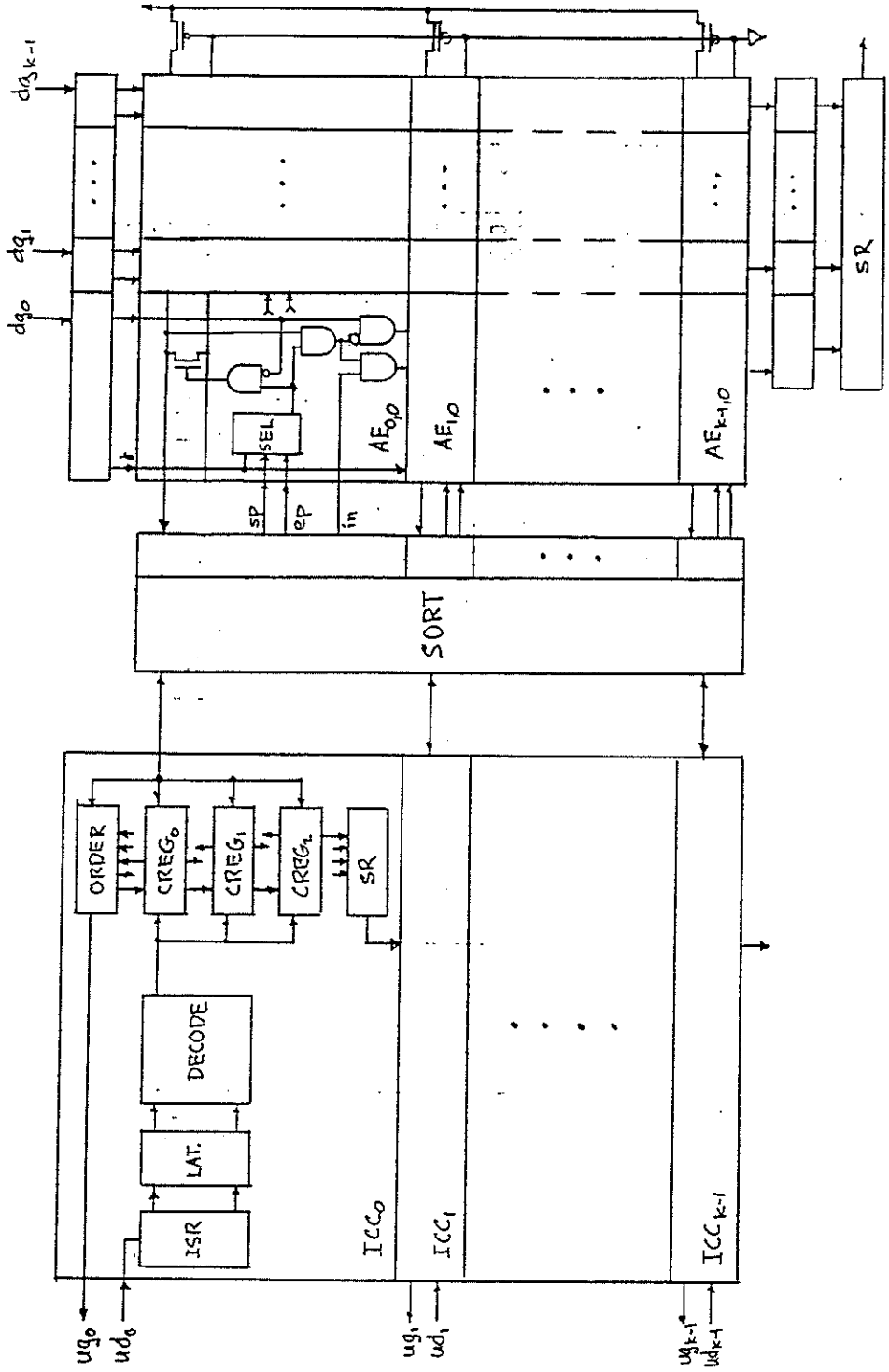

Figure 2.4: Control Slice

a 32 port control slice, approximately 160,000 transistors, making it clearly within reach of current CMOS technology. The complexity of the data slice is determined largely by the output crossbar complexity and the memory requirements, which in turn depend on the packet length and the degree of parallelism, since systems constructed from many parallel bit slices need store fewer bits per slice. If we let L denote the packet length, m the amount of parallelism, b the number of buffers provided in each IC, x_1 the number of transistors per bit of memory and x_2 the number of transistors per output crosspoint, then the data slice complexity is approximately

$$kLb/m + k^2x_2$$

If we let $L = 4096$, $m = 32$, $b = 3$ and estimate x_1 and x_2 at about ten transistors, we require about 64,000 transistors for a 16 port data slice and about 133,000 transistors for a 32 port data slice.

Based on this design we have estimated that a packet switching system comprising a Benes network with 32 bit wide data paths and supporting 4096 fiber optic data links would require about twelve standard equipment cabinets. Another four cabinets would be required for the link interfaces and packet processors. If such a system was operated with a clock rate of 100 Mb/s (an achievable rate even with CMOS), its internal data paths would operate at an effective rate of 3.2 Gb/s, which is sufficient to support external link speeds of 1.6 Gb/s. This represents an order of magnitude improvement over the speeds being achieved by current research efforts. While there are limits to how far such techniques can go, it appears likely that almost another order of magnitude is possible, through a combination of higher clock rates and greater parallelism.

3. Performance Studies

Faculty	Jonathan Turner
Research Associate	Riccardo Melen

The evaluation of any switching system architecture is determined in large part by performance issues. We have addressed performance issues in several different ways. Previous reports have described extensive simulation studies examining several aspects of the performance of the proposed Broadcast Packet Switch fabric. In this report, we examine two broader performance issues. First, we consider a general method of evaluating the loading characteristics of packet switching fabrics that dynamically distribute their load across all available paths. As we shall see, such fabrics can be made robust in the face of arbitrary traffic patterns with minimum complexity. We refer to the analysis method as *fluid flow loading analysis*, and using it, we derive several fundamental results for both point-to-point and multipoint packet switching fabrics. In section 3.2, we consider a class of fabrics in which all packets belonging to a particular connection are constrained to follow the same path. The prime motivation for making such a constraint is to eliminate the possibility of packet mis-ordering. As we shall see, this consideration leads to a natural generalization of the classical theory of non-blocking networks; in this report we define that generalization, outline the important problems and present several fundamental theorems. We close this chapter with a brief description of simulation results which quantify the potential for packet misordering in the proposed broadcast packet switch design. These results are intended to to be used to help design a mechanism to resequence misordered packets on a switch module basis.

3.1. Fluid Flow Loading Analysis

In this section we introduce a systematic method of analyzing the effects of a given traffic configuration on packet switching fabrics that dynamically distribute

load across all available paths, and apply it to the analysis of several proposed architectures. Our method allows us to prove theorems characterizing the worst-case loading for various switching fabrics. The section gives several such theorems, both as illustrations of our method and for their inherent interest. Proofs are omitted for brevity; readers are referred to [94] for further details.

We note that the method is fairly easy to apply and leads to useful insights that can guide the switching system architect to better designs. It is not a complete characterization, as it ignores queueing and contention, but when used in conjunction with queueing and simulation models based on uniform random traffic, it can provide the designer and performance analyst with a more complete understanding of system performance.

Networks for Point-to-Point Communication

We define a *packet switching network* (or simply network) as a directed graph $G = (N, L)$ consisting of a set of nodes N and a set of directed arcs or *links* L. In addition, G contains a set of distinguished *input nodes* I and a set of distinguished *output nodes* O. Input and output nodes are also referred to as *ports.* Each input port has a single outgoing link and no incoming links, while each output port contains a single incoming link and no outgoing links.

We limit ourselves to networks in which the number of input nodes equals the number of output nodes. When we refer to an n port network, we mean a network with n input nodes and n output nodes, numbered from 0 to $n-1$. We also limit ourselves to networks, which can be divided into a sequence of stages. We say that input ports are in stage 0 and for $i > 0$, a node v is in stage i if for all links (u, v), u is in stage $i-1$. A link (u, v) is said to be in stage i if u is in stage i. In the networks we consider, all output ports are in a separate stage by themselves. When we refer to a k stage network, we mean that there are k stages containing internal nodes; that is, we neglect the input and output stages.

When describing particular networks, we will find it convenient to use a composition operation. We denote a composition of two networks X_1 and X_2 by $X_1 \textcircled{h} X_2$, where h is a positive integer. The composition operation yields a new network consisting of one or more copies of X_1 connected to one or more copies of X_2, with h links joining each pair of subnetworks. More precisely, if X_1 is an n_1 port network and X_2 is an n_2 port network then $X_1 \textcircled{h} X_2$ is formed by taking n_2/h copies of X_1 numbered from 0 to $(n_2/h)-1$ followed by n_1/h copies of X_2, numbered from 0 to $(n_1/h)-1$. Then, for $1 \leq i \leq n_1$, $1 \leq j \leq n_2$, we join $X_1(i)$ to $X_2(j)$ using h links; these links connect output port $(n_1/h)m + j$ of $X_1(i)$ to input port $(n_2/h)m + i$ of $X_2(j)$, where $0 \leq m < h$. Finally, we eliminate the former input and output nodes

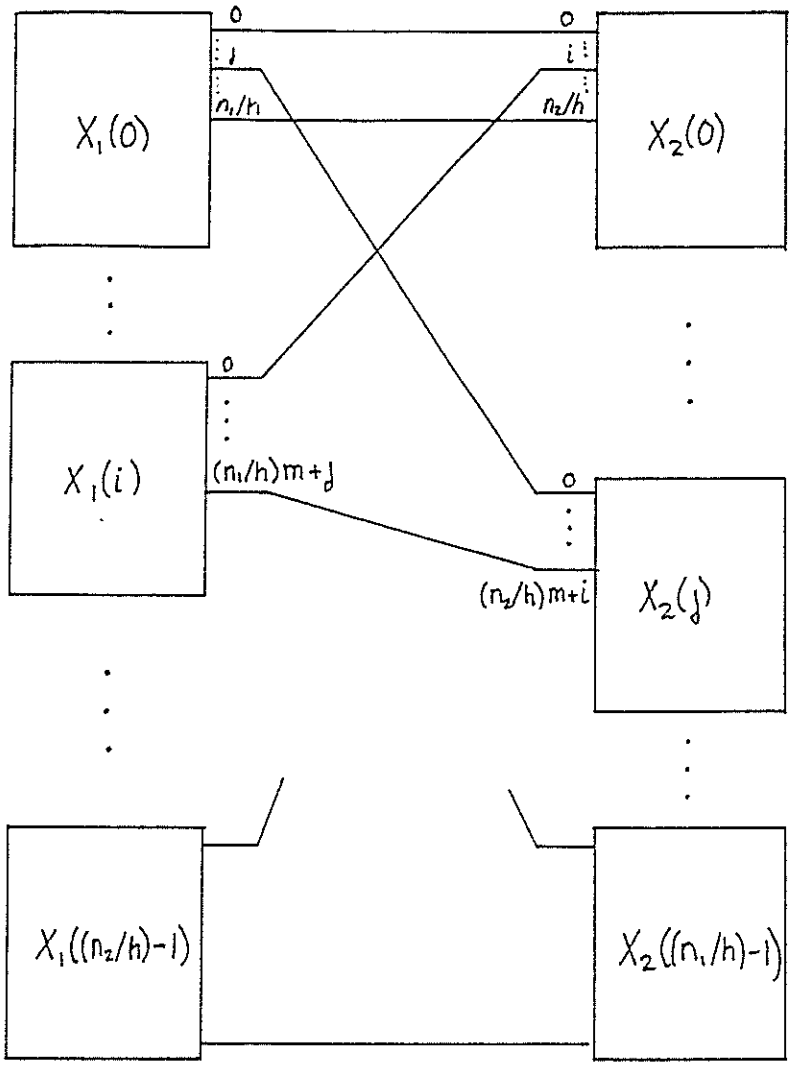

Figure 3.1: Composition Operation

that are now internal and renumber the input and output nodes of the network as follows; if u was input port i of $X_1(j)$, it becomes input jn_1+i in the new network; similarly if v was output port i of $X_2(j)$, it becomes output jn_2+i. We also allow composition of more than two networks; the composition $X_1 \textcircled{h} X_2 \textcircled{i} X_3$ is obtained by letting $Y_1 = X_1 \textcircled{h} X_2$ and $Y_2 = X_2 \textcircled{i} X_3$, then identifying the copies of X_2 in Y_1 and Y_2. This requires of course that the number of copies of X_2 generated by the two initial compositions be the same. Note this is not the same as $(X_1 \textcircled{h} X_2) \textcircled{i} X_3$. Finally, we use the symbol o in place of ① when appropriate. The composition operation is illustrated in Figure 3.1.

A *connection* through a network is defined as a triple (x, y, ρ) where $x \in I$,

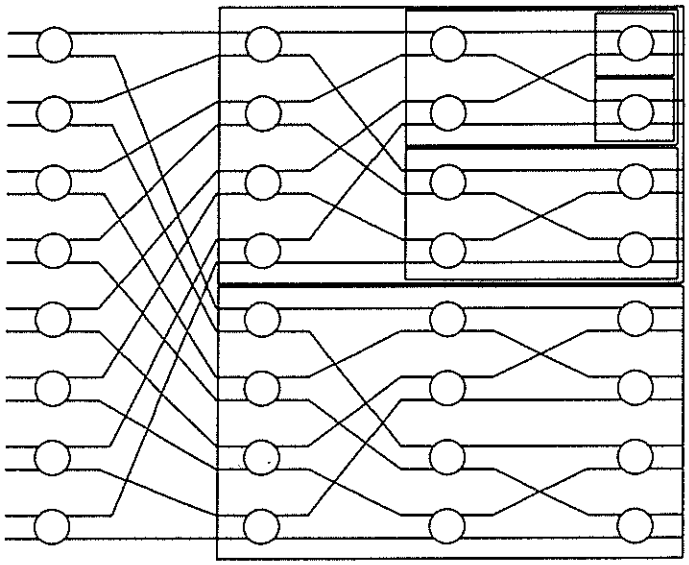

Figure 3.2: Recursive Construction of Delta Network

$y \in O$ and $0 < \rho < 1$. A connection *induces a load* on the various links that lie on paths joining the connection's input and output ports. The load induced by a connection (x, y, ρ) on the link leaving x is defined to be ρ. The magnitude of the induced loads on the internal links depends on the types of the nodes and the topology of the network. In this section, we will consider only a single node type. If α is the sum of the loads induced by a connection (x, y, ρ) on the input links of a node u, and u has i output links that lie on paths from x to y, then the load induced by the connection on each of these output links is α/i and the load induced on all other output links is 0.

A *configuration* is defined as a set of connections. The load induced by a configuration on a link ℓ is simply the sum of the loads induced by the individual connections and is denoted $\lambda_\ell(C)$. A configuration $C = \{c_1 \ldots c_\tau\}$ is α-bounded if for all input and output links ℓ, $\lambda_\ell(C) \leq \alpha$. We say that a configuration is *legal* if it is 1-bounded and that a network is *robust* if for every legal configuration C, $\lambda_\ell(C) \leq 1$ for *all* links ℓ.

Delta networks form a well-known class of useful switching networks [21,22,23, 29]. We can define these recursively using the composition operation. Let D_1 be a network with two input ports and two output ports connected to a single internal node. We then define $D_i = D_1 \circ D_{i-1}$ for all $i \geq 2$. We refer to D_k as a k stage delta network; note that D_k has $n = 2^k$ ports. An example of a 4 stage delta

network is given in Figure 3.2.

Delta networks have been widely studied and have many interesting properties. Most useful is the *self-routing property* that allows paths from inputs to outputs to be easily determined. A related property is that there is a single path connecting any input node to any output node. For the purposes of our loading analysis, this means that a connection (x, y, ρ) induces a load of ρ on all links that lie on that path and a load of 0 on all other links. To illustrate our method of loading analysis, we start with a simple theorem which characterizes the worst-case loading for a delta network.

THEOREM 3.1.1. *Let $C = \{c_1, \ldots, c_r\}$ be an α-bounded configuration for D_k. Then $\lambda_\ell(C) \leq \alpha\sqrt{n}$ for all links ℓ.*

The bound in Theorem 3.1.1 can be achieved; that is, there exist worst-case patterns that induce a load approaching $\sqrt{n}$ on some of the internal links. We note that delta networks are readily generalized to networks in which each internal node has m input ports and m output ports. The worst-case loading in such networks is the same as for networks constructed from two port nodes.

The bound in Theorem 3.1.1 and the fact that there are traffic patterns that achieve the bound, lead to the conclusion that the binary (and m-ary) delta networks can perform poorly in the worst-case. This has been observed previously and various approaches have been proposed to remedy the situation. We review two such approaches here. The first is to add one or more stages of distribution nodes at the front of a delta network.

We denote a delta network with k routing stages and d distribution stages as $D_{k,d}$, which we define by $D_{k,d} = D_d \circ D_{k-d} \circ D_d$. This is illustrated in Figure 3.3. If we consider the load induced by a connection (x, y, ρ) on the links in such a network, we note that for any node u in the first d stages that lies on a path from x to y, both of u's output links lie on paths from x to y, hence the incoming load from the connection is distributed across u's output links. In contrast, any node v in the last k stages is on at most one path from x to y. We refer to the nodes in the first d stages as distribution nodes and the nodes in the last k stages as routing nodes.

THEOREM 3.1.2. *Let $C = \{c_1, \ldots, c_r\}$ be an α-bounded configuration for $D_{k,d}$. Then $\lambda_\ell(C) \leq \alpha n 2^{-\lceil (k+d)/2 \rceil}$ for all links ℓ.*

The bound in Theorem 3.1.2 is the best possible; that is, there exist traffic patterns approaching the given bound. Theorem 3.1.2 tells us that every time we add

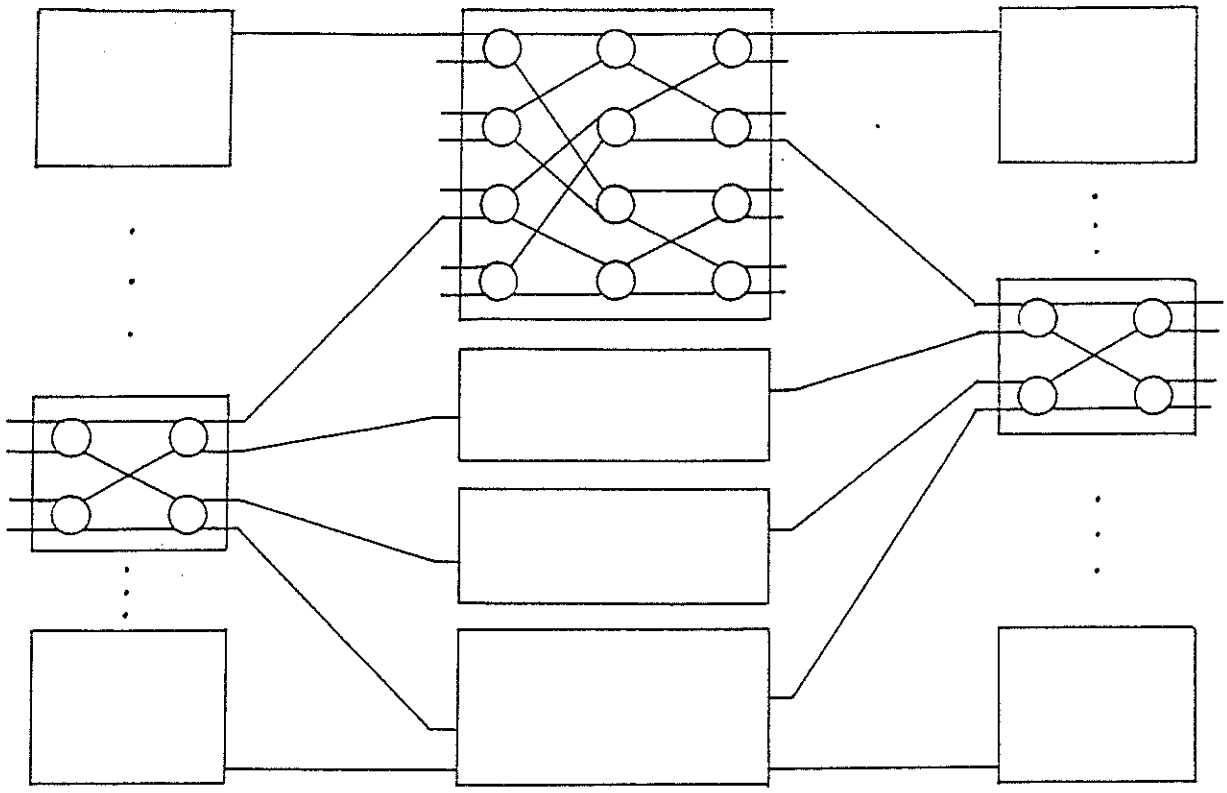

Figure 3.3: Construction of Delta Network with Distribution Stages

two distribution stages, we reduce the worst-case load by a factor of 2. To achieve a robust network, we require $d = k - 1$. Also note that with respect to worst-case loading, it never makes sense to have $k + d$ an even number, since a network with one fewer distribution stage has the same worst-case loading characteristics. We note that this result can be readily generalized to networks with nodes having m input and output ports. The bound in the statement of the theorem becomes $nm^{-\lceil (k+d)/2 \rceil}$ (with $k = \log_m n$).

In [55], Lea proposes a variant of the delta network that we refer to as the *alternate routing network*. We can define this network recursively using the composition operation. The base network is denoted by A_1 and consists of four input ports and four output ports connected to a single internal node. For $i > 1$, $A_i = A_1 \textcircled{2} A_{i-1}$. An example of an alternate routing network is given in Figure 3.4. Note that an alternate routing network with k stages has $n = 2^{k+1}$ ports. Given any connection (x, y, ρ), if u is in the first $k - 1$ stages and lies on some path from x to y, then two of u's output links lie on paths from x to y. Consequently, whatever load is induced on the input links of u will be shared by two of u's output links. The following theorem characterizes the worst-case loading of an alternate routing fabric. We note that essentially the same result is stated (in somewhat different terms)

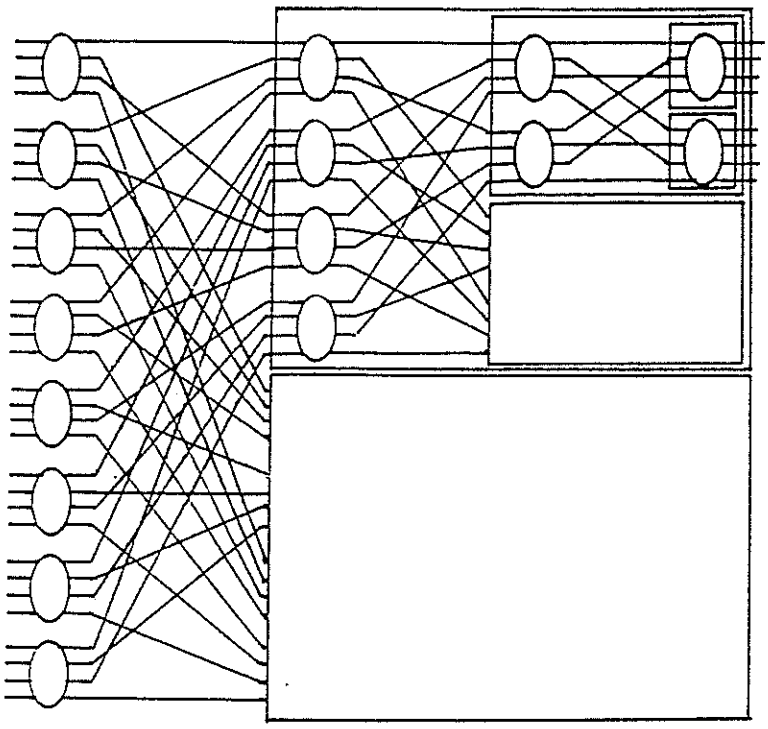

Figure 3.4: Construction of Alternate Routing Network

in [55].

THEOREM 3.1.3. *Let* $C = \{c_1, \ldots, c_\tau\}$ *be an* α*-bounded configuration for* A_k. *Then* $\lambda_\ell(C) \leq \alpha n^{1/3}$ *for all links* ℓ.

Loading in Copy Networks

The broadcast packet switch of [89] is one of several proposed systems for multipoint communication. In this section we study the worst-case loading of the copy network, which gives that system the ability to handle multipoint communication. We also consider several variants.

When dealing with copy networks, we must modify our definition of connection. In the current context, we define a connection to be an an ordered triple (x, F, ρ), where x is the input port of the copy network where packets belonging to the connection enter, F is the *fanout* of the connection and ρ is the load induced by the connection at the input port x. The fanout of the connection is the number

of copies that must be produced by the copy network for each input packet. We say that a traffic configuration C is *α-bounded* if $\lambda_\ell(C) \leq \alpha$ for all input ports ℓ and $\sum_{(x,F,\rho)\in C} \rho F \leq \alpha n$, where n is the number of input and output ports. A legal configuration is one that is 1-bounded.

Reference [89], describes a copy network that is topologically identical to a delta network. However, the nodes of a copy network replicate received packets under certain conditions. Specifically, a node may replicate a packet if the number of output ports reachable from that node is less than $2F$, where F is the fanout of the connection the packet belongs to. Packets that are not replicated are routed to an arbitrarily selected output. Hence, if α is the load induced on the input links of a node u by a connection (x, F, ρ), then the load induced on each of u's output links is α, if the number of output ports that can be reached from u is $< 2F$ and $\alpha/2$ otherwise.

Given these definitions, we find that for a connection $c = (x, F, \rho)$ and a link ℓ in stage i,

$$\lambda_\ell(c) = \begin{cases} 0 & \text{if there is no path from input } x \text{ to link } \ell \\ \rho 2^{-i} & \text{if there is a path and } 0 \leq i \leq k - \lceil \log_2 F \rceil \\ \rho 2^{-(k-\lceil \log_2 F \rceil)} & \text{if there is a path and } i \geq k - \lceil \log_2 F \rceil \end{cases}$$

Our first theorem, which was first proved in [10] shows that the worst-case loading in a copy network is bounded.

THEOREM 3.1.4. *Let $C = (c_1, \ldots, c_r)$ be any α-bounded configuration for an n-port copy network. Then, $\lambda_\ell(C) \leq 3\alpha$ for all links ℓ.*

There exist legal traffic patterns approaching the bound in Theorem 3.1.4. Copy networks can also be constructed using nodes with $m > 2$ input and output ports. In such networks, a node replicates a packet m times if the number of reachable output ports is less than mF. Surprisingly, the worst-case performance of such a copy network is worse than for a copy network constructed from binary nodes.

THEOREM 3.1.5. *Let $C = (c_1, \ldots, c_h)$ be an α-bounded configuration for an n-port copy network constructed from m-port nodes. Then $\lambda_\ell(C) < \alpha(m+1)$, for all links ℓ.*

The proof of this is very similar to that of Theorem 3.1.4. Again, the bound is the best possible; there exist legal traffic configurations that induce loads approaching $m + 1$ on some internal links.

As with routing networks, we can improve the worst-case performance of a copy network by adding distribution stages. The topology of such a network is identical to a routing network with added distribution stages. The effect on the worst-case loading is captured by the following theorem.

THEOREM 3.1.6. *Let $C = (c_1, \ldots, c_h)$ be an α-bounded configuration for a copy network with k copy stages and d distribution stages. Then $\lambda_\ell(C) < \alpha(1 + 2^{1-d})$, for all links ℓ in stages 0 to $k + d - 1$; $\lambda_\ell < 2\alpha$ for all links ℓ in stage $k + d$.*

Theorem 3.1.6 shows that the worst-case loading in a copy network can be brought very close to α in all but the last stage links, by adding a few distribution stages. We note that Theorem 3.1.6 can be generalized to copy networks constructed with m-ary nodes. In this case the bound on the worst-case loading becomes $\alpha(1 + m^{1-d})$ for all but the last stage and m for the last stage.

3.2. Nonblocking Multirate Networks

In this section we introduce a generalization of the classical theory of nonblocking switching networks to model communications systems designed to carry connections with a multiplicity of data rates (details can be found in [60]). This theory can be used to model packet switching fabrics in which all packets in a given connection are constrained to follow the same path. The theory of nonblocking networks was motivated by the problem of designing telephone switching systems capable of connecting any pair of idle terminals, under arbitrary traffic conditions. From the start, it was recognized that crossbar switches with N terminals and N^2 crosspoints could achieve nonblocking behavior, only at a prohibitive cost in large systems. In 1953, Charles Clos [16] published a seminal paper giving constructions for a class of nonblocking networks with far fewer crosspoints, providing much of the initial impetus for the theory that has since been developed by Benes [6,7], Pippenger [70] and many others [3,12,27,51,57,58,66].

We start with some definitions. A *connection* in a network is a triple (x, y, ω) where $x \in I$, $y \in O$ and $0 \leq \omega \leq 1$. We refer to ω as the *weight* of the connection and it represents the bandwidth required by the connection. A *route* is a path joining an input node to an output node, with intermediate nodes in $V - (I \cup O)$, together with a weight. A route r *realizes* a connection (x, y, ω), if x and y are the input and output nodes joined by r and the weight of r equals ω.

A set of connections is said to be *compatible* if for all nodes $x \in I \cup O$, the sum of the weights of all connections involving x is ≤ 1. A *state* of a network G is a set of routes. The weight on an edge in a particular configuration is just the

sum of the weights of all routes including that edge. A state is legal if for all edges $(u, v) \in E$, the weight on (u, v) is ≤ 1. A set of connections is said to be *realizable* if there is a legal state that realizes that set of connections. If we are attempting to add a connection (x, y, ω) to an existing state, we say that a node u is *accessible* from x if there is path from x to u, all of whose edges have a weight of no more than $1 - \omega$.

A network is said to be *rearrangeably nonblocking* (or simply *rearrangeable*) if every set of compatible connections is realizable. A network is *strictly nonblocking* if for every legal state R, realizing a set of connections C, and every connection c compatible with C, there exists a route r that realizes c and such that $R \cup \{r\}$ is a legal state. For strictly nonblocking networks, one can choose routes arbitrarily and always be guaranteed that any new connections can be satisfied without rearrangements. We say that a network is *wide-sense nonblocking* if there exists a routing algorithm, for which the network never blocks; that is, if we use the routing algorithm to select routes for each new connection request, it is always possible to realize a new connection by adding a route to the current configuration.

Sometimes, improved performance can be obtained by placing constraints on the traffic imposed on a network. We will consider two such constraints. First, we restrict the weights of connections to the the interval $[b, B]$. We also limit the sum of the weights of connections involving a node x in $I \cup O$ to β. Note that $0 \leq b \leq B \leq \beta \leq 1$. We say a network is strictly nonblocking for particular values of b, B and β if for all sets of connections for which the connection weights are in $[b, B]$ and the total port weight is β, the network cannot block. The definitions of rearrangeably nonblocking and wide-sense nonblocking networks are extended similarly. The practical effect of a restriction on β is to require that a network's internal data paths operate at a higher speed that the external transmission facilities connecting switching systems, a common technique in the design of high speed systems. The reciprocal of β is commonly referred to as the *speed advantage* for a system.

Two particular choices of parameters are of special interest. We refer to the traffic condition characterized by $B = \beta$, $b = 0$ as unrestricted packet switching (UPS), and the condition $B = b = \beta = 1$ as pure circuit switching (CS). Since the CS case is a special case of the multirate case, we can expect solutions to the general problem to be at least as costly as the CS case and that theorems for the general case should include known results for the CS case.

Strictly Nonblocking Networks

A three stage Clos [16] network with N input and output ports is denoted by $C_{N,k,m}$, where k and m are parameters, and is defined as: $C_{N,k,m} = X_{k,m} \circ X_{N/k,N/k} \circ$

$X_{m,k}$, where $X_{r,s}$ denotes an $r \times s$ crossbar. The standard reasoning to determine the nonblocking condition (see [16]) can be extended in a straightforward manner, yielding the following theorem.

THEOREM 3.2.1. *The Clos network $C_{N,k,m}$ is strictly nonblocking if*

$$m > 2 \max_{b \le \omega \le B} \left\lfloor \frac{\beta k - \omega}{s(\omega)} \right\rfloor$$

where $s(\omega) = \max\{1 - \omega, b\}$.

Using Theorem 3.2.1, we can construct a wide-sense nonblocking network for unrestricted traffic by placing two Clos networks in parallel and segregating connections in the two networks based on weight. In particular if we let $m = 4k-1$, the network $X_{1,2} \circ C_{N,k,m} \circ X_{2,1}$ is wide-sense nonblocking if all connections with weight $\le 1/2$ are routed through one of the Clos subnetworks and all the connections with weight $> 1/2$ are routed through the other.

A k-ary Benes network [6], built from $k \times k$ switching elements (where $\log_k N$ is an integer) can be defined recursively as follows: $B_{k,k} = X_{k,k}$ and $B_{N,k} = X_{k,k} \circ B_{N/k,k} \circ X_{k,k}$. A k-ary Cantor network of multiplicity m is defined as $K_{N,k,m} = X_{1,m} \circ B_{N,k} \circ X_{m,1}$. The next theorem captures the condition on m required to make the Cantor network strictly nonblocking.

THEOREM 3.2.2. *The Cantor network $K_{N,k,m}$ is strictly nonblocking if*

$$m \ge 2 \frac{\beta}{s(B)} \frac{k-1}{k} \log_k N$$

COROLLARY 3.2.1. *The Benes network $B_{N,k}$ is strictly nonblocking if*

$$\beta \le \left[\frac{2}{s(B)} \frac{k-1}{k} \log_k N \right]^{-1}$$

When we apply the theorem to the CS case for $k = 2$, we find that the condition on m reduces to $m \ge \log_2 N$ as is well known. For the UPS case with $k = 2$, we have $m \ge 2(\beta/(1-\beta)) \log_2 N$; that is, we again need a speed advantage of two to match the value of m needed in the CS case.

We can construct wide-sense nonblocking networks for $\beta = 1$ by increasing m. We divide the connections into two subsets, with all connections of weight $\le 1/2$ segregated from those with weight $> 1/2$. Applying Theorem 3.2.2 we find that $m \ge 4((k-1)/k) \log_k N$ is sufficient to carry each portion of the traffic, giving a total of $8((k-1)/k) \log_k N$ subnetworks.

Rearrangeably Nonblocking Networks

The Benes network is rearrangeable in the CS case [6] and efficient algorithms exist to reconfigure it [57,66]. The next theorem gives conditions under which it is rearrangeable in the multirate case as well.

THEOREM 3.2.3. *$B_{N,k}$ is rearrangeable when*

$$\beta \leq \left[1 + \frac{k-1}{k}(B/\beta)\log_k(N/k)\right]^{-1}$$

As an example, if $N = 2^{16}$, $k = 16$ and $B = \beta$, it suffices to have $\beta \leq 0.26$. The proof of Theorem 3.2.3 is fairly straightforward (see [60]). A more detailed analysis yields the following theorem.

THEOREM 3.2.4. *$B_{N,k}$ is rearrangeable when*

$$\beta \leq [\max\{2, \lambda - \ln\lfloor\beta/B\rfloor\}]^{-1}$$

where $\lambda = 2 + \ln\log_k(N/k)$.

So, for example if $N = 2^16$, $k = 16$ $B = \beta$, we can have $\beta = 0.32$. The following theorem gives conditions under which the Cantor network is rearrangeable.

THEOREM 3.2.5. *Let $\epsilon > 0$ and $\lfloor\beta/B\rfloor \leq \log_k(N/k)$. $K_{N,k,m}$ is rearrangeable if*

$$m \geq \lceil(1+\epsilon)(\lambda - \ln\lfloor\beta/B\rfloor)\rceil + 2\,(2 + \log_2\lambda + \log_2(B/c))$$

where $\lambda = 2 + \ln\log_k(N/k)$ *and* $c = 1 - \beta\lambda/(1+\epsilon)(\lambda - \ln\lfloor\beta/B\rfloor)$.

We can also provide conditions for rearrangeability fo networks that "expand" at each level of recursion. Let $C^*_{k,k,m} = X_{k,k}$ and for $N = k^i$, $i > 1$, let $C^*_{N,k,m} = X_{k,m} \circ C^*_{N/k,N/k} \circ X_{m,k}$.

THEOREM 3.2.6. *$C^*_{N,k,m}$ is rearrangeable if*

$$\beta \leq \left[1/\gamma^c + \frac{m-1}{m}\frac{B}{\beta}\frac{1-1/\gamma^c}{1-1/\gamma}\right]^{-1}$$

where $\gamma = m/k$ *and* $c = \log_k(N/k)$.

Remarks

In recent years, there has been a growing interest in switching systems capable of carrying general multirate traffic, in order to support a wide range of applications including voice, data and video. Several research teams have constructed high speed switching systems of moderate size [18,39,89,101], but little consideration has yet been given to the problem of constructing very large switching systems using such modules as building blocks. The theory we have developed here is a first step to understanding the blocking behavior of such systems.

We have introduced what we feel is an important research topic and have given some fundamental results. There are several directions in which this work may be extended. While we have good constructions for strictly nonblocking networks, we expect that our results for rearrangeably nonblocking networks can be improved. In particular, we suspect that the Benes network can be operated in a rearrangeable fashion with just a constant speed advantage. Another interesting topic is nonblocking networks for multipoint connections. While this has been considered for space-division networks [3,27,51,75], it has not been previously studied for networks supporting multirate traffic. We have recently devised two novel switching structures that are rearrangeably nonblocking for multipoint connections. These will be described in a later report. Another area to consider is determination of blocking probability for multirate networks. We expect this to be highly dependent on the particular choice of routing algorithm.

3.3. Packet Misordering

This section presents some initial simulation results which attempt to assess the likelihood that packets passing through a broadcast packet switch become misordered. The results presented here are for a configuration consisting of a copy network, distribution network and routing network, all with 64 input and output ports and all comprising binary switch elements with two buffer slots per input.

Our results are summarized in the two plots shown in Figure 3.5. The plot on the left gives the distribution of the delay incurred by packets passing through a switch fabric. Note that for an offered load of $\rho = 0.4$ (the maximum allowed under normal operating conditions) the vast majority of the packets pass through the switch fabric in under ten packet times and at this loading level, only about one packet in 10^5 is delayed as much as 20 packet times. Since the switch fabric has a 2:1 speed advantage over the external links, packets that arrive 10 packet times apart on an external link are very unlikely to get misordered by the switch fabric. This in turn suggests that resequencing packets on the output side of the switch

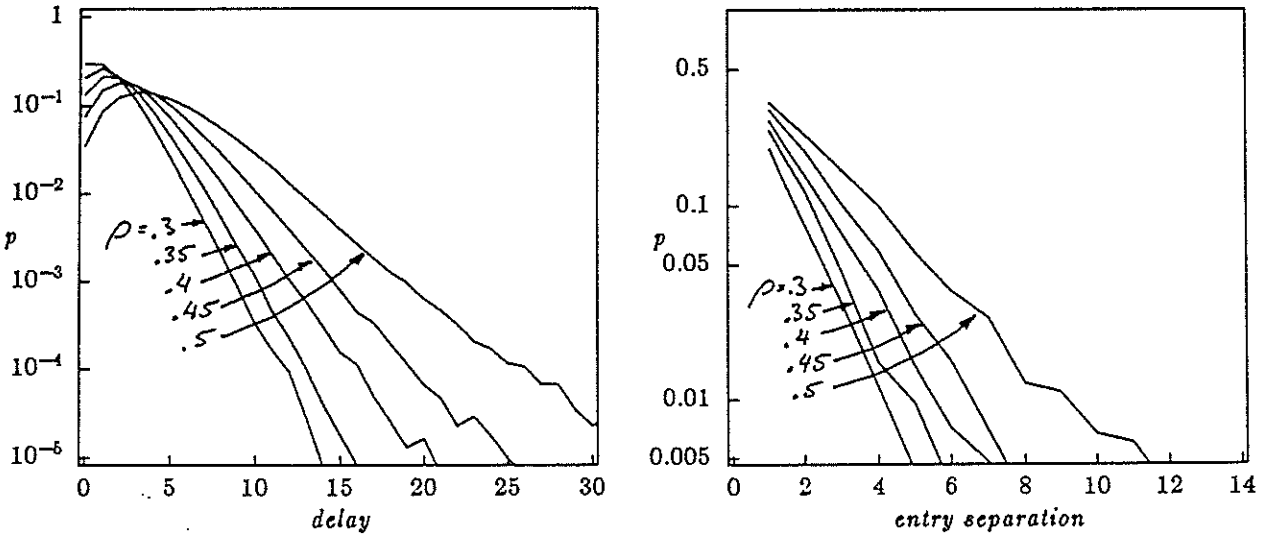

Figure 3.5: Distribution of Delay and Misordering Probabilities

fabric may be viable using only a small resequencing buffer. In particular, one could time-stamp packets on receipt from a link, then order them by time-stamp in the transmit buffer. If in addition, one never transmits a packet that isn't at least say 10 packet times old, the likelihood of misordering can be reduced to a very small level. While the logic to put packets in time-stamp order adds some complexity to the output buffer, the incremental cost is fairly small.

The plot on the right examines, the likelihood of packets being misordered more closely. In the simulations on which these results are based, all packets entering the copy network on input port 0 were treated as belonging to a point-to-point connection going to output port 0 of the routing network. The plot shows the fraction of all pairs of packets entering the copy network with a separation of x packet times that were misordered. So for example, for $\rho = 0.4$, about 1% of the pairs arriving six time units apart were misordered. If one extrapolates from these curves, one finds that less than one pair of packets in 10^6 that arrive with a separation of 20 packet times are misordered.

4. Prototype Hardware Design

Faculty	Jonathan Turner
Research Associate	Pierre Costa
Graduate Students	Neil Barrett
	Shabbir Khakoo
	Tony Mazraani
	George Robbert
	James Sterbenz
	Einir Valdimarsson

A prototype of a BPN switch module is being designed. The purpose of this prototyping effort is to provide a convincing demonstration of feasibility, allow detailed examination of implementation issues and provide a testbed for future experimental efforts at higher levels.

During the past year, we have designed several integrated circuits in order to obtain a detailed understanding of implementation issues and to deepen our experience with the design process. The first chips we designed had four bit wide internal data paths and implement the packet switch element (PSE) and broadcast translation circuit (BTC). A photograph on one of the chips used in the PSEappears as Figure 4.1. It contains the main data path for one of the two input circuits of the PSE, including a long shift register in which the packet is buffered. The four bit chips have been fabricated and are being tested now. Testing has been completed on one of the two BTC chips and while the tests have verified the logical correctness of the design, the yield was disappointing. Of 18 chips, only one functioned properly. At this point the reasons for the low yeild are unclear; we should have a clearer picture when the remaining chips have been tested. We are now designing a set of integrated circuits with eight bit wide data paths to be used in our prototype system. These circuits incorporate a number of fundamental improvements based on our experience with the trial chips and related performance studies. We expect to operate the new chips at a clock speeds of 40 to 50 Mb/s, giving data rates on the internal data paths between 300 and 400 Mb/s. Using

Figure 4.1: Main Data Path for Packet Switch Element

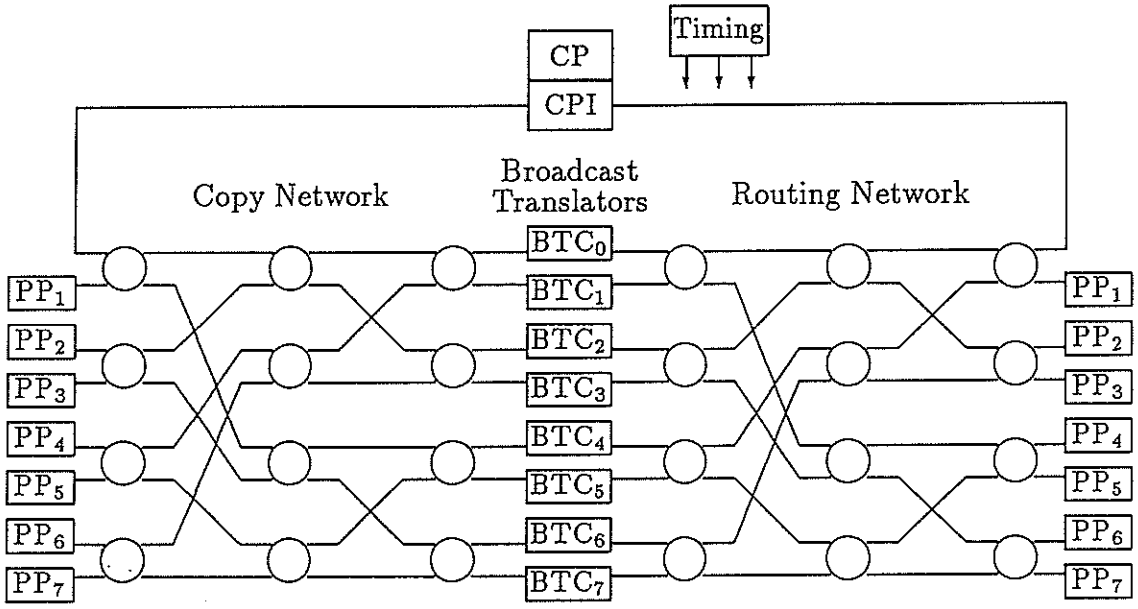

Figure 4.2: Prototype Switch Module

these chips, we plan to construct a prototype switching system supporting links speeds of up to 150 Mb/s.

The overall structure of the prototype packet switch is shown in Figure 4.2. The *Connection Processor* (CP), shown at the top of the figure, is a general purpose computer that provides overall control of the system, including connection establishment. The heart of the system is an eight port *Switch Fabric* (SF) comprising a *Copy Network* (CN), a set of *Broadcast Translation Circuits* (BTC) and a *Routing Network* (RN). A set of *Packet Processors* (PP) provide the interface between the SF and the high speed *Fiber Optic Links* (FOL) that are used to interconnect different switches. The CP communicates with the rest of the system through the CP *Interface* (CPI). The system is operated in a highly synchronous fashion, with global timing provided by the single timing circuit shown at the top of the figure.

Custom integrated circuits are being designed for the switch elements, BTCs and Packet Processors (PP). The BTC and switch element designs will require one chip apiece, the PP design will require two or three chips. A total of approximately 50 custom chips are required to implement the prototype switch module. At this time, trial designs have been completed for the BTCs and switch elements. These are being fabricated currently and will be tested on return. We are currently revising these designs, in part to meet the speed objectives for the prototype, and at the same time are developing detailed designs for the PP. Design of the CP interface and timing circuitry is also underway. The remainder of this chapter

describes the format of the packets used within the prototype and gives a top level description of the design of the various components.

4.1. Packet Formats

This section describes the formats of packets used in the switch. There are two primary packet formats: external and internal. Packets are carried in external format on the fiber optic links connecting switches, and in internal format within each switch. The PP translates between these two formats. Note that higher level processes may define additional packet formats; this section details only those fields that are of direct concern to the prototype hardware.

External Packet Format.

Each external packet is organized as a sequence of 8 bit wide words. Each packet contains exactly 76 words, the first 3 of which constitute the packet *header*. The last word of the packet is used for a frame checksum. When transmitted on the external transmission links, external packets are separated by a SYNC pattern that allows the receiver to identify packet boundaries. The meanings of the external fields are given below.

- *Packet Type* (PTYP). Identifies one of several types of packets, including ordinary data packet (1), test packets (2) and control packets (4).
- *External Logical Channel Number* (ELCN). Logical channel numbers are used to identify which connection a packet belongs to. For the prototype, only 256 distinct logical channels are recognized.
- *Information* (I). Normally contains user information. In the case of control packets, may contain additional control information. Individual words are denoted I[0], I[1], I[2], ... with I[0] being the first word of the I field.
- *Frame Check* (FC). The frame check is used to detect errors in the packet. A simple check sum over the first 75 bytes of the packets is used.

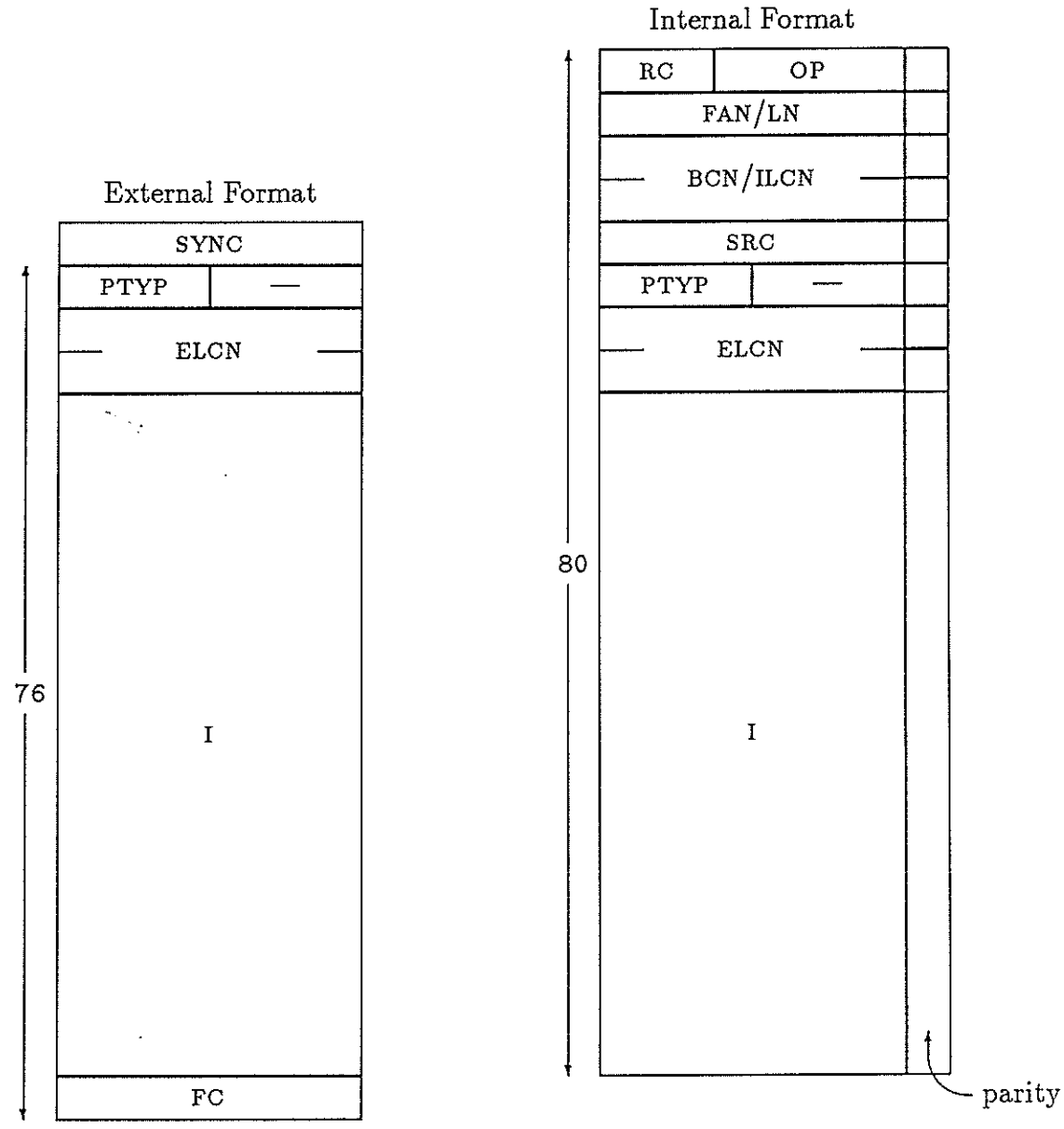

Figure 4.3: Packet Formats

Internal Packet Format

Each internal packet is organized as a sequence of nine bit wide words, including an odd parity bit. Each packet contains exactly 80 words, the first five of which constitute the packet *header*. The structure of the packet is shown in Figure 4.3. The meanings of the fields are given below.

- *Routing Control* (RC). This field determines how the packet is processed by the switch elements. The possible interpretations are listed below.

  0 *Empty Packet Slot*

  1 *Point-to-Point Data Packet*

  2 *Broadcast Packet*

  4 *Specific-Path Packet*

- *Operation* (OP) This field specifies which of several control operations is to be performed for this packet. The possible values of the field and the corresponding functions are listed below.

  0 *Vanilla Packet.* No control functions.

  1 *Read* LCXT *Block.* Directs PP to read a block of 16 entries from the Logical Channel Translation Table. I[0] specifies which block to read. The data is copied into I[1]–I[64] and the packet is returned to the CP.

  2 *Write* LCXT *Block.* Directs PP to write a block of 16 entries to the Logical Channel Translation Table. I[0] specifies the block to write. The data to be written is in I[1]–I[64].

  3 *Read* PP *Parameter Block.* Read the contents of the PP parameter block into I[1]–I[64] of the packet and return the packet to the CP.

  4 *Write* PP *Parameter Block.* Write the contents of I[1]–I[64] into the PP parameter block.

  5 *Switch Test Packet.* When received by a PP is returned to the SF with a new routing field. The new routing field is obtained by rotating the entire contents of the packet by five words.

  6 *Read* BTT *Block.* Directs BTC to read and return a block of 16 entries from the Broadcast Translation Table (BTT). I[0] field specifies which block to read. The data is copied into I[1]–I[64] and the packet returned to the CP.

  7 *Write* BTT *Block.* Directs BTC to write information into a block of 16 entries of the BTT. I[0] field specifies which block to write. The data to be written is in I[1]–I[64].

8 BTT *Single Entry Update.* Directs all BTCs in a given group to update an entry in their BTTs. BCN gives the broadcast channel number of the connection, I[0] is the new fanout, I[1] is the block of Broadcast Copy Indices that are to be updated (16 BCIs to a block) and I[2]–I[65] contains the block of 16 entries. Each BTC calculates its broadcast copy index, j, for the connection and if $\lfloor j/16 \rfloor$ equals the block number in I[1], it copies I[$4(j \bmod 16)$]–I[$(j \bmod 16)+3$] to BTT[BCN].

9 *Read* BCIT *Block.* Directs the BTC to read the contents of one block of 64 BCIT entries into I[1]–I[64] and return the packet to the CP. I[0] specifies the block to read.

A *Write* BCIT *Block.* Directs the BTC to write the information in words I[1]–I[64] into one block of the BCIT. I[0] specifies the block to write.

C–FF *Reserved.*

- *Destination* (DST). The interpretation of these three words depend on the value of RC.
  - *Fanout* (FAN). If RC = *Broadcast Packet*, the second word of the packet is taken to be the fanout, that is the number of switch fabric output ports that require copies of the packet.
  - *Broadcast Channel Number* (BCN). If RC = *Broadcast Packet*, the third and fourth words of the packet are taken to be the broadcast channel number. All packets within a particular multi-point channel have the same broadcast channel number. Only 256 distinct BCNs are recognized.
  - *Link Number* (LN). If RC = *Point-to-Point Packet*, the second word of the packet is taken to be the number of the outgoing link to which the packet should be delivered.
  - *Internal Logical Channel Number* (ILCN). If RC = *Point-to-Point Packet*, the third and fourth words of the packet are taken to be the internal logical channel number. This will become the external logical channel number when the packet exits the switch module.
  - *Specific Path Specification.* If RC = *Specific-Path Packet*, the three words of the DST field specify output ports for each of the three networks. The packet will be routed through each of these.
- *Source* (SRC). The number of the most recent PP through which the packet has passed. For vanilla packets, this will be the number of the link on which the packet entered the switch. For test packets it will be changed as the packet passes through different PPs.

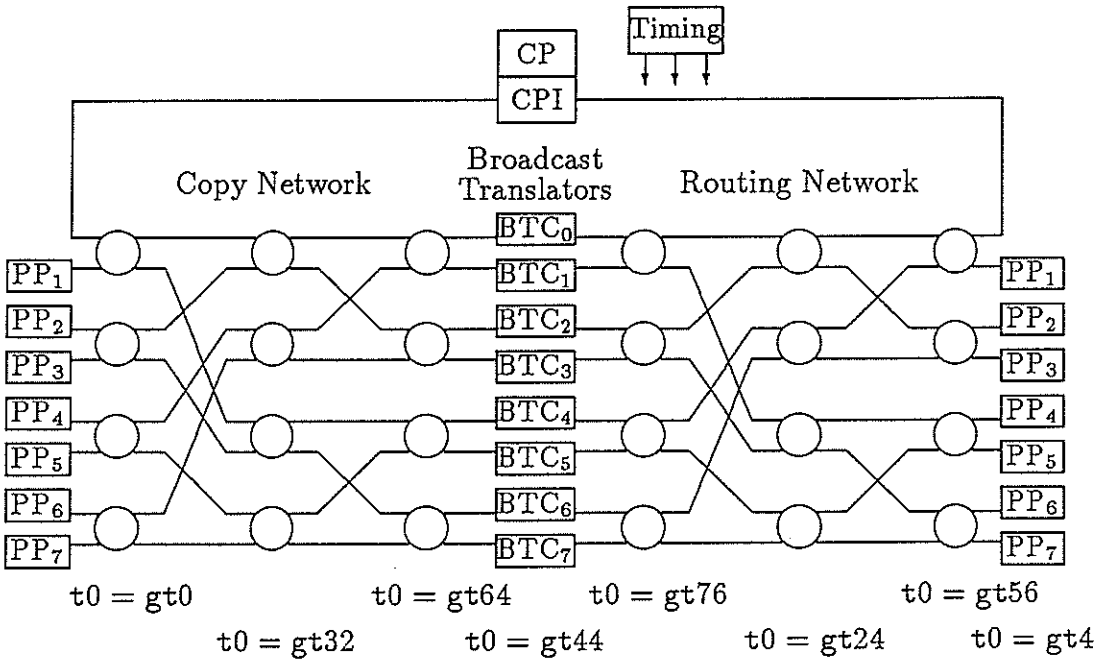

Figure 4.4: Local and Global Timing Relationships

4.2. Timing

The system is operated in a highly synchronous fashion. All packets are the same length and pass through the switch fabric in synchrony with one another. There is a global packet cycle that determines the timing of all events within the system. Incoming packets are received by the packet processors and synchronized to this packet cycle. Each cycle is referred to as an *epoch*. The length of an epoch is 84 clock times or 1.68 μs. This allows time for one packet to be processed and leaves a guard time of four clock periods between packets.

The global timing generator provides the base 50 MHz clock that drives the system plus a set of signals that define various instants within the global time reference. The notation gt$i$ is used to denote clock cycle i in the global time reference. By definition, gt0 is the time at which packets start to enter the leftmost stage of the copy network. The nodes of the switch fabric delay packets passing through them for exactly 32 clock times and the BTC delays packets for exactly 64 clock times. Consequently, packets pass from the leftmost stage of the copy network to the next stage at gt32 and so forth.

Every component in the system has a local time reference which is typically synchronized to the point in the global time reference at which that component

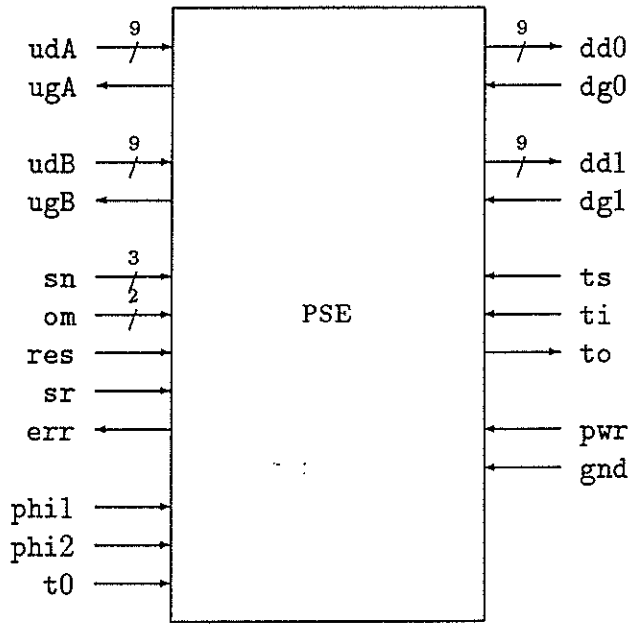

Figure 4.5: External Interface for Packet Switch Element Chip

can start to receive a packet on one of its input links. The notation t0 denotes the starting point of the epoch for a particular component's local time reference. Each of these local time references is synchronized to the global time reference as shown in Figure 4.4.

4.3. Packet Switch Element

The Packet Switch Element chip (PSE) is the 2×2 VLSI switch element used in the binary routing, copy and distribution networks. The PSE directs packets to one or both outputs based on packet type (point-to-point, broadcast, or specific-path), switch operation mode (routing, copy, or distribution), and the contents of the LN/FAN field.

The prototype version of the PSE differs from the initial trial chip in several important respects. The objective of these changes has been to eliminate constraints on the speed of operation of the PSE. The design outlined below is expected to run at clock speeds of 50 Mb/s, as opposed to 10 Mb/s for the trial chip. This improvement is due largely to changes in some basic design decisions. The most important change is to modify the way in which grant propagation is handled.

In the system as described in [89], grants are propagated from the output of the routing network back through the inputs to the copy network before packets can flow forward. This design makes best use of the nodes' internal buffers but places tight constraints on the number of clock cycles a node can delay a packet. In the new design a node makes decisions on its upstream grants independent of the status of the downstream grants. This change greatly relaxes the constraint on the number of clock cycles a node can delay a packet, which in turn makes it possible to increase the speed of the clock. Because this change reduces the effectiveness of node buffers, we have also decided to switch from a design with a single buffer per input to one with two buffers per input.

External Interface

The external leads of the switch element are shown in Figure 4.5 and described briefly below.

- *Upstream data leads* (udA,udB). Incoming data from upstream neighbors. Nine bits wide, including parity.
- *Upstream grants* (ugA,ugB). Grant signals to upstream neighbors. When asserted, grants permission to transmit packet on corresponding data leads during subsequent epoch.
- *Downstream data leads* (dd0,dd1). Outgoing data to downstream neighbors. Nine bits wide, including parity.
- *Downstream grants* (dg0,dg1). Grant signals from downstream neighbors. When asserted, grants permission to transmit packet on corresponding data leads during subsequent epoch.
- *Stage number* (sn). Three bit stage number. Each network has up to eight stages (columns), numbered from 0, with stage 0 being the last (rightmost) stage in a network.
- *Operating mode* (om). Two bit code identifying which of three possible operating modes the switch element implements. 1 for route, 2 for distribute, 3 for copy.
- *Reset* (res). Initialize all internal control registers; this causes any packets in the node to be discarded.
- *Soft Reset* (sr). Clear the error flag.

- *Error* (err). Report parity violation or other error.
- *Clock* (phi1,phi2). Two-phase, non-overlapping clock.
- *Start of Packet Cycle* (t0). Goes high when first word of packet is present on ud leads.
- *Test Shift* (ts). Shift lead for controlling shifting of test data.
- *Test In* (ti). Input lead for test data.
- *Test Out* (to). Output lead for test data.
- *Power* (pwr).
- *Ground* (gnd).

Global Operation

A single PSE circuit is used to implement the routing, copy and distribution networks. Packets are handled based on the information in the packet headers and either forwarded to the appropriate output (or outputs) or held until the required output(s) is available. The grant signals are used by nodes to control the arrival of packets from their upstream neighbors. In general, a node asserts a grant, allowing a new packet to arrive if it has an available buffer in which to store the packet. Each node can store up to four complete packets in its internal buffers.

PSE routing decisions are based on the operation mode and RC field, as specified below.

- For om =route; use bit sn of the LN field to select an output port, where sn is the stage number.
- For om =copy; if RC is broadcast, and FAN exceeds 2^{sn}, where sn is the stage number, send copies of packets to both output ports. If RC is specific-path, use bit sn of LN field to select an output port. Otherwise, distribute.
- For om =distribute; if RC is specific-path, use bit sn of LN field to select an output port. Otherwise, distribute.

When arbitrary routing choices can be made, the following policies are used to make decisions:

- Ties among input ports for a given output port are arbitrarily broken based on the last input port favored, to avoid individual starvation.

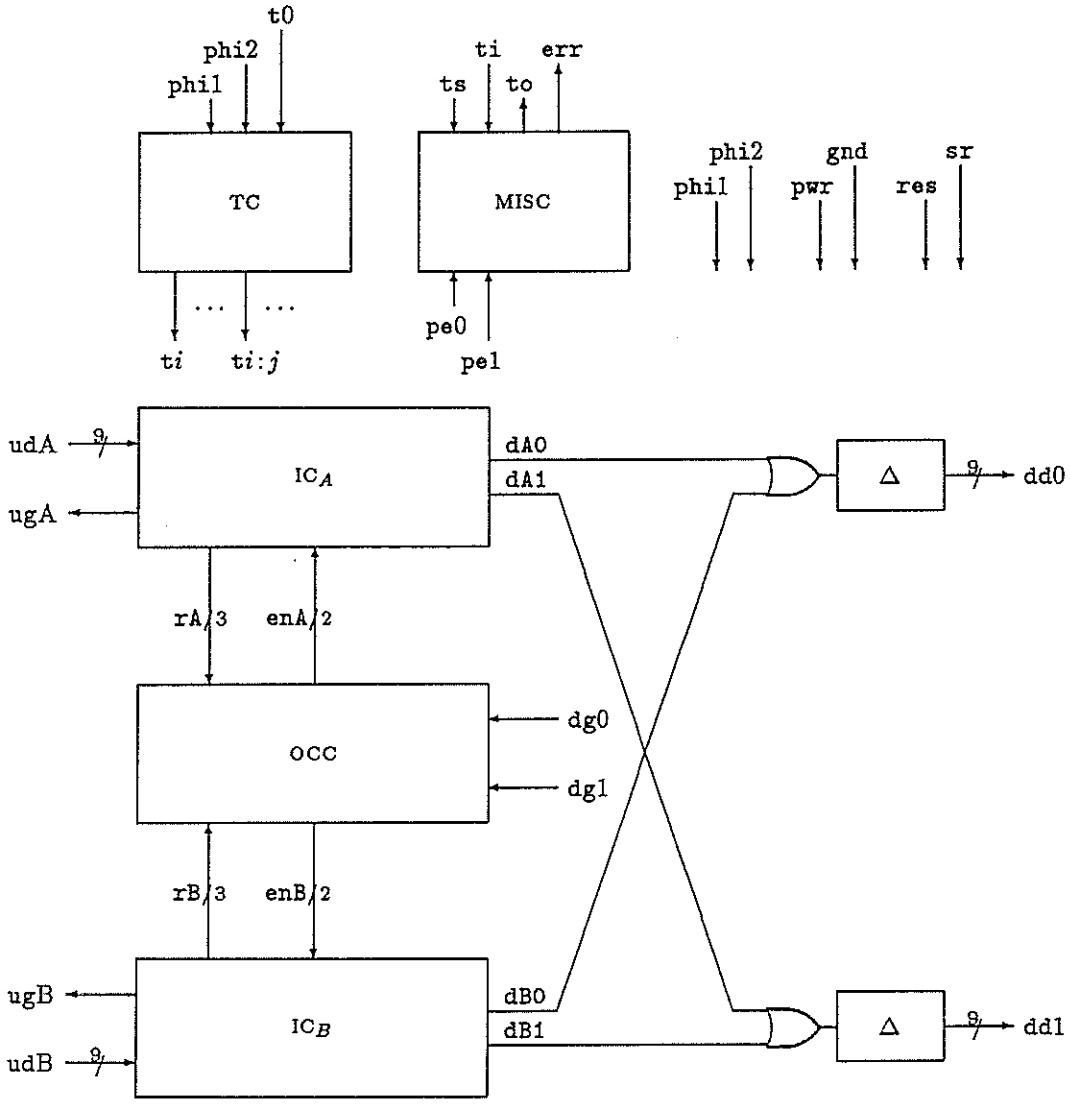

Figure 4.6: Block Diagram of Packet Switch Element Chip

- Packets that can proceed to either output are uniformly and arbitrarily distributed (all packets in distribution network, point-to-point packets and broadcast packets not replicated in copy network).
- Packets requesting both outputs in the copy network are favored over packets requiring only one.
- Packets requesting a specific output are favored over packets which can use either.

The clock period during which the first word of a packet appears on the upstream data leads is called t0 and in general, the clock period during which word i appears is called ti. The delay through a node is 32 clock times, or 640 ns. So, if an incoming packet can be switched through a node without buffering, the first byte will appear on the output at t32. Each node makes its upstream grant signals available at t32 in the node's frame of reference and holds the grant leads in that state until t32 of the subsequent cycle. Consequently, the grant signal is available to the upstream neighbor any time after t64 in the neighbor's frame of reference.

Internal Components

A block diagram of the PSE appears in Figure 4.6. The major components are described below.

- *Output Control Circuit* (OCC). The OCC arbitrates access to the two output ports, based on the downstream grant signals and port requests received from the input circuits. The port requests are given in the form of three bit request vectors, rA and rB; a value of 101 requests access to a output port 0, 110 requests output port 1, 111 requests both output ports and 100 requests a single output port, with either one being acceptable. The individual bits of these three bit codes are assigned the names rn, r1, and r0 with the suffix A or B included when necessary to designate a specific side. The response is given in the form of two bit enable vectors enA and enB; a value of 01 grants access to port 0, a value of 10 grants access to port 1 and a value of 11 grants access to both. The individual bits have the names en1 and en0.

- *Input Circuits* (ICA,ICB). There is one input circuit for each input port. Each IC includes two buffers large enough to hold a single packet, plus control circuitry to extract information from the packet header, generate the request vector for the OCC and use the resulting enable vector to make decisions on the disposition of the packet. It also modifies the packet header when necessary.

- *Timing Circuit* (TC). This circuit generates signals of the form ti and ti:j, for various values of i, j. Signal ti is high during clock period ti of the epoch; in particular it goes high during phi2 of the preceding clock cycle and goes low before phi2 goes high again. Signal ti:j is similar; it is high during ti and stays high through tj.

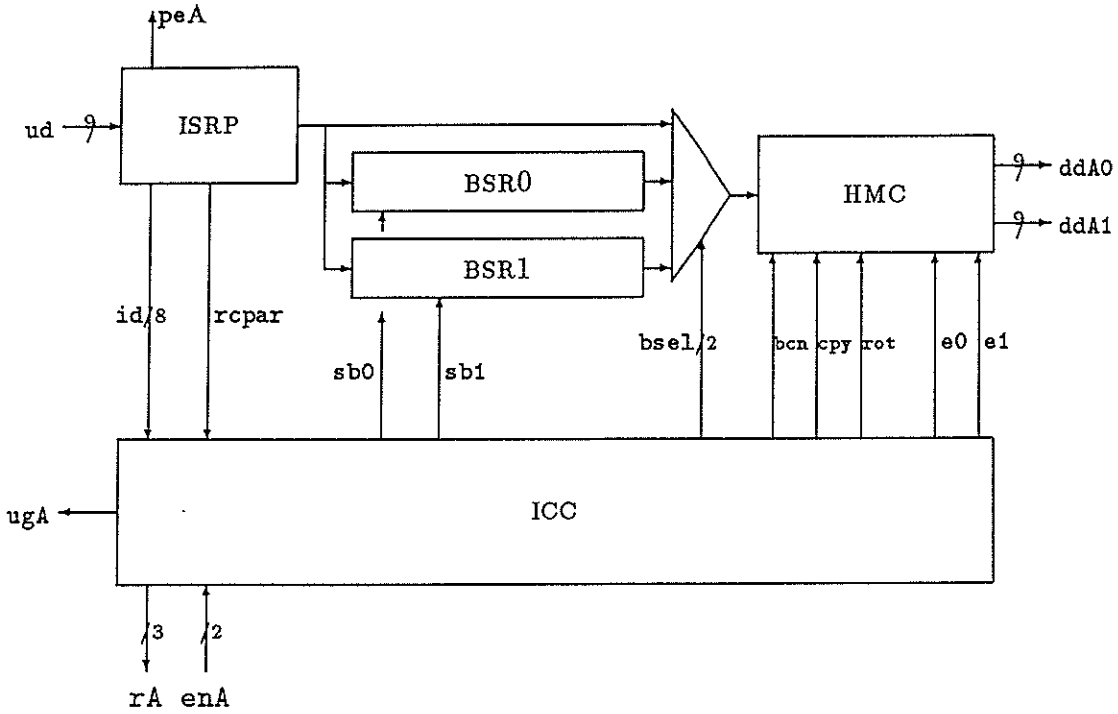

Figure 4.7: Input Circuit

Output Control Circuit. The Output Control Circuit is a PLA with ten inputs and six outputs plus two flip flops which store the values of a pair of tie-breaker variables. The flip flop ui specifies the input port that was most recently favored the last time a tie was broken; in particular, if input port A was most recently favored ui is 0, otherwise it is 1. Similarly, uo gives the number of the output port that was most recently used during an epoch when only one output port was used.

Input Circuit. The structure of the input circuit is shown in Figure 4.7. The main blocks are summarized below.

- *Input Shift Register* (ISR). The input shift register is a 20 stage static shift register with an output tap after the first stage and a parity checker. Packets are shifted into the ISR from the upstream data lines. The first stage of shift delay provides synchronization. The remaining stages allow time for control and routing decisions to be made by the input and output control circuits. The leads hd are connected to the output of the first shift register stage (data bits only) and provide access to the header information. The signal rcpar

is 1 if the parity of the first byte of the packet is incorrect. This is used to suppress copying of packets with incorrect routing control.

- *Input Control Circuit* (ICC). The ICC controls the flow of packets through the input circuit. It extracts and decodes header information from incoming packets and stores the decoded information for packets stored in the packet buffers. Using this information, it requests output ports from the OCC and based on the results, controls the flow of packets through the IC. It also generates the upstream grant signals. A more detailed description of the ICC appears below.

- *Buffer Shift Register* (BSR). Each BSR is a static 80 stage shift register, with the shift control provided by the ICC. A total of two BSRs are provided. The buffers are followed by a multiplexor also controlled by the ICC, which selects from one of the buffers or the bypass path.

- *Header Modification Circuit* (HMC). This component makes minor changes to the header as specified by the ICC. If the `rot` bit is asserted, words 1–3 of the routing field are rotated, with word 1 becoming word 3, word 2 becoming word 1 and word 3 becoming word 2. If the `cpy` bit is asserted the packet is sent to both output ports and the fanout fields of the copies are modified. The `bcn` bit determines which copy gets the "extra" when the fanout value is odd. The `en0` and `en1` signals control the passage of data onto the output links, with `en0` enabling output 0 and `en1` enabling output 1.

Figure 4.8 details the Input Control Circuit. The ICC contains several major components. The Header Register and Decode Logic (HRDEC) latches various fields of an incoming packet's header and decodes those fields into six bits. The `cpy` bit is 1 if the packet must be copied to both outputs. The `bcn` bit specifies which output receives the "extra" when the fanouts of the two copies are modified. The `rot` bit is 1 if words 1 to 3 should be rotated. The `rn` bit is 1, if there is an incoming packet. The `r0` bit is 1, if output 0 is required and the `r1` bit is 1, if output 1 is required.

The buffer control registers BCREG0, BCREG1 store the decoded control bits for packets stored in BSR0 and BSR1. Each BCREG has six data inputs and six tri-state data outputs. In addition, the `rn` signal has a non tri-state output. The BCREGs have two control inputs. If `latch` is high at `t16`, the input control bits are latched. When `sbc` is high, the six stored bits are placed on the tri-state outputs.

The PLA at right provides overall control of the ICC. At the start of the epoch it selects one of the HRDEC or BCREGs to provide a request vector to the Output Control Circuit. Then, based on the response, it controls the steering of data

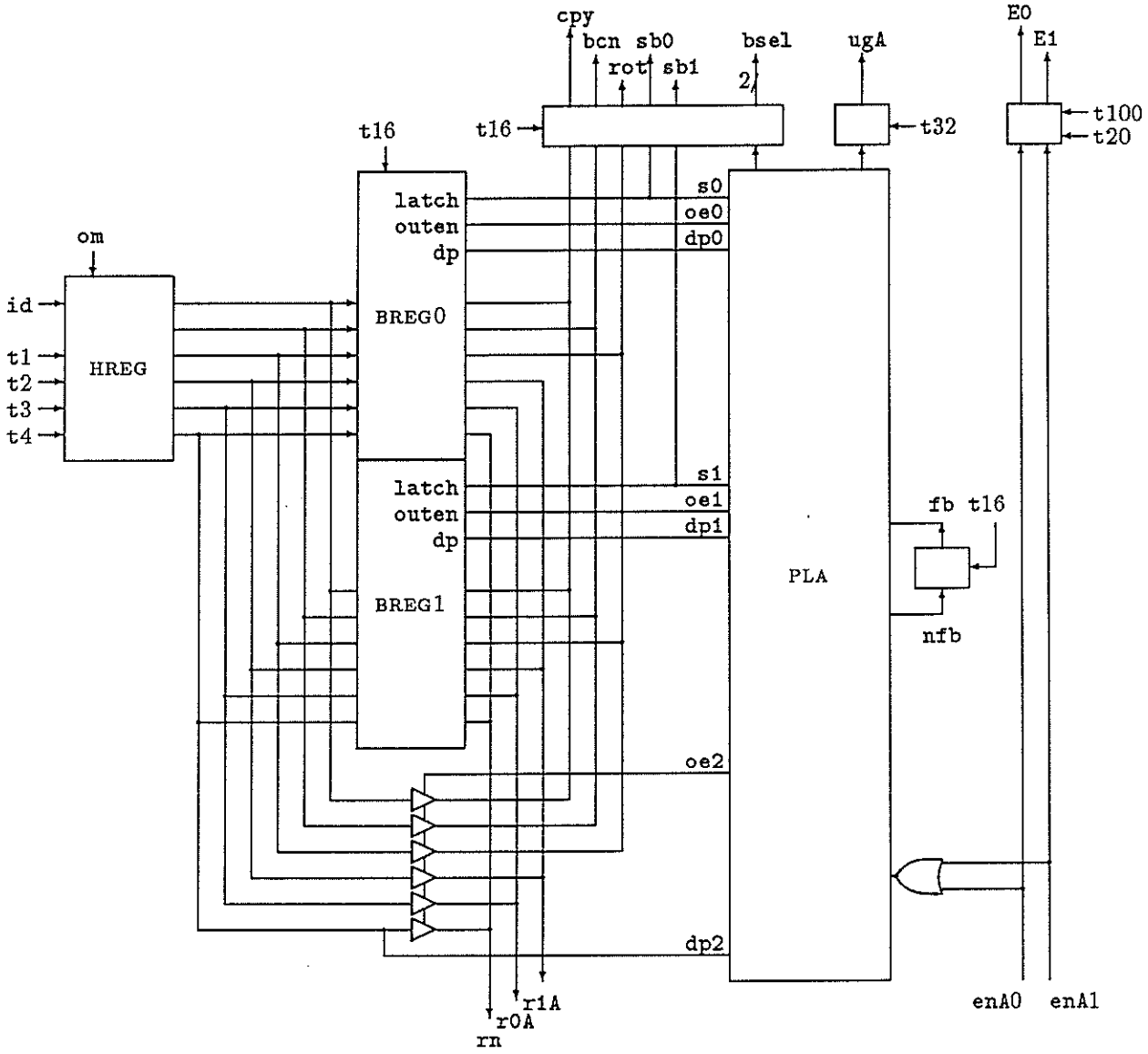

Figure 4.8: Input Control Circuit

to and from the buffer shift registers and controls updating the BCREGs. It also generates the upstream grant signals. The latches at the top of the figure simply hold the control signals for the duration of the epoch and are latched at the times indicated.

The FIFO to the right of the PLA is used to keep track of the order of packets stored in the buffers. The FIFO is two bits wide and two deep. The output of the FIFO gives the number of the buffer containing the packet which is to be output first.

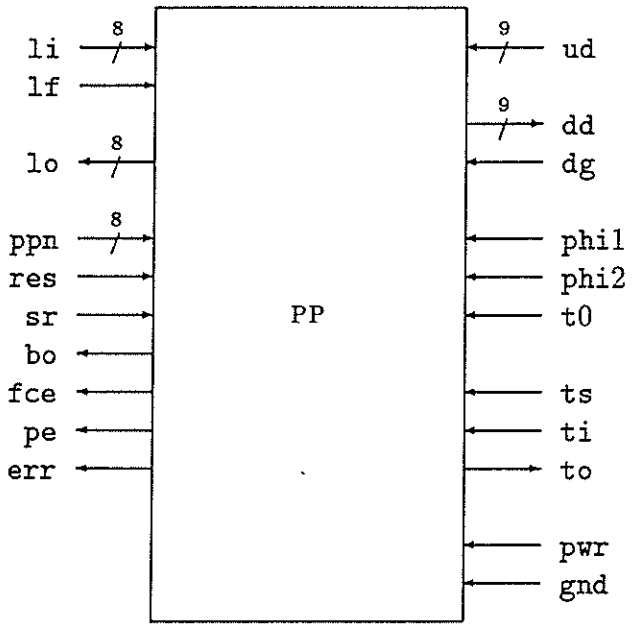

Figure 4.9: External Interface for Packet Processor

4.4. Packet Processor

The Packet Processors (PP) form the interface between the external fiber optic links and the switch module's internal data paths. They perform all the link level protocol functions, including the determination of how packets are routed.

External Interface

The external leads of the packet processor are shown in Figure 4.9 and summarized briefly below.

- *Upstream data from SF* (ud). Data from switch fabric. Nine bits wide including parity.
- *Downstream data to SF* (dd). Data to switch fabric. Nine bits wide including parity.
- *Downstream grant from SF* (dg). When asserted, allows PP to transmit packet in subsequent epoch.

- *Data from link* (`li`). Data stream from FOL. Eight bits wide.
- *Link framing* (`lf`). Link framing signal. Goes high at start of packet.
- *Data to link* (`lo`). Data stream to FOL. Eight bits wide.
- PP *number* (ppn). Eight bit number identifying PP.
- *Reset* (`res`). Resets the entire PP when it is asserted, causing any packets stored in the PP to be discarded.
- *Soft reset* (`sr`). Resets PP error flags.
- *Buffer overflow* (`bo`). Asserted whenever a packet is discarded by the PP due to buffer overflow.
- *FC error* (`fce`). Asserted when the PP receives a packet containing a bad frame check field.
- *Parity error* (`pe`). This signal is asserted whenever the PP detects a parity error.
- *Error* (`err`). Asserted when the PP detects any error, including those signaled above.
- *Clock* (`phi1`,`phi2`). Two-phase, non-overlapping clock.
- *Start of epoch* (`t0`). Goes high when first word of packet is present on ud leads.
- *Test shift* (`ts`). Shift lead for controlling shifting of test data.
- *Test in* (`ti`). Input lead for test data.
- *Test out* (`to`). Output lead for test data.
- *Power* (`pwr`).
- *Ground* (gnd).

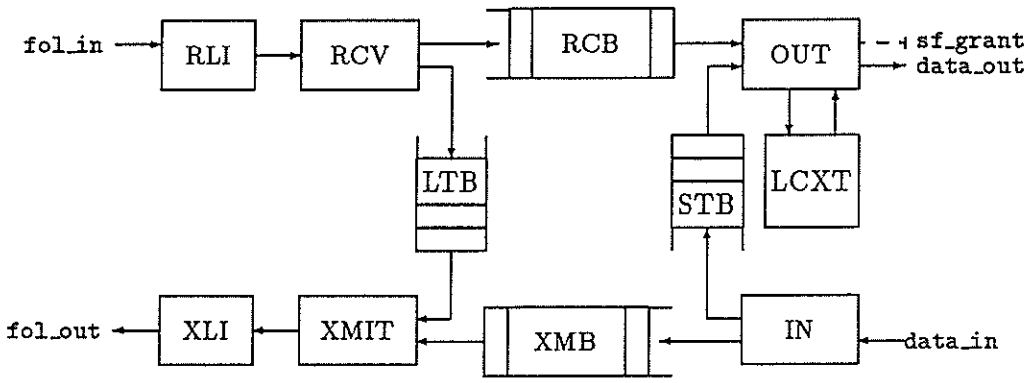

Figure 4.10: Packet Processor Circuit

Global Operation

The processing of packets by the PPs is determined by the PTYP field for external packets (received from FOL) and by the OP field for internal packets (received from SF).

- *External Data Packet.* Converted to internal format, with the routing field determined by a lookup in an internal *Logical Channel Translation Table* (LCXT). The packet is then transmitted to the switch fabric.
- *External Link Test Packet.* The PTYP field is changed to external data packet, and the packet is returned on the outgoing FOL.
- *External Control Packet.* Converted to internal format, with the LN field set to 0 and the RC set to ordinary data packet. Transmitted to SF.
- *Internal Data Packet.* Converted to external format, with contents of internal LCN field transferred to external LCN field. Transmitted to FOL.
- *Switch Test Packet.* The RC field is set to 0 and then the first five words of the packet are moved to the end of the packet and everything else shifted up. In other words, the whole packet is rotated by five words. The packet is then returned to the SF.

Internal Components

A block diagram of the PP appears in Figure 4.10. The various components are described briefly below.

- *Buffers.* The PP contains four packet buffers. The *Receive Buffer* (RCB) is used for packets arriving from the FOL and waiting to pass through the SF. The *Transmit Buffer* (XMB) is used for packets arriving from the SF that are to be sent out on the FOL. The *Link Test Buffer* (LTB) and *Switch Test Buffer* (STB) provide paths for test packets used to verify the operation of the FOL and SF respectively. The RCB has a capacity of 16 packets, the XMB has a capacity of 32. The LTB and STB can each hold two packets. Together, the four buffers require a total of about 35 Kbits of memory.

- *Receive Link Interface* (RLI). Converts the incoming optical signal to an eight bit electrical format, synchronized to the local clock.

- The *Receive Circuit* (RCV). Checks incoming packets for errors, adds parity, strips off FC, routes test packets to the LTB and other packets to the RCB.

- *Output Circuit* (OUT). Adds five bytes of header information to the front of each packet received from the RCB. Performs logical channel translation and sends packets to the SF. Also reads switch test packets, LCXT read/write packets and PP parameter block read/write packets from the STB and processes them appropriately.

- *Logical Channel Translation Table* (LCXT). Lookup table used to translate an incoming logical channel number to the routing information needed by the switch fabric.

- *Input Circuit* (IN). Routes internal data packets to the XMB, removing the first five bytes of header information and routes all other packets to the LTB. Performs the rotation required for switch test packets.

- *Transmit Circuit* (XMIT). Takes packets from the XMB, adds the SYNC field, strips parity and computes the frame check. Also processes test packets from the LTB.

- *Transmit Link Interface* (XLI). Converts from eight bit electrical format to optical format.

The RLI and XLI will be implemented separately from the integrated circuit that implements most of the PP functions. These will be implemented using commercially available components. We are currently evaluating the TAXI chip set for this purpose. The circuits that make up the bulk of the PP chip can be divided into three basic types; synchronous streams processors (SSP), packet buffers and lookup tables.

An SSP is a circuit that has several typed I/O ports over which it sends and receives data in a highly synchronous fashion and which transforms the contents

of certain fields as the data passes through. These can be readily described in a high level specification language that we have designed and can then be compiled into a special-purpose circuit implementing the given specification. Some details of the language and the translation process are given in the following chapter of this report. The RCV, XMIT, OUT and IN circuits are all examples of SSPs and our plan is to implement them using the automatic translation process just described.

A packet buffer is a memory designed to hold packets. The XMB, RCB, LTBand STB are all specific instances. As with the SSPs, we are designing a program that will automatically generate a specific packet buffer given a description of its desired characteristics (packet length, data path width, timing requirements, etc.). This program will be used to design the particular packet buffers required for the PP. This is described further in the following chapter.

A lookup table is also a memory, but has a somewhat different interface than a packet buffer, since it must offer random access to the table entries rather than implementing a buffer. We will soon begin work on a similar program that generates a lookup table from a given specification. The LCXT is the only lookup table in the PP, but other chips in the system also contain lookup tables, so we felt it advantageous to design a general tool to create them.

4.5. Broadcast Translation Circuit

The Broadcast Translation Circuit (BTC) provides unique addresses for each of the copies of a broadcast packet replicated by the copy network. It also provides a hardware assist for updating the table of new addresses for a single broadcast channel.

External Interface

The external leads of the BTC are shown in Figure 4.11 and described briefly below.

- *Upstream data leads* (ud) Incoming data from upstream neighbors. Nine bits wide including parity.
- *Downstream data leads* (dd) Outgoing data to downstream neighbors. Nine bits wide including parity.
- *Reset* (res). Resets the entire BTC when it is asserted, causing any packets stored in the BTC to be discarded.
- *Soft reset* (sr). Resets PP error flags.

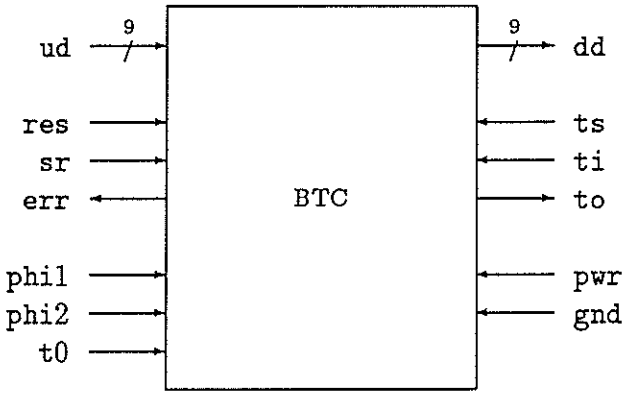

Figure 4.11: External Interface for BTC

- *Error* (err). Asserted when the BTC detects any error.
- *Clock* (phi1,phi2). Two-phase, non-overlapping clock.
- *Start of epoch* (t0) Goes high when first word of packet is present on udleads.
- *Test shift* (ts). Shift lead for controlling shifting of test data.
- *Test in* (ti). Input lead for test data.
- *Test out* (to). Output lead for test data.
- *Power* (pwr).
- *Ground* (gnd).

Global Operation

The BTC's operation depends upon the type of packet passing through it.

- *Ordinary Data Packet.* These packets are passed straight through the main shift register unchanged.
- *Broadcast Data Packet.* The routing field is replaced with a new field selected from an internal *Broadcast Translation Table* (BTT). The new field is selected using the BCN of the packet.
- *Read/Write* BTT *Block.* These two packet types are used for updating the BTT in large chunks and for reading it for auditing and testing purposes.

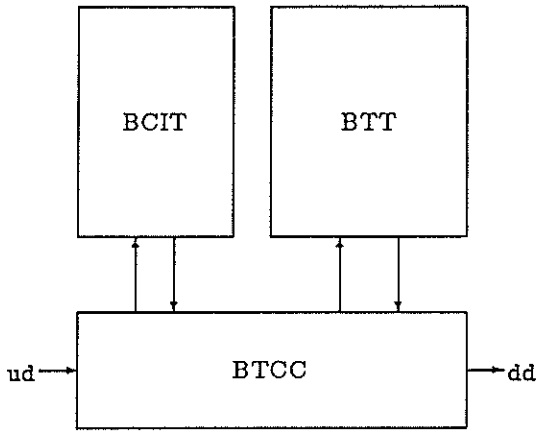

Figure 4.12: Block Diagram of Broadcast Translation Chip

- *Read/Write* BCIT *Block.* These packets read and update the entire broadcast copy index table.
- *Single Entry Update.* This packet supplies a set of new routing fields for this BCN. The BTC chooses which one to write into the BTT depending on the value of the *broadcast copy index*, which is determined from an internal *Broadcast Copy Index Table* (BCIT). I[0] specifies the new fanout for the connection, I[1] specifies which group of routing fields are contained in the packet. I[2]–I[65] contains the routing fields.
- All other packets are passed through like ordinary data packets.

Internal Components

The internal components that make up the BTC are similar to those in the PP. In particular, the BTC can be described as a single SSP along with two lookup tables, one for the BTT and the other for the BCIT. This structure is illustrated in Figure 4.12 and the components are described briefly below.

- Broadcast Translation Table (BTT). This is the table used to store the routing information for packets belonging to to multipoint connections. The prototype version will consist of 256 entries, each four bytes long, plus parity.

- Broadcast Copy Index Table (BCIT). This table is used to compute the broadcast copy index of the BTC, for use in updating the BTT when the fanout of a connection changes. It contains 512 single byte entries, plus parity.
- BTC Control (BTCC). This is the control circuit that processes all received packets using the BTT and BCIT.

Our plan is to implement the two lookup tables using a lookup table generator, which is described briefly in the next section. The BTCC will be implemented as a synchronous streams processor using the program we are developing for that purpose.

5. Tools for Design of Communication Circuits

Faculty	Jonathan Turner
Research Associate	Pierre Costa
Graduate Student	Neil Barrett
	George Robbert
	James Sterbenz
	Einir Valdimarsson

The implementation of the prototype packet switch for the ACS project will require several custom VLSI chips. In the past year, we have designed preliminary versions of two chips and gained considerable insight into both the impact of low-level design issues on architecture and on the design process itself. In this chapter, we review our efforts at constructing special purpose tools to reduce the amount of manual effort associated with the design of integrated circuits. Our most ambitious effort in this area is the design of a circuit generator for a particular class of circuits that arise frequently in our work and which we refer to as synchronous streams processors. We have also begun several other projects which seek to automate or partially automate other design tasks.

5.1. Synchronous Streams Processors

Many of the circuits required in a fast packet switching system contain a large number of functional modules that accept packets on one or more input ports, modify the packet headers and transfer the packets to one or more output ports. The various modules operate in tight synchronism because of the use of fixed length packets. We have come to view each of the specific modules as special cases of a generic *synchronous streams processor* or SSP.

An SSP, is a module with one or more typed input and output *ports*, a local clock synchronized by external timing signals and a function which can be described in

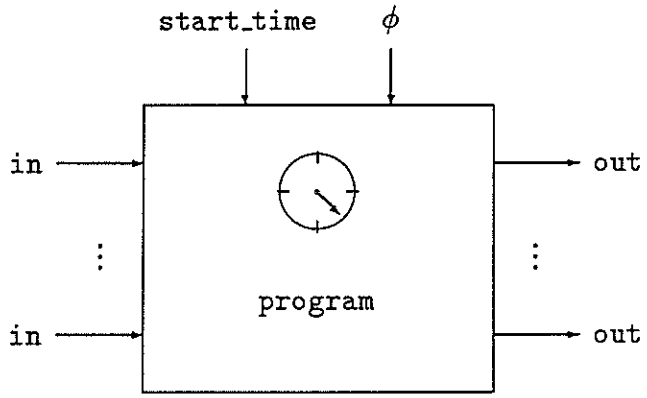

Figure 5.1: Generic Synchronous Stream Processor

a style similar to a conventional programming language (see Figure 5.1). The local clock is set to 0 when the external synchronization signal `start_time` is received, and is then incremented on every tick of the global system clock ϕ. The period between successive `start_time` signals is referred to as an *epoch* and all events happen at specific times during an epoch.

Each port has a type associated with it. The base type is *bit* and complex types can be constructed using arrays and structures. As an example, the following declaration defines the format of an internal packet in the BPN prototype.

```
struct ipfmt {
    bit op[5], rc[3];
    bit fan_ln[8];
    bit bcn_ilcn[16];
    bit src[8];
    bit fill[4], ptyp[4];
    bit elcn[16];
    bit info[72][8];
};
```

In addition to its type, a port has a *start time* and a *width*. The start time defines at what point in each epoch the data item defined for that port begins to appear on the port. The width of the port defines the number of bits available to carry the data. These pieces of information are sufficient to define when in an epoch and where on a port, specific items of data appear. This allows a designer

to describe the function of an SSP in terms of actions on port fields, ignoring the details of timing and bit location.

We now turn to a simple example to illustrate how an SSP can be described. The circuit we will describe combines the functions of the IN and XCVT circuits of the packet processor. It accepts an input packet from the switch fabric on one port and based on the control field directs the packet to either the XMB or STB. For test packets, the RC field is set to 0 and the packet is rotated by five words. In addition, packets sent to the XMB are placed in a format that is intermediate between the internal and external formats; basically, it is the internal format with the first five words removed. As part of this transformation, the logical channel number in the ILCN field is copied to the ELCN field.

```
struct hpfmt {              // Hybrid packet format
    bit fill[4], ptyp[4];
    bit elcn[16];
    bit info[72][8];
};
module inxcvt(port[8] struct ipfmt  <sfp@0, >ltbp@2,
              port[8] struct hpfmt >xmbp@6 )
    if sfp.op == OP_DATA ->
        xmbp.fill = sfp.fill;
        xmbp.ptyp = sfp.ptyp;
        xmbp.elcn = sfp.bcn_ilcn;
        xmbp.info = sfp.info;
     | sfp.op == OP_STEST ->
        ltbp:(0 .. 74*8) = sfp:(5*8 .. 79*8);  /* Rotate */
        ltbp:(75*8 .. 79*8) = sfp:(0 .. 4*8);
        ltbp:(75*8 .. 75*8+2) = 0;             /* Clear rotated RC */
     | sfp.op != OP_DATA && sfp.op != OP_STEST ->
        ltbp = sfp;
    fi;
end
```

The example module defines three eight bit wide ports; sfp is an input port (indicated by <) carrying data in mpfmt, starting at time 0; ltbp is an output port, also carrying data in mpfmt, starting at time 2; xmbp is an output port carrying data in hpfmt, starting at time 6. The program specifies the appropriate action based on the OP field of the incoming packet. The assignments define the contents of the various output fields. Unspecified output fields are filled with zeros. The notation port:(i..j) refers to a range of bits on the given port, providing a simple low level mechanism for rearranging large blocks of a packet. Notice that the only

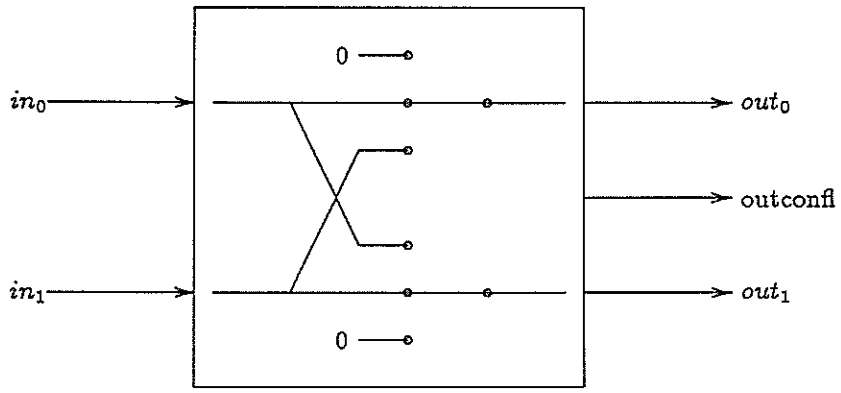

Figure 5.2: Unbuffered Switch Element

times that must be defined explicitly are the times at which data items start to flow across ports.

Our second example is a 2×2 unbuffered switch element that could be used in a self-routing switching network. A block diagram of the switch element is shown in Figure 5.2. Each switch element uses the first (low order) bit of the address (addr) to select which port to output the packet on. The switch element also rotates the bits of the address to place the next address bit in the correct place for the next switch element to use in routing this packet. It may occur that both input packets to the switch element request the same output port. If one of these does not contain useful data (ptyp = `PT_NONE`), no problem exists since that packet is dropped anyway. However, if both packets contain valid data (ptyp = `PT_PPOINT`), the "straight through" packets are given priority. That is, if both packets request output 0, the packet on input 0 is passed on and the packet on input 1 is discarded. Whenever a packet containing valid data is discarded, an error signal is asserted. The specification of this circuit is shown in Figure 5.3.

This example illustrates some simple field re-arrangement and the use of the C preprocessor to ease the function description. The macro `Shipout` illustrates several features of the specification language. It handles "assigning" the given input to the given output. It also rotates the address field of the packet to set up the address bits for the next routing element. The first line sets up the entire packet on the destination port. The next two lines override this for the address field, replacing it with a rotated copy. This example also shows the concurrent use of the **if**. Note that the guards are not mutually exclusive. If several of the guards evaluate to **true**, the statements associated with all of them are "executed." For example, in the case of routing two "straight through" packets ($in_0 \rightarrow out_0$ and $in_1 \rightarrow out_1$), the first two guards are both true and both of their actions are

```
typedef struct {
    bit ptyp[2];        /* packet type              */
    bit addr[6];        /* destination address      */
    bit data[72][8];    /* data in packet           */
}packet;

/* various packet types */
#define PT_NONE     0   /* no data in this packet  */
#define PT_PPOINT   1   /* point-to-point packet   */

#define Shipout(in,out) \
        out = in; \
        out.addr[5] = in.addr[0]; \
        out.addr:(0 .. 4) = in.addr:(1 .. 5);

router (port[8] packet
                <in0@0,  <in1@0,        /* input data ports  */
                >out0@4, >out1@4;       /* output data ports */
          port[1] bit >outconfl@4)      /* output conflict   */

{
    outconfl = 0;
    if (in0.addr[0] == 0 && in0.ptyp != PT_NONE) ->
        Shipout(in0,out0);
    |  (in1.addr[0] == 1 && in1.ptyp != PT_NONE) ->
        Shipout(in1,out1);
    |  (in0.addr[0] == 1 &&
              (in1.addr[0] != 1 || in1.ptyp == PT_NONE)) ->
        Shipout(in0,out1);
    |  (in1.addr[0] == 0 &&
              (in0.addr[0] != 0 || in0.ptyp == PT_NONE)) ->
        Shipout(in1,out0);
    |  (in0.ptyp != PT_NONE && in1.ptyp != PT_NONE &&
        ((in0.addr[0] == 0 && in1.addr[0] == 0) ||
         (in0.addr[0] == 1 && in1.addr[0] == 1))) ->
            outconfl = 1;
    fi
}
```

Figure 5.3: Specification of Unbuffered Switch Element

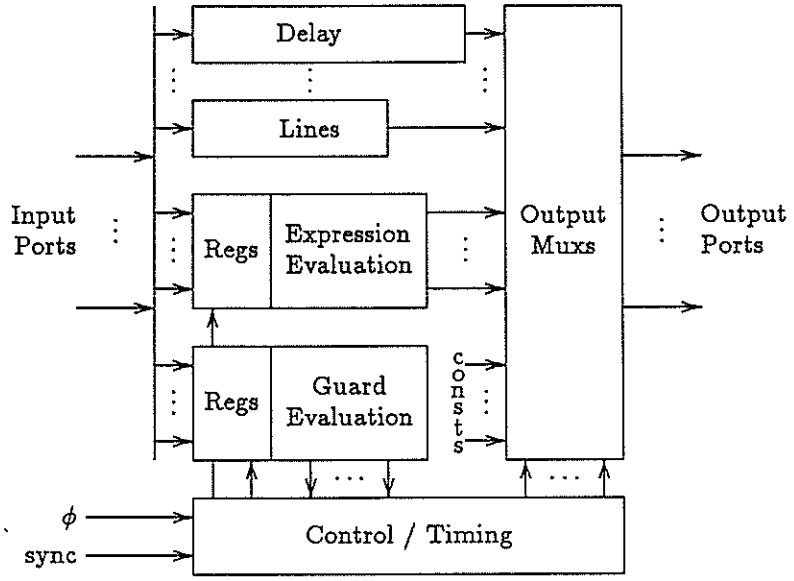

Figure 5.4: Target SSP Architecture

performed. It also uses the feature that all unspecified outputs are 0. Thus, if no packet is routed to an output, an all zero packet (ptyp = PT_NONE) is output.

The simple paradigm of typed, synchronous ports can also be used to define control signals that must be exchanged between different modules. This allows us to define more complicated interfaces such as are required on the output side of various buffers. Modules can also have local variables that can be used to save information across time epochs.

5.2. Implementation of SSPs

SSPs that perform simple functions, as are typical in the Packet Processors, fit nicely into a common architecture illustrated in Figure 5.4. This architecture supports several input and output ports of varying widths. Input ports connect to a common input bus and outputs to a common output bus. Between these are a set of processing elements (PE). Each processing element has data registers which latch selected input fields. The *guard evaluation logic* in addition, contains the combinational logic to evaluate the conditions in if-statements. The *expression evaluation logic* evaluates expressions on the right side of assignments. The *delay lines* are used to delay the passage of certain fields to the output bus in order to satisfy timing constraints. The control and timing element provides timing signals for latching input data and controlling access to the output bus.

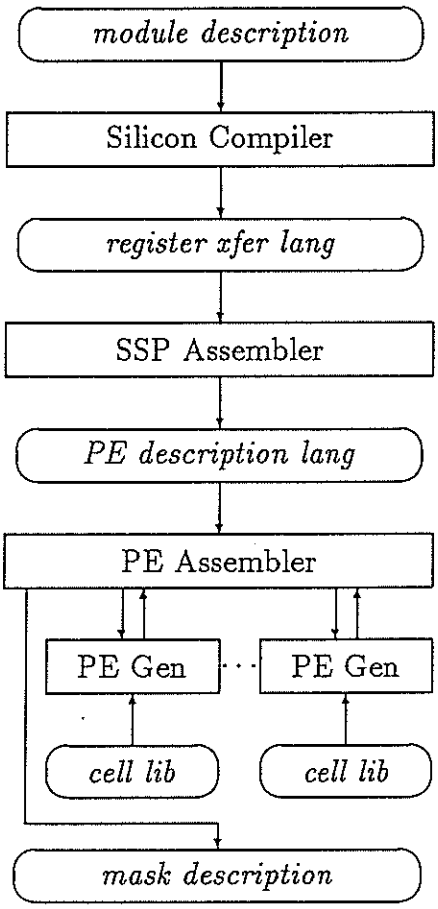

Figure 5.5: Structure of SSP Generator

We have developed a circuit generator that takes a high level description of an SSP and creates a circuit implementing it, by tailoring the target architecture. We have divided the translation into several parts as illustrated in Figure 5.5. The *compiler* takes the high level module description and translates it to a simple *register transfer language*. This is further processed by an SSP*assembler* which translates it further to a PE description language. This is further processed by a PE *assembler* which generates the actual mask-level description of the module, using a library of standard cells and a set of PE generators, which include existing tools such as a PLA generator. An initial version of the compiler has been written which is capable of generating most of the circuits needed within the project. Several extensions are planned as well.