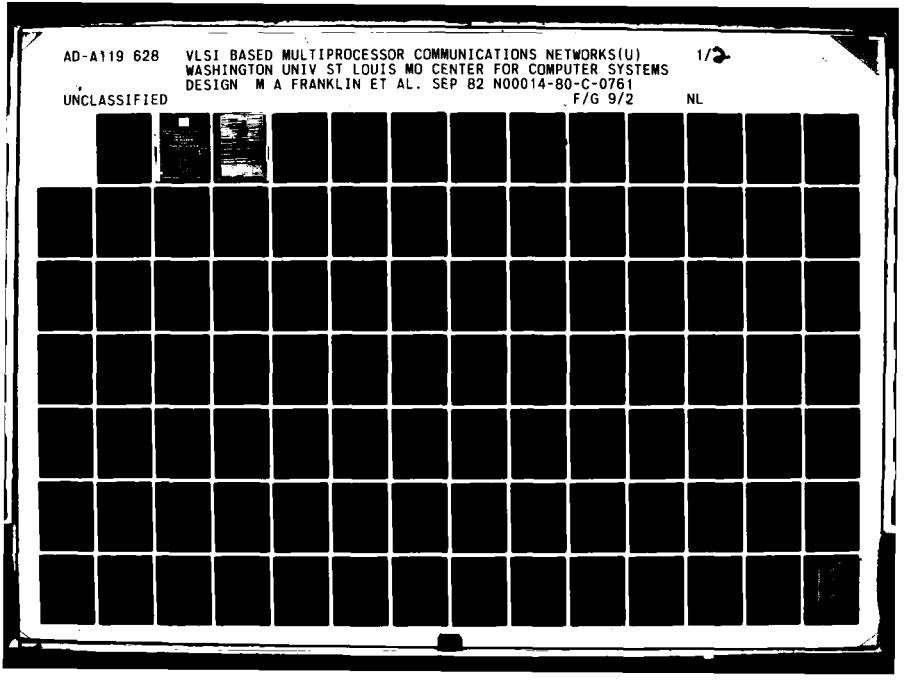
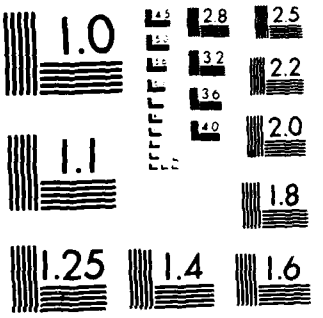
AD-A119 628 VLSI BASED MULTIPROCESSOR COMMUNICATIONS NETWORKS(U) 1/2
WASHINGTON UNIV ST LOUIS MO CENTER FOR COMPUTER SYSTEMS
DESIGN M A FRANKLIN ET AL. SEP 82 N00014-80-C-0761

UNCLASSIFIED F/G 9/2 NL

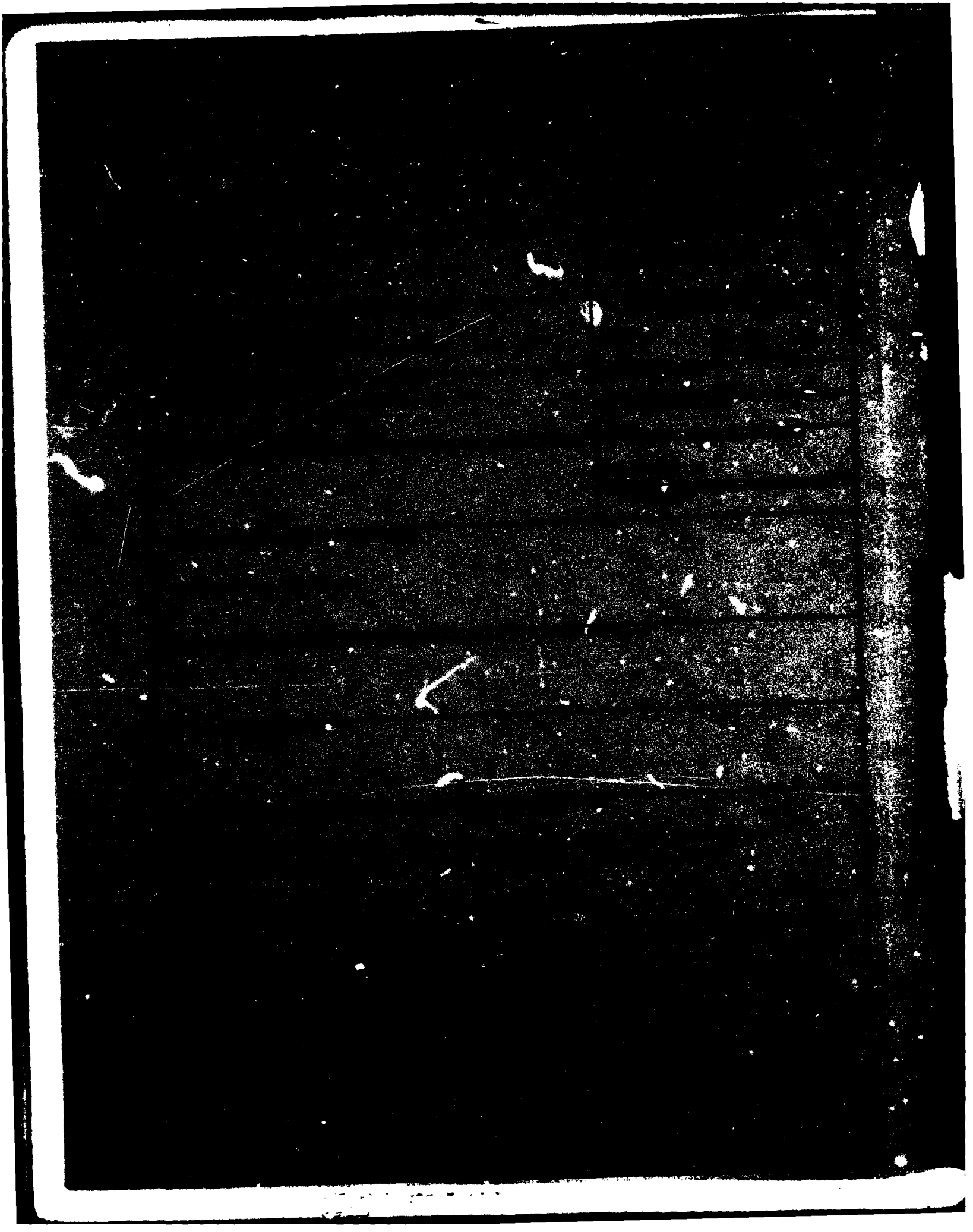




WASHINGTON UNIVERSITY

SCHOOL OF ENGINEERING AND APPLIED SCIENCE

[redacted]



ANNUAL PROGRESS REPORT

for

THE OFFICE OF NAVAL RESEARCH

NR#:375-033 CONTRACT #:N00014-80-C-0761

Submitted

by

Washington University

Center for Computer Systems Design

St. Louis, Missouri 63130

Contract Title

VLSI BASED MULTIPROCESSOR COMMUNICATIONS NETWORKS

Principal Investigators

Mark A. Franklin
Professor: Electrical Engineering

Donald F. Wann
Professor: Electrical Engineering

Date

September 1982

TABLE OF CONTENTS

VLSI BASED MULTIPROCESSOR COMMUNICATIONS NETWORKS

Progress Report: Year 2

Mark A. Franklin and Donald F. Wann

1. Introduction

This document is the Annual Progress Report for the Office of Naval Research contract number N00014-80-C-0761, (NR#: 375-033) entitled "VLSI Based Multiprocessor Communications Networks". The contract began on September 1,1980 and was approved on scientific/technical grounds for a duration of three years. Incremental funding was approved for year three of the research and this work has just begun. This report documents research progress and major achievements during the second year of the contract. Research plans for year three are also presented.

Need for a research effort in the area of VLSI based communication networks was discussed in depth in the original proposal and will therefore only be briefly reviewed here.The research is motivated by four basic factors:

1- Until recently, increases in computational power have resulted principally from the increased performance associated with advances in component technology. In the future, as we push against the basic physical limitations of various component technologies, large performance increases based on such advances are likely to be increasingly costly. Another approach to achieving high computational performance is through the use of parallel processing techniques. The research discussed in this report is central to the further development of these techniques. It focuses on tightly coupled multiple processor computer systems consisting of large numbers of low cost microprocessors tied together by a communication network.

2- The communication network is a central and critical factor in multiprocessor system performance. A poorly designed network can rapidly become a performance bottleneck as the number of processors and the amount of network traffic increases. Furthermore, the complexity and cost of many high bandwidth networks grows faster than the growth of processors in the system. With large networks, the cost of the network may dominate overall system costs. The bulk of our research is concerned with the design and performance of such networks.

3- Communication and interconnection network research has often focussed on functional and protocol issues. The advent of VLSI however has made it important to examine just how such networks can be designed to exploit the cost and performance opportunites available with this technology. Much of our research work has thus centered on questions of design in the VLSI environment. In this context, the questions of pin constraints, geometric regularity, network partitioning and network control are among the research questions considered. In addition, fabrication experiments related to different network control schemes have been undertaken.

4- Finally, the role of circuit topology (i.e. circuit planarity, and overall circuit geometries) is clearly of great importance in VLSI design. Some of our earlier research has examined this topic in the context of interconnection networks. Such networks are very similar (from topology and control points of view), to various regular parallel processing networks such as systolic arrays. These systolic arrays represent another approach, as opposed to conventional multiprocessors, to using parallelism to achieve high computational performance. Extending our network research to these types of regular arrays is a current research area being investigated.

The major accomplishments of the research performed during the last year are reviewed in Section 2 to follow. Section 3 discusses our plans for year three of the contract. Section 4 concludes with a summary discussion of the research thus far.

A number of appendices follow the main body of the report and constitute the detailed results of our research. These include research papers which have been published or submitted for publication, and several working papers which discuss research in progress. Other research performed under this contract is documented in last year's annual report.

2. Research Summary and Major Accomplishments

2.1 Interconnection Networks: Asynchronous versus Clocked Design Methodologies

For large multiprocessor systems, high bandwidth interconnection networks will require numerous "network chips", with each chip implementing some subnetwork of the original larger network. Modularity and growth are important properties for such networks since multiprocessor systems may vary in size. The problem of partitioning such networks is thus a critical one and must be dealt with if the high bandwidth properties of large idealized networks are to be preserved when they are actually implemented as aggregates of subnetwork chips. Two components of the partitioning problem are addressed in this report. The first, considered in this section, relates to the broad question of network timing control. The second, considered in section 2.2, is concerned principally with the problem of pin limitations.

Timing control concerns just how data movement is synchronized in an interconnection network. Consider, for instance a mesh connected crossbar network where messages move from the input to the output ports. Assume that local routing capabilities are present at each crosspoint (i.e. any message

proceeding through the network contains header information that is successively examined by each crosspoint switch to determine message routing through the switch) and, once a path through the network has been established, that path is held for the duration of the message.

There are two principal methods which can be used in controlling data movement along such a path. These are referred to as asynchronous and synchronous (or clocked) control schemes, and while they are well known as general and often opposing methodologies, there have been few cases where a quantitative comparison has been made between them on the basis of a common system design problem. Research was undertaken to develop appropriate models so that such a quantitative comparison could be made. Two types of interconnection networks were examined. First the standard mesh connected crossbar, and then an NlogN network referred to as a Banyan network. Details of this work can be found in Appendices I and II. Appendix I was presented at the last International Conference on Computer Architecture where it was very well received. It is now being published in the IEEE Transactions on Computers.

In practice, clocked designs have usually been preferred due to their relative simplicity and generally lower hardware costs. When systems become physically large however, when their size cannot be predicted in advance, or when there are numerous system inputs which operate independently (and on separate clocks), then the advantages of asynchronous design begin to mount. An example of this is in modular computer design where expandability and arbitrary system restructuring are key system features. In such modular systems determining the appropriate clock period is difficult, if not impossible, since the final size and configuration of the system is not known in advance. Not knowing this final size makes estimation of clock skew and the design of clock distribution schemes problematic. Designing for a maximum size

system, on the other hand, would require such large clock periods that system speed would be inordinately slow. The use of asynchronous control schemes in such environments is a natural solution to designing for system growth. Design of such control schemes are however difficult, and in general engineers have shunned this approach focusing instead on extending conventional clocked schemes. An interesting tradeoff can be seen here. On the one hand with clocked schemes the design of the control logic is simple while clock distribution is difficult. On the other hand with asynchronous schemes the design of the control logic is difficult while there is no clock distribution problem with which to contend.

All of this relates directly to the control problem encountered with interconnection networks used in multiprocessor systems. The ideal network should be modular and expandable so that it can be readily used to support a wide range of multiprocessor system sizes. This points to the use of an asynchronous scheme. Furthermore, since network chips will tend to be pin limited rather than component limited, any extra logic components needed for implementation of the asynchronous control could be easily absorbed on the chip. But, any asynchronous scheme would have to be implemented on a per port basis. That is, each input and ouput port of the network or subnetwork would require extra control lines. An asynchronous scheme would thus tend to have heavy pin requirements in a situation which already has pin constraints. Synchronous schemes, on the other hand will not be as pin intensive, however, will also not yield broadly modular and expandable designs.

The research, detailed in Appendices I and II, concentrated on developing models so that the asynchronous/clocked design decision could be made in a quantitative fashion. The performance measure used was network bandwidth, and thus the various models concentrated on determining the time

delays associated with each of the options. For the synchronous case the use of a simple two phase clocking scheme was assumed. A novel clock distribution arrangement based on a tree structure was presented. This implementation assures equidistant clock paths to all switch nodes and thus eliminates one of the important sources of clock skew. Other sources remain however, and a model of clock skew based on a MOS VLSI implementation of the network was developed. These models allow one to determine whether a synchronous or a clocked control scheme results in higher overall network bandwidth for a given clock skew and set of fabrication parameters. Decision curves can thus be obtained which aid in the design process. For example, with the crossbar network and a reasonable set of MOS VLSI fabrication parameters, our results show that if a clock skew below 100 nanoseconds can be achieved, then a clocked control scheme would yield the highest bandwidth network.Similar design decisions with regard to the use of crossbar versus Banyan networks have been quantified and are presented in Appendix II.

2.2 Interconnection Networks: Pin Limitations and Partitioning

In last year's annual report the problem of pin constraints was discussed. This is a fundamental problem which is having an increasing impact on the design of VLSI circuits. The source of the problem is that the amount of logic that can be placed on a chip varies roughly with the area of the chip, and chip size is increasing as fabrication techniques improve. At the same time logic densities are increasing, also due to better fabrication techniques (e.g. smaller line widths).

Standard dual-in-line packaging, on the other hand, typically places pins on the periphery of the package. These pins must have a minimum dimension and separation due to mechanical constraints, and these constraints have not

relaxed appreciably over the years. At the same time other mechanical and space constraints limit the size of the package which can be built. Thus the number of pins available per package has increased roughly linearly over the past years. The result is that the increase in available pins/chip has not matched the increase in the amount of logic/chip.

The input/output and pin requirements of a particular logical device will of course depend on the function of the device and details of its design. There is, however, a general relationship between the number of logic devices a system requires and its pin requirements. Empirically, this relationship has been found to be reasonably approximated by:

Number of Pins = KC**b

K is a constant which depends on the device function and design. C is the number of logical circuits, and b has a value of about 0.5 . For interconnection networks of the sort needed in multiprocessor systems, the value of K tends to be much larger than values found for the "average" circuit. That is, interconnection networks are very pin intensive.

The effective design of large interconnection networks is thus tied to handling the pin limitation problem. Our research here has centered on determining network partitioning strategies which satisfy given pin constraints, while at the same time minimizing performance measures of network time delay and chip count. In the partitioning process an important synchronization problem was also revealed. Details of this research on partitioning and associated synchronization problems may be found in Appendix III which is to be published in the November 1982 IEEE Transactions on Computers.

2.3 Interconnection Networks: VLSI Design and Implementation

During the past year work was begun on designing and implementing two small VLSI network chips. Both chips implement the basic crosspoint used in a modular crossbar network. In one chip clocked control procedures were employed, while in the other, asynchronous procedures were used. The designs were done for an NMOS fabrication process and one of the chips is currently being fabricated at the MOSIS facility at UCLA (ISI).

The goals of this design and implementation work are threefold. First, once the chips are available, performance testing will allow us to evaluate the accuracy of our design models. It is clear that mathematical models, such as those discussed in the previous sections, require verification and validation before they can be accepted in the design community. The production of actual subnetwork chips will permit such a comparison of models and reality.

The second goal relates to evaluating the methodologies and tools available in the logic design and implementation process. This is of particular importance when considering the design of asynchronous logic. Design methods here have by and large been ad hoc and unstructured in nature. Recently however, a rigorous design procedure for asynchronous circuits has been developed by C. Molnar and T. Fang of the Washington University Computer Systems Laboratory. This procedure holds out the promise of yielding asynchronous circuits which can be guaranteed by design to work. The asynchronous interconnection chip has been a good vehicle for testing out this design procedure. If successful, this may well lead to greater acceptance and use of asynchronous design solutions to digital systems problems.

The final goal is a long term one. The chips which are developed and tested are viewed as small scale prototypes of the interconnection chips

which are necessary for the design of future multiprocessor systems. A long term objective is to develop and test such a multiprocessor system.

Appendices IV and V document the procedures used in developing each of the two interconnection network chips discussed above, and present the resultant designs. The synchronous chip is currently being fabricated, and should be returned for testing within the next few months. The asynchronous chip will be submitted for fabrication shortly. Future experiments will involve replicating the basic switch design so that larger network modules can be placed on a single chip.

2.4 Systolic Arrays: Asynchronous versus Clocked Design Methodologies.

Although, as we have described above, increases in processing power can be achieved using multiple processors interconnected via a switching network, another architectural style that seems to offer similar performance increases has recently received considerable attention. This is the systolic array. This architecture has the desirable properties of being modular, can be pipelined, and has the potential for high speed concurrent processing. It also appears to be quite attractive for VLSI implementation because of its regular structure. That is, it only requires the design of a single basic module for each of the linear, rectangular or hexagonal arrays.

There are numerous papers describing the applications of such arrays, particularly in the signal processing field (see references of Appendix VI). However, most of this literature is concerned with the presentation of algorithms and there is only a limited discussion of how such arrays can be controlled. We also are not aware of any reports on how one determines the actual performance of these arrays in terms of bandwidth or data rate. As with the multiprocessor interconnection networks, both synchronous and

asynchronous control can be used.

Consideration of this architectural style then is a natural extension of our work on interconnection networks and we recently have begun some preliminary studies aimed at determining accurate delay based models for both types of control strategies. These models will allow us to make comparisons of the control strategies and will also permit us to predict the performance (e.g. data rates) of both structures. A working paper is included in Appendix VI that illustrates some of our initial research efforts in this area. The single dimensional linear systolic array is examined and delay models similar to those for the crossbar networks have been constructed. From these models the worst case computational periods have been extracted and the data rates obtained. The results for the synchronous and asynchronous structures are :

$$DRs = 1/(dcompute + dpath + alpha + delta)$$

$$DRa = 1/(dcompute + 2dpath)$$

where dcompute is the processing delay associated with the module, dpath is the propagation path delay associated with the transfer of data from module to module, and alpha and delta are related to the skew of the clock in the synchronous system.

It should be emphasized here that the systolic array architecture is substantially different than the multiprocessor architecture discussed earlier and this has important fabrication implications. For a moderate number of processors an interconnection network can probably be placed on a single chip or a bit slice approach can be used. The interconnection network, being simple, is therefore not chip area limited, but pin limited. On the other hand, with the systolic array, each module contains a processor. This increased logical complexity will require chip area and thus more of a balance may be achieved between area and pin constraints. We plan to investigate some

of these issues during the coming year in addition to developing similar models and bandwidth relations for square and hexagonal systolic arrays.

2.5 Reinforced Delta Networks: Formal Models and Network Performance

Much of the research work we have pursued on interconnection networks has focused on modelling and comparing two fundamental network types: Crossbar and NlogN networks. Our work has considered the chip area and pin requirements, and the performance (i.e. bandwidth and cost) of these networks using different control schemes.

NlogN or Banyan type networks have two main attractions. First, they have moderate component counts since the number of components grows roughly as NlogN versus N**2 for crossbar networks. Second, network control for establishing connections can be decentralized. This latter property permits design of modular networks which can be more easily partitioned.

An undesirable characteristic of these networks, however, is their inability to realize arbitrary connections (mappings) between input and output ports. Networks which cannot perform all such mappings are said to be blocking type networks, as opposed to non-blocking networks such as the crossbar network. This loss in connection concurrency is one of the tradeoffs involved in keeping the component count down in Banyan networks. A number of studies have been made which formalize the properties and performance of such networks, and one of the most general is that of Patel (see references of Appendix VII). Patel proposed a broad class of NlogN networks called Delta networks and presented performance models for operation of these networks in a random access environment. In such an environment, random connections need to be established between input and output ports, and such an environment approximates the operation of a large MIMD (Multiple Instruction

stream Multiple Data stream) computer.

During the past year research has been pursued on extending the Delta network class to include networks which have certain improved properties. This new class of network is referred to as a "Reinforced" Delta network. There are three reasons for the developement of this new network class. First, Delta networks are only defined for network sizes which are powers of the subnetwork module size (this will typically correspond to the network which can be placed on a single chip). For a given subnetwork size, this restricts the size of the overall network which can be constructed. Reinforced Delta networks relax this constraint and permit a greater number of overall network sizes for a given subnetwork module size.

Second, as indicated, Delta networks are blocking networks and hence have poorer bandwidth properties then crossbar networks with equal numbers of input and output ports. The reinforced Delta network introduces extra links into the Delta network construction in a structured fashion. This increases the bandwidth of the network. Thus network bandwidths which range from pure Delta networks to crossbar networks can be designed, and a spectrum of networks with varying bandwidth and chip requirements are possible.

Third, Delta networks, and for that matter all standard NlogN networks, have a single path from input to output. If a link along this path is broken, then certain connections cannot be made. This represents a severe reliability problem in many applications. Reinforced Delta networks, by providing extra links, improve the reliability of the network.

Appendix VII presents a formal model for reinforced Delta networks, indicates how overall networks of different sizes can be constructed from various submodule networks, and analyzes network bandwidth.

3.0 Research Tasks: Year Three

3.1 Experiments and Tests

As indicated earlier, we have just completed the design and submitted a layout for the basic module of the synchronous controlled crossbar network to the UCLA ISI MOSIS facility for implementation in NMOS. This is scheduled to be fabricated soon and several chips should be returned to us in the next few months. This design will be tested, and based on the experimental results, any necessary modifications in the layout will be made. We then plan on replicating several of the modules on a single chip so that a 2 by 2 or 4 by 4 crossbar network can be obtained. When this larger interconnection network is fabricated we will be in a position to measure the performance of overall networks (e.g. delays, data rate, bandwidth), obtain information about other important parameters (e.g. power consumption), and compare these to the theoretical results that we have obtained. This should allow us to refine our theoretical models so that they more realistically predict circuit performance.

During this period the design and layout of the asynchronously controlled crossbar module presented in Appendix V will be finished and submitted for fabrication. Plans here are similar to those for the synchronously controlled crossbar: to test and evaluate the basic module, to modify and resubmit it if necessary, and to then have a larger (4 by 4) asynchronously controlled network fabricated Since the asynchronous switch is far more complex than the synchronous switch, the testing procedure will be more time consuming.

The results of these studies and measurements should give us substantial insight into the accuracy of our models of the two control structures, and should provide practical results that allow these VLSI

interconnection networks to be applied in multiprocessor systems.

3.2 Theoretical Studies

3.2.1 Banyan and Reinforced Delta Networks

We have begun work on determining how the Banyan interconnection network should be controlled and have constructed models for evaluating its performance. We plan on continuing this work and comparing it to the crossbar implementations described above. Attempts to convert these results into a physical NMOS layout and implementation may be undertaken if it seems to be warranted and if time and manpower permit. If possible, reliability models of reinforced Delta networks will also be developed and reliability comparisons with standard NlogN networks studied.

3.2.2 Protocols and Software Support

Research will be pursued on related issues (other than the module physical implementation) that must be understood before an actual interconnection network can be successfully employed. These include additional studies of source/destination protocol problems which relate directly to higher level software support, and place additional constraints on network behaviour and performance.

3.2.3 General Physical Constraints

Three principal physical constraints are associated with VLSI chip design. These are chip area, number of pins, and heat (power) dissipation. We have already investigated the impact of pin limitations on interconnection network design. The development of more general models which integrate all three constraints is now being studied. This research will continue with the goal of obtaining an overall constraint model to apply to interconnection networks of interest.

3.2.4 Systolic Arrays

Our knowledge of pin limitations, asynchronous and synchronous control structures, module modelling, and performance evaluation which have been acquired in our research on crossbar and Banyan networks will be applied to systolic array designs that are common in signal processing applications. These arrays are also constructed by the replication of interconnected modules and our experience with modular interconnection networks should be extremely useful in this research area. We hope to further expand our initial work on one dimensional linear arrays (Appendix VI) to square and hexagonal arrays. The goal of this research is to be able to predict the optimum type of systolic array control structure as a function of array size and clock skew. This should be of significant interest to the signal processing community.

4.0 Conclusions

This annual report has documented research progress and achievements which have occurred during year two of ONR contract N00014-80-C-0761 entitled "VLSI Based Multiprocessor Communications Networks". The work was performed at the Washington University Center for Computer Systems Design, St. Louis, Missouri. This work has been motivated by the potential for increased speed and high reliability associated with the implementation of multiple processor systems, recognition of the importance of the interconnection network over which the processors communicate, and by the availability of new design options afforded by the ongoing VLSI technology revolution. In addition, interconnection networks are one example of a digital system which has regular topological properties, and consists of basic functional units which are replicated and connected in a structured fashion. Systolic arrays are another example of such regular systems and thus some of the design models and

techniques used on interconnection networks may be applicable to such arrays.

During year 2 of our research, further progress was made on a host of problems. Of particular importance has been the work developing models for interconnection network control structures. Our analysis of asynchronous and clocked control schemes is unique, and represents one of the only published results which constrasts these two approaches in a quantitative fashion on a realistic digital design problem. The models developed allow one to decide on the control technique which yields the highest bandwidth for a given set of design parameters. As discussed in Section 2.4, this research has been extended to one dimensional systolic arrays. Control of systolic arrays has generally been a neglected area of research even though development of systolic array chips requires an understanding of this problem . Further research here may provide the designer with a fuller understanding of important control section design decisions, and thus limit the use of ad hoc design methods. We intend to do further research work in this area during year 3 of this contract.

In order to evaluate the interconnection network models and general design methodologies which we have developed, some time was spent during the year on the design and layout of two NMOS chips. Each chip implements the basic crosspoint element in a modular crossbar network, with one chip implementing a clocked control scheme, and one an asynchronous control scheme. The clocked chip has been completed and sent out for fabrication. The asynchronous chip will soon be completed and will also be fabricated. Testing will begin when the fabricated chips are obtained.

Finally, general network modelling research was also pursued during the year. Research continued on the pin limitation and partitioning problem, and a new type of network referred to as the reinforced Delta network was pro-

posed and investigated. This network design is both more reliable and has better bandwidth properties than standard Delta networks. Models were developed to predict the blocking and bandwidth capabilities of this network.

Proposed research during year three is outlined in Section 3 of this report. Most of this work represents a continuation of tasks begun during years 1 and 2 of the project, and reflect the work discussed initially in our original proposal. Our expectations are that the coming year will continue to be productive and that further meaningful progress will be made.

APPENDIX I

Proceedings of the 1982
International Conference
on
Computer Architecture

IEEE TRANSACTIONS
ON COMPUTERS (in press)

ASYNCHRONOUS AND CLOCKED CONTROL STRUCTURES FOR VLSI BASED INTERCONNECTION NETWORKS*

Mark A. Franklin and Donald F. Wann

Department of Electrical Engineering
Washington University
St. Louis, Missouri 63130

ABSTRACT

A central issue in the design of multiprocessor systems is the interconnection network which provides communications paths between the processors. For large systems, high bandwidth interconnection networks will require numerous 'network chips' with each chip implementing some subnetwork of the original larger network. Modularity and growth are important properties for such networks since multiprocessor systems may vary in size. This paper is concerned with the question of timing control of such networks. Two approaches, asynchronous and clocked, are used in the design of a basic network switching module. The modules and the approaches are then modelled and equations for network time delay are developed. These equations form the basis for a comparison between the two approaches. The importance of clock distribution strategies and clock skew is quantified, and a network clock distribution scheme which guarantees equal length clock paths is presented.

1.0 INTRODUCTION

The principal issues in computer architecture have continually evolved in response to changes in basic computer technology and in recent years the advent of VLSI technology has once again shifted the design space. While the implications of this shift are not yet fully understood, certain changes in direction are becoming apparent. This paper is concerned with a problem of central importance in the design of large multiprocessor computer systems. Such systems are typically based on the use of inexpensive yet powerful VLSI microprocessor chips and chip sets.

Over the past few years the availability of such microprocessors has led to numerous proposals for the design of a variety of physically local, closely coupled multiprocessor systems (SWAN77, DENN74, SEJN80, SULL77). The communications network utilized by such closely coupled processor configurations is of central importance to the performance of these systems. Various studies have focused on functional properties of such networks (BENE65, GOKE73, LAWR75, PEAS77), on their complexity and performance (PATE79, SIEG79, MALE80), and to a certain extent, on their actual design (FRAN79, QUAT81, FRAN81A). Some of these studies have emphasized the VLSI implementation of such connection networks (HOEY80, THOM80, FRAN81B, PADM81) and in particular just how the topology of various networks affects their chip area and time delay properties.

This paper considers the problem of timing control structures for large interconnection networks where numerous VLSI chips are required for full network implementation. A general interconnection network is shown in Figure 1. The network is taken to have N' input ports and N' output ports where each port has B' data lines. Other lines not shown in the figure will also be required for control and synchronization of data transfer through the network. Clearly for N' and B' beyond a certain size, the network will require partitioning into a number of subnetworks where a single VLSI chip can be associated with each subnetwork. A principal motivation for partitioning the network relates to chip pin limitations and, as discussed in FRAN81B, there are several partitioning strategies available. One simple approach is shown in Figure 2.

Related to the partitioning problem is the broader question of timing control. That is, how is data movement synchronized in the network? Consider, for instance the mesh connected crossbar shown in Figure 2 and assume that the network is to pass messages from input to output ports. Assume that local routing capabilities are present at each crosspoint in the network (i.e., any message proceeding through the network contains header information that is successively examined by each crosspoint switch to determine message routing through the switch) and that once a path through the network has been established, that path is held for the duration of the message. Assume furthermore that the individual switches have limited memory so that a message can be pipelined along a captured input/output path.

There are two principal methods which can be used in controlling data movement along such a path. These are referred to as asynchronous and synchronous (or clocked) control schemes and while they are well known as general and often opposing methodologies there have been few cases where a quantitative comparison has been made between them

*This work was supported in part by NSF Grant MCS-78-20731 and ONR Contract N00014-80-C-0761

on the basis of a common system design problem. This paper presents such a comparison, where the system in question is a crossbar interconnection network of the general type described above. Though not pursued here, the analysis presented extends to other network topologies as well.

In practice, clocked designs have usually been preferred due to their relative simplicity and generally lower hardware costs. When systems become physically large however, or when their size cannot be predicted in advance, or when there are numerous system inputs which operate independently (and on separate clocks), then the advantages of asynchronous design begin to mount. One example of this is in processor bus design where asynchronous control schemes are common (DIGI81, SUTH79). Another is in modular computer design where expandability and arbitrary system restructuring are key system features (CLAR67). In such modular systems determining the appropriate clock period is difficult, if not impossible, since the final size and configuration of the system is not known in advance. Not knowing this final size makes estimation of clock skew and the design of clock distribution schemes problematic. Designing for a maximum size system, on the other hand, would require such large clock periods that system speed would be inordinately slow. A similar type of problem arises when designing timing control structures in the VLSI domain which are robust over a range of feature sizes, that is, which operate properly as the physical dimensions of the components are scaled (SEIT79). The use of asynchronous control schemes in such environments is a natural solution to designing for system growth (or shrinkage). In this context it has been pointed out that use of asynchronous techniques can also be viewed as a structured design discipline in the time (or actually sequence) domain akin in spirit to structured programming in the program domain (SEIT80). Design of such control schemes are however difficult, and in general engineers have shunned this approach focusing instead on extending conventional clocked schemes. An interesting tradeoff can be seen here. On the one hand with clocked schemes the design of the control logic is simple while the clock distribution problem is difficult. On the other hand with asynchronous schemes the design of the control logic is difficult while there is no clock distribution problem with which to contend.

All of this relates directly to the control problem encountered with interconnection networks used in multiprocessor systems. The ideal network should be modular and expandable so that it can be readily used to support a wide range of multiprocessor system sizes. This points to the use of an asynchronous scheme. Furthermore, since network chips will tend to be pin limited rather than component limited, any extra logic components needed for implementation of the asynchronous control could easly be absorbed on the chip. In contrast, any asynchronous control scheme would have to be implemented on a per port basis. That is, each input and output port of the network or subnetwork would require extra control lines (e.g., request, acknowledge). An asynchronous scheme would thus tend to have heavy pin requirements in a situation which already has pin constraints. The full range of considerations here is currently being studied.

This paper concentrates on a principal component of this study, that is, what are the relative speeds that can be achieved when implementing an interconnection network using asynchronous versus clocked control schemes. The section that follows defines asynchronous and clocked protocols for a simple but reasonable network switch module. Time delay models are then developed and general expressions for the delay presented and discussed. These models show that clock skew is a key factor in determining which of the two approaches leads to a faster switching module. The central role of clock skew is further examined in the succeeding section and the physical origins of this skew are investigated under the assumption that the switch modules are fabricated using standard NMOS technology. Clock path layout is also important in determining skew and an interesting tree structured layout is presented which provides for equal length paths to each switching module. The paper concludes with a detailed example demonstrating the use of the time delay models in a particular switch design situation.

2.0 PROTOCOLS

A brief description of suggested protocols for the synchronous and asynchronous realizations of the crossbar interconnection network is given in this section. A complete interconnection network would require control provision for:

Path establishment
Transfer of data from source to destination
Detection of a blocked path
Indication of end of transmission
Path clearing

We have described how all of these requirements could be satisfied (FRAN81C). We have also shown that for many cases of practical importance, a bit slice architecture in which the network is partitioned into planes, each plane switching one bit of the incoming data words, is optimumal from a chip count viewpoint (FRAN81P). For this reason the analysis here is restricted to a one bit plane network as shown in Figure 2a. Figure 2b illustrates what positions may be established in an individual switch. Further, since data rate is the performance measure, only the protocol necessary to transfer data from module to module (assuming that the path from source to destination has already been established) is described.

2.1 Asynchronous Protocol

A delay insensitive protocol is adopted for the asynchronous modules, that is, insertion of any fixed or time varying delay in any of the paths between modules will not cause the network to fail - although its speed may be modified. One such delay insensitive protocol that has the minimum number of signal changes (and thus probably will have the maximum data rate) uses transition sensitive logic. This protocol can be illustrated in Figure 3 by considering a single switch pair

(i,j) in which data is to be transmitted from a source at the left to a destination at the right. The protocol is as follows: If module i wants to send a logic zero to module j it makes a change in the R0 line - if it wants to send a logic one to module j it makes a change in the R1 line. Upon receipt of a change in R0 or R1, module j accepts the data and returns an acknowledge to module i by changing the A line. If module i has some new data (e.g. received from the module to its left) it may now transmit it to module j by again changing the appropriate line, R0 or R1. Note that this transaction technique is independent of any delay that is inserted in the lines between modules i and j since the change will eventually reach its intended module and the module cannot proceed with its next exchange until the previous exchange has been completed. This protocol will be assumed in the further discussions on asynchronous interconnection networks.

2.2 Synchronous Protocol

For the synchronous system the standard clocked-data protocol is adapted in which a level sensitive two-phase clock is employed. Two such communicating modules (i,j) are shown in Figure 4 along with an entire network. This arrangement operates in the following manner: Let data be available at the input to module i. This data is captured by module i (i.e. stored) upon the assertion of the phase-one clock. On the assertion of the phase-two clock the data is transferred to the output of module i and propagates over the communication pathway to the input of module j. This data at the input of module j is then captured on the assertion of the phase-one clock and the procedure is repeated. Note that this protocol is not delay insensitive since, if the period of the clock (e.g. the time between successive assertions of the phase-one clock) is too short, then the data captured at the input to module i at an assertion of the clock will not have had adequate time to propagate through module i, across the interconnecting pathway and be available at the input to module j on the next clock assertion.

3.0 DELAY MODELS

3.1 Asynchronous Delay Model

For the asynchronous switching elements shown in Figure 3 consider only a single request from left to right with the corresponding acknowledge from right to left. The Huffman model (that satisfies the protocol discussed in the previous section) for a pair of modules along a path from source to destination is represented in Figure 5. Consider module i in this figure. Let the propagation delay of the combinational logic be d_{Li}, the propagation delay of the feedback path be d_{Fi}, and the propagation delay along the request path from module i to module j be d_{Pij}. Similarly for module j the logic and feedback delays are d_{Lj}, and d_{Fj}, while for the acknowledge path from module j to module i, the delay is d_{Pji}. To ensure race free operation it has been shown by FANG81 that the delays must satisfy the relations:

$$d_F \geq d_L \qquad [3.1]$$

$$d_P \geq 0 \qquad [3.2]$$

It is now possible to compute the minimum interarrival time between successive requests to module i. Assume that there is an acknowledge present at module i and a first request to this module arrives from the module to its left. This request propagates along the dotted path shown in Figure 5 and arrives back at the combinational logic input to module i. If the module to the left of module i has produced a new request in response to the acknowledge generated by module i, this second request could be processed immediately. Thus the time between servicing successive data bits (e.g. requests) for the asynchronous architecture is given by the loop delay, d_A, where

$$d_A = d_{Li} + d_{Pij} + d_{Lj} + d_{Fj} + d_{Pji} \qquad [3.3]$$

The values for the individual delays will vary from module to module due to processing and fabrication nonuniformities and the next question is just what values of delays to adopt. Note that the network operates in a pipeline manner, and thus the data rate of the network is determined by the maximum value of this loop time (i.e. the maximum value of one of the pipe delays). For a network with N sources and N destinations the average path through the network contains N modules. For large N there is a high probability that a loop between a pair of adjacent modules will be encountered that contains propagation delays that all have their largest values. This probability approaches one as N grows and to ensure worst case conditions it will be assumed that such a maximum loop delay, $\overline{d}_A$, is encountered. (It is interesting to observe here that because of the pipeline nature of this architecture the **maximum** delay determines its performance. This is in contrast to many applications of asynchronous systems in which a good estimate of the performance is given by the **average** delay). These maximum delays will be identified by removing the subscripts i and j in Equation 3.3. Hence

$$\overline{d}_A = 2d_L + d_F + 2d_P \qquad [3.4]$$

3.2 Synchronous Delay Model

The model for two modules in the synchronous system of Figure 4 can be constructed using a finite state machine representation of each module. This machine is designed to implement the classical two-phase level sensitive clocking scheme protocol described in Section 2.2 and can be depicted as shown in Figure 6. This model has combinational logic delay d_L, memory delay d_M, interconnection data path delay d_P, and delay along the clock line d_C. The delays d_L and d_P are assumed to have a distribution of values identical to those used for the asynchronous model, thus no distinguishing subscript is needed. The delay d_C is used to represent propagation delay along the path over which the clock is distributed.

Each module contains two memory elements (i.e. a master-slave configuration) and these are identified by an additional subscript (e.g. d_{Mi1}, d_{Mi2}) corresponding to whether they receive a phase-one or a phase-two clock.

Consider data stored in a memory element on a particular clock phase. This data can propagate along a path to a memory element and, to guarantee deterministic behavior, this data must arrive and be stable prior to the next occurrence of this clock phase. This imposes constraints on the minimum clock period. The module pair shown in Figure 6 has four such paths along which data is transmitted: a path from memory i1 to memory j1, a path from memory i2 to memory j2, an internal feedback path from i1 to i1 and an internal feedback path from j1 to j1. Each of these path delays places a constraint on the clock period and the maximum delay determines the acceptable period. The constraint for Path 1 is: Data at the input to memory i1 at an occurrence of the phase-one clock at memory i1 must propagate via d_{Mi1}, d_{Mi2}, d_{Pij}, and d_{Lj} and be stable at the input to memory j1 at the next occurrence of the phase-one clock at memory j1. A timing diagram for this constaint is shown in Figure 7 and the period of the clock is specified as T. The occurrence of the phase-one clock at the two memory elements i1 and j1 is influenced by the value of the two clock line delays d_{Ci1} and d_{Cj1}. From the timing diagram the clock period can be expressed as:

$$T > d_{Mi1} + d_{Mi2} + d_{Pij} + d_{Lj} + (d_{Ci1} - d_{Cj1}) \quad [3.5]$$

In a similar manner the other three paths consist of data propagating through the following elements:

Path 2: d_{Mi2}, d_{Pij}, d_{Lj}, and d_{Mj1}

Path 3: d_{Mi1}, d_{Mi2}, and d_{Li}

Path 4: d_{Mj1}, d_{Mj2}, and d_{Lj}

The constraint relations for these three paths can be found in a manner similar to that for Path 1 and are:

$$T > d_{Mi2} + d_{Pij} + d_{Lj} + d_{Mj1} + (d_{Ci2} - d_{Cj2}) \quad [3.6]$$

$$T > d_{Mi1} + d_{Mi2} + d_{Li} \quad [3.7]$$

$$T > d_{Mj1} + d_{Mj2} + d_{Lj} \quad [3.8]$$

Since in most designs the last two constraints imposed on T are smaller than either of the first two, they will not be considered further here. Note that the bracketed quantity in Equation 3.5 is the difference in arrival of the phase-one clock at the corresponding memory elements of the two modules. Likewise for the bracketed quantity in Equation 3.6 and the phase-two clock. These differences are called <u>clock skew</u> and will be defined as

$$\delta_{C1} = d_{Ci1} - d_{Cj1} \quad [3.9]$$

$$\delta_{C2} = d_{Ci2} - d_{Cj2} \quad [3.10]$$

The delay values are statistically distributed and, due to the pipeline argument, the delays for the module pair under consideration are assumed to represent the largest delay encountered in the distribution. Then Equations 3.5 and 3.6 are *identical*. Removing the subscripts to indicate this worst case condition allows the maximum delay for the synchronous architecture to be written as

$$d_S = d_L + 2d_M + d_P + \delta \quad [3.11]$$

where the clock skew is $\delta = d_{Ci} - d_{Cj}$.

4.0 DATA RATE COMPARISON

The data rate for the asynchronous system, DR_A, and the data rate for the synchronous system, DR_S, are the inverses of the corresponding delay times given in Equations 3.4 and 3.10. Therefore

$$DR_A = 1/(2d_L + d_F + 2d_P) \quad [4.1]$$

$$DR_S = 1/(d_L + 2d_M + d_P + \delta) \quad [4.2]$$

The condition under which the asynchronous data rate is larger than the synchronous data rate can be written as

$$\delta > d_L + d_P + d_F - 2d_M \quad [4.3]$$

Equations 4.1 and 4.2, which provide the data rates for the asynchronous and synchronous systems, and Equation 4.3, which indicates how clock skew affects the data rate comparisons between the two systems, are the major developments of this work. Once the design team has information about the specific implementation parameters and can determine their numerical delay values, these design equations can be used to compare system performance and make a selection of an appropriate system architecture.

The interpretation of this rate comparison can be simplified further by observing that the delay of the memory will normally be equal to the delay of an elemental transistor, d. The combinational logic delay can be expressed as a multiple of this delay, that is, $d_L = kd$. The minimum delay constraint for d_F from Equation 3.1 will be used so that $d_F = d_L = kd$. Using this nomenclature the condition of Equation 4.3 then becomes

$$\delta/d > 2(k-1) + d_P/d \quad [4.4]$$

This relation is shown in Figure 8 and illustrates the regions in which $DR_A > DR_S$ and in which $DR_A < DR_S$. For large d_P the synchronous system data rate is higher than that of the asynchronous system. This is because d_P appears only <u>once</u> in the path between two synchronous switch modules (see Figure 6 and Equation 4.2) while, in the asynchronous system, the handshake protocol requires a round trip, thus d_P occurs <u>twice</u> (see Figure 5 and Equation 4.1). Likewise as δ is increased, the performance of the synchronous system is degraded (see Equation 4.2). The interaction between these two variables can be

shown graphically (for a specific value of k and d_p) by plotting a normalized data rate, d'DR versus δ/d. An example of this is illustrated in Figure 9 for the special case of zero propagation delay ($d_p = 0$) between modules. The role of clock skew becomes clear from this figure. As the skew increases, there is a distinct crossover point beyond which an asynchronous design is superior from a data rate performance viewpoint. The effect of the intermodule propagation delay is also apparent. As this delay increases the synchronous design becomes superior.

5.0 DETERMINATION OF SYSTEM DELAY RELATIONS

Let the crossbar network be implemented via NMOS technology with a collection of N^2 switching modules on a chip and let a number of such chips be placed on a printed circuit board and interconnected via printed circuit wiring. In this section the factors that affect the values of the various system delays are examined and simple expressions for each of them are developed.

5.1 Combinational Logic, Memory and Feedback Delays

The synchronous module can be described by a 40 row state table and a PLA implementation of this table was chosen. A design methodology for determing a PLA having the minimum delay was applied (ANAN82) and yielded a combinational delay, d_L = 37.5 ns and memory delay d_M = 2 ns. Although a complete state table for the asynchronous case was not developed, the preliminary analysis indicates that its complexity would be comparable, so this same value of d_L is used for both architectures. The feedback delay, d_F, for the asynchronous case is equal to the combinational logic delay.

5.2 Path Delay

There are two intermodule paths that must be examined: paths between two adjacent modules on the same integrated circuit chip and paths between two adjacent modules on different chips. Because of the topology of the crossbar network, modules that communicate can be implemented in close physical proximity. Compared to the other circuit delays, the intermodule path delay between modules on the same chip is then very small and can be ignored. This, however, is not true when a module on one chip must communicate with a module on another chip. The delay is related to the resistance and capacitance of the path and, since the interconnection path will be short and will use metal conductors, its resistance is small and can be neglected. The propagation delay is then determined by the ratio of the capacitance of an elemental gate, C_g and the capacitance of the interconnection line, C_L. It has been shown (MEAD80) that if one uses an exponential buffer this delay is given by the expression

$$d_p = de[\ln(C_L/C_g)] \qquad [5.1]$$

5.3 Clock Skew

The clock skew is the maximum delay between the clock signals that control a horizontally or vertically adjacent module pair in Figure 4. Consider the simple model for this situation in which two modules M1 and M2 are driven from a common clock point P, with clock lines from this common point of lengths L1 and L2. Assume that the clock is asserted at t = 0. The clock skew is then the difference in time between when M1 responds to C1 and when M2 responds to C2. This difference in response is determined by four factors:

Differences in the line parameters (e.g. resistivity, dielectric constant) that that determine the line time constant.

Differences in the threshold voltages of the two modules M1 and M2.

Differences in delays through any active elements inserted in the lines (e.g. clock buffers).

Differences in the line lengths L1 and L2.

It is reasonable to assume that the clock will be generated external to the chip so that sufficient drive for all the on-chip modules is available. Hence buffers will not be needed and the third factor in the above list can be ignored. In the next section a clock distribution that guarantees equal clock line lengths for all paths is developed. Thus the fourth factor can be eliminated. A model for the clock path skew that includes the first two factors is developed in the Appendix and the clock skew is found in terms of the maximum and minimum time constants of a clock line ($\overline{RC}$ and $\underline{RC}$) and the maximum and minimum values of the threshold voltage of a typical logic gate ($\overline{V_T}$ and $\underline{V_T}$). The result of that derivation gives the clock skew as

$$\delta = \overline{RC} \ln \overline{V_T} - \underline{RC} \ln \underline{V_T} \qquad [5.2]$$

6.0 CLOCK DISTRIBUTION

As shown in the Appendix, it is important to maintain the same on-chip clock line length to each of the N^2 modules. One technique that accomplishes this utilizes a binary tree layout for the clock path. An example of this distribution for a single-phase clock supplied to an 8x8 network is shown in Figure 10. Note that as this tree is traversed from its root (clock input) to any leaf (module) the length to each switch is constant. Let the dimension of one edge of the network array equal E, then the length of each of the N^2 clock paths (for this example) is 11E/8. As N increases (i.e. for large E) the length, L of each path becomes L = 3E/2. In order to estimate the clock skew, the RC time constant of the clock path must be computed. Thus the length and type of conducting material for each branch of the clock tree must be known. Although the clock could be distributed using metal for the horizontal clock paths and diffusion for the vertical paths, metal should be used wherever possible. Some short sections of diffusion will be necessary, however, in order to bridge intermodule communication, power, ground and reset lines (assuming only a single layer of metal is available). An actual

layout of a synchronous module indicates that these sections can be shortened so that only 20% of the clock line is in diffusion. The distribution of the two-phase clock requires two such binary trees - this second tree can be constructed by merely displacing the first tree in the vertical and horizontal directions by the minimum line separation. The length of the metal branches and the length of the diffusion branches for a large chip with side E then becomes

$$L_M = 1.2E \qquad L_D = 0.3E \qquad [6.1]$$

where $L = L_M + L_D = 1.5E$

7.0 EXAMPLE

Consider a network constructed on a 25cm x 25cm printed circuit board which contains 64 packaged chips each with a package size of 2.5cm x 2.5cm. The actual chip size is approximately 1cm x 1cm. (If an individual chip had 100 pins this arrangement could handle one bit slice of a 48x48 asynchronous network). Assume copper printed circuit connections between chips. The pin capacitance for this type of construction is about 4 pf and the capacitance of an elemental gate is about 0.02 pf. Then the maximum delays for this type of circuit are approximately $d_L = 37.5$ ns, $d_M = 2$ ns, $d_F = 37.5$ ns, and $d_p = 43$ ns.

Substituting these values into Equation 4.3 shows that in order for the asynchronous data rate to exceed the synchronous system data rate the clock skew would have to be 114 ns or greater. Next an estimation of the clock skew is computed for this example.

Since there is negligible clock skew due to the printed circuit board paths, the clock skew is dominated by intrachip parameters. From the Appendix the RC value for the chip clock line is found as 50 ns. Discussion with commercial fabricators have indicated a variation of $\pm$ 20% in RC. Thus $\overline{RC}$ = 61 ns and $\underline{RC}$ = 40 ns. For a supply voltage of 5 volts, equal noise margins can be obtained with V_T = 2.5 volts. The range of the threshold variation is also about $\pm$ 20%, so $\overline{V}_T$ = 3.0 and $\underline{V}_T$ = 2.0 volts. Using these values in Equation 5.2 gives the clock skew as 39 ns. Figures 11 and 12 illustrate a numerical comparison between the two architectures. Note that the synchronous system architecture yields a higher data rate than the asynchronous architecture for these network parameters.

8.0 CONCLUSIONS

This paper has presented a comparison of asynchronous and clocked timing control structures in the context of the design of an interconnection network. The network has local routing control, is pipelined, and has a mesh connected crossbar topology. It is intended for use in a message based multiprocessor environment. The selection of the appropriate control structure is critical if a high bandwidth, modular interconnection network is to be achieved. Designing for growth and size uncertainty both in the large (i.e. the number of processors in the final system may change), and in the small (i.e. the VLSI feature size may change) appears to make an asynchronous approach attractive. Such an approach, however, will tend to have heavy pin requirements. Before a complete comparison of the methodologies can be achieved, it is necessary that fundamental models which allow one to compare their relative speeds are developed. This is the principal contribution of this paper.

First two switch modules, one asynchronous and one clocked, and their respective data synchronization protocols were defined. Based on this a model of each module's operation was developed. These models allow one to determine the data transmission speed associated with the modules (and therefore for an entire network) and thus compare the timing control methodologies in question. The model equations yield decision curves which can be used to compare these two control structures under a variety of design parameters. The equations indicate the key role played by clock path delay and clock skew, and clearly shows how the speed of clocked systems must be lowered as the clock skew increases. Key equations are derived for path delay and clock skew, and a tree structured clock layout scheme is presented which results in equal length clock paths to each switching module. An example is developed where the switching modules are assumed to be fabricated in the NMOS technology and for this example the synchronous control is shown to yield a faster system.

APPENDIX

The waveform of the clock at the input to one of the modules in Figure 10 due to a step change at the clock input is approximated by an exponential of the form

$$v(t) = V_{DD}[1 - \exp(-t/RC)] \qquad [A1]$$

where R and C are the lumped resistance and capacitance of the clock path and V_{DD} is the supply voltage. The time constant, RC, of two clock lines will be different because of the variability in the line parameters. Let $\overline{RC}$ and $\underline{RC}$ be the maximum and minimum values of this variability. The clock skew is also dependent on the differences in the threshold voltages of the two adjacent modules. Let the mean threshold voltage of a module be V_T with a range of $\overline{V}_T$ to $\underline{V}_T$. Then if one module has the maximum threshold and the maximum RC line time constant, and the adjacent module has the minimum threshold and the minimum RC line time constant, this causes the maximum time difference in the response of the two modules to the clock. This is illustrated in Figure A1 where the delay difference is $\overline{t} - \underline{t}$. Substituting these worst case conditions into Equation A1 allows the values of $\overline{V}_T$ and $\underline{V}_T$ to be expressed in terms of $\overline{t}$ and $\underline{t}$ as

$$\overline{V}_T = V_{DD}[1 - \exp(-\overline{t}/RC)] \qquad [A2]$$

$$\underline{V}_T = V_{DD}[1 - \exp(-\underline{t}/\underline{RC})] \qquad [A3]$$

Combining these two equations yields the clock skew due to both time constant and threshold variations as

$$\delta = t - \underline{t} = \overline{RC}\ \ln(V_T) - \underline{RC}\ \ln(\underline{V_T}) \qquad [A4]$$

Next the value of RC is determined. The physical geometry of a conductor can be represented as shown in Figure A2 where the resistance and the capacitance are given by

$$R = \rho L/(Wh_c) \quad \text{and}$$
$$C = \varepsilon LW/(h_o) \qquad [A5]$$

In these relations ρ is the resistivity of the conductor, ε is the dielectric constant of the oxide, h_o is the thickness of the oxide, and h_c is the thickness, W is the width and L is the length of the conductor. Hence the time constant of the line can be expressed as

$$RC = \rho\varepsilon L^2/(h_c h_o) \qquad [A6]$$

Note that the time constant is independent of the width of the line. If two clock lines have different lengths, the clock skew is related to the square of the differences in line lengths. It is for this reason that the binary tree distribution illustrated in Figure 10, which provides equal lengths, has been chosen.

Consider one of the paths in the binary tree and let the diffusion length and width be L_D and W_D, and the metal length and width be L_M and W_M. The resistance will be expressed in terms of ohms per square and capacitance in terms of pf per unit area (in this case square microns). These relations can be related to the basic parameters shown in Figure A2 as

$$R/sq = \rho/h_c \quad \text{and} \quad C/ar = \varepsilon/h_o \qquad [A7]$$

Since the resistivity of diffusion is very large compared to the resistivity of metal, the contribution of the resistance of the metal path to the total line resistance will be negligible. However, the capacitance of both the diffusion and metal lines must be included in the computation. Therefore the total line resistance and total line capacitance is approximately

$$R = (R_D/sq)(L_D/W_D) \qquad [A8]$$

$$C = C_M + C_D$$
$$= (C_M/ar)W_M L_M + (C_D/ar)W_D L_D \qquad [A9]$$

If the minimum feature size is λ then MEAD80 shows that the minimum width of a diffusion conductor is 2λ and the minimum width of a metal conductor is 3λ. Equation 6.1 gives the lengths of metal and diffusion in terms of the chip dimension. Hence the total resistance and capacitance can be written as

$$R = (R_D/sq)(0.3E)/(2\lambda) \qquad [A10]$$

$$C = (C_M/ar)(3\lambda)(1.2E) + (C_D/ar)(2\lambda)(0.3E) \qquad [A11]$$

The line time constant is then

$$RC = (R_D/sq)(E^2)[0.54(C_M/ar) + 0.09(C_D/ar) \qquad [A12]$$

For the example discussed in Section 7 the following representative values for the line parameters are used:

$$R_D/sq = 20\ \text{Ohms/sq}$$
$$E = 1cm = 10^4\ um \qquad [A13]$$
$$C_M/ar = 0.3 \times 10^{-4}\ pf/um^2$$
$$C_D/ar = 10^{-4}\ pf/um^2$$

Substituting these values into Equation A12 gives RC = 50 nanoseconds.

REFERENCES

ANAN82: Anantharaman, S. Delays in PLAs: Analysis, Reduction and Computer Aided Optimization. M.S. Thesis, Dept. of E.E. Washington Univ. St. Louis (May 1982)

BENE65: Benes, V.E. MATHEMATICAL THOERY OF CONNECTING NETWORKS AND TELEPHONE TRAFFIC, Acad. Press, N.Y. (1965)

CLAR67: Clark, W.A. Macromodular Computer Systems, AFIPS Proc. SJCC (Apr 1967)

DENN74: Dennis, J.B. et.al A Preliminary Architecture for a Basic Data-Flow Processor, Proc. 2nd Ann. Sym. Comp. Arch. (Dec 1974)

DIGI81: Digital Equipment Corp, MICROCOMPUTERS AND MEMORIES, DEC, Maynard, Mass (1981)

FANG81: Fang, T.P. On the Design of Hazard Free Circuits, Computer Sys. Lab., T.M. 285, Washington Univ., St. Louis (Nov 81)

FRAN79: Franklin, M.A. et.al. Design Issues in The Development of a Modular Multiprocessor Communications Network, Proc. 6th Ann. Symp. Comp. Arch. (Apr 1979)

FRAN81A: Franklin, M.A. and Wann, D.F. Pin Limitations and VLSI Interconnection Networks, Proc. 1981 Conf. on Parallel Processing (1981)

FRAN81B: Franklin, M.A. VLSI Performance Comparison of Banyan and Crossbar Switching Networks, IEEE Trans. Comp. C-30,4 (Apr 1981)

FRAN81C: Franklin, M.A., Wann, D.F. and Thomas, W.J. Word Inconsistency in Partitioned VLSI Interconnection Networks, Center for Computer Systems Design TM 81FR-B, Washington Univ. St. Louis (1981)

GOKE73: Goke, L.R. and Lipovski, G.J. Banyan Networks for Partitioning Multiprocessor Systems, Proc. 1st Ann. Symp. Comp. Arch (1973)

HOEY80: Hoey, D. and Leiserson, C.E. A Layout for the Shuffle-Exchange Network, Proc.Inter. Conf. on Parallel Processing (1980)

LAWR75: Lawrie, D. H. Access and Alignment of Data in an Array Processor, IEEE Trans. Comp. C-24,12 (Dec 1975)

MALE80: Malek, M and Myres, W.W. Figures of Merit for Interconnection Networks, Proc. Workshop on Interconnection Ntwks. Purdue Univ. (Apr 1980)

MEAD80: Mead, C. and Conway, L. INTRODUCTION TO VLSI SYSTEMS, Addison Wesley, Reading, Mass. (1980)

PADM81: Padmanabhan, K. Multiprocessor Interconnection Networks in a VLSI Environment, M.S. Thesis, Dept. of E.E., Washington Univ., St. Louis (Dec 1981)

PATE79: Patel, J.H. Processor-Memory Interconnection Networks for Multiprocessors, Proc. 6th Ann. Symp. Comp. Arch. (Apr 1979)

PEAS77: Pease, M.C. The Indirect Binary n-Cube Microprocessor Array, IEEE Trans. Comp. C-26,5 (May 1977)

QUAT81: Quatember, B. Modular Crossbar Switch for Large Scale Multiprocessor Systems - Structure and Implementation, AFIPS Proc. Nat. Comp. Conf. (1981)

SEIT79: Seitz, C.L. Self-Timed VLSI Systems, Proc. Caltech. Conf. on VLSI. (Jan 1979)

SEIT80: Seitz, C.L. System Timing, Chapter 7 in INTRODUCTION TO VLSI SYSTEMS by Mead, C. and Conway, L., Addison-Wesley. Reading, Mass. (1980)

SEJN80: Sejnowski, M.C., et.al. An Overview of the Texas Reconfigurable Computer, AFIPS Proc., Nat. Comp. Conf. (1980)

SIEG79: Siegel, H.J. A Survey of Interconnection Methods for Reconfigurable Parallel Processing Systems, AFIPS Proc., Nat. Comp. Conf. (June 1979)

SULL77: Sullivan, H. and Bashkow, T.R. A Large Scale Homogeneous, Fully Disbtirbuted Parallel Machine I, Proc. 4th Ann. Symp. Comp. Arch. (Mar 1977)

SUTH79: Sutherland, I.E., et.al. The TRIMOSBUS, Proc. Caltech. Conf. on VLSI, (Jan 1979)

SWAN77: Swan, R.J. et.al. Cm* - A Modular Multi-Microprocessor, AFIPS Proc. Nat. Comp. Conf. (1977)

THOM80: Thompson, C.D. A Complexity Theory for VLSI, Proc. 11th Ann. ACM Symp. on Theory of Computing (Apr 1979)

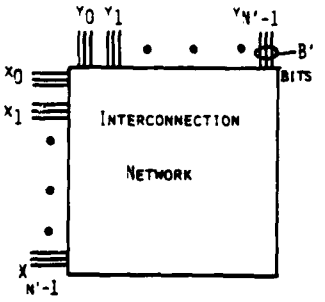


FIGURE 1: AN N'*N' NETWORK

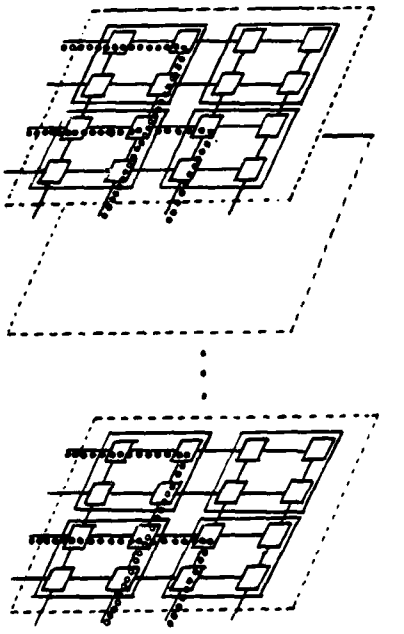


FIGURE 2a. A MESH CONNECTED CROSSBAR (N'=4,B'=8). BIT SLICE PARTITIONING WITH 2*2 SUBNETWORKS PER CHIP. TWO SAMPLE INPUT/OUTPUT PATHS SHOWN

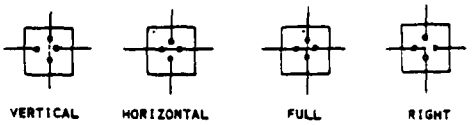


FIGURE 2b. CROSSPOINT POSITIONS FOR MESH CONNECTED CROSSBAR

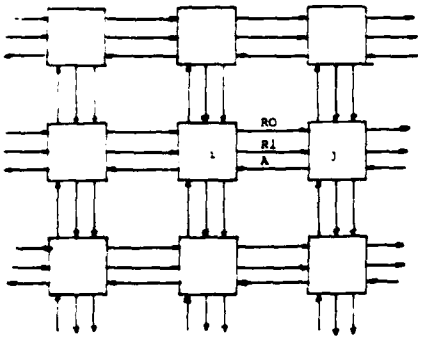


FIGURE 3. CROSSBAR INTERCONNECTION NETWORK USING ASYNCHRONOUS MODULES

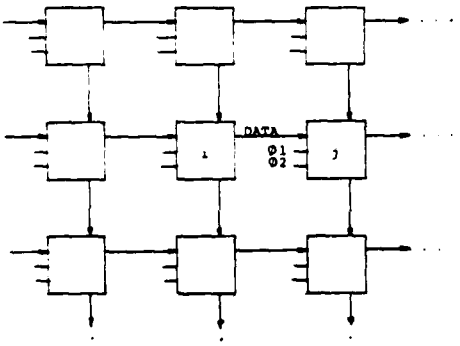


FIGURE 4. CROSSBAR INTERCONNECTION NETWORK USING SYNCHRONOUS MODULES

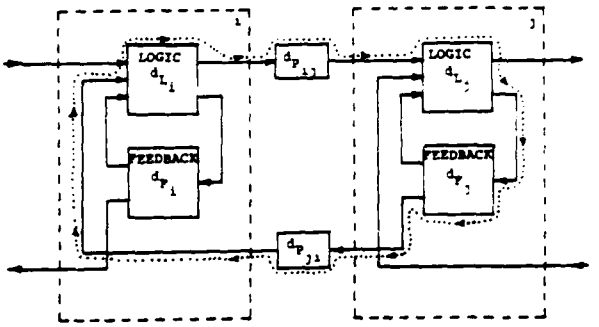


FIGURE 5. DELAY MODULE FOR TWO ADJACENT ASYNCHRONOUS MODULES

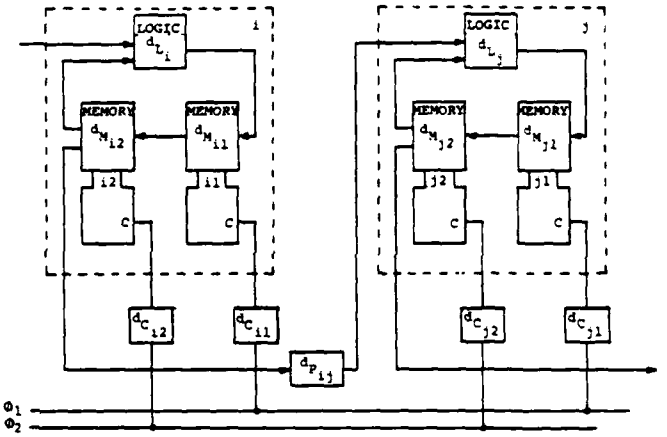


FIGURE 6. DELAY MODEL FOR TWO ADJACENT SYNCHRONOUS MODULES

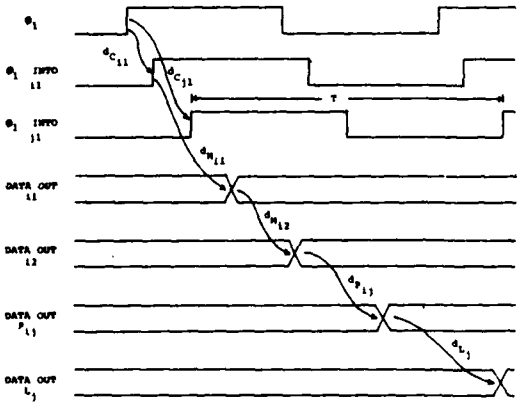


FIGURE 7

TIMING DIAGRAM CORRESPONDING TO EQ. 3.5

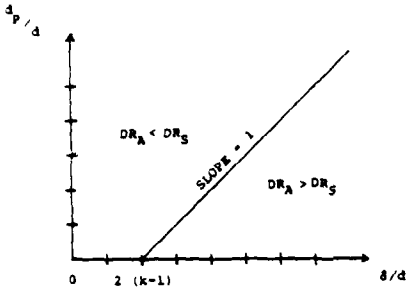


FIGURE 9. DESIGN SPACE AS FUNCTION OF CLOCK SKEW AND PATH DELAY FOR ASYNCHRONOUS-SYNCHRONOUS ARCHITECTURES

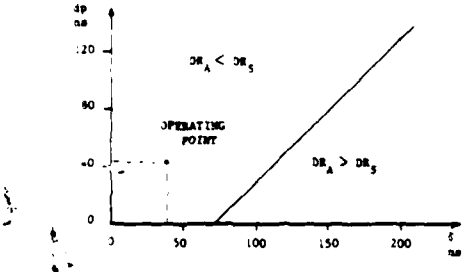


FIGURE 11: LOCATION OF OPERATING POINT IN DESIGN SPACE FOR EXAMPLE SYSTEM.

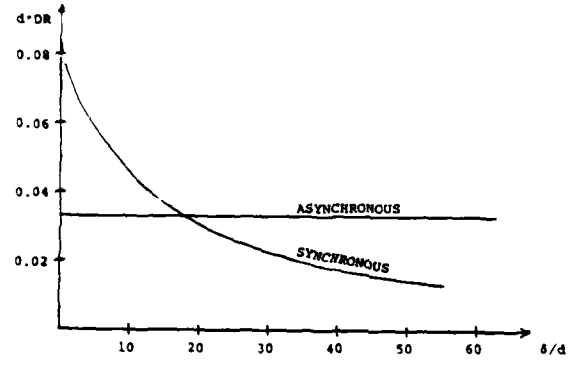


FIGURE 9. NORMALIZED DATA RATE FOR ASYNCHRONOUS AND SYNCHRONOUS SYSTEMS AS FUNCTION OF NORMALIZED CLOCK SKEW (FOR k=10 AND d_p=0)

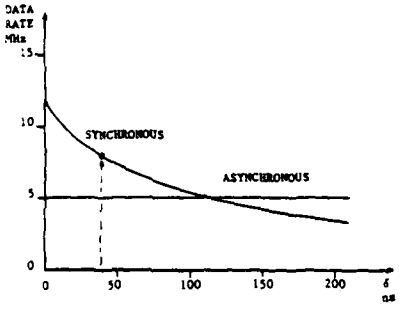


FIGURE 12: DATA RATES FOR ASYNCHRONOUS AND SYNCHRONOUS ARCHITECTURES FOR EXAMPLE SYSTEM. ARROW INDICATES ACTUAL CLOCK SKEW.

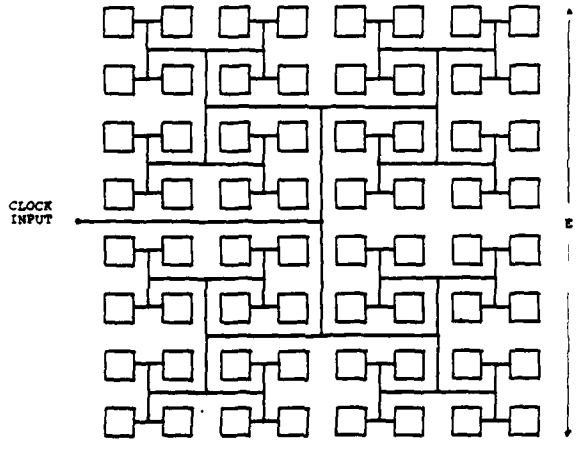


FIGURE 10. CLOCK DISTRIBUTION FOR AN 8*8 NETWORK USING A BINARY TREE

DELAY COMPARISON OF VLSI BASED BANYAN AND CROSSBAR NETWORKS

S. Dhar, M. Franklin and D. Wann
Washington University,
St. Louis, Missouri.

1.0 Introduction

Advances in VLSI technology have made available a number of low cost yet powerful microprocessor chips. This has led to a host of proposals [SWANN77, SULL77, SEJN80] for the design of closely coupled multiple processor systems in which a number of processors are connected together by a communications network. The communications network handles interprocessor communication and makes sharing of common resources possible. A key issue in the design of such systems is the design of this network, and overall system performance is dependent to a large extent on the performance of the network. The choice of the type of communications network therefore becomes a major factor in the design process.

A number of studies [FRAN81A, FRAN81C, KAHN78, FRAN79] have focussed on Crossbar and NlogN interconnection networks in a VLSI environment. The planar and modular construction of the Crossbar network makes it very suitable for VLSI implementation. NlogN networks, such as Banyan networks, while not planar, generally require fewer chips for implementation of networks of equal size.

The data rate that can be achieved in a network is an important performance measure that determines the suitability of a particular network in a particular system. A high data rate is generally desirable and several methods have been evolved which reduce the delay through a network to maximize the data rate. The pipelined mode of data transfer is a commonly used technique for achieving high data rates.

Consider an N*N crossbar network built from the basic module shown in Figure 1. With uniformly addressed input and output ports, the average number of modules in an input/output path is N. If the pipelined mode of data transfer is applied to such a system, and long messages are transferred, there would be an increased data rate of approximately N times over the non-pipelined case. A recent paper [FRAN81A] has evaluated the delays in Crossbar and Banyan networks under a non-pipelined mode of data transfer. In such a system the delay is directly proportional to the length of a data path and it has been shown that for large networks requiring multiple chips, the Banyan network has a higher data rate due to a smaller path length. However, in a pipelined system the length of the data path has a much smaller influence on the delay, especially when large messages are transferred. The data rate of the network then is determined by other factors.

Another important issue in the design of the communications network is the type of control scheme to be used for control of data movement. There are two principal methods that can be used in controlling data movement along the network; these are referred to as the synchronous (or clocked) and the asynchronous (or self-timed) schemes. Franklin and Wann [FRAN81D] have made a quantitative comparison of the two control schemes for a Crossbar network. The synchronous control scheme has been traditionally favoured, especially in small systems, because of the simplicity of logic design. However, for large systems where size cannot be predicted in advance, or where a number of subsystems operate independently, the maximum clock skew may not be known in advance and the asynchronous control scheme seems to be more suitable.

This paper is an extension of the work presented by Franklin and Wann. Their analysis focussed on a single network, the Crossbar, and how

the use of asynchronous or clocked control schemes affected network performance. As pointed out above however, for large networks, the use of a Banyan (or other NlogN) networks can significantly reduce the number of chips required for overall network implementation. This paper therefore compares both Crossbar and Banyan network performance (e.g. data rate or bandwidth) over the asynchronous/clocked control design options. The analysis is based on a finite state machine model of the basic switch modules, and a clock line layout which minimizes clock skew in both the Crossbar and Banyan networks. The analysis provides a quantitative approach to making the Crossbar/Banyan, asynchronous/synchronous design decisions. For a particular example, decision curves are provided to illustrate the procedure.

2.0 Protocol Issues

A complete interconnection network requires control provisions for:

a) Path establishment.

b) Transfer of data from source to destination.

c) Detection of a blocked path.

d) Indication of end of transmission with path clearing.

The way in which these requirements can be satisfied have been described in FRANSID and is not considered further here. The present analysis assumes that the path from source to destination has already been established and focuses on data rates achievable after path establishment.

It is also assumed that the network has been partitioned following a bit slice architecture approach with one bit per plane, that is, if a word size of sixteen is desired , then sixteen network planes are implemented. The present analysis is therefore restricted to one bit plane of the network.

For the asynchronous network, a delay insensitive control structure is adopted. That is, insertion of any delay between modules will not cause the network to malfunction. Transition sensitive logic is employed and with this approach the total number of signal changes necessary to complete the transfer of one bit of data is two (one on the request line and one on the acknowledge line). Figure 2(a) shows the interconnection between two asynchronous modules. A transition on R1 indicates the presence of a "1" bit, while a transition on R0 indicates the presence of a "0" bit. The "A" line supplies the acknowledge response signal. Interconnection of two synchronous modules is shown in Figure 2(b). For the synchronous network, the standard two phase level sensitive clock is used for the data transfer. Data at the input of a module is captured at phase one of the clock and is transferred to the output of the module at phase two of the clock.

3.0 A Banyan Network Model

3.1 Banyan Asynchronous Delay Model

The logical implementation of the asynchronous module is achieved by means of a finite state machine. The Huffman representation of such an implementation is shown in Figure 3. If the input of the combinational logic is a function of the output of the combinational logic, then it has been shown [FANG81] that the sufficient conditions on the various delays to achieve hazard free operation are given by the relations

$$dF >= dL \qquad (3.1)$$

$$dO >= dF + dL \qquad (3.2)$$

For the purpose of determining the data rate consider Figure 2(a) where a single request moves from left to right with the corresponding

acknowledge proceeding from right to left. Since the network is pipelined, the data rate in a given source/destination path is determined by the the maximum delay between any two communicating modules in that path. A pair of communicating modules i and j in a path k is modelled as shown in Figure 4(a). Considering module i, the maximum propagation delay from any input to any output of the combinational logic is dLi while the propagation delay of the feedback path is dFi. dPij is the propagation delay of a request in going from module i to module j over the physical distance between module i and module j. The delay term "max(dPij, dFi)" in the request path between modules i and j shown in Figure 4(a) is necessary to satisfy the constraint given by (3.2) and is explained later on. Similarly for module j, we have dLj and dFj with the propagation delay in the acknowledge path from module j to module i over the physical distance between modules j and i being dPji. The propagation delay of the path from the output of the combinational logic of module i, through the combinational logic of module j to the input of the combinational logic of module i corresponds to the term dO in Figure 3. If relation (3.1) is satisfied, then the delay terms "max(dFi, dPij)" and "max(dFi, dPji)" ensure that relation (3.2) is always satisfied.

The minimum delay between two requests for the pair of Asynchronous BAnyan modules i and j is equal to the maximum loop delay as given below.

$$dABAij = dLi + \max(dPij, dFi) + dLj + \max(dpPi, dFi) \qquad (3.3)$$

The inverse of the delay dABAij gives the data rate between the pair of modules i and j. However, we are interested in determining the data rate of the entire network. In an asynchronous network the data rate in different paths of the network will in generel be different. Hence the average data rate that the network can support will be considered. Obviously, the average data rate is dependent on the relative frequency of usage of the

different paths in the network. We will assume that all paths in the network are equally used. In the remaining part of this section we will develop expressions for the minimum delay between two requests for a path in the network, and then by obtaining the average of such delays for all paths in the network we will arrive at the expression for the average delay between two requests for the entire network.

The expression (3.3) is for a particular pair of modules i and j in a path k. Consider next all pairs of such communicating modules in the path k. Each pair of communicating modules has an associated maximum loop delay which determines the minimum time between two requests. The path k can then be modelled as shown in Figure 4(b) where each pair of communicating modules and the maximum loop delay associated with that pair is shown. Because the network is pipelined the maximum of the delays dABA12, dABA23,...., dABA(n-1)n will determine the minimum time between two requests for the path k. This minimum time can then be expressed as

$$dABAk = 2dL + 2(\max(dPk, dF) \qquad (3.4)$$

where dL and dF are the maximum values associated with the combinational logic and feedback delays; dPk is the maximum delay between modules along the request acknowledge lines for path k, that is

$$dPk = \max(dPij, dPji) \text{ for all } i \text{ in path } k,\ j=i+1 \qquad (3.5)$$

Notice that dPk will be dependent on the particular path since this delay reflects the layout of module chips on a printed circuit board and that layout is in turn dependent on the topology of the network being considered. Figure 5 shows a layout for a 64*64 Banyan network composed of 4*4 chip modules. It is clear from this figure that different source/destination paths will encounter different delays, and that within a given path there is some particular module to module delay which is largest. This results from the

nonplanar topology of the Banyan network which will generally require longer paths between modules than a Crossbar network.

In order to obtain the average delay between two requests for the entire network, the average of dABAk over all possible paths in the network should be obtained. This can be expressed as

$$dABA = \left(\sum_{k=1}^{M} dABAk\right)/M \qquad (3.6)$$

where there are a total of M paths in the network. We will assume here that the maximum delays associated with the combinational logic and feedback are constant for all paths in the network. Then equation (3.6) can be expressed as

$$dABA = 2dL + 2\left(\sum_{k=1}^{M} \max(dPk, dF)/M\right) \qquad (3.7)$$

If dPk >= dF for all k then equation (3.7) can be written as

$$dABA = 2dL + 2\left(\sum_{k=1}^{M} dPk\right)/M \qquad (3.8)$$

Letting dPBA be the average of dPk over all paths we obtain

$$dABA = 2dL + 2dPBA \qquad (3.9)$$

where

$$dPBA = \sum_{k=1}^{M} dPk/M \qquad (3.10)$$

Evaluation of individual delay parameters is deferred to Section 3.3.

3.2 Banyan Synchronous Delay Model

The synchronous module may also be represented as a finite state machine. As in the case of the asynchronous system, a pair of modules in a path from a source to a destination is modelled. The model shown in Figure 6 has combinational logic delays dL, memory delays dM, interconnection path delay dP and clock delay dC. Each module contains two memory elements and these are identified by the subscripts 1 and 2. A two phase clocking scheme is

used to clock the memory elements 1 and 2.

Constraints on the time-period of the clock can be obtained by noting that the following conditions must exist to ensure proper operation:

(1) Data at input of memory i1 at phase 1 of the clock must propagate via dMi1, dMi2, dPij and dLj and be stable at the input of memory j1 before the next occurrence of the phase 1 clock.

(2) Data at input of memory i2 at phase 2 of the clock must propagate via dMi2, dPij, dLj and dMj1 and be stable at the input of memory j2 before the next occurrence of the phase 2 clock.

(3) Data at input of memory i1 at phase 1 of the clock must propagate via dMi1, dMi2 and dLi and be stable at the input of memory i1 before the next occurrence of the phase 1 clock.

These constraints can be expessed in terms of the clock period and various delays as shown below.

$$T >= dMi1 + dMi2 + dPij + dLj + (dCi1 - dCj1) \quad (3.11)$$

$$T >= dMi2 + dPij + dLj + dMj1 + (dCi2 - dCj2) \quad (3.12)$$

and

$$T >= dMi1 + dMi2 + dLi \quad (3.13)$$

In most designs the third constraint on T is smaller than either of the first two; hence it will not be considered further. The quantity (dCi1-dCj1) is the difference between the times the phase 1 clock arrives at the corresponding memory elements of the two modules. The same applies to the quantity (dCi2-dCj2) and the phase 2 clock. These terms are referred to as the clock skew and are defined as

$$\delta_{C1} = dCi1 - dCj1 \quad (3.14)$$

$$\delta_{C2} = dCi2 - dCj2 \quad (3.15)$$

The clock period will be determined by the pair of modules for which the constraint on T is the largest. If we assume that the largest individual

delays are encountered for this pair of modules (this gives the worst case condition), then relations (3.11) and (3.12) are identical (or if not, we select the most limiting condition). This worst case condition clock time period for the Synchronous BAnyan network represents the delay, dSBA, between two requests and is given by

$$dSBA = dL + 2dM + dPmax + \delta \qquad (3.16)$$

where dPmax is the maximum path delay between any pair of communicating modules over the entire network, that is,

$$dPmax = \max(dPij, dPji) \text{ for all } i \text{ in the network, } j=i+1 \qquad (3.17)$$

and δ = the clock skew = $dCi - dCj$ (3.18)

The next section considers the individual delay parameters for the Banyan network.

3.3 Delay Parameters for Banyan Network

We assume that the network will be implemented with NMOS technology with a number of modules on a chip and a number of chips on a printed circuit board. The minimum feature size for the NMOS technology is assumed to be 2.5 microns. Consider next the various delay parameters needed for evaluation of dABA and dSBA in Equations (3.9) and (3.16) respectively.

The synchronous module was designed and can be described by a 40 row state table. Using a PLA implementation, the maximum delay through the combinational logic was determined to be 45 nsec. The asynchronous module was also designed and the combinational logic delay after the path has been established was calculated as 34 nsec.

Hence	dL = 45	nsec for the synchronous module
and	dL = 34	nsec for the asynchronous module.
We choose	dF = dL = 45	nsec for the synchronous module
and	dF = dL = 34	nsec for the asynchronous module

such that the condition given by (3.1) is satisfied. The memory element typically is an inverter with pullup to pulldown transistor ratio of 8:1, and hence we have dM = 2d where d is the delay of a 4:1 inverter. Next consider the delays dPBA and dPmax. The path delay involves delay in paths on chip and delay in paths off chip. The former is negligible compared to the latter because the capacitances in paths off chip are orders of magnitude larger than those in paths on chip. In order to minimize the delay in driving a load capacitance CL external to the chip we use exponential drivers. Assuming that a signal present at the input to an elemental gate (an inverter) is to be transferred to the load capacitance CL, the delay using exponential drivers is given by [MEAD80]

$$dP = d*e*\ln(CL/Cg) \tag{3.19}$$

where Cg is the capacitance of an elemental gate. Expressions for dPBA and dPmax in terms of the network size N' and the module size in a single chip N, are derived in Appendix A as

$$dPBA = d*e*\ln(2Cpin+L1+((N^2+1)(N-1)/2N^4)N'*L2/Cg) \tag{3.20}$$

$$dPmax = d*e*\ln((2Cpin+L1+(N-1)N'*L2/N^2)/Cg) \tag{3.21}$$

where Cpin is the pin capacitance, and L1 and L2 are the chip separations for the Banyan network as shown in Figure 5.

We next consider the clock skew. The clock skew between two modules is the difference in the delay in the clock lines to the two modules. Consider the simple model shown in Figure 7. Two modules M1 and M2 are driven from a common clock point P with clock line lengths of L1 and L2. The clock skew is the difference in the time when modules M1 and M2 respond to the assertion of the clock at point P. This difference can be attributed to :

(a) Differences in line lengths.

(b) Differences in the passive line parameters like resistance,

dielectric constant that determine the line time constant.

(c) Differences in the threshold voltages of the two modules. The delay in a line with a lumped resistance R and a lumped capacitance C is proportional to the time constant RC. As shown in Appendix B the time constant RC is proportional to the square of the line length. Hence the contribution of (a) to the clock skew grows as the square of the difference in the clock line lengths. One possible clock distribution scheme that guarantees equal length paths, thus eliminating (a) from consideration is shown in Figure 8. Given equal length paths the clock skew can be found as:

$$\delta = (RC)min*ln(1-(VTmin/Vdd)) - (RC)max*ln(1-(VTmax/Vdd)) \quad (3.22)$$

where (RC)max and (RC)min are the maximum and the minimum RC time constants for the clock line, VTmax and VTmin are the maximum and minimum values of the threshold voltage of a typical logic gate, and Vdd is the power supply voltage.

In order to determine (RC)max and (RC)min, the path associated with the clock line must be analyzed in terms of its length and material. Consider a chip which contains an N*N Banyan network. Let the length of the chip be AN ; then assuming that the switch modules are distributed as shown in Figure 8, the width of the chip WN is given by

$$WN = AN(\log_2 N-1)/((N/2)-1) \ , \ N > 2 \quad (3.23)$$

The clock could be distributed using metal for the horizontal clock paths and diffusion for the vertical clock paths. However, since the line resistance of diffusion is much larger than that of metal, it is desirable to use metal only. Some small sections of diffusion will, however, be necessary to bridge intermodule communication, power, ground and reset lines (only one layer of metal is assumed here). Actual layout of the network indicates that only about 20% of the total clock line need be in diffusion.

The total clock line length is calculated as

$$LBA = AN/2 + WN \tag{3.24}$$

Hence the lengths of the metal and diffusion sections of the clock line are given by

$$LMBA = 0.8LBA \tag{3.25}$$

$$LDBA = 0.2LBA \tag{3.26}$$

As mentioned earlier, AN in the length of an N*N Banyan network chip. The length of such a chip, as seen from Figure 8, is proportional to the size of the network, that is , AN is proportional to N.

4.0 A Crossbar Network Model

4.1 Crossbar Asynchronous Delay Model

The delay model for the Crossbar network is obtained in the same way as in the Banyan network. A pair of communicating modules in a path from a source to a destination is modelled in the same way as shown in Figure 3. The maximum Asynchronous, CrossBar loop delay for modules i and j is then obtained as

$$dACBij = dLi + \max(dPij, dFi) + dLj + \max(dPji, dFi) \tag{4.1}$$

Since data is pipelined in the same manner as in the Banyan network, following the same reasonings as in the Banyan network, we obtain the data rate as the inverse of dACB where

$$dACB = 2dL + dF + 2dPCB \tag{4.2}$$

where dPCB is the path delay between two communicating modules. It should be noted that because of the planar construction of the Crossbar network, the distance between two interchip communicating modules is a constant independent of network size. Hence the path delay between two communicating modules is not

dependent on any particular path being considered. This is a key difference in the analysis of the Banyan and the Crossbar networks.

4.2 Crossbar Synchronous Delay Model

The synchronous delay model for the Crossbar network is similar to the model developed for the Banyan network (Figure 6). The data rate in the network is then given by the inverse of dSCB where

$$dSCB = dL + 2dM + dPCBmax + \delta \qquad (4.3)$$

where dPCBmax corresponds to the delay in the longest path between two communicating modules in the network.

4.3 Delay Parameters for Crossbar Network

The delay parameters dL, dF and dM are same as for the Banyan network. The term dPCB is estimated in Appendix A. Also, the clock distribution is similar to the Banyan network giving equal clock path lengths for all the modules. A clock distribution for a 16*16 Crossbar network is shown in Figure 9. Using metal and diffusion to distribute the clock as in the Banyan network, the total clock line length LCB is given by

$$LCB = 1.5AN \qquad (4.4)$$

where AN is the length (or width) of an N*N Crossbar network. The length of clock line in metal and diffusion then are given by

$$LMCB = 1.2AN \quad \text{for the metal line} \qquad (4.5)$$

$$LDCB = 0.3AN \quad \text{for the diffusion line} \qquad (4.6)$$

5.0 Example

As an example let us consider a N′*N′ network built from N*N size module chips which are laid on copper printed circuit boards. The pin capacitance for this type of construction is about 4pf and the capacitance of

an elemental gate is 0.01pf. The various delays are:

d = 2.0 nsec

dM = 4.0 nsec

dL = 45 nsec for the synchronous module

= 34 nsec for the asynchronous module

dF = 45 nsec for the synchronous module

= 34 nsec for the asynchronous module

An estimate of the clock skew is given in Appendix B. The synchronous module was designed and a VLSI implementation was achieved via a PLA. The area required by a 2*2 network was estimated and from this data it was calculated that the largest network that can be implemented on a 1cm*1cm chip is 16*16. We will assume that the size of the largest chip available is 2cm*2cm; a 32*32 network can be implemented in a chip of these dimensions. The delays dABA and dACB for the asynchronous modules and the delays dSBA and dSCB for the synchronous modules of the Banyan and Crossbar networks are plotted in Figures 10 and 11 against N', the network size. In Figure 12 the same delays are plotted against N, the module size.

5.1 Comparison of Asynchronous and Synchronous Network Delays

Banyan Network

Consider the delay dABA for the asynchronous module. From Figure 10 we observe that the delay increases with an increase in the network size N' for a constant N, the module size; the delay decreases with increase in N for a constant N'. The increase of dABA with N' can be attributed to the term dPBA, the average of the maximum path delays. As the size of the network increases, the physical path lengths between communicating chips increase and since the off-chip path determines the path delay, there is an increase in

delay. On the other hand, with increase in the module size, the implementation of a given size network requires fewer chips and hence the physical path length between communicating chips decreases; this results in a decrease in the delay.

Next consider the delay dSBA for the synchronous module. The synchronous module delay increases with N' for a constant N and increases with N for a constant N'. The increase of dSBA with N' can be attributed to the term dPmax, the maximum path delay between two communicating chips. As mentioned with reference to the asynchronous module, with increase in the network size the physical path lengths increase resulting in an increased delay. On the other hand, increase of dSBA with increase in N is due to the increase in clock skew; as the physical size of the chip increases (with increase in N) the clock line lengths within the chip increase and hence result in an increased clock skew. A key assumption in this is that there is negligible clock skew associated with clock distribution to the individual chips.

Crossbar Network

The delay for the asynchronous module is independent of N' and N. This is because of the planar construction of the Crossbar network. The distance between communicating chips is a constant independent of network size and hence the delay is also constant.

The delay for the synchronous module is independent of N'; this is because of the same reasons as in the asynchronous module. The delay increase with N in the asynchronous case is because the clock skew increases with N as explained earlier.

The variation of dABA and dSBA with N for different N' has been shown in Figure 12. From this figure we can make a comparison of the delays associated with the asynchronous and synchronous control schemes. It is

clear that in the case of the Banyan network, for small N the synchronous control scheme results in a smaller delay. Consider a particular network size N'. From the intersection of the curves for the asynchronous and synchronous control schemes for this value of N', we can obtain the range of values of N for which a particular control scheme is best. For example, for N'=512 we get the following result:

BANYAN { Synchronous control better for N < 23
BANYAN { Asynchronous control better for N > 23

The same interpretation can be extended to the Crossbar network curves. Notice that the Crossbar network delays are independent of the network size because the intermodule distances are constant independent of the module or network size. Thus we get the following result:

CROSSBAR { Synchronous control better for N < 19
CROSSBAR { Asynchronous control better for N > 19

5.2 Comparison of Banyan and Crossbar Network Delays

We first consider the asynchronous control scheme. From Figures 10 and 11 we find that the Crossbar asynchronous network always has a smaller delay than the Banyan asynchronous network. For small network sizes, the difference in the Banyan and the Crossbar network delays is not significant; however for large network sizes, the Crossbar network has a significantly less delay. The reason for this difference again is because of the planar construction of the Crossbar network. A more important characteristic of the Crossbar network is that the delay is constant independent of the network size; this could become a significant advantage for networks which are designed with future expansion in mind.

We next consider the synchronous control scheme. Again, the

Crossbar network has a smaller delay for small values of N, while for large values of N the delay of the Crossbar network is significantly higher than that of the Banyan network. From Figure 12 we find that the delay of the Crossbar synchronous network increases more rapidly as N increases than that of the Banyan network. The reason for this is the clock skew. In the Crossbar network the clock path lengths increase much more rapidly with increase in the module size in a chip than in the case of the Banyan network since the Crossbar network requires more number of modules to implement a given size network than the Banyan network.

In order to minimize hardware costs (i.e. reduce total number of chips), the maximum module size would be selected. However it should be noted that given current chip sizes and pin constraints, if one wants to use the maximum module size possible (i.e. $16 < N < 32$), then unfortunately this is just the region of greatest uncertainty in terms of selecting the option with minimum delay.

6.0 Conclusions

This paper has presented a comparison of the Banyan and Crossbar networks on the basis of the type of control scheme used for routing of data. The networks are intended for use in multiprocessor systems for interprocessor communication. The choice of the type of network and control structure is important in achieving high bandwidth and modularity. The analysis used in this paper can be utilized in making certain important decisions regarding the bandwidth and modularity of the network.

The Banyan and the Crossbar networks have been modelled according to the type of control scheme used. These models were used to obtain the delay associated with each type of network and control scheme. The delay equations were then used to obtain the delay curves of Figures 10 and 11 and

Figure 12.

Comparison of the delays in the Banyan and the Crossbar network were made for both types of control schemes, the synchronous and the asynchronous. The modular and planar structure of the Crossbar network was found to be very suitable for VLSI implementation; this was also the reason for the delays in the Crossbar network for both types of control schemes being less than that in the Banyan network. Delay curves were obtained for a particular example clearly showing the delay tradeoffs for the Banyan and Crossbar networks, and the synchronous and the asynchronous control schemes.

References

SWAN77
Swan, R. J. et. al. 'Cm* A Modular Multi-Microprocessor', AFIPS Proc. Nat. Comp. Conf. (1977).

SEJN80
Sejnowski, M. C., et. al. 'An overview of the Texas Reconfigurable Computer', AFIPS Proc., Nat. Comp. Conf. (1980)

SULL77
Sullivan, H. and Bashkow, T. R. 'A Large Scale Homogeneous, Fully Distributed Parallel Machine I', Proc. 4th Ann. Symp. on Comp. Arch. (March 1977).

FRAN81A
Franklin, M. A. 'VLSI Performance Comparison of Banyan and Crossbar Switching Networks', IEEE Trans. on Comp. ,C-30, 4 (April 1981)

FRAN81B
Franklin, M. A., Wann, D. F., Thomas, W. J. 'Word Inconsistency in Partitioned VLSI Interconnection Networks', Center for Computer Systems Design, T. M. 81FR-B, Washington Univ., St. Louis (1981)

FRAN81C
Franklin, M. A., Wann, D. F. 'Pin Limitations and VLSI Interconnection Networks', Proc. 1981 Conf. on Parallel Processing (1981).

FRAN81D
Franklin, M. A., Wann, D. F. ' Asynchronous and Clocked Control Structures for VLSI Based Interconnection Networks'

KAHN78
Kahn, S. A., 'Design Issues of a modular Crossbar Network for a Multiprocessor' M. S. Thesis, Washington University, St. Louis, Mo (Dec 1978)

FRAN79
Franklin, M. A., Kahn, S. A. and Stucki, M. J. , 'Design Issues in the Development of a Modular Multiprocessor Communication Network', Proc. 1979 Int. Symp. on Comp. Architectur (April 1979)

MEAD80
Mead, C. and Conway, L. , INTRODUCTION TO VLSI SYSTEMS, Addison-Wesley Pub. Co. Reading, MA (1980)

FANG81
Fang, T. P. "On the Design of Hazard Free Circuits", Computer Systems Lab., Tech. Mem. 285, Washington University, St. Louis, Mo (Nov. 81)

MOLN81
Molnar, C. E., Fang, T. P. "An Asynchronous Design Methodology", Computer Systems Lab., Tech. Mem. 287, Washington University, St. Louis, Mo (Nov. 81)

APPENDIX A

Determination of dPBA

The layout of a 64*64 Banyan network built from 4*4 module chips is shown in Figure 5 (only paths from source 1 to all destinations are shown for clarity). The analysis for the delay will be made for network sizes N′ given by

$$N' = N^{**}n' \quad \text{where } n' \text{ is an integer.}$$

As shown in the layout of the 64*64 network, the number of rows in the layout of a N′*N′ network is given by $N'/N = N^{**}(n'-1)$. A column will be referred to as the level of the network; the number of columns is given by $\log_N N' = n'$.

We consider source 1 and evaluate the maximum delays in each of the paths from source 1 to all the destinations. We will first identify the maximum intermodule communication distances and the levels at which they occur for each of these paths.

For destinations (N**(n′-1)+1) to N′, the maximum distance between two communicating modules is the distance between the module at level (n′-1) and the module it communicates with at level n′. Hence the maximum path delays from source 1 to destinations (N**(n′-1)+1) to N′ are due to the path delays between the (n′-1)th level and the n′th level of the network.

We now analyze the communication distances between modules at level (n′-1) and level n′ for the paths from source 1 to destinations (N**(n′-1)+1) to N′. We will consider N**(n′-1) paths at a time starting from the path to destination (N**(n′-1)+1), that is, the groups of paths to be considered at a time are:

Paths to destinations: N**(n'-1)+1 to 2N**(n'-1)

2N**(n'-1)+1 to 3N**(n'-1)

3N**(n'-1)+1 to 4N**(n'-1)

.

.

(N-1)N**(n'-1)+1 to N**n'

Consider the paths to destinations (N**(n'-1)+1) to 2N**(n'-1). The maximum distance between two communicating modules is between the modules at level (n'-1) and level n' and are equal for all these paths. Let this be S2. Similarly, considering the next N**(n'-1) destinations (destinations (2N**(n'-1)+1) to 3N**(n'-1)), the maximum distance between two communicating modules are equal for all these paths; this is denoted by S3. Proceeding in the same manner, let SN be the maximum distance between two communicating modules for destinations (N-1)N**(n'-1)+1 to N'.

Consider now the paths to destinations 1 to N**(n'-1). The maximum distances between two communicating modules for these paths are not the intermodule communication distances between levels (n'-1) and n'. Let S1 be the average of the maximum intermodule communication distances for paths to destinations 1 to N**(n'-1). Then the average of the maximum distances between two communicating modules for all paths from source 1 is given by

$$Sav = \sum_{i=1}^{N} Si/N = (S1/N) + \sum_{i=2}^{N} Si/N \qquad (A1)$$

We now evaluate S1. In order to do this, we consider the intermodule communication distances between levels (n'-2) and (n'-1). Clearly, for paths to destinations N**(n'-2)+1 to N**(n'-1), the maximum intermodule communication distances are those between modules at levels (n'-2) and (n'-1). We define S12, S13,...., S1N in the same way as we defined S2, S3,...., SN where the the first number after S denotes that S12, S13,...., S1N are

components of the term S1. S11 is defined in the same way as S1, that is, S11 is the average of the maximum intermodule communication distances for paths to destinations 1 to N**(n'-2). Hence we can express S1 as

$$S1 = \sum_{i=1}^{N} S1i/N = (S11/N) + \sum_{i=2}^{N} S1i/N \qquad (A2)$$

Hence

$$Sav = (S11/N^2) + \sum_{i=2}^{N} S1i/N^2 + \sum_{i=2}^{N} Si/N \qquad (A3)$$

We assume that the chips are going to be laid out on the printed circuit board as an array such that distance between two adjacent chips in the same row is L1 and the distance between two adjacent chips in the same column is L2. The first term in the expression for Sav is small compared to the third term; we take S11 as equal to the smallest distance between two communicating modules, that is, L1. Hence

$$Si = L1 + (i-1)*(N**(n'-2))*L2 \quad , \; i=2 \text{ to } N \qquad (A4)$$

$$S1i = L1 + (i-1)*(N**(n'-3))*L2 \; , \; i=2 \text{ to } N \qquad (A5)$$

$$S11 = L1 \qquad (A6)$$

where we assume that L1 < L2.

Then

$$Sav = L1 + ((N^2 + 1)*(N-1)/2N^4)N'*L2 \qquad (A7)$$

Sav gives the average of the maximum intermodule communication path lengths for all paths from source 1. Under the layout shown, source 1 clearly has the longest paths . Hence Sav is an overestimation of the average of the maximum path lengths when all sources are considered. However, we use Sav in our calculations as this would give a worse case value.

The external capacitance that has to be driven from a chip consists of the printed circuit board path capacitance and two pin capacitances . The capacitance per inch of standard printed circuit boards is generally about 1pf whereas the pin capacitance is about 4pf. If the external

load CL is driven by an exponential driver then the delay is given by

$$dP = d*e*\ln(CL/Cg) \quad (A8)$$

Hence $$dPBA = d*e*\ln(CL/Cg) \quad (A9)$$

where $$CL = 2Cpin + Cpath$$

$$= 2Cpin + L1 + ((N^2 + 1)*(N-1)/2N^4)N'*L2 \quad (A10)$$

We consider some numerical values for L1 and L2. We take

$$L1 = 1 \text{ inch}$$

$$L2 = 2 \text{ inches}$$

Then $$CL = 9 + (N^2+1)(N-1)N'/N^4 \quad pf$$

$$Cg = \text{gate capacitance of a minimum sized transistor}$$

$$= 0.01 \quad pf$$

$$d = \text{delay through a basic gate ie. an inverter}$$

$$= 2.0 \text{ nsec}$$

Then $$dPBA = d*e*\ln((9+(N^2+1)(N-1)N'/N^4)/0.02) \quad (A11)$$

Determination of dPmax

The maximum distance between any two communicating modules for the Banyan network is SN.

$$SN = L1 + (N-1)*N'*L2/N^2 \quad (A12)$$

Hence $$dPmax = d*e*\ln((2Cpin+L1+(N-1)N'*L2/N^2)/Cg) \quad (A13)$$

Determination of dPCB and dPCBmax

The layout of a 16*16 Crossbar network is shown in Figure 9 and it is clear that because of the modular and planar construction of the Crossbar network, the intermodule communication distance is independent of N and N'. The intermodule communication distance is either L1 or L2. Hence the

average communication distance is (L1+L2)/2. Hence

$$dPCB = d*e*\ln((2Cpin+0.5*(L1+L2))/Cg) \quad (A14)$$

The maximum intermodule communication distance for the Crossbar network is L2

Hence $$dPCBmax = d*e*\ln((2Cpin+L2)/Cg) \quad (A15)$$

APPENDIX B

Determination of clock skew

The resistance R and the capacitance C of a line of length L and width W is given by

$$R = \rho L/(WHc) \quad (B1)$$

$$C = \epsilon LW/Ho \quad (B2)$$

where Hc and Ho are the thickness of the conductor and oxide respectively, ρ is the resistivity of the conductor and ϵ is the dielectric constant of the oxide. Hence the time constant of the line can be expressed as

$$RC = \rho\epsilon L^2/(HcHo) \quad (B3)$$

This expression shows that the time constant varies as the square of the line length. A detailed analysis of the clock skew has been made in FRAN81D; we present the final result here: the clock skew δ given by

$$\delta = (RC)min*\ln(1-(VTmin/Vdd)) - (RC)max*\ln(1-(VTmax/Vdd)) \quad (B4)$$

where (RC)max = maximum time constant of the clock path

(RC)min = minimum time constant of the clock path

VTmax = maximum threshold voltage

VTmin = minimum threshold voltage

Banyan network

The length of the metal line was shown to be

$$LMBA = 0.8LBA \quad (B5)$$

and the length of the diffusion line was shown to be

$$LDBa = 0.2LBA \quad (B6)$$

The resistance R of the clock path is:

$$R = (RD/\square)LDBA/2\lambda \quad (B7)$$

where we neglect the contribution of the metal to the resistance as it is small compared to that of diffusion. The capacitance of the clock path is given by

$$C = (CM/a)3\lambda LMBA+(CD/a)2\lambda LDBA \quad (B8)$$

Hence

$$RC = (RD/\square)(LDBA/2)[(CM/a)3LMBA+(CD/a)2LDBA \quad (B9)$$

where

$RD/\square$ = resistance per square of diffusion

$AN = 1\ cm = 10^{4}\ um$ for N=16

CM/a = capacitance per unit area of metal

$= 0.3*10^{-4}\ pf/um^{2}$

CD/a = capacitance per unit area of diffusion

$= 10^{-4}\ pf/um^{2}$

λ = minimum feature size

= 2.5 microns

For VLSI chips , the variation of the time constant and the threshold voltage is about 20%. Hence

$$(RC)max = 1.2RC \quad (B10)$$

$$(RC)min = 0.8RC \quad (B11)$$

The typical threshold voltage VT = 2.5 V and with a variation of 20% we have

$$VTmax = 3\ V$$

$$VTmin = 2\ V$$

Hence

$$\delta = (RC)min*ln(0.6) - (RC)max*ln(0.4) \quad (B12)$$

Crossbar Network

The length of the metal line was shown to be

$$LMCB = 1.2AN \tag{B13}$$

and the length of the diffusion line was shown to be

$$LDCB = 0.3AN \tag{B14}$$

Hence the resistance and capacitance of the clock line are given by

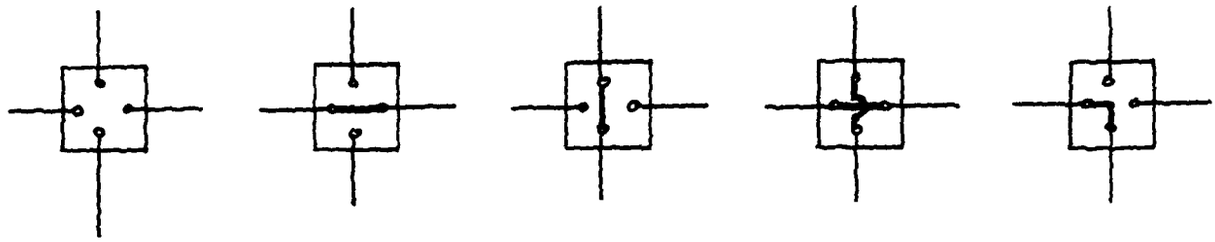
$$R = (RD/\square)LDCB/2\lambda \tag{B15}$$

$$C = (CM/a)3\lambda\, LMCB+(CD/a)2\lambda\, LDCB \tag{B16}$$

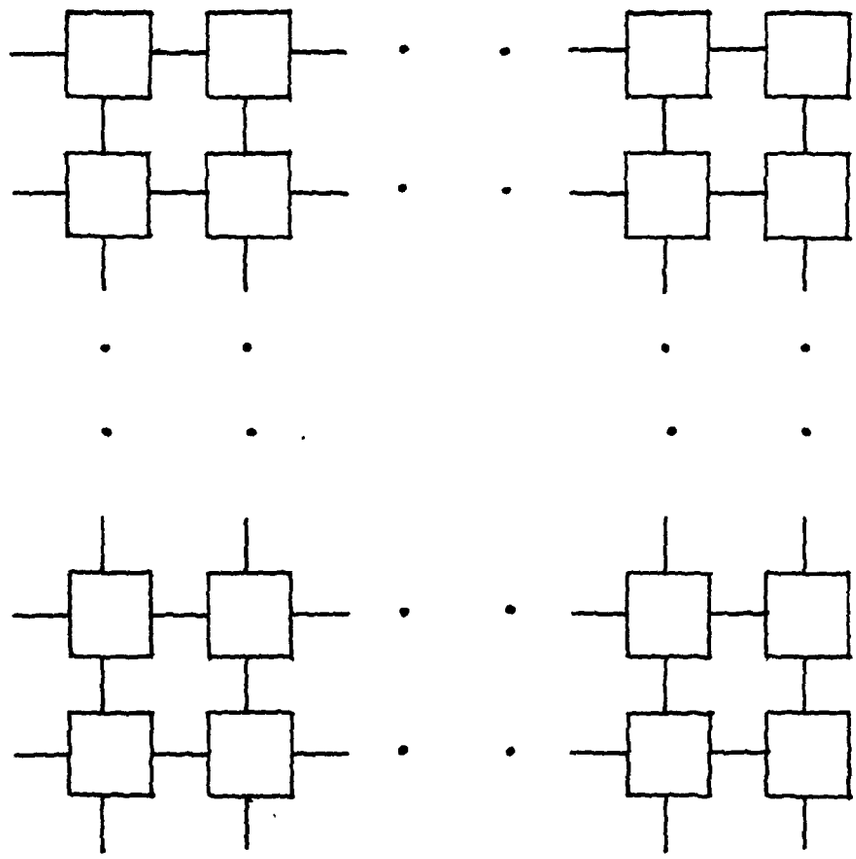
Hence

$$RC = (RD/\square)(LDCB/2)[(CM/a)3LMCB+(CD/a)2LDCB] \tag{B17}$$

$$\delta = (RC)min*ln(0.6) - (RC)max*ln(0.4) \tag{B18}$$

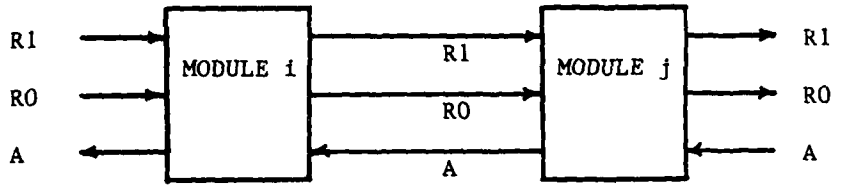


(a) Crossbar switch and the different path connections

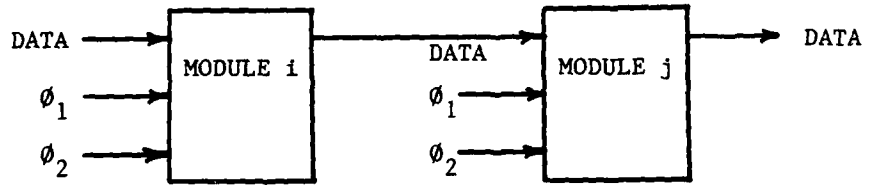


(b) An N*N Crossbar network

Figure 1: The basic Crossbar switch and an N*N Crossbar network.



(a) Interconnection of two asynchronous modules.



(b) Interconnection of two synchronous modules.

Figure 2: Interconnection of synchronous and asynchronous modules.

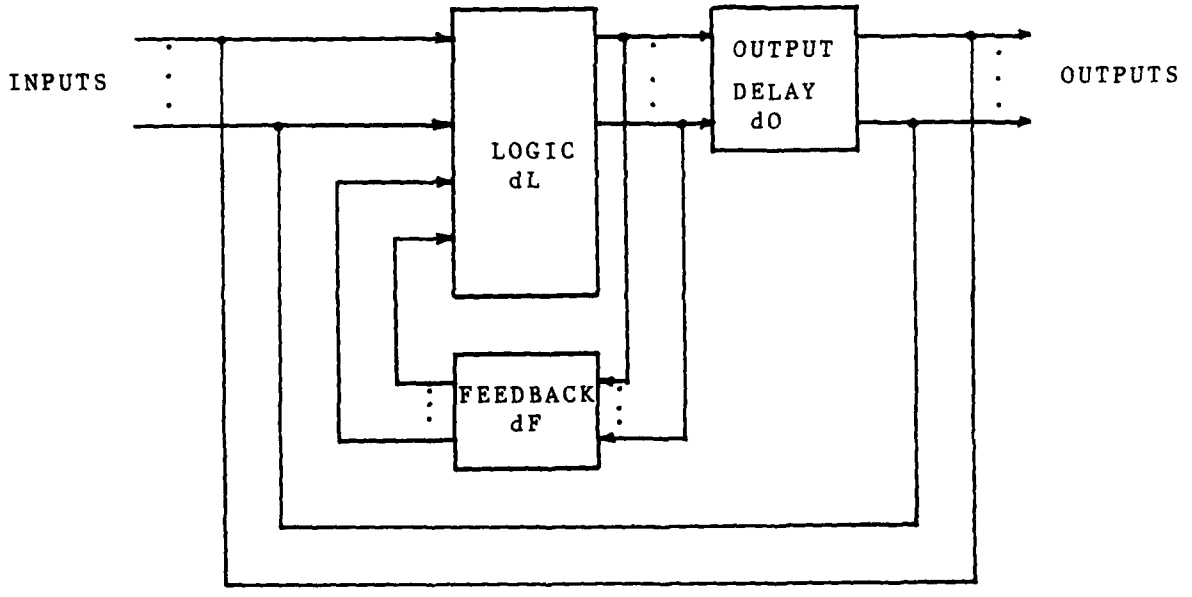


Figure 3: Huffman asynchronous circuit model.

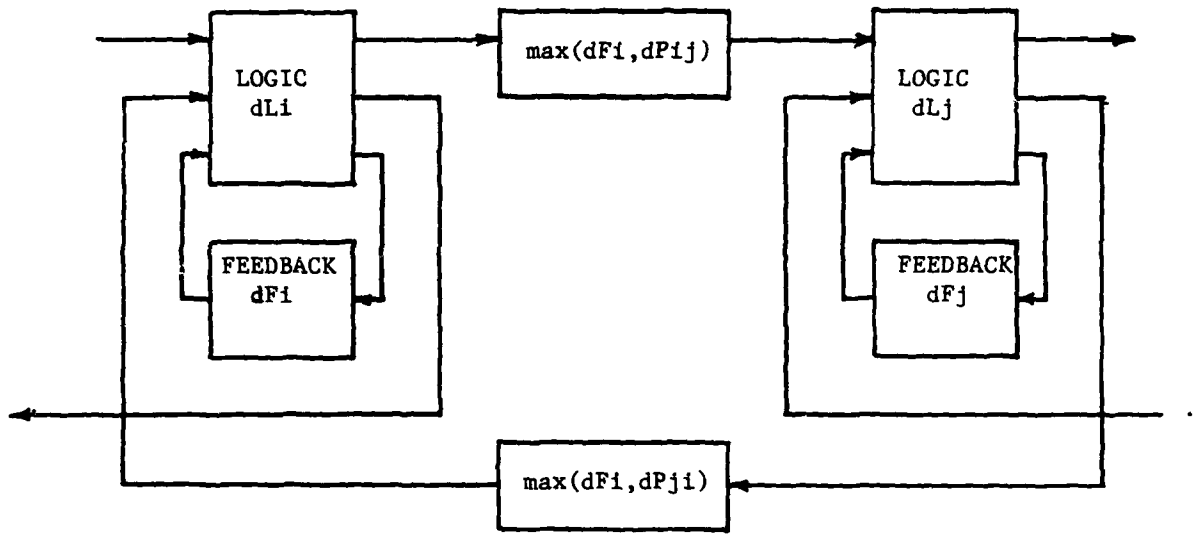


Figure 4(a): Delay model for two adjacent asynchronous modules.

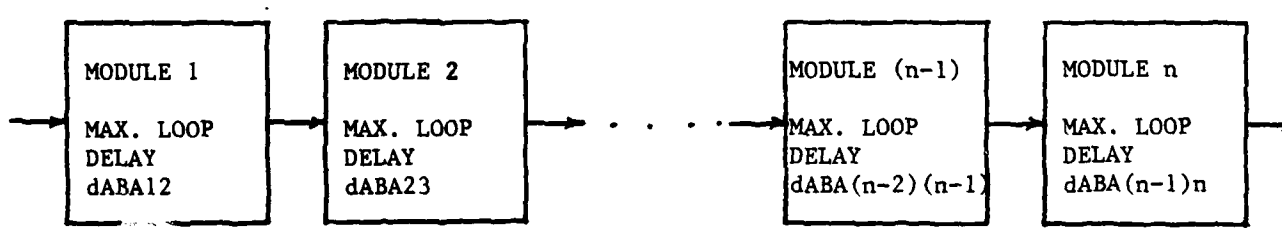


Figure 4(b): Delay model for a path in the asynchronous Banyan network.

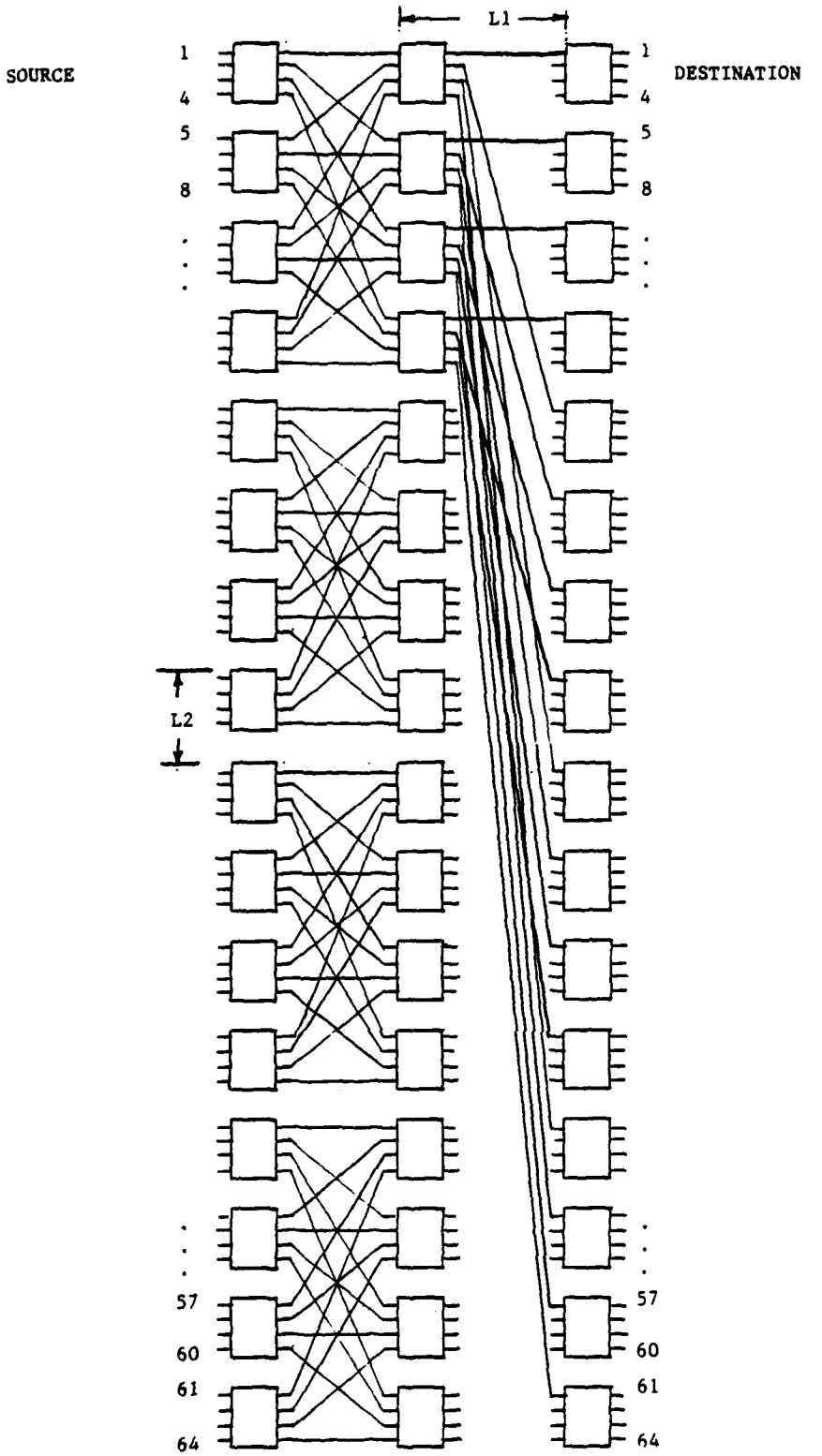


Figure 5: Layout of a 64*64 Banyan network built from 4*4 module chips.

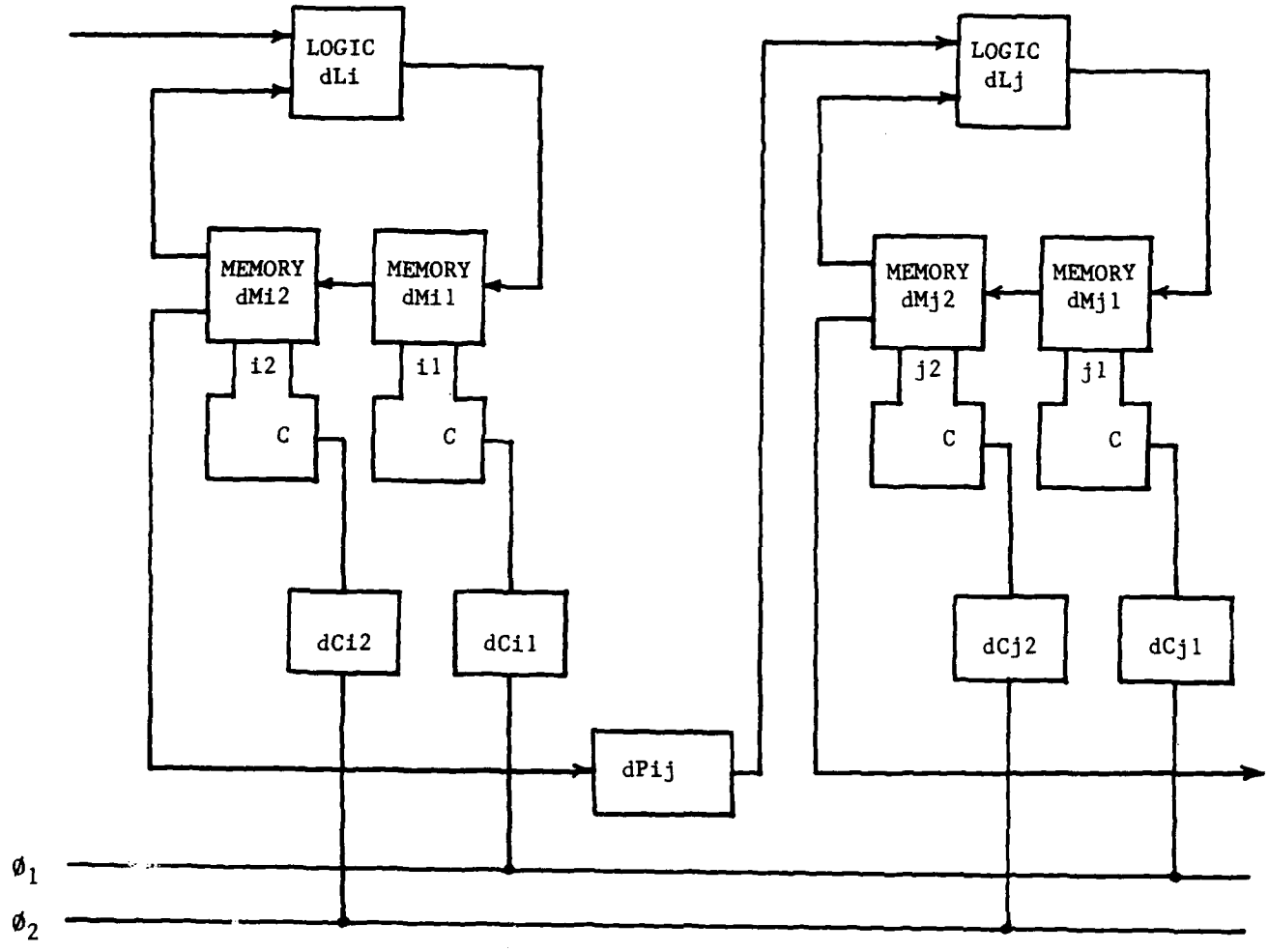


Figure 6: Delay model for two adjacent synchronous modules.

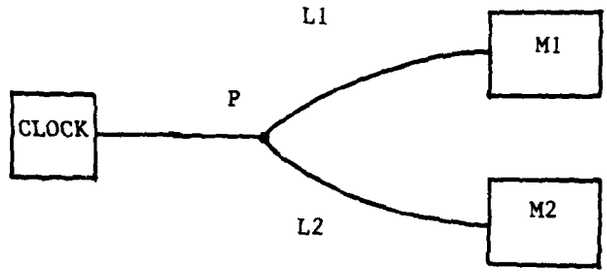


Figure 7: Clock distribution to two modules with clock path lengths L1 and L2.

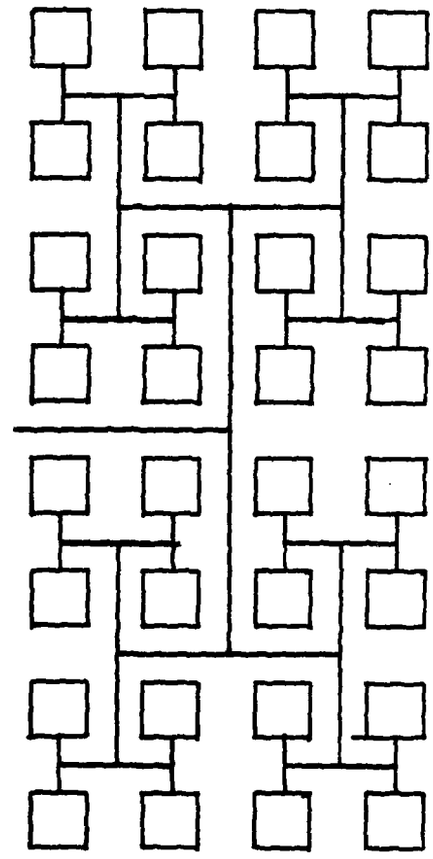


Figure 8: Clock distribution for a 16*16 Banyan network.

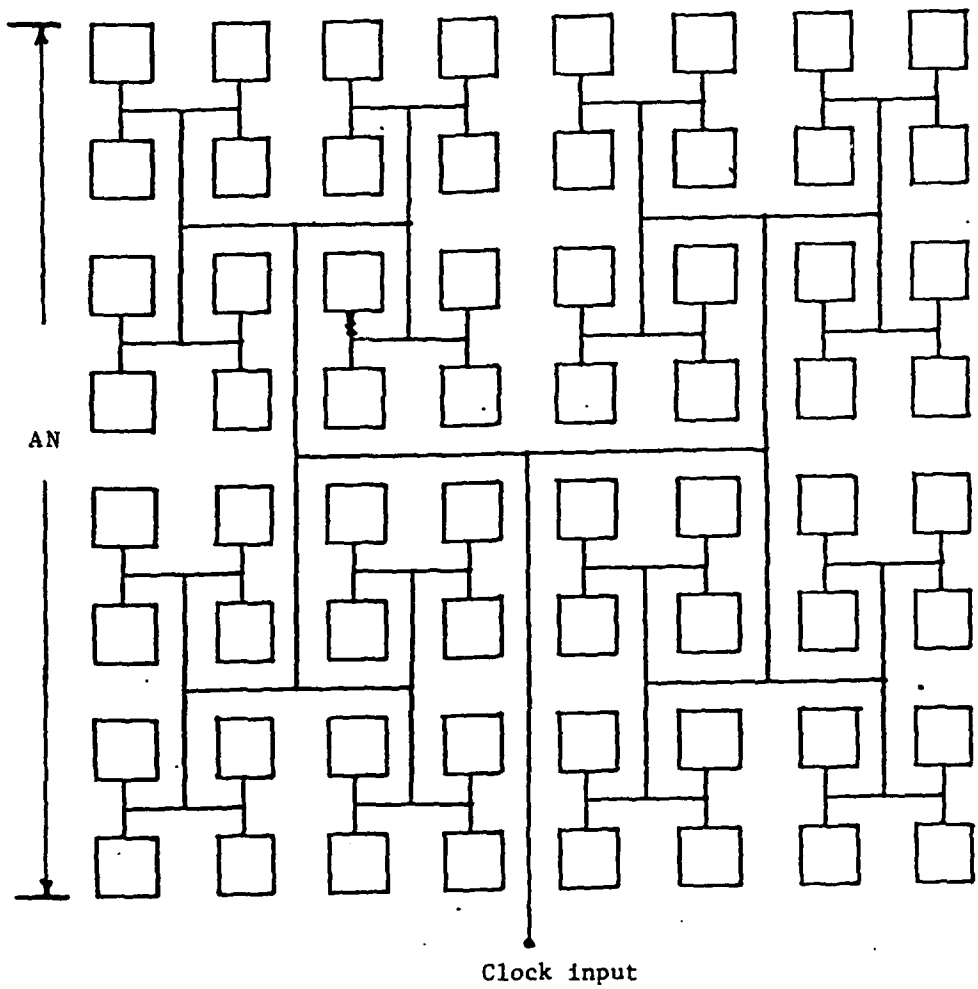


Figure 9: Clock distribution for an 8*8 Crossbar network

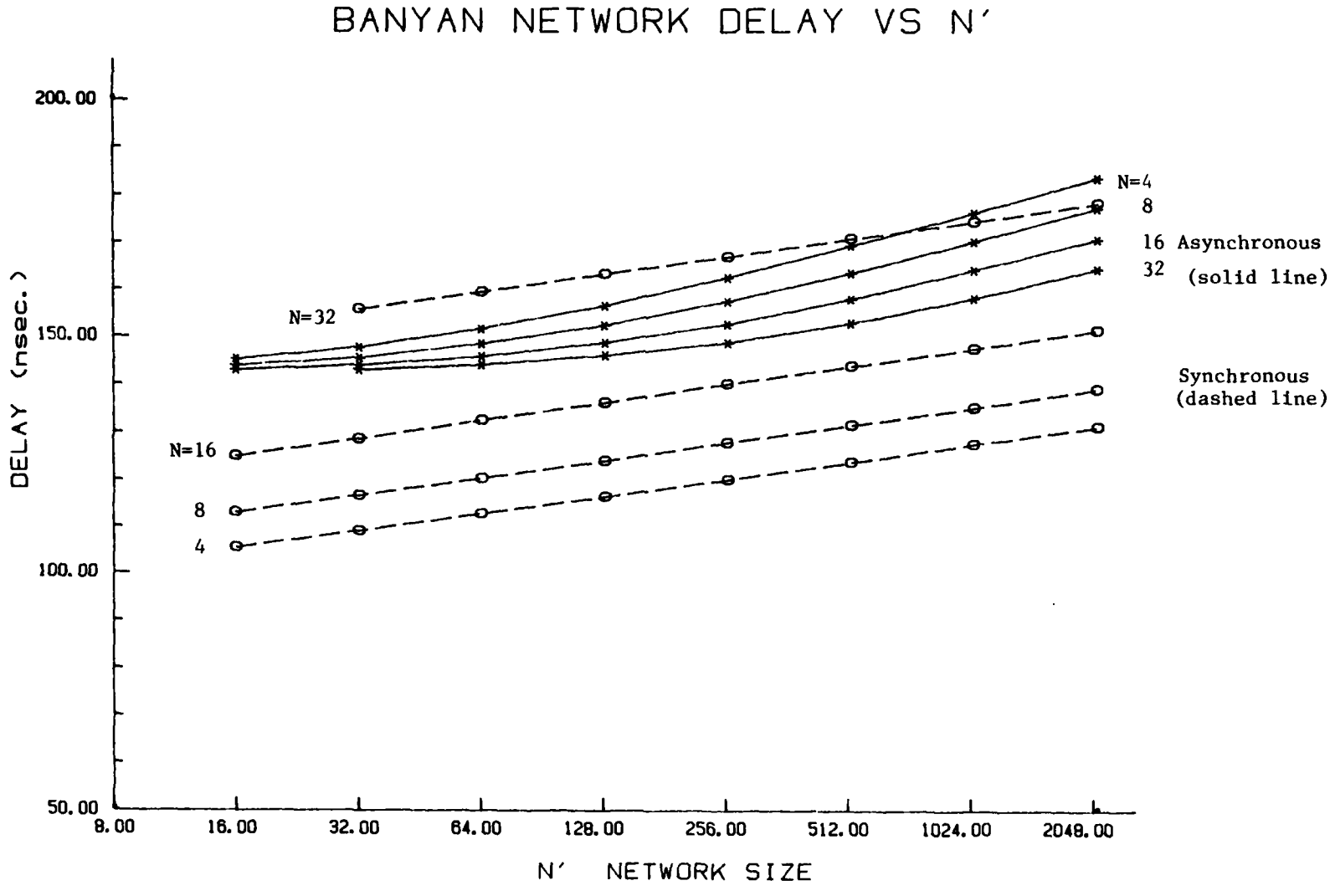


Figure 10: Delay variation of Banyan network for different network sizes.

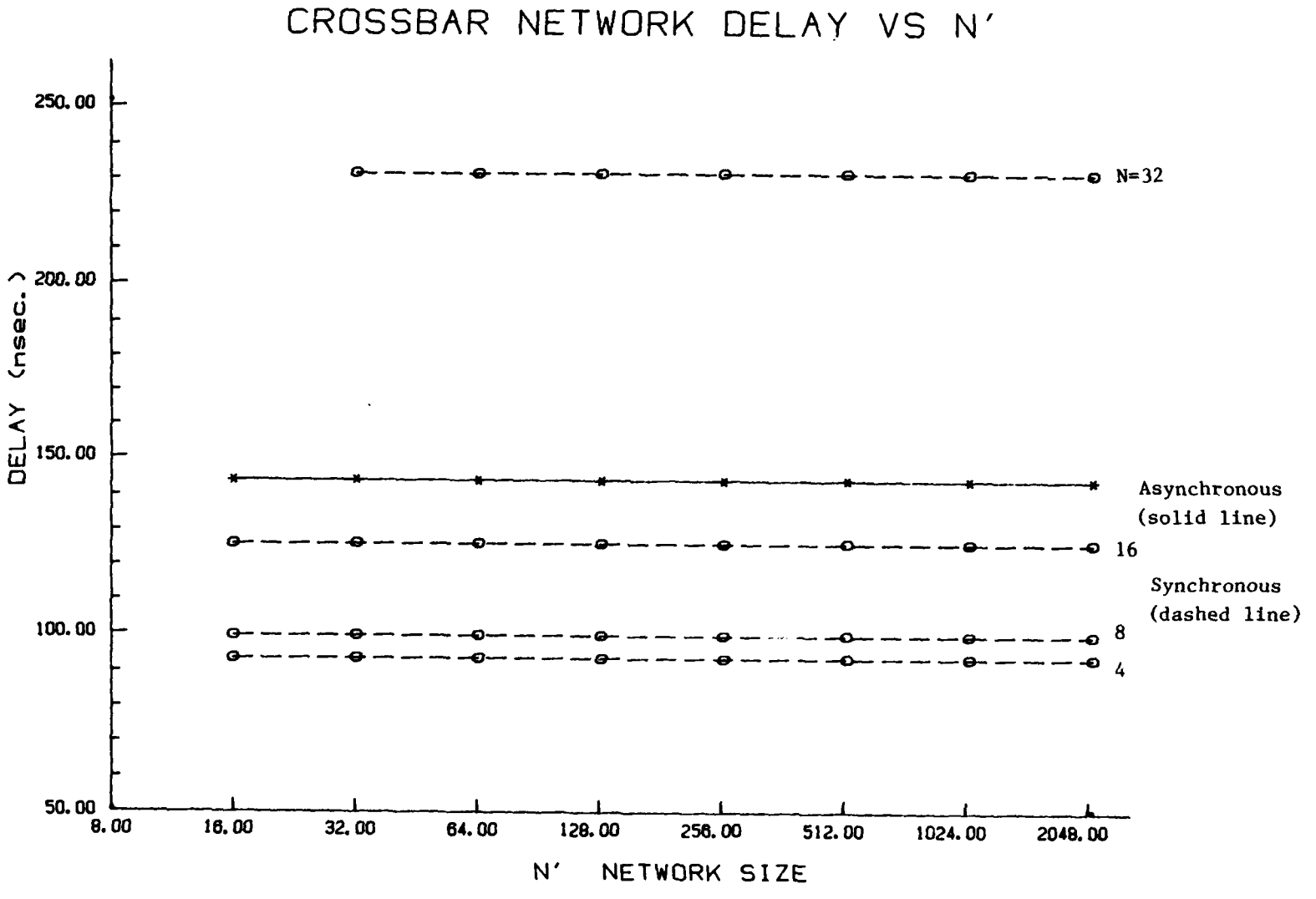


Figure 11: Delay variation of Crossbar network for different network sizes.

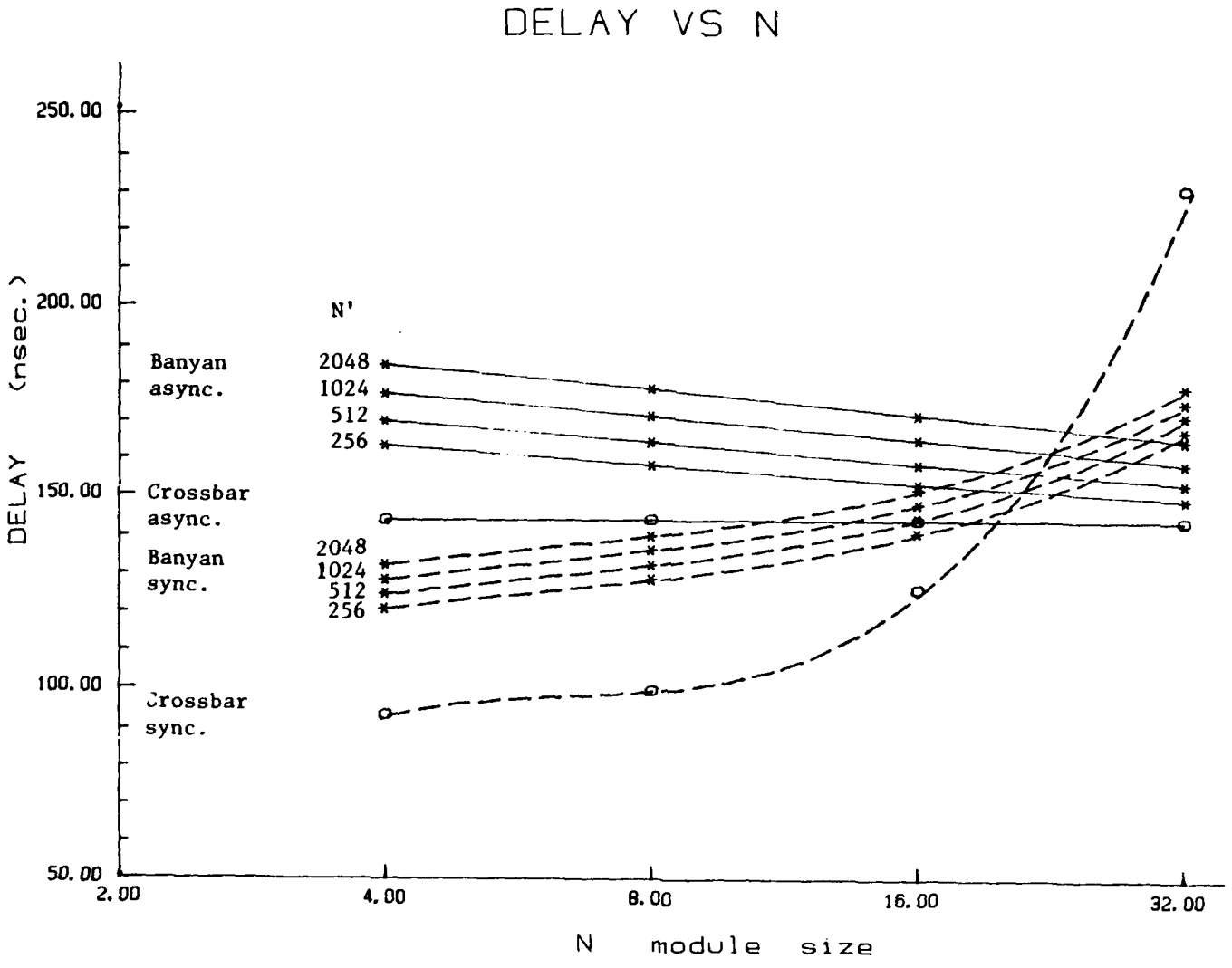


Figure 12: Delay variation in Crossbar and Banyan networks for different module size.

APPENDIX III

IEEE TRANSACTIONS
ON COMPUTERS
(NOV. 1982)

Pin Limitations and Partitioning of VLSI Interconnection Networks

MARK A. FRANKLIN, DONALD F. WANN, AND WILLIAM J. THOMAS

Abstract—Multiple processor interconnection networks can be characterized as having N' inputs and N' outputs, each being B' bits wide. A major implementation constraint of large networks in the VLSI environment is the number of pins available on a chip, N_p. Construction of large networks requires partitioning of the $N' * N' * B'$ network into a collection of $N * N$ switch modules with each input and output port being B ($B \leq B'$) bits wide. If each module corresponds to a single chip, then a large network can be implemented by interconnecting the chips in a particular manner. This correspondence presents a methodology for selecting the optimum values of N and B given values of N', B', N_p, and the number of control lines per port. Models for both banyan and crossbar networks are developed and arrangements yielding minimum: 1) number of chips, 2) average delay through the network, and 3) product of number of chips and delay, are presented.

Index terms—Banyan, crossbar, interconnection networks, multiprocessors, pin limitations, multiprocessors, synchronization.

I. Introduction

Recently a variety of physically local, closely coupled multiple processor computer systems have been proposed and, in some cases, built [1]-[4]. One key issue in the design of such systems concerns the communications network used by these multiprocessor systems. Various studies have focused on the functional properties of such networks (i.e., permutations, control algorithms), on their complexity, and on performance issues [5]-[9]. Network complexity has often been measured by the number of elementary switching components needed, while performance has been determined by the average number of components through which a message must pass (i.e., average delay). Complexity and performance questions have been examined in the context of VLSI implementation of such interconnection networks. Franklin [10] has compared crossbar and banyan networks operating in circuit-switched mode in terms of their space (area) and time (delay) requirements. Wise [11] presents a three-dimensional VLSI layout arrangement for banyan networks. Thompson [12] and Padmanabhan [13] derive lower bounds on the area and time complexity for a number of networks.

Closer examination of VLSI network implementation problems however show that pin limitations, rather than chip area or logical component limitations, are a major constraint in designing very large networks. Consider a network with N' inputs, M' outputs, and with each output being B' bits wide ($N' * M' * B'$). The number of required pin connections (ignoring power, ground, and general control) for a single chip implementation is given by $B'(N' + M')$. For a square network of size thirty-two, with $B' = 16$, the number of pins required would thus be 1024. This is much larger than is commonly available on commercial integrated circuit carriers and is near the limits of advanced ceramic modules where the entire area of one module surface is used for pin placement.

Manuscript received January 11, 1982; revised June 17, 1982. This work was supported in part by NSF under Grant MCS-78-20731 and ONR under Contract N00014-80-0761.

The authors are with the Department of Electrical Engineering, Washington University, St. Louis, MO 63130.

This correspondence focuses on two important problems encountered in the design of interconnecting networks. First we examine how partitioning the network can be used to overcome the pin limitation constraint. We develop relations for optimum partition configurations as a function of the major network parameters including total number of integrated circuit chips and average message delay through the network. Secondly, we identify an unusual problem, called word inconsistency, that is created when local control of the partitioned network (which may be highly desirable for its ease of modular expansion/contraction) is used, and propose a control structure and protocol that overcomes this problem.

One approach to partitioning is to implement a large network ($N' * N'$) requiring many pins as an interconnected set of smaller subnetworks ($N * N$) where each smaller subnetwork can be contained on a single chip whose packaging design meets the pin constraints.

Another approach is to slice the network so that one creates a set of network planes with each plane handling one or more bits (e.g., B bits) of the B' wide data path. It is this bit slicing procedure which can lead to synchronization problems. Consider the situation where message routing through the network is via local control logic present at each network switch node. If several messages arrive close together in time and request the same output, it is possible that the output word will contain bits from a number of input sources and thus be in error. This situation is referred to as a word inconsistency.

The next section of this paper deals with determining the "best" combination of data path slice size B and subnetwork size N such that both chip count and bandwidth of the overall $N' * N'$ network are minimized and the chip pin constraints are satisfied. Then the word inconsistency problem is treated, methods for its detection are discussed, and a protocol and associated asynchronous switch module are presented.

II. The Basic Model

The following parameters specify the space over which the data path slice B and the chip subnetwork size N must be selected.

1) N': An overall network size ($N \leq N'$).
2) B': A data path width ($B \leq B'$).
3) T: An intrachip network type (e.g., the interconnection network implemented within the $N * N$ chip might be a crossbar).
4) T': An interchip network type (e.g., the interconnection network implemented between the $N * N$ chips to achieve the overall $N' * N'$ network might be an Omega network).
5) N_p: The maximum number of pins allowed on a chip.
6) N_k: The number of pins on a chip allocated to power, ground, and control.

Figs. 1, 2, and 3 illustrate a general $N' * N'$ network, and a possible decomposition of a sample $16 * 16 * 8$ network ($N' = 16$, $B' = 8$). The network shown in Fig. 2 uses $4 * 4 * 2$ subnetwork chips are also possible and each implementation will have a different total chip count, time delay, and chip pin requirements.

The basic model consists of two parts. The first relates to the chip count, while the second concerns network time delay. Only square, fully connected networks (i.e., there is a path from each input port to each output port) are considered. Note that certain input/output paths may have a common subpath and this may result in messages being temporarily blocked.

Two types of interchip networks (T') are considered: the incremental crossbar, CB, and the modified banyan, BA [14], [15]. Thus $T' = CB$ or $T' = BA$. While there are many ways of designing a crossbar network (e.g., demultiplexer/multiplexer configuration, switched multiple buses, etc.), the incremental crossbar design (Fig. 4) can be expanded on a unit basis by adding basic switch modules in a row-column arrangement. This modularity property permits flexible expansion while retaining the nonblocking and full connection properties of the crossbar. A price is paid for these properties in terms of number of switches and pins required on a switch module. While the number of switches required per switch module may not be a serious constraint with VLSI technology, the problem of pin constraints is often severe. For the incremental crossbar, the modularity property requires $4NB$ data pins to implement an $N * N * B$ switch module while the banyan, a blocking network, requires $2NB$ data pins. The modified banyan networks are assumed to be of the form shown in Fig. 2 where all output ports of a switch module are used in establishing connections between successive module rows.

To make global comparisons simpler, and to eliminate blocking at the switch module level, it is assumed that the intrachip network used is the incremental crossbar ($T = CB$). This is reasonable since component limitations within a chip are unlikely to be present in the VLSI network environment assumed. Furthermore, over the range of intrachip network sizes considered, the differences between the CB and BA in terms of space-time product measures are not substantial [10].

A. Chip Count Model

The number of $N * N * B$ chips required to implement an $N' * N' * B'$ incremental crossbar network is given by

$$N_{cb} = \lceil B'/B \rceil \lceil N'/N \rceil^2. \tag{1}$$

The banyan network is one of the class of blocking networks whose logical complexity grows as $0(N \log N)$ rather than $0(N ** 2)$ and the number of $N * N * B$ chips needed to implement an $N' * N' * B'$ banyan network is given by

$$N_{ba} = \lceil B'/B \rceil \lceil N'/N \rceil \lceil \log_N N' \rceil. \tag{2}$$

The first term in this expression is the number of bit slices that is required. The second term represents the number of chips at each level, while the third term is the number of levels.

B. Time Delay Model

A model giving the average time for a signal to propagate through a network must include the time to traverse each of the chips, the time to propagate from chip to chip, and since the bit slice approach separates the bits in a data word, the additional time that is needed to make certain that all the data bits have completed their movement through the $N' * N' * B'$ network.

The average delay associated with a basic switch module will be designated as D_{cb} since these modules are assumed to have a crossbar configuration. Path establishment delays are not considered here. The delay of a pin driver and associated interconnection wires between modules is denoted by D_{im}. The intermodule delays for the CB and BA networks are different and will be denoted as D_{imcb} and D_{imba}. Additional synchronization delay introduced by the designer to assure that all data bits have traversed the network will be represented by D_{syncb} and D_{synba}. For the CB network the average delay can be determined by examining Fig. 4. Assume that each switch module implemented on a single chip represents an $N * N$ CB network. The pin drivers for each module are also located on the chip. For this arrangement the number of modules in an average path is $\lceil N'/N \rceil$ and each intermodule path has the same delay D_{imcb}. Therefore the average network delay D'_{cb} is given by

$$D'_{cb} = \lceil N'/N \rceil D_{cb} + \lceil N'/N \rceil D_{imcb} + D_{syncb}. \tag{3a}$$

Note that a circuit-switched design is assumed here with no pipelining between modules.

For the BA network the number of switch modules and the number of intermodule connections is $\lceil \log_N N' \rceil$. Here, because of the connection topology, the intermodule paths are not constant in length. The average delay, D_{ba}, through such a network is given by

$$D_{ba} = \lceil \log_N N' \rceil D_{cb} + \lceil \log_N N' \rceil D_{imba} + D_{synba}. \tag{4a}$$

The delays discussed above are due to the physical nature of the devices being used. In addition, delays will be present due to logical conflicts which arise when there is contention for either common output ports or common network paths. These delays are probabilistic in nature and their evaluation requires a model for general network operation. Assume a fully saturated system where messages are always available to be sent at each input port, that messages are of equal length, begin and end simultaneously, and that output ports are uniformly addressed. Since several inputs may select the same output port, output contention may occur. In the case of the banyan network there is the additional probability that a path may not be available. Thus the probability that a message will be blocked due to output contention is a function of network size in the case of crossbar networks, and is a function of both network size and module (subnetwork) size in the case of banyan networks. This blocking probability is denoted by $P_{N',N}$ and is given in Fig. 5. These curves were derived in the manner suggested by Patel [16].

A protocol must be provided to handle those cases where messages are blocked. This protocol influences the overall delay encountered by a message and must be accounted for in delay calculations. Assume that a message retry protocol is used where blocked messages reenter the system with the next message batch and that each message batch is roughly a random and independent grouping of messages. To minimize the effect of message length and possible delays introduced by output port devices, assume also that messages are very short and that output port devices are very fast (i.e., they are never a bottleneck). Under these conditions, the average delay D^*_{cb} through the CB network is given by

$$D^*_{cb} = D'_{cb}(1 - P_{N',N}) + 2D'_{cb}(1 - P_{N',N})P_{N',N} +$$

$$3D'_{cb}(1 - P_{N',N})P^2_{N',N} \cdots$$

which can be reduced to

$$D^*_{cb} = D'_{cb}/(1 - P_{N',N}). \tag{3b}$$

Similarly, the average delay through the BA network is given by

$$D^*_{ba} = D'_{ba}/(1 - P_{N',N}). \tag{4b}$$

C. Pin Constraints

For a square $N * N * B$ chip with N_k pins allocated to power, ground, and control, the pin constraint is given by

$$N_p \geq KB \cdots N_k \tag{5}$$

where $K = 4$ for the CB network and $K = 2$ for the BA network. The equality will be used since it is generally advantageous to utilize as many available pins as possible. Two cases may be considered. Case 1 is the situation where the number of data pins is much larger than N_k. This is typical of a clocked system where a small number of clock lines are needed to synchronize all the data lines.

For case 2, N_k is not negligible and there is a control line overhead associated with the data paths. Assume that the number of control lines is proportional to the number of ports, N, on an individual chip (i.e., $N_k = QNK$ where Q is a constant). This model would be appropriate if chips communicated with each other in an asynchronous manner and the control line overhead consisted of request/acknowledge pairs (i.e., $Q = 2$).

These two cases can be expressed as

$$N = \begin{cases} \lfloor N_p/KB \rfloor & \text{[case 1]} \quad (6) \\ \lfloor N_p/(KB + Q) \rfloor & \text{[case 2]}. \quad (7) \end{cases}$$

D. Chip Count Minimization

The pin limitation constraints can now be incorporated by substituting 7 into 1 and 2 to yield

$$N_{cb} = \lceil B'/B \rceil \left\lceil \frac{N'}{\lfloor N_p/K(B+Q) \rfloor} \right\rceil^2 \tag{8}$$

$$N_{ba} = \lceil B'/B \rceil \left\lceil \frac{N'}{\lfloor N_p/K(B+Q) \rfloor} \right\rceil \left\lceil \frac{\log N'}{\log \lfloor N_p/K(B+Q) \rfloor} \right\rceil. \tag{9}$$

The resulting expressions are functions of the path slice B and, for a given network size, number of pins, and control structure, the value of B (and thus N) which minimizes the chip count can be obtained. An approximate way of doing this is to consider situations where N', B', and N_p are large, thus permitting the ceiling and floor function to be removed with the resulting expressions being continuous.

For case 1 the chip count is minimized when B is minimized (i.e., $B = 1$) and N should be selected as N_p/K. Note that the continuous result does not always hold when N', B', or N_p are small. For instance for the BA network with $N_p = 60$, $N' = 128$, and $B' = 16$, a $B = 1$ solution yields $N_{ba} = 160$, while if a two-bit slice is used then the number of required chips is only $N_{ba} = 144$.

For case 2 derivatives can be taken to obtain the optimal B. For the CB network the number of chips is minimized when $B = Q$. This continuous approximation is inadequate over a wide range of values and direct search techniques should be employed. For instance with $Q = 2$, $N' = 512$, $B' = 16$, and $N_p = 90$, a direct search procedure gives an optimum of $B = 3$ and a chip count $N_{ba} = 1026$. Using $B = 1$ in this case results in $N_{ba} = 1680$.

Equations 1 and 2 were evaluated using optimal values of N and B obtained by direct search procedures. Fig. 6 illustrates how the total number of chips varies as a function of the network size for two different values of Q and N_p. For a given N', N_p, and Q, the BA network requires fewer chips than the CB. The curves agree with the observation that the CB grows as $0(N'^2)$, while the BA grows as $0(N' \log N')$. As expected, increasing Q or N' requires a larger number of chips for both networks while increasing the number of pins/chip reduces the total number of chips.

D. Network Delay Minimization

To evaluate the network delays as expressed in (3b) and (4b), it is necessary to determine D_{cb}, D_{im}, and D_{syn}.

The value of D_{cb} has been developed by Franklin [10] using NMOS NOR gates for construction of the crossbar module and is given as

$$D_{cb} = N[2.5mf\tau + \tau(1 + 2.25\alpha_{cb})] = NA. \tag{10}$$

The various parameters of 10 are defined in Table I. The equation assumes a circuit switched CB, and uniformly distributed addressing of module output ports. The first term in the brackets represents the delay through an individual switch within a module, and the second term is the delay between switches within a chip module.

The intermodule delay, D_{im}, is encountered when a signal goes off the chip, propagates along an interconnecting path, and enters another switch module (chip). A buffer (e.g., a series of inverters) must be included within the module to allow the minimum size transistor to drive the module pin and associated load with minimum delay. This delay is given by [17]

$$D_{im} = \tau e \log_e \beta \tag{13}$$

where β is the ratio of the buffer load capacitance to the buffer input transistor gate capacitance. To determine the load capacitance, assume that the driving and receiving pin capacitances are equal and each has a value of C_{pin}, and that modules are placed on a circuit board and interconnected via printed circuit copper paths with capacitance C_{path}. Thus

$$\beta = (2C_{pin} + C_{path})/C_g. \tag{14}$$

For the CB network, the planar topology of the network ensures that the spacing between modules will generally be less than one inch. Then pin capacitance will dominate and $\beta_{cb} = 2C_{pin}/C_g$.

Given a limited number of circuit board connect layers, the nonplanar topology of the BA network results in a varying distance between modules and C_{path} will vary according to the banyan level. Since the number of levels required for a specific configuration is not known *a priori*, the inclusion of a variable C_{path} complicates the computation. As a simple approximation C_{path} can be taken as the longest path encountered by the network with this longest path corresponding to the last level of the network. The capacitance of a typical printed circuit path is approximately 1 pF/in and thus for a reasonable size board, C_{path} can be taken as 12 pF. By decreasing the pin driver area as the banyan level decreases this value applies to all levels.

The final delay component corresponds to the synchronization delay which depends upon the design technique used to determine that all bits have traversed the network. For clocked designs, a standard design practice is to include a tolerance region that is proportional to the average delay time. Thus

$$D^*_{cb} = K_1\lceil N'/N \rceil (D_{cb} + D_{imcb})/(1 - P_{N',N}) \tag{15}$$

and

$$D^*_{ba} = K_1\lceil \log_N N' \rceil (D_{cb} + D_{imba})/(1 - P_{N',N}) \tag{16}$$

where $K_1 = 1 + K_s$. K_s determines the guard region and might for instance be taken as 0.1.

The pin constraint equations derived earlier can now be substituted into 15 and 16 and overall expressions for the delay as a function of the path width B obtained. Optimum values for B and N which minimize the overall delay can be found following the same procedures used in minimizing chip count. Fig. 7 shows the overall delay as a function of parameters N_p, N', and network type T'. These delays use the optimum B and N values.

For the CB network the optimum occurs when $B = 1$ for all but very small networks. Delay clearly increases as the number of control pins increases, and as the number of pins on the chip decreases. This is due to the fact that fewer pins available for data paths result in more chips and more chip-to-chip (intermodule) transfer delays.

For the banyan network a wide range of optimal B and N values is obtained. Due to the fact that the delay does not change over the Q and N_p ranges considered ($Q \in [0, 2]$ and $N_p \in [60, 120]$) only a single BA curve is shown in Fig. 7. To understand this consider the range of N and B. The minimum values of N and B are 1 while the maximum values may be obtained from the pin constraint expression 5. For example with $N' = 512$, $B' = 16$, $Q = 0$, and $N_p = 90$, the pin constraint equation yields $N \in [1, 45]$ and $B \in [1, 45]$. For $Q = 2$ one obtains $N \in [1, 15]$ and $B \in [1, 43]$. The value of N which minimizes the delay (i.e., $N = 8$) and the resulting B for $Q \in [0, 2]$ and $N_p \in [60, 120]$ are both within their allowable ranges. Thus changing Q and N_p will effect the value of B; however, the values of the optimal N will remain constant, and thus the curves for delay will be the same over wide Q and N_p ranges. In effect the optimum N (and thus minimum delay) is not on the constraint boundary.

F. Chip Count-Delay Product Minimization

The chip count-delay product, P, can be obtained by multiplying the appropriate equations given previously. Earlier discussion indicated that for large networks ($N' \geq 64$, $B' \geq 16$), both chip count and delay were generally minimized in the CB case with $B = 1$. Consequently, the product is also minimized with this choice.

For the BA network, the situation is more complex and search methods must be employed to obtain the optimal B and N values. Consider the case of $Q = 0$ and $N' = 512$. Table II shows the values of N, B which optimize the number of chips, the delay, and chip count-delay product. The B and N values required for minimizing P fall between those needed for minimization of the chip count and delay measures by themselves. The count minimization is achieved by attempting to place as large a network as possible on a given chip. Delay minimization is achieved by balancing the delays associated with the module network and the delays associated with increasing the number of levels in the overall network. In this case placing as large a network as possible on a chip is not the best strategy from a delay point of view.

Values for N and B which minimize P were obtained for both network types. P is plotted as a function of N' for several values of N_p and Q in Fig. 8. As expected, P increases with increasing N_p. Once again the BA network does better than the crossbar on this overall performance measure.

III. Other Partitioning Issues

An important problem arises when a network is partitioned across the bits in a data word. Consider a particular source, S_i whose B' bits are partitioned into Z planes ($Z = \lceil B'/B \rceil$). Thus S_i can be represented as

$$S_i = (S_i^1, S_i^2, \cdots S_i^p \cdots S_i^z) \tag{17}$$

where the superscript identifies the plane. A specific plane, p, then can interconnct only the bits from the pth partition of each source, (S_i^p), to the pth partition of each destination, (D_k^p). This is illustrated in Fig. 9. Since the interconnection network supports concurrent transactions, two sources, say S_i and S_j, may make simultaneous requests to be connected to, say D_k. The arbitration of these requests, and the appropriate path selection through the network, can be handled either on a central basis (i.e., individual crosspoints controlled from one central location) or on a distributed basis (i.e., each crosspoint making its own decision). Because of reliability and extendability considerations—of particular importance in VLSI—a modular decentralized control structure has significant realization and perhaps performance advantages. Unfortunately, if a distributed control arbitration arrangement is used a nonhomogeneous word can be received. To understand this, suppose that S_i and S_j are both requesting a path to D_k at about the same time. It is possible that a path from S_i to D_k will be established on plane p (S_i^p to D_k^p) while on plane q a path from S_j to D_k will be established (S_j^q to D_k^q). This is illustrated for the case where $B' = 16$, $B = 1$ in Fig. 10 in which S_j captures the path of plane 14. The received D_k is connected to

$$D_k = (S_i^1 \cdots S_i^{13}, S_j^{14}, S_i^{15}, S_i^{16}) \tag{18}$$

and an inconsistency occurs at plane 14. The path from S_i^{14} to D_k^{14} is blocked because of the connection to S_j^{14}.

Notice that there is a switching module at which requests from S_i and S_j intersect, and at which an arbitration may occur. Neglecting the arbitration uncertainty, which is a small effect (18), (19), the path to the destination is awarded to the request that arrives first. But a source request is divided into Z requests, one for each plane. Therefore even if one of the sources makes a request prior to another source, the Z plane level requests may not arrive at their respective arbitration modules first on all planes. This can happen because the propagation delay along a path is not a constant from plane to plane. It is thus possible that the propagation delay S_i to the arbitration module on plane p will be greater than the propagation delay from S_j, while on plane q the propagation delay from S_i will be less than from S_j. Thus a received B' bit word can contain a nonhomogeneous set of bits if the interconnection network utilizes a distributed methodology for establishment of the paths through each of the partitions, rather than a single path controller assignment strategy. If this local distributed control structure is selected (because of its other desirable features) then some mechanism must be included to detect that each of the paths established to D_k through the partitions are from the same source. A word received at the destination is inconsistent if all its B' bits are not from the same source. Thus an important issue is to determine what types of decentralized control allow modularity and also permit the detection of inconsistent words.

Decentralized control implies that there is no global sequencing; thus communication between chips is accomplished via some form of self-timed protocol. Within a chip, however, the transactions may be carried out with a local clock or with a self-timed methodology. A typical protocol would include provision for 1) detection of word inconsistency, 2) detection of a blocked path, and 3) initiation of a retry in either case. Part 1) of the protocol can be accomplished in either a deterministic or a probabilistic manner and the detection can be made at either the source or the destination. Consider detection at the destination. If each D_k^p can determine the source to which it is connected, the word consistency can be evaluated. Since there are N' possible sources, this requires at most $\log_2 N'$ address bits. One possible protocol is to establish the path and then have the source serially transmit the same code word (of length $\log_2 N'$) on each plane. For consistency, this transmitted word must be the same for each of the destination partitions and this could easily be detected by comparing all the received bits in a word. This is a deterministic destination validation procedure.

It is also possible to have the destination perform a probabilistic validation. Using this technique an extra parity bit is added to the word width, so it becomes $B' + 1$ bits wide. The destination checks the parity of each of the received words. An inconsistent connection can result in a parity error. Although there is a nonzero probability that one or more words can be received with proper parity even if the connection is inconsistent, this probability declines very rapidly as the number of exchanged data words increases (assuming random data).

When the detection of blocking is included in the protocol, thus requiring the transmissin of control signals from the network back to the source, deterministic consistency validation at the source becomes feasible, and our investigations have indicated that this is the method of choice if a completely delay insensitive protocol is desired. A possible switch module design for an incremental crossbar network, which has provision for path arbitration, path establishment, blocking detection, consistency evaluation, and path clearance, is shown (for $B = 1$) in Fig. 11. All signals are transition encoded. Path establishment is achieved by transmitting a header word. Assertion of R^1 indicates a 1 and R^0 at 0. To select the destination in column C a word containing C bits and having a single 1 in the last position (e.g., 10 $\cdots$ 0) is transmitted. An arbitration unit handles simultaneous requests from the left and top of a module. If the first header bit received is a zero, the module absorbs the bit and sets its data path to the horizontal position—otherwise, it passes the bit on to the next module to its right. When a module senses a first header bit that is nonzero, it indicates that the selected column has been reached and the data path is placed in the corner position. Each bit exchange is acknowledged via the generation of the A signal. If a desired module path is blocked, then NA is generated and sent back to the source. This signal resets the pathway previously established by the header bits, thus allowing the next source request to be accepted. If only acknowledge signals are received after sending the header word, then a path to the destination has been completed. The completion of each of the $\lceil B'/B \rceil$ paths can be detected in this manner. If all paths have been established and thus word consistency determined, then a message can be transmitted. An end of message word received by the destination causes it to generate an NA signal, which clears the path on its return to the source. If the source does not receive acknowledge signals from all the paths, then this indicates word inconsistency and a retry may be instituted after some arbitrary delay.

While word inconsistencies can be detected and corrected through the use of protocols having retry procedures, it is important to quantify how often such retries must be initiated so that performance degradation of the network can be evaluated. A detailed investigation of these issues using a model in which each source has a request that is Poisson distributed is currently underway. Preliminary studies indicate that for many practical arrival rates the probability of retry is relatively small (i.e., under 0.05).

IV. Summary and Conclusions

This correspondence examines the design of multiple processor interconnection networks and presents a methodology for selecting the optimum values of N and B (a switch module size and path width), given values of parameters N', B', T', Q, and N_p (interconnection network size, path width, type, number of control lines, number of pins per switch). Models for both banyan and crossbar networks were developed and arrangements yielding minimum: number of chips, average delay through the network, and product of number of chips and delay were presented. The results show that for the crossbar network a bit slice approach ($B = 1$) produces the optimum arrangement, while for the banyan the optimum is achieved with multiple bits per module. The impact of the number of control lines

on chip count, delay, and product were also discussed.

The issue of bit plane synchronization was presented and a potential problem involving output port reception of words containing bits from several different input ports was characterized. Methods for dealing with this word inconsistency problem were proposed and the functional design of an asynchronous switch module was given.

Research is continuing in this general area with current efforts being directed at two related problems. The first concerns quantification of the asynchronous versus clocked design methodologies in the context of interconnection networks (20). This directly relates to the value of Q used in the analysis presented. The second concerns the incorporation of other physical constraints, such as heat dissipation, into the design process in a manner similar to the use of pin constraints in the work presented here.

REFERENCES

[1] R. J. Swan *et al.*, "Cm*—A modular multi-microprocessor," in *Proc. Nat. Comput. Conf.*, 1977.

[2] J. B. Dennis and D. P. Misunas, "A preliminary architecture for a basic data-flow processor," in *Proc. 2nd Annu. Symp. Comput. Arch.*, Dec. 1974.

[3] M. C. Sejnowski *et al.*, "An overview of the Texas reconfigurable computer," in *Proc. Nat. Comput. Conf.*, 1980.

[4] H. Sullivan and T. R. Baskow, "A large scale, homogeneous, fully distributed parallel machine I," in *Proc. 4th Annu. Symp. Comput. Arch.*, Mar. 1977.

[5] Y. E. Benes;, Mathematical Theory of Connecting Networks and Telephone Traffic. Academic, NY: 1965.

[6] H. J. Siegel, R. J. McMillen, and P. T. Muller, "A survey of interconnection methods for reconfigurable parallel processing systems," in *Proc. Nat. Comput. Conf.*, June 1979.

[7] G. A. Anderson and E. D. Jensen, "Computer interconnection structures. Taxonomy, characteristics, and examples," *Ass. Comput. Mach. Comput. Survey*, vol. 7, Dec. 1975.

[8] K. J. Thurber, "Interconnection networks—A survey and assessment," in *Proc. Nat. Comput. Conf.*, May 1974.

[9] M. A. Franklin, S. A. Kahn, and M. J. Stucki, "Design issues in the development of a modular multiprocessor communications network," in *Proc. 6th Annu. Symp. Comput. Arch.*, Apr. 1979.

[10] M. A. Franklin, "VLSI performance comparison of banyan and crossbar communications networks," *IEEE Trans. Comput.*, vol. C-30, Apr. 1981.

[11] D. S. Wise, "Compact layouts of banyan/FFT networks," in *VLSI Systems and Computations*, H. T. Kung *et al.*, Eds. Rockville, MD: Comput. Sci. Press, 1981.

[12] C. D. Thompson, "Area time complexity for VLSI," in *Proc. 11th Annu. Ass. Comput. Mach. Symp. Theory Comput.*, Apr. 1981.

[13] K. Padmanabhan, "Multiprocessor interconnection networks in a VLSI environment," M.S. thesis, Dep. Elec. Eng., Washington Univ., St. Louis, MO, Dec. 1981.

[14] L. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," in *Proc. 1st Annu. Symp. Comput. Arch.*, 1973.

[15] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Comput.*, vol. C-24, Dec. 1975.

[16] J. H. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Trans. Comput.*, vol. C-30, Oct. 1981.

[17] C. Mead and L. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980.

[18] T. J. Chaney and C. E. Molnar, "Anomalous behavior of synchronizer and arbiter circuits," *IEEE Trans. Comput.*, vol. C-22, Apr. 1973.

[19] G. R. Couranz and D. F. Wann, "Theoretical and experimental behavior of synchronizers operating in the metastable region," *IEEE Trans. Comput.*, vol. C-24, June 1975.

[20] M. A. Franklin and D. F. Wann, "Asynchronous and clocked control structures for VLSI based interconnection networks," in *Proc. 9th Annu. Symp. Comput. Arch.*, 1982.

Parameter	Symbol	Units	Typical value
minimum feature size	λ_{min}	um	3
minimum gate area	$A_{min} = 41^2$	$(um)^2$	36
gate capacitance	C_g	pf	$1.4*10^{-2}$
switch module pin capacitance	C_{pin}	pf	5
transit time	τ	nsec	0.5
NOR gate logic levels per crosspoint	m	—	2
NOR gate fanout	f	—	2
metalization path capacitance to transistor gate capacitance ratio (within switch module)	γ_{CB}	—	0.1
guard region multiplier	K_g	—	0.1
printed circuit path capacitance	C_{path}	pf	1pf/inch
length of side of printed circuit board	S	inches	12

TABLE I
TIME DELAY PARAMETERS

	N	B	CHIP COUNT	CHIP DELAY (nsec)	COUNT-DELAY PRODUCT ($*10^3$)
$N_p = 60$					
COUNT MINIMIZATION	30	1	576	791	455
DELAY MINIMIZATION	8	3	1152	452	521
PRODUCT MINIMIZATION	27	1	608	725	441
$N_p = 90$					
COUNT MINIMIZATION	45	1	384	1112	427
DELAY MINIMIZATION	8	5	768	452	347
PRODUCT MINIMIZATION	11	4	564	553	312
$N_p = 120$					
COUNT MINIMIZATION	60	1	288	1423	410
DELAY MINIMIZATION	8	7	576	452	260
PRODUCT MINIMIZATION	27	2	304	725	220

TABLE II
BANYAN NETWORK MINIMIZATION RESULTS ($N' = 512$, $Q = 0$, $B' = 16$)

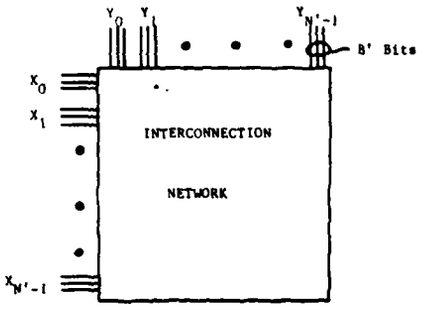


Fig. 1. An $N' \times N'$ network (B' wide data paths).

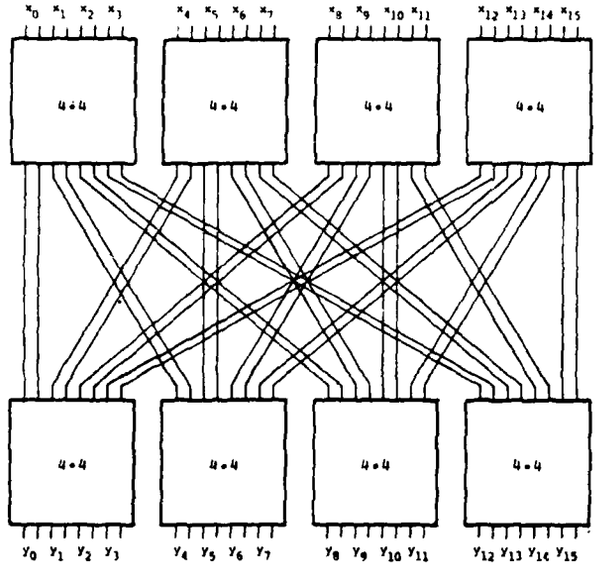


Fig. 2. Decomposition of a 16 • 16 • 8 network using 4 • 4 • 2 network chips (one plane of four shown).

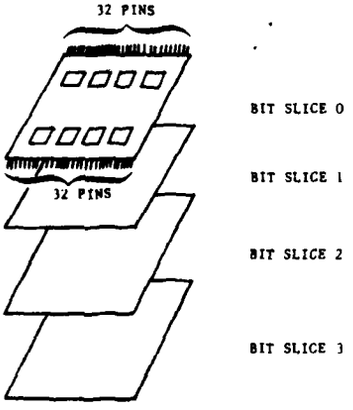


Fig. 3. Four planes used to implement a 16 • 16 • 8 network.

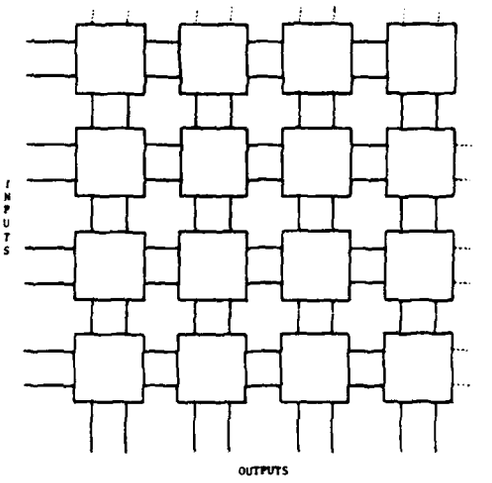


Fig. 4. An 8 • 8 • 1 network composed of 2 • 2 • 1 chips arranged in an incremental crossbar configuration.

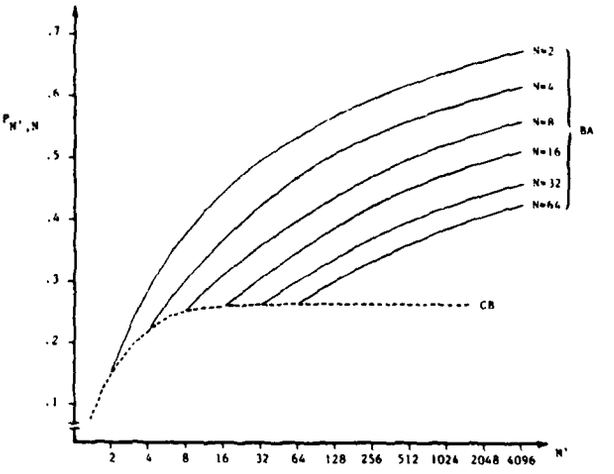


Fig. 5. Probability of blocking $P_{N',N}$ versus network size N'.

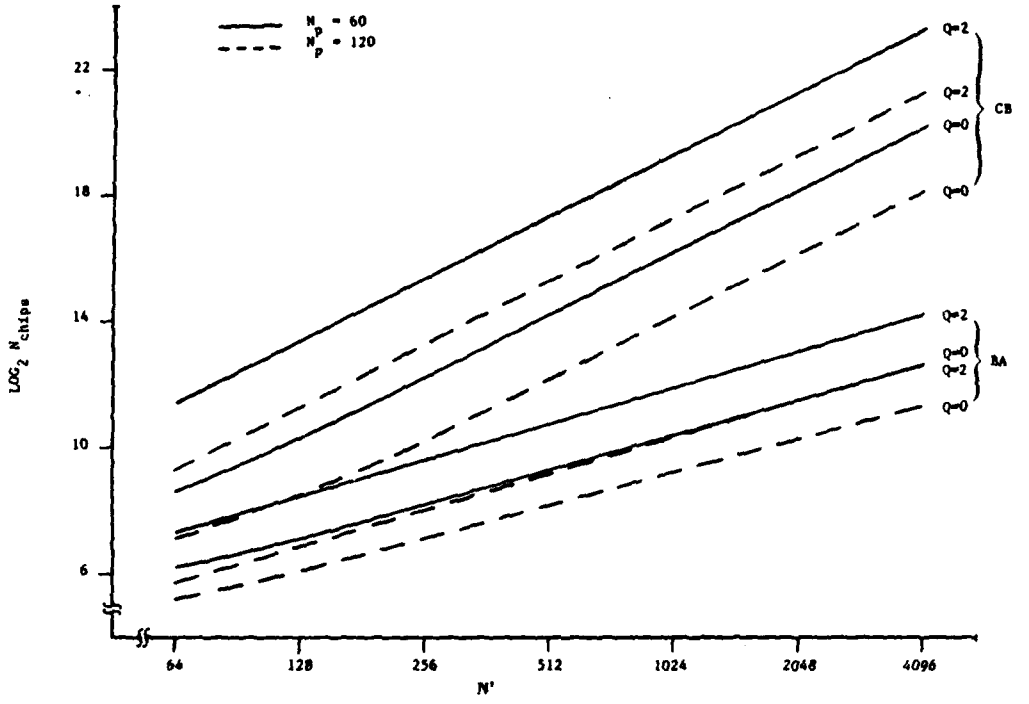


Fig. 6. Optimal number of chips, N_{chips}, versus network size N'.

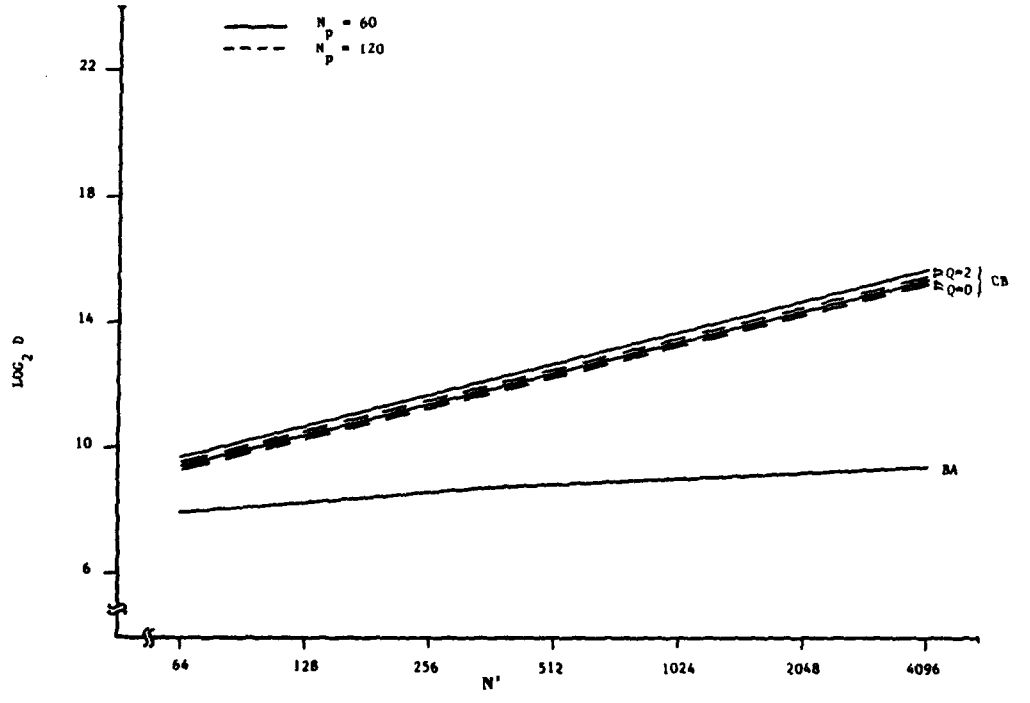


Fig. 7. Optimal delay, D, versus network size N'.

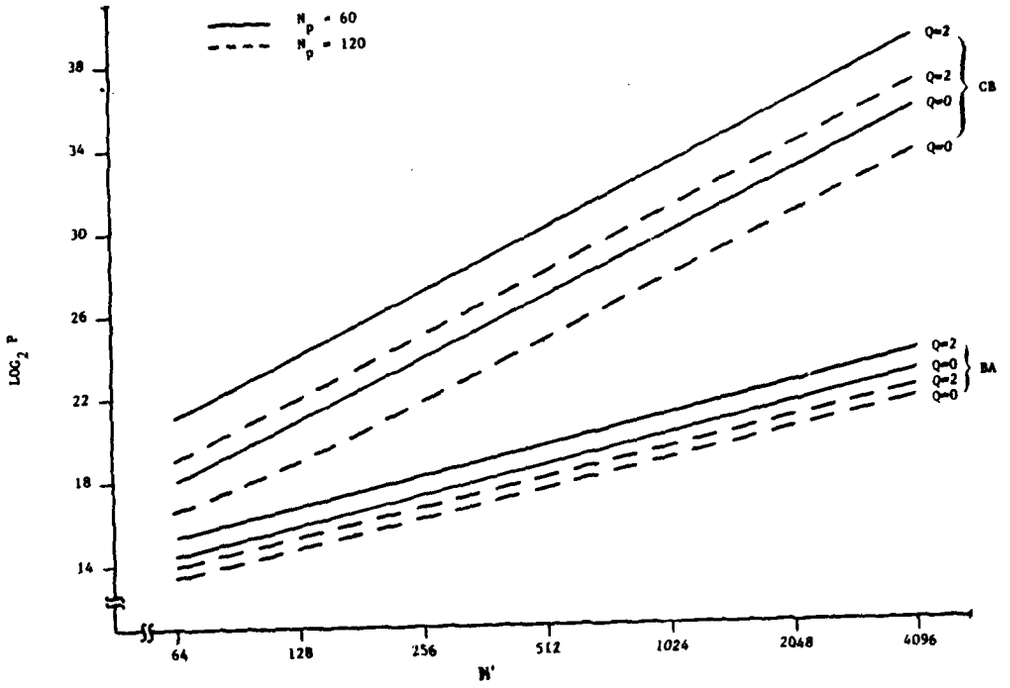


Fig. 8. Optimal performance measure, P, versus network size N'.

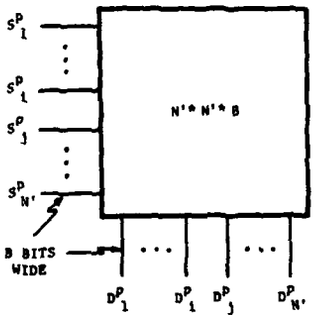


Fig. 9. Interconnection network for Pth plane using $N * N * B$ chips.

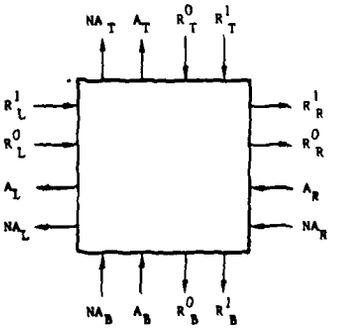


Fig. 11. Delay-insensitive switch module (L = left, R = right, T = top, B = bottom).

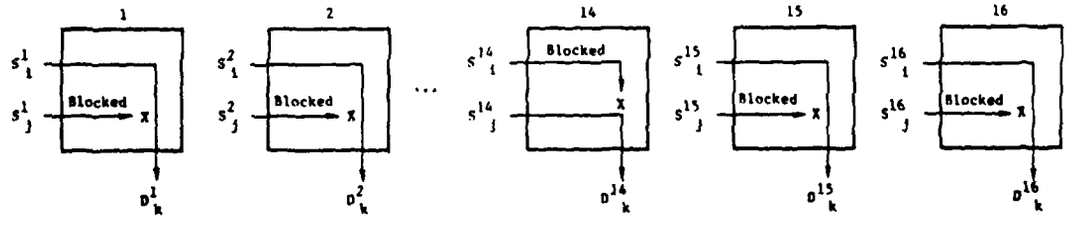


Fig. 10. Connections for S_i and S_j requesting D_k with inconsistency in 14th plane (B' = 16, B = 1).

DESIGN AND VLSI IMPLEMENTATION OF A SYNCHRONOUS CROSSPOINT SWITCH

Sanjay Dhar and Praveen Bhatia
Department of Electrical Engineering,
Washington University,
St. Louis, MO 63130.

1.0 Introduction

Exploitation of hardware concurrency has been made possible by the availability of low cost yet powerful microprocessor chips. Design of systems which have a large number of processors has received particular attention and a number of such systems have been proposed [SWANN77, SULL77, SEJN801. A central issue in the multiprocessor system design is related to the network which maintains the communication among the individual processors. The Crossbar network has received particular attention [FRAN81A, FRAN81D] because of its planar and modular construction which leads to easy VLSI implementation.

A basic building block of the Crossbar network, the crosspoint switch, has been suggested in PADM81. It is possible to construct a full Crossbar network by interconnection of such crosspoint switches. A 4 * 4 Crossbar network formed by interconnection of such switches is shown in Figure 1.

In the crossbar network, any source i can communicate with any destination j provided destinaton j is not busy. The network should therefore have provisions for path establishment, transfer of data from source to destination, detection of a blocked path and path clearing. The mode of data transfer in the network can be circuit switched or pipelined. The pipelined mode of data transfer, in which the data forms a queue in the network on its way from the source to the destination, results in a larger data rate as compared to the circuit switched mode. The data transfer through the network

is therefore chosen to be pipelined.

2.0 Functional Description

2.1 Synchronous Switch Characteristics

Figure 2 illustrates a synchronous crosspoint switch and the signals associated with it. The switch module has two connections per side. The sides are identified by the second letter of the signal (i.e. L=left, T=top, R=right, B=bottom). The first letter of a signal denotes the function of the signal: "D" indicates that the signal is used for data transfer while "N" indicates that the signal is used for acknowledge or not-acknowledge of the data transfer signal. Consider a module i in a particular path. The left side of module i has a path DL that is used when data (a binary 1 or a binary 0) is sent to module i from the left. Similarly, the top side has a path DT that is used to send data to module i from the top. The right side has a path DR that is used when data is sent from module i to the module at the right. Similarly, the bottom side has a path DB that is used to send data from module i to the module at the bottom. The pipelined mode of data transfer is achieved by transferring one bit of data from one crosspoint switch to the next crosspoint switch all along the path. The not-acknowledge lines are used to indicate a blocked path. The left side has a path NL that is used by module i to indicate a blocked path to the module at the left. Similarly, the path NT is used to indicate a blocked path to the module at the top. The path NR is used by the module at the right to indicate a blocked path to module i. Similarly, the path NB is used by the module at the bottom to indicate a blocked path to module i. Thus, there are eight paths of which four are input and four are output paths. In addition to these eight paths, two more paths

for a two phase clock are necessary. A reset line is present whose function is explained later on.

2.2 Switch states

The crosspoint switch can have five data connection states. These are shown in Figure 3 and correspond to: the horizontal and vertical paths inactive (state I); the horizontal path active (state H); the vertical path active (state V); the horizontal and vertical paths active (state HV); and the corner path from the left to the bottom side active (state C). It is to be noted that a switch state corresponding to a path from the top to the right side is not present as it is not necessary in the crossbar network. The actual number of states required for this implementation is more than five; the remaining states and the reason why they are necessary will be explained in sections 2.3 and 2.5.

2.3 Path Establishment

In order to be able to transmit data it is necessary that a path be established from the source to the destination. We now describe the protocol by which such a path is established.

Assume that all switch modules are in their inactive state. In order to establish a path from source Si to destination Dk (Figure 4), it is necessary that all switches in row i from column 1 to column (k-1) be set to their horizontal states, the switch at the intersection of row i and column k be set to its corner state and all switches in column k from row (i+1) to row N be set to their vertical states. In order to achieve this the source sends a path establishment vector of length (k+1) bits. The first bit of the vector is always a 1 and will be referred to as the switch enable bit; the next (k-1) bits are 0; the (k+1)th bit is a 1. This final bit will be referred

to as the column select bit. Thus if the destination is D4, the path establishment vector is 10001.

The response of a switch module to the path establishment vector bits arriving from the left is as follows: A module which is in the inactive state (state I) examines the bits it receives on the DL line. If this bit is a 0 the module takes no action (remains in state I). If the bit on the DL line is a 1 (the switch enable bit) the module changes to the enabled state (state I-En). A module which is in state I-En responds to a 0 on the DL line by changing to the horizontal state and transmitting a 1 on the DR line; however, if the bit on the DL line is a 1 the module changes to the corner state and transmits a 1 to the bottom on the DB line. A module in the horizontal state passes on the bits on the DL line on to the DR line and the state of the switch remains the same while a module in the corner state passes the bits on the DL line to the DB line with no change in state. Consider the example where a path has to be established between source Si and destination Dk. The modules in row i from column 1 to column (k-1) enter the horizontal state while the module in column k enters the corner state. The module immediately below in row (i+1) and column k, which is in the inactive state, receives a 1 on the DT line from the top. It responds by changing to the vertical state and transmitting a 1 to the bottom on the DB line. This is repeated for all the modules in column k and thus all modules in column k from row (i+1) to row N enter the vertical state. The path from source i to destination k is set and transmission of data can continue. It is to be noted that a new state (state I-EN) was introduced. Consider the data lines. A 0 or a 1 on the data line indicates the presence of a data bit. However when no paths are established in the network (all modules in state I), the data lines will be 0. Thus we need to differentiate between a data bit that is 0 and the

absence of a data bit. Since only one line has been used for data transmission, we circumvent this problem by introducing a new state I-EN.

The Crossbar network permits the establishment of concurrent paths. This is manifest in the switch module by the presence of the data connecton path HV. A module in state H on receiving a 1 on DT changes to state HV and transmits a 1 to the bottom on DB. A module in state V goes through a different sequence of state changes to be in state HV. If it receives a 1 on DL it changes to a new state V-EN. If the next bit on DL is a 0 then the module changes to state HV and transmits a 1 to the right on DR. Notice that if the module in state V-EN receives a 1 on DL (request for the corner path) then the corner path cannot be set (as the vertical path is already set) and some not-acknowledge signal has to be generated. This is explained in detail in section 2.5.

2.4 Transmission of Data

The source Si transmits data on the DL line of the module in row i, column 1. This module passes the data bit to the next module in the established path. This is repeated by all the modules in the established path till the data reaches the destination.

2.5 Blocked Path

The above description had assumed that no other paths were in use at the time source Si was establishing the path to destination Dk . If another source has previously established a path to Dk then it is necessary to generate a signal indicating to the source Si that a blocked or unavailable path has been encountered. Also, it is necessary to clear the partially established path so that some other source can use it. This procedure is adopted rather than waiting to avoid any deadlock condition. Indication of a

blocked path is accomplished by generating a not-acknowledge signal which is also used to clear any partially set path.

The not-acknowledge signal can originate from two situations. Consider a module in the corner state. If it receives a 1 on DT (request for the vertical path) it has to indicate to the requesting module at the top that this path is blocked. It does that by sending a 1 on NT to the top. On the other hand, if a module in state V-EN receives a 1 on DL (request for the corner path) it indicates to the module on the left that the path is blocked by sending a 1 on NL.

Once a blocked path is encountered, the partially established path must be cleared so that some other source can use it. Thus a module whose vertical path is active (states V and HV) should respond to a 1 on the NB line by resetting the vertical path and transmitting a 1 on the NT line. The 1 on the NT line starts the same sequence of events in the module at the top. However notice the pipelined mode of data transfer. Consider a module i which has just reset the vertical path on receiving a not-acknowledge signal on NB. It outputs a 1 on NT to the module at the top. If the module at the top transmits a 1 on DT to module i on the same clock period, then the vertical path in module i will be set. However the 1 transmitted by the module at the top on the line DT was a data bit, being part of the data of the path that had already been set up. It was not intended to be a path establishment vector bit. Thus it becomes necessary to wait for one clock cycle before resetting a path when a not-acknowledge signal is received. In order to accomplish this, in addition to the seven states that have already been introduced, seven more states have to be introduced.

2.6 End of Transmission

The protocol described so far allows for the path establishment, transmission of data, detection of blocked path and clearing of blocked path. It is also necessary that once a path has been established and data transmission completed, the path be cleared to allow other sources to use it. This is done by encoding a particular bit stream to indicate the end of transmission. The destination has the responsibility for continuously decoding incoming data bits and detecting the end of transmission bit stream. When such a stream has been detected it responds by sending a not-acknowledge signal to the crosspoint switch immediately above it. The not-acknowledge signal is then transmitted back through the switches in the path clearing the path at the same time. It is to be noted that the not-acknowledge signal is used to indicate a blocked path as well as end of transmission. The source however can distinguish between these two cases because it has generated the end of transmission.

2.7 Design Methodology for the Synchronous Switch

The design methodology to be followed for the synchronous crosspoint switch consists of obtaining an incompletely specified state table. The incompletely specified state table is then used to obtain the minimized logic functions. The VLSI implementation of the logic functions is achieved by a PLA.

3.0 Statetable of the Crosspoint Switch

The statetable of the crosspoint switch describes the switch behaviour in the form of state changes and outputs events in response to the input events. Since a crosspoint switch has fourteen states, four bits S3, S2,

S1 and S0 are necessary to encode the states. The correspondence between the states of crosspoint switch and the bits S3, S2, S1 and S0 is listed in Table 1.

There are three special cases which need to be explained and these are shown in Figure 5.

Case (i): Here the state of the crosspoint switch is I-En and a 1 is received on DL as well as on DT (simultaneous requests for the corner and vertical paths). The corner path request is given precedence and the corner path is set and a not-acknowledge signal is transmitted to NT.

Case (ii): Here the state of the crosspoint switch is V-En and a 1 is received on DL as well as on NB (a request for the corner path and a blocked path signal from the bottom). The vertical path is cleared and a 1 is transmitted on NT as is required, and a blocked path is also indicated to the left by transmitting a 1 on NL.

Case (iii): When the switch is in the corner state and a 1 is received on DT as well as on NB (request for the vertical path and a blocked path signal from the bottom), the switch changes to the inactive state and a 1 is transmitted on NT.

The unminimized statetable is presented in the appendix.

4.0 Design and Implementation

The crosspoint switch module has a finite number of states and hence a finite state machine implementation is appropriate. Figure 6 shows the model on which the logic implementation is performed . The finite state machine is constructed to realize a two phase, level sensitive clocking scheme.

The design process can be broadly classified into three major

sections:

(i) Implementation of the combinational logic in the form of a PLA.

(ii) Feedback arrangement of the four state variables and clocking of the inputs and the outputs of the PLA.

(iii) I/O pads and drivers

4.1 The PLA

In the implementation of the state table logic a highly regular structure like the PLA is preferred. An NMOS geometric layout was obtained using a PLA program [ANAN81] that generated the CIF code for the PLA. The inputs to the program are number of inputs, outputs and minterms in the statetable and the state table. The PLA layout is shown in Figure 7 and has nine inputs and eight outputs. The nine inputs of the PLA are the four state variables S3, S2, S1, S0 , the four inputs to the crosspoint switch DL, DT, NB and NR and the reset input. In order to reset the crosspoint switch to the inactive state, the state variables S3, S2, Sland S0 must be reset to 0 and the four outputs of the switch DB, DR, NL and NT must be reset to 0. This is accomplished by the reset line. The eight outputs of the PLA are the four state variables S3, S2, S1, S0 and the four outputs of the crosspoint switch DB, DR, NL and NT.

4.2 State Variable Feedback

The feedback arrangement of the four state variables S3, S2, S1 and S0 from the output of the PLA to the input of the PLA is shown in Figure 8. During phase 1 of the two-phase clock, the pass transistor P1 is closed and the state variable is stored on the gate capacitance of inverter I1; during phase 2 of the clock, the pass transistor P2 is closed and the state variable is stored on the gate capacitance of the input buffer of the

PLA. The detailed layout of the feedback arrangement is shown in Figure 9.

4.3 I/O Pads and Drivers

Input and output pads with appropriate driving logic are necessary to interface the VLSI chip with the external world and are therefore fundamental components in all VLSI designs. A variety of designs for input/output pads with appropriate logic are available for general use through a component library developed at the California Institute of Technology. Two different pads were selected from this component library, one for input pads and one for output pads. It is to be noted that the output pad is driven by a depletion mode transistor (high logic level is about 3.5 V). In order to make the crosspoint switch cascadable (i.e. the output of one switch can be directly fed to the input of another), the inputs of the switch should be able to recognize a high logic level of about 3.5 V. Hence all input pads of the crosspoint switch are fed to inverters with pullup to pulldown transistor ratios of 8:1.

5.0 Estimation of Delay, Data Rate and Area

In order to determine the delay through the crosspoint switch it is necessary to estimate the delay through the PLA and the delay in the feedback path. The PLA generator software [ANAN81] generated a layout which minimized the maximum delay through the PLA. The minimization was done by searching for the particular path from the input to the output of the PLA that involved the maximum delay and then varying the size of the buffers until this path delay was minimized. Since the absolute delay in NMOS circuits is a function of the minimum feature size, we will assume that the minimum feature size is 2.5 microns. The maximum delay through the PLA was obtained as 45

nsec. The delay through the feedback path is estimated as follows: the path consists of two pass transistors and two inverters; the pass transistors are of minimum size and the pullup to pulldown ratio of the inverters are 8:1. Hence the delay through a pass transistor and inverter pair is 9τ where τ is the time required to charge the gate of minimum size transistor through a minimum size transistor. Thus the delay in the feedback path is 18τ. For a minimum feature size of 2.5 microns, τ is about 0.3 nsec. Hence the delay in the feedback path is 5.4 nsec. An analysis of the data rate of a pipelined synchronous crossbar network has been made in [FRAN81D]. The clock period T is given there as:

$$T >= dL + 2dM + dP + \delta$$

where

dL : delay through the PLA.

2dM : delay in the feedback path.

dP : delay to go off-chip.

δ : clock skew

In this analysis we assume that there is no clock skew. The delay to go off-chip dP, is estimated as 35 nsec. Hence the minimum clock period is given as

$$Tmin = 45 + 5.4 + 35 = 85.4 \text{ nsec.}$$

Hence the estimate of the maximum data rate is

$$DRmax = 1/Tmin = 11.7 \text{ MHz.}$$

The dimensions of the bounding box of the NMOS layout was approximately 500 lambda by 600 lambda where lambda is the minimum feature size (2.5 microns). The area occupied by the circuit including the pads is 1.875 sq. mm.

6.0 Summary and Conclusions

This paper presented the design and VLSI implementation of a synchronous crosspoint switch module which can be used as the basic building block for Crossbar networks. It was shown how a number of crosspoint switches can be interconnected to form a modular, Crossbar network. The functional description of the crosspoint switch module was presented and the logic design was outlined. A VLSI implementation following the design rules of Mead and Conway [MEAD80] was shown. Finally the performance measures of the circuit like data rate and area were determined.

References

SWAN77
Swan, R. J. et. al. 'Cm* A Modular Multi-Microprocessor', AFIPS Proc. Nat. Comp. Conf. (1977).

SEJN80
Sejnowski, M. C., et. al. 'An overview of the Texas Reconfigurable Computer', AFIPS Proc., Nat. Comp. Conf. (1980)

SULL77
Sullivan, H. and Bashkow, T. R. 'A Large Scale Homogeneous, Fully Distributed Parallel Machine I', Proc. 4th Ann. Symp. on Comp. Arch. (March 1977).

FRAN81A
Franklin, M. A. 'VLSI Performance Comparison of Banyan and Crossbar Switching Networks', IEEE Trans. on Comp., C-30, 4 (April 1981)

FRAN81B
Franklin, M. A., Wann, D. F., Thomas, W. J. 'Word Inconsistency in Partitioned VLSI Interconnection Networks', Center for Computer Systems Design, T. M. 81FR-B, Washington Univ., St. Louis (1981)

FRAN81C
Franklin, M. A., Wann, D. F. ' Asynchronous and Clocked Control Structures for VLSI Based Interconnection Networks'

KAHN78
Kahn, S. A., 'Design Issues of a modular Crossbar Network for a Multiprocessor' M. S. Thesis, Washington University, St. Louis, Mo (Dec 1978)

FRAN79
Franklin, M. A., Kahn, S. A. and Stucki, M. J., 'Design Issues in the Development of a Modular Multiprocessor Communication Network', Proc. 1979 Int. Symp. on Comp. Architectur (April 1979)

MEAD80
Mead, C. and Conway, L., INTRODUCTION TO VLSI SYSTEMS, Addison-Wesley Pub. Co. Reading, MA (1980)

ANAN81
Anantharaman, S., 'Generation of Conventional Programmable Logic Arrays with Minimized Delays', M. S. Thesis, Dept. of Elec. Engr., Washington University, St. Louis, MO (Aug, '82).

State	S3	S2	S1	S0
I	0	0	0	0
I-EN	0	0	0	1
V	0	1	0	0
H	0	0	1	1
HV	0	1	1	1
C	0	1	0	1
V-EN	0	0	1	0
I-NL	1	0	0	0
I-EN-NT	1	0	0	1
I-NT-NL	1	0	1	0
H-NT	1	0	1	1
V-NL	1	1	0	0
C-NT	1	1	0	1
I-NT	1	1	1	0

Table 1: Encoding of the fourteen states into the four state variables.

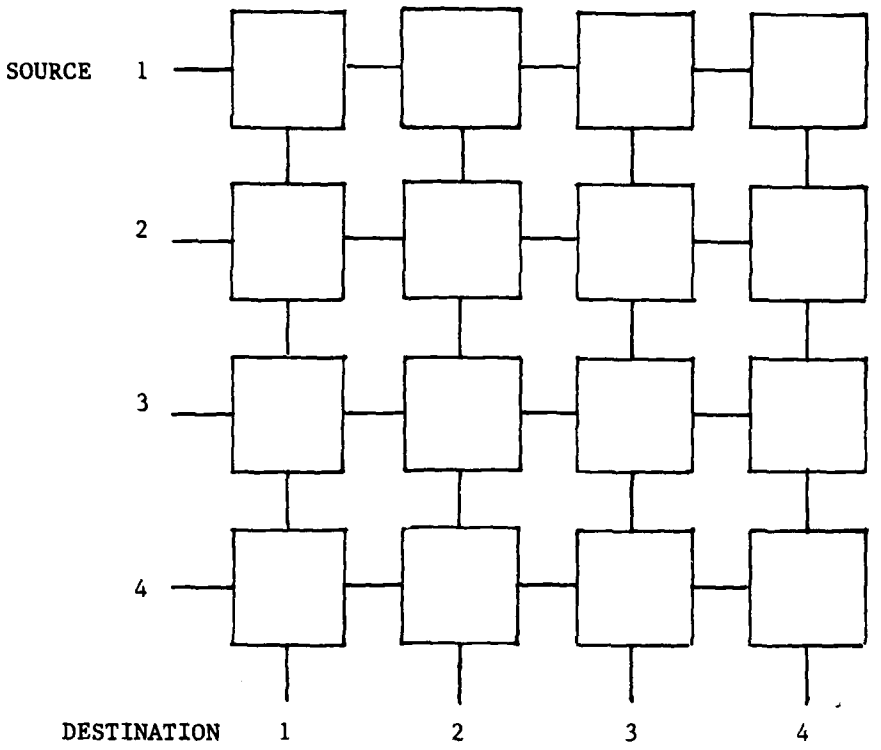


Figure 1: A 4*4 Crossbar network.

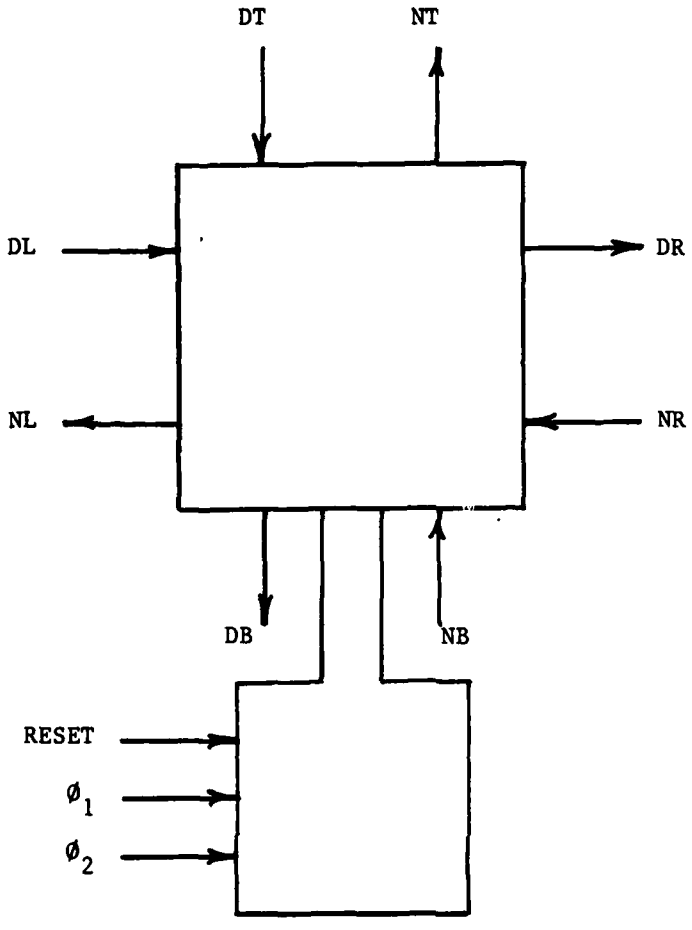


Figure 2: The synchronous switch module.

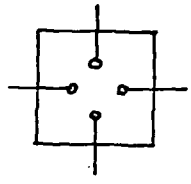
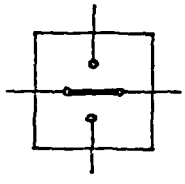
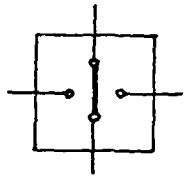
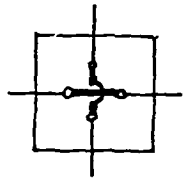
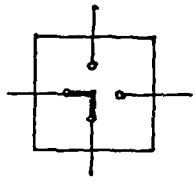
STATE	PATH CONNECTION
I	
H	
V	
HV	
C	

Figure 3: Data connection states of the synchronous switch.

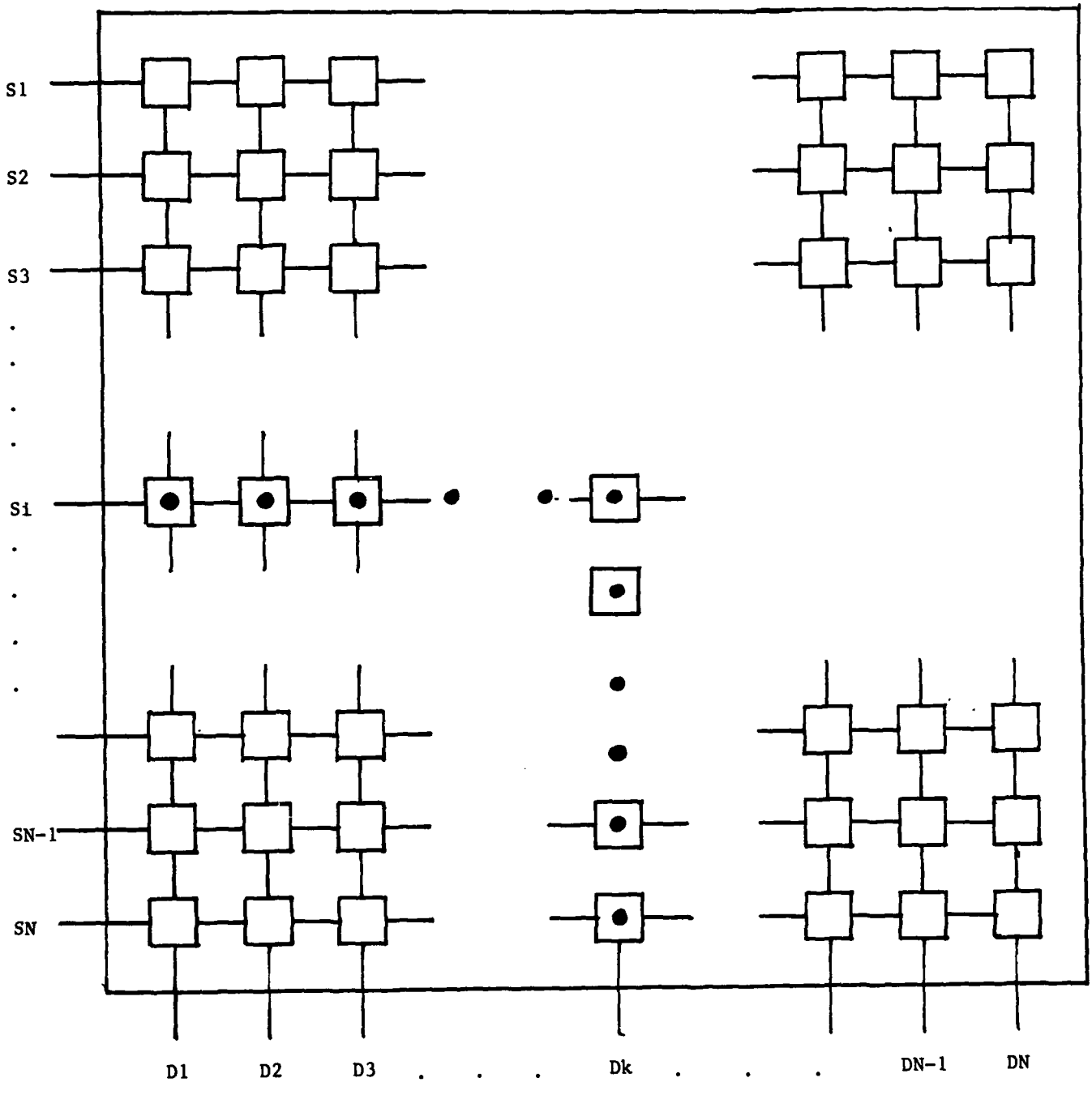


Figure 4: Path establishment from source Si to destination Dk.

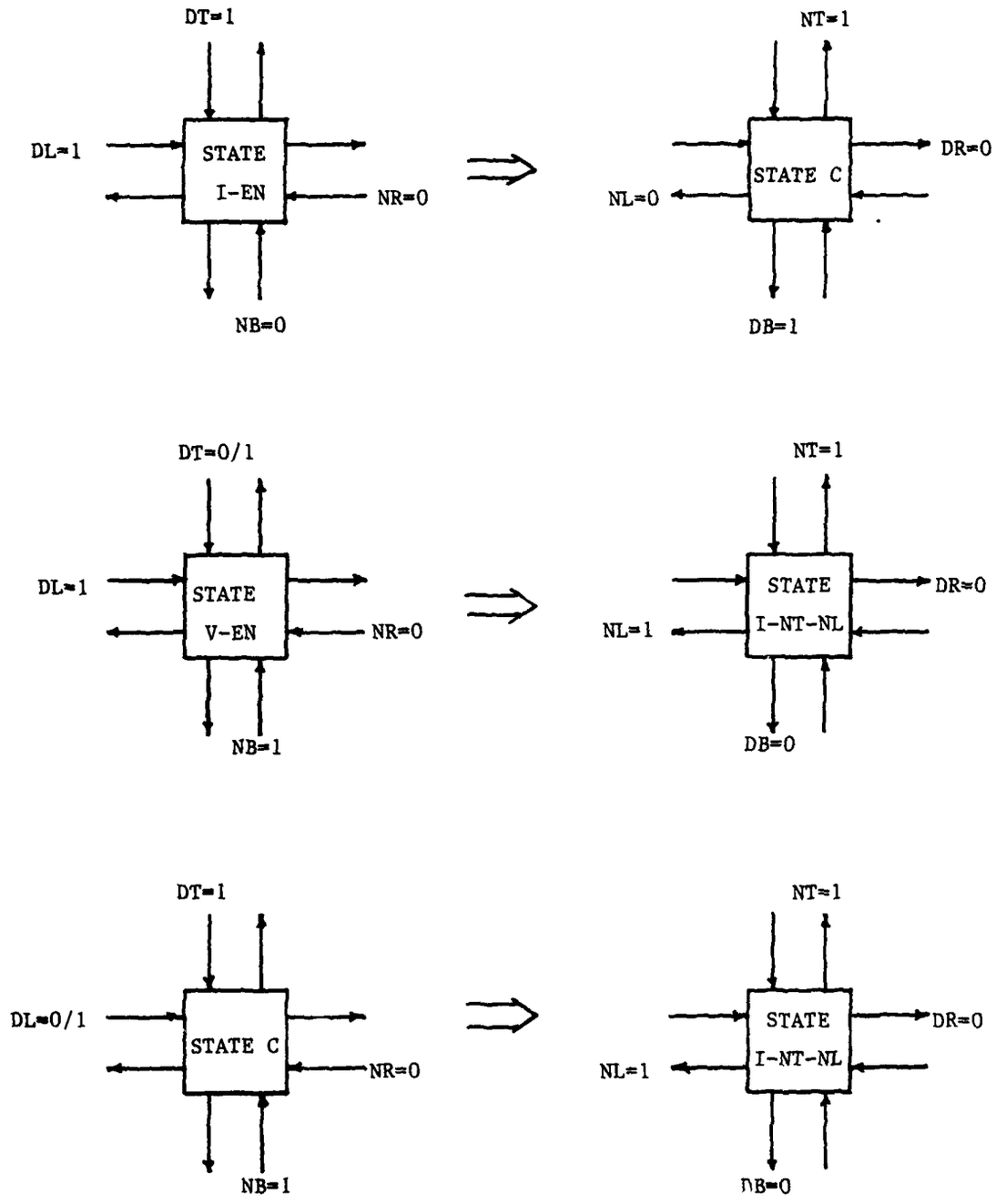


Figure 5: Conflict situations and results.

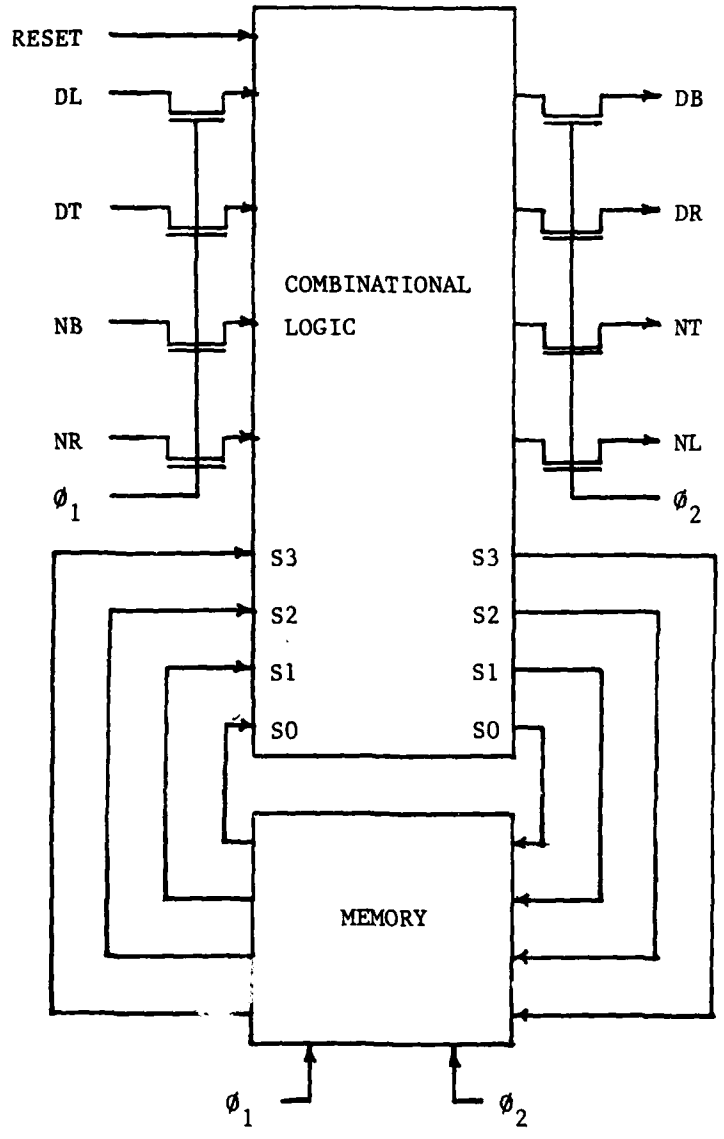


Figure 6: Synchronous switch implementation model.

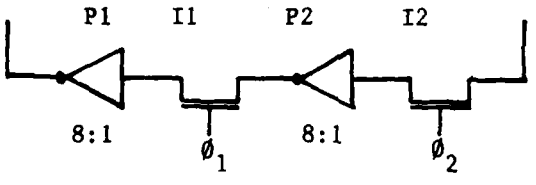


Figure 7: The memory arrangement.

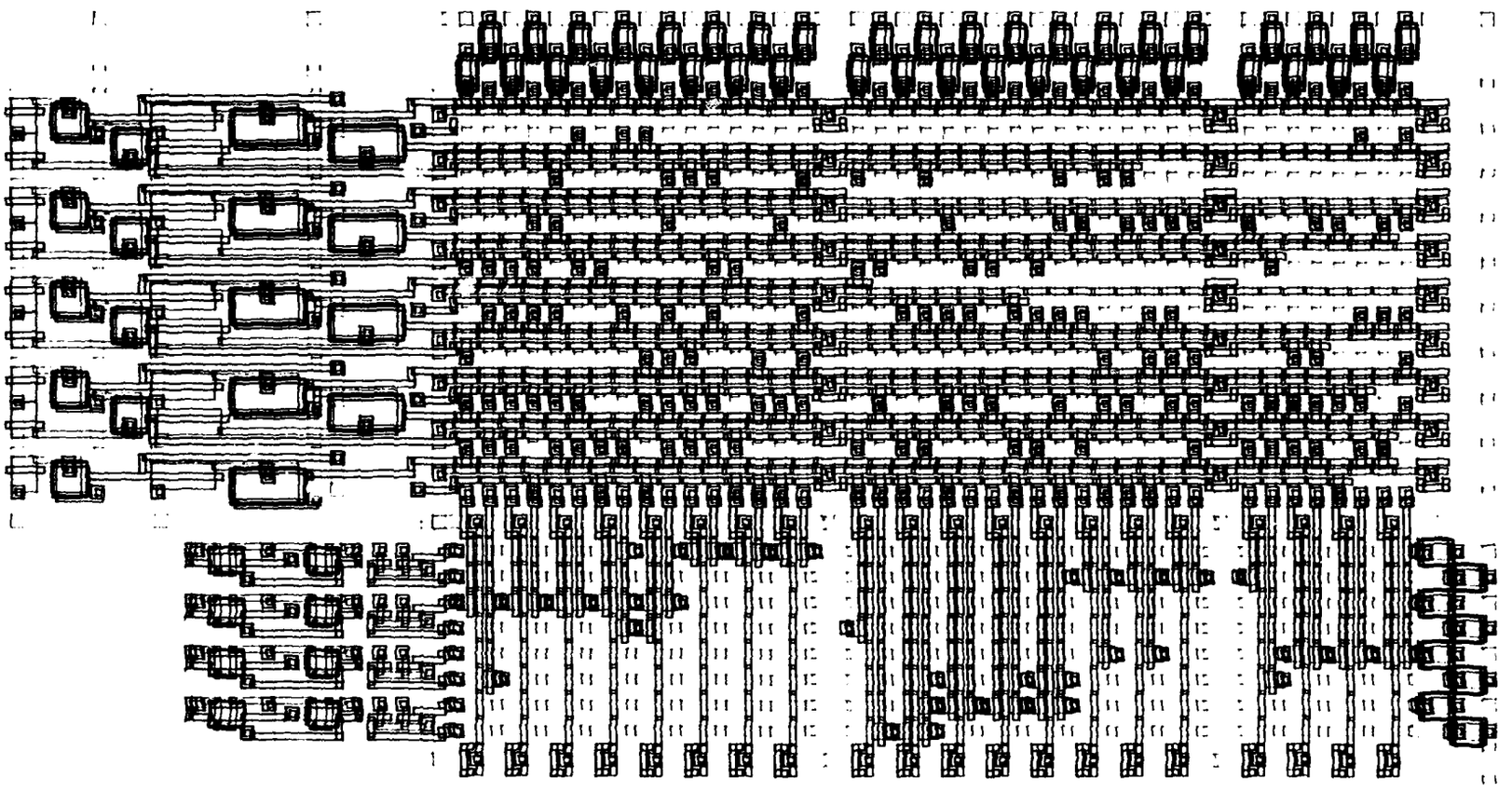
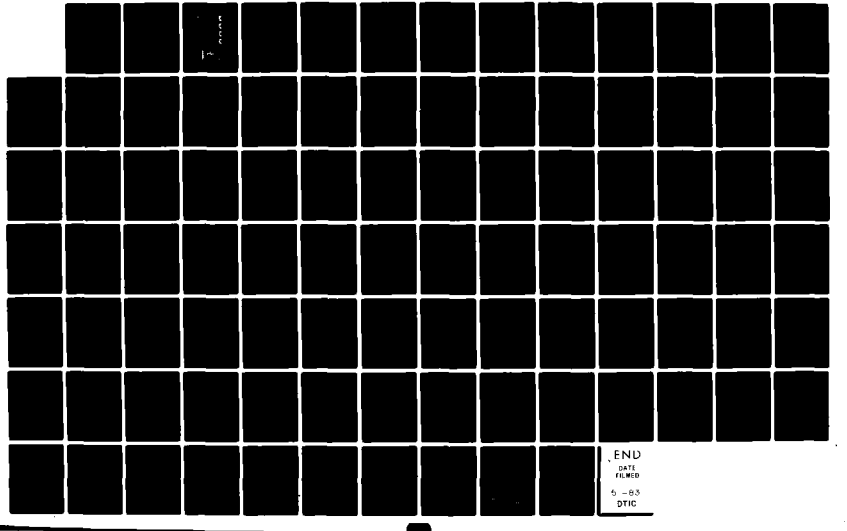
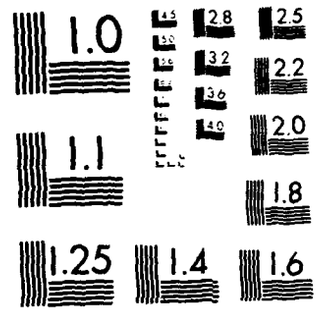


Figure 8: Layout of the PLA.

AD-A119 628 VLSI BASED MULTIPROCESSOR COMMUNICATIONS NETWORKS(U) 1/2
WASHINGTON UNIV ST LOUIS MO CENTER FOR COMPUTER SYSTEMS
DESIGN M A FRANKLIN ET AL. SEP 82 N00014-80-C-0761
UNCLASSIFIED F/G 9/2 NL





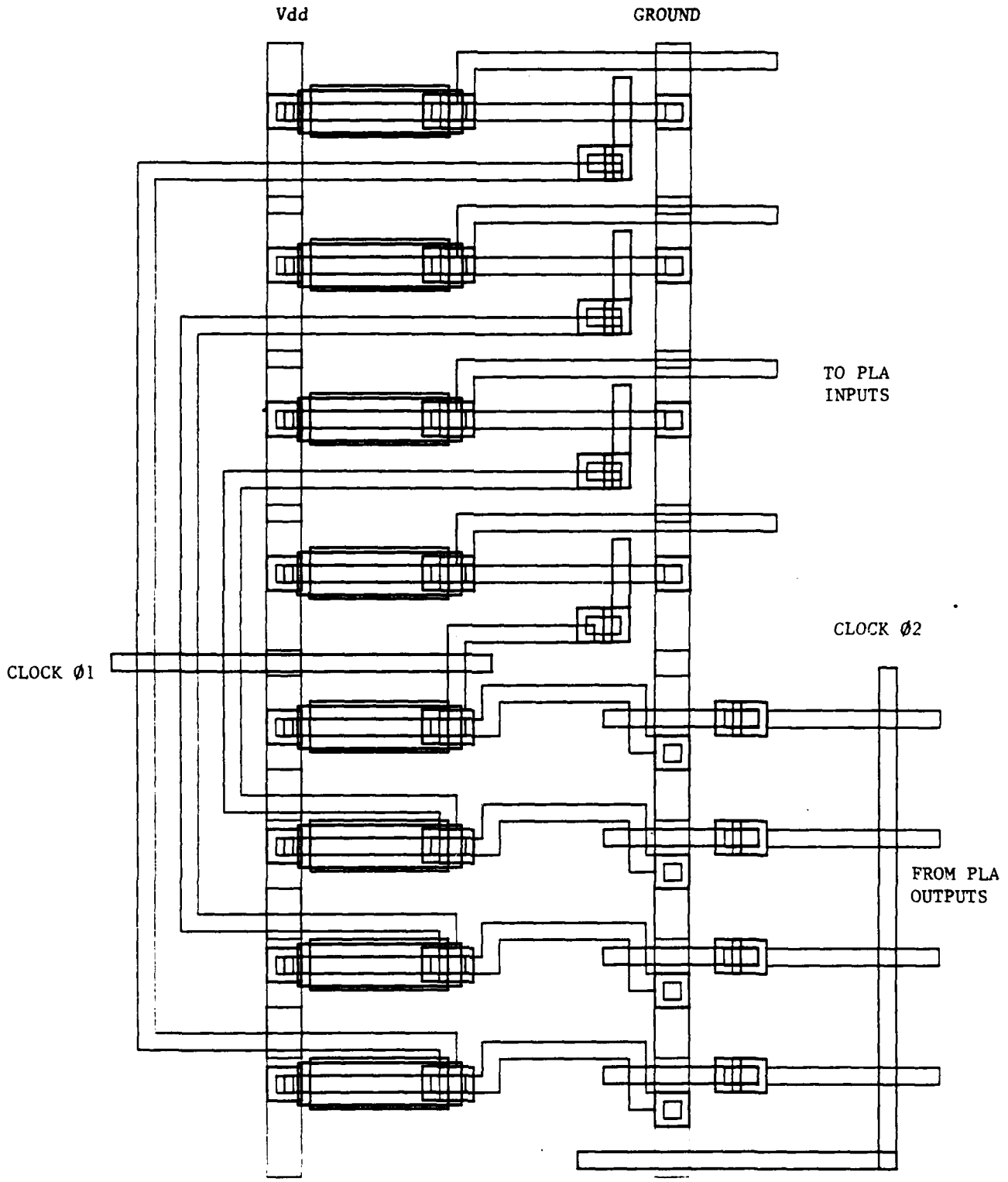


Figure 9: Layout of the synchronous switch memory.

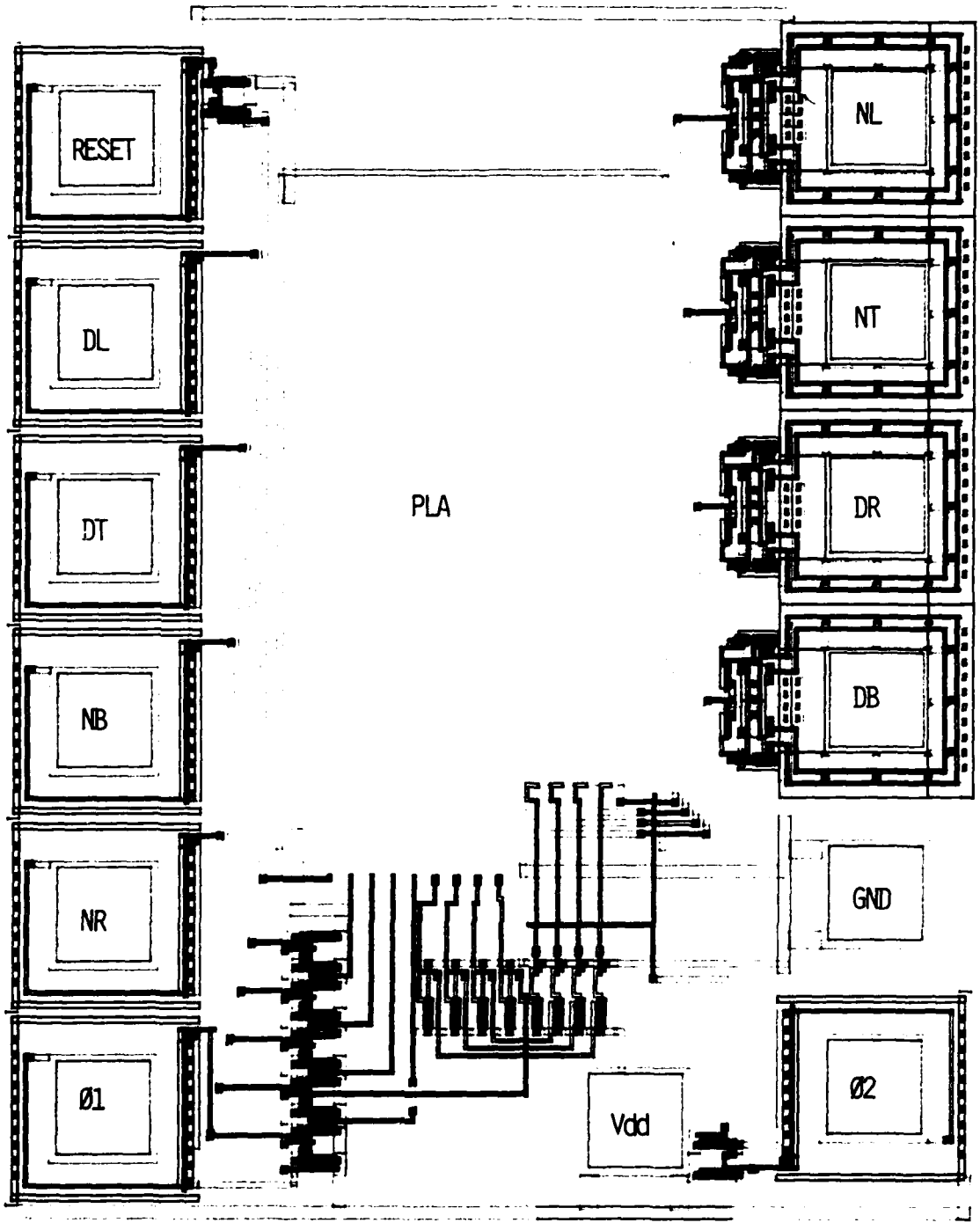


Figure 10: Layout of the synchronous crosspoint switch.

Appendix

State Table of the Synchronous Crosspoint Switch

Last State				Input				New State				Output			
S3	S2	S1	S0	DL	DT	NB	NR	S3	S2	S1	S0	DB	DR	NT	NL
0	0	0	0	0	0	X	X*	0	0	0	0	0	0	0	0
0	0	0	0	0	1	X	X	0	1	0	0	1	0	0	0
0	0	0	0	1	0	X	X	0	0	0	1	0	0	0	0
0	0	0	0	1	1	X	X	0	0	1	0	1	0	0	0
0	0	0	1	0	0	X	X	0	0	1	1	0	1	0	0
0	0	0	1	0	1	X	X	0	1	1	1	1	1	0	0
0	0	0	1	1	0	X	X	0	1	0	1	1	0	0	0
0	0	0	1	1	1	X	X	1	1	0	1	1	0	1	0
0	0	1	1	0	0	X	0	0	0	1	1	0	0	0	0
0	0	1	1	0	0	X	1	1	0	0	0	0	0	0	1
0	0	1	1	0	1	X	0	0	1	1	1	1	0	0	0
0	0	1	1	0	1	X	1	1	1	0	0	1	0	0	1
0	0	1	1	1	0	X	0	0	0	1	1	0	1	0	0
0	0	1	1	1	0	X	1	1	0	0	0	0	0	0	1
0	0	1	1	1	1	X	0	0	1	1	1	1	1	0	0
0	0	1	1	1	1	X	1	1	1	0	0	1	0	0	1
0	1	0	0	0	0	0	X	0	1	0	0	0	0	0	0
0	1	0	0	0	0	1	X	1	1	1	0	0	0	1	0
0	1	0	0	0	1	0	X	0	1	0	0	1	0	0	0

* : X denotes a don't care condition

Last State				Input				New State				Output			
S3	S2	S1	S0	DL	DT	NB	NB	S3	S2	S1	S0	DB	DR	NT	NL
0	1	0	0	0	1	1	X	1	1	1	0	0	0	1	0
0	1	0	0	1	0	0	X	0	0	1	0	0	0	0	0
0	1	0	0	1	0	1	X	1	0	0	1	0	0	1	0
0	1	0	0	1	1	0	X	0	0	1	0	1	0	0	0
0	1	0	0	1	1	1	X	1	0	0	1	0	0	1	0
0	0	1	0	0	0	0	X	0	1	1	1	0	1	0	0
0	0	1	0	0	0	1	X	1	0	1	1	0	1	1	0
0	0	1	0	0	1	0	X	0	1	1	1	1	1	0	0
0	0	1	0	0	1	1	X	1	0	1	1	0	1	1	0
0	0	1	0	1	0	0	X	1	1	0	0	0	0	0	1
0	0	1	0	1	0	1	X	1	0	1	0	0	0	1	1
0	0	1	0	1	1	1	X	1	0	1	0	0	0	1	1
0	1	0	1	0	0	0	X	0	1	0	1	0	0	0	0
0	1	0	1	0	0	1	X	1	0	0	0	0	0	0	1
0	1	0	1	0	1	0	X	1	1	0	1	0	0	1	0
0	1	0	1	0	1	1	X	1	0	1	0	0	0	1	1
0	1	0	1	1	0	0	X	0	1	0	1	1	0	0	0
0	1	0	1	1	0	1	X	1	0	0	0	0	0	0	1
0	1	0	1	1	1	0	X	1	1	0	1	1	0	1	0
0	1	0	1	1	1	1	X	1	0	1	0	0	0	1	1
0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0
0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	1
0	1	1	1	0	0	1	0	1	0	1	1	0	0	1	0
0	1	1	1	0	0	1	1	1	0	1	0	0	0	1	1
0	1	1	1	0	1	0	0	0	1	1	1	1	0	0	0

Last State				Input				New State				Output			
S3	S2	S1	S0	DL	DT	NB	NB	S3	S2	S1	S0	DB	DR	NT	NL
0	1	1	1	0	1	0	1	1	1	0	0	1	0	0	1
0	1	1	1	0	1	1	0	1	0	1	1	0	0	1	0
0	1	1	1	0	1	1	1	1	0	1	0	0	0	1	1
0	1	1	1	1	0	0	0	0	1	1	1	0	1	0	0
0	1	1	1	1	0	0	1	1	1	0	0	0	0	0	'
0	1	1	1	1	0	1	0	1	0	1	1	0	1	1	0
0	1	1	1	1	0	1	1	1	0	1	0	0	0	1	1
0	1	1	1	1	1	0	0	0	1	1	1	1	1	0	0
0	1	1	1	1	1	0	1	1	1	0	0	1	0	0	1
0	1	1	1	1	1	1	0	1	0	1	1	0	1	1	0
0	1	1	1	1	1	1	1	1	0	1	0	0	0	1	1
1	0	0	0	X	0	X	X	0	0	0	0	0	0	0	0
1	0	0	0	X	1	X	X	0	1	0	0	1	0	0	0
1	0	0	1	0	X	X	X	0	0	1	1	0	1	0	0
1	0	0	1	1	X	X	X	0	1	0	1	1	0	0	0
1	0	1	0	X	X	X	X	0	0	0	0	0	0	0	0
1	0	1	1	0	X	X	0	0	0	1	1	0	0	0	0
1	0	1	1	0	X	X	1	1	0	0	0	0	0	0	1
1	0	1	1	1	X	X	0	0	0	1	1	0	1	0	0
1	0	1	1	1	X	X	1	1	0	0	0	0	0	0	1
1	1	0	0	X	0	0	X	0	1	0	0	0	0	0	0
1	1	0	0	X	0	1	X	1	1	1	0	0	0	1	0
1	1	0	0	X	1	0	X	0	1	0	0	1	0	0	0
1	1	0	0	X	1	1	X	1	1	1	0	0	0	1	0
1	1	0	1	0	X	0	X	0	1	0	1	0	0	0	0

Last State				Input				New State				Output			
S3	S2	S1	S0	DL	DT	NB	NB	S3	S2	S1	S0	DB	DR	NT	NL
1	1	0	1	0	X	1	X	1	0	0	0	0	0	0	1
1	1	0	1	1	X	0	X	0	1	0	1	1	0	0	0
1	1	0	1	1	X	1	X	1	0	0	0	0	0	0	1
1	1	1	0	0	X	X	X	0	0	0	0	0	0	0	0
1	1	1	0	1	X	X	X	0	0	0	1	0	0	0	0

DESIGN OF AN ASYNCHRONOUS CROSSPOINT SWITCH

Sanjay Dhar
Department of Electrical Engineering,
Washington University,
St. Louis, MO 63130.

1.0 Introduction

The performance of large, closely coupled multiple processor systems depend to a great extent on the performance of the network through which interprocessor communication takes place. The importance of the communication network in such systems has resulted in a number of proposals [SWAN77, SEJN80, SULL77] for the design of the communication network. Interest has also been focussed on the Crossbar network, principally because of its modular and planar construction which leads to easy VLSI implementation. A basic building block of the Crossbar network, the crosspoint switch, has been suggested in PADM81. It is possible to implement a full Crossbar network by interconnection of these crosspoint switches.

An important issue in the design of such communication networks relates to the problem of control of data movement, that is, as data moves along the network what methods are adopted for synchronization of data. This problem has been dealt with in detail in FRAN81D. The asynchronous and synchronous (or clocked) are the two principal methods used for control of data movement. Obviously each scheme has its own advantages and disadvantages. This paper considers the design of an asynchronous crosspoint switch which can be interconnected to form a full Crossbar network.

2.0 Asynchronous Switch Module Characteristics

The asynchronous crosspoint switch proposed has provisions for path establishment, transfer of data from source to destination, detection of

a blocked path and path clearing associated with end of transmission. The data transfer through the network is pipelined.

Figure 1 illustrates the asynchronous crosspoint switch module and the signals associated with it. Each side of the switch module is identified by the first letter of the input or output variable, that is, L=left, R=right, T=top and B=bottom . The asynchronous module has four connections per side for a total of 16 connections; of these, 8 are input connections and 8 are output connections. Each side has two request connections; the second letter of a variable identifies its function. An "O" or a "Z" means that it is used for data transmission ("O" for One, "Z" for Zero). An "A" or an "N" means that it is used for the acknowledge or not-acknowledge signal to indicate the acceptance of a request; this is necessary to achieve self-timed operation. The not-acknowledge connection on each side is used to indicate a blocked path. Two separate data lines (one for tranmission of 1 and another for transmission of 0) and two separate acknowledge lines (acknowledge and not-acknowledge) are necessary because of the asynchronous or self-timed nature of the crosspoint switch.

2.1 Path Establishment

In order to establish a path from source Si to destination Dk a path establishment vector of length k is sent by the source. The path establishment vector consists of (k-1) 0's followed by a 1, that is, if a path to destination 5 is to be set, the path establishment vector would be 00001. The way in which the path is established is described in detail in FRANSIB and will not be discussed further.

2.2 Transmission of Data

The switch module transmits data bits on the request lines. Each bit is acknowledged or not-acknowledged by the switch module it is communicating with. The next bit of data is transmitted only after receipt of an acknowledge signal.

2.3 Blocked Path

A blocked path occurs when a source tries to establish a path to a destination that is already in communication with another source. The blocked path condition is indicated by a signal on the not-acknowledge line. For instance, a switch module in the vertical connection path must indicate a blocked path in response to a request for the corner path. A switch module that receives a not-acknowledge signal responds to any further requests by resetting that path and sending a not-acknowledge signal to the requesting source. For instance, when a switch module in the horizontal connection path receives a signal on line RN, it responds to any signal on lines LO or LZ by generating a signal on line LN and also resetting the horizontal path. In this way the partially established path is cleared for use by another source.

2.4 End of Transmission

It is necessary to have the ability to clear a path that was established at the end of transmission. This is done by reserving a special bit stream to indicate end of transmission; the destination continuously decodes the incoming bit stream and on detecting the end of transmission bit stream sends a not-acknowledge signal. This not-acknowledge signal has to be transmitted back along the path. As mentioned earlier, a switch module sends a not-acknowledge signal only in response to a request signal. Hence the

source must continue to send in dummy request signals after sending the end of transmission bit stream to clear the path. It stops sending dummy request signals only after receiving a not-acknowledge signal.

3.0 Formal Specification of the Asynchronous Switch

A speed independent system, like the asynchronous crosspoint switch, can be partitioned into a module and its environment with connections from the module to the environment and connections from the environment to the module. A specification of the interaction between the module and the environment in terms of the sequence of changes of the input and output variables at the module-environment interface would then be a sufficient specification of the system.

Because the crosspoint switch supports concurrent data and control flow, it is not possible to specify the module using a state table. Instead a special class of Petri net, called the Interface net, will be used for the specification of this interaction. This method has been developed at the Washington University Computer Systems Laboratory by Charles Molnar and T. P Fang [MOLN81].

3.1 Asynchronous Switch Design Methodology

The methodology that will be followed in designing the asynchronous crosspoint switch consists of:

(1) Specification of the crosspoint switch by constructing an interface net.

(2) Generation of the interface state graph (ISG) from the interface net.

(3) Removal of all cases that have two or more different conditions of

the output variables for the same condition of the input variables either by adding sequence constraints or by introducing internal variables.

(4) Generation of an incompletely specified truthtable for all output variables from the ISG.

(5) Minimization of the logic functions obtained.

3.2 The Interface Net

The first step in constructing the interface net of the switch is to recognize some basic characteristics of the switch:

The switch has three data connection paths, the vertical path, the horizontal path and the corner path. The corner path and the vertical and horizontal paths are mutually exclusive, that is, if the switch is in the corner path then it cannot be in the vertical or the horizontal paths and vice versa. Also, the vertical and the horizontal paths can be used concurrently and the establishment or release of the vertical (horizontal) path is independent of the establishment or release of the horizontal (vertical) path.

Having recognized these properties, we arrive at the initial structure shown in Figure 2. The boxes marked C, H and V represent interface nets that represent the sequence of events that happen once the corner, horizontal and vertical paths respectively have been established. LO, LZ and TO are input signals to the switch as shown in Figure 1. The initial markings are shown and represent the condition when none of the data connection paths have been established.

3.3 Explanation of the Notations to be Used

The interface net consists of a set of nodes interconnected by arcs through a set of transitions; the interface variable (input or

output) that changes due to the transition is listed beside the transition. At this stage we do not specify the type of signalling to be used (for example, two stroke (NRZ) or four stroke (RZ)). Each transition simply signifies that the particular event has occurred.

3.4 Interface Net for the Corner Path

In a speed independent system, communication between modules typically occurs by means of handshake, that is, the sender sends the data and the receiver acknowledges the receipt of the data. It is important to understand the types of handshake used in the crosspoint switch. One type of handshake is between a request and the corresponding acknowledge (or not-acknowledge) like that between LO or LZ and LA or LN. A second type of handshake exists which is not obvious. Consider the case when the corner path has been established. A request comes from the left on LO or LZ; this initiates an output on BO or BZ at the bottom. The request LO or LZ is acknowledged on LA and a new request comes in. This request cannot be acknowledged until the sequence of events at the bottom started by BO or BZ has gone to completion. Specifically, the request on BO or BZ must receive its acknowledge or not-acknowledge on BA or BN before the new request on LO or LZ can be serviced.

The corner interface net can now be constructed as shown in Figure 3. The net is found to consist of two identical stages, one stage starting at nodes E and F and the other stage starting at nodes O and P. This structure can be directly attributed to the two types of handshakes just mentioned. The position of the transition TO will play an important part and will be discussed in more detail later.

3.5 Interface Nets for the Vertical and Corner Paths

Constructing the interface nets for these two paths is now quite simple. The sequence of events that take place in the vertical path is nearly identical to that of the corner path. Notice that when a request on TO is received while the corner path is set, a not-acknowledge on TN is given. On the other hand, if a request on LO is received while the vertical path is set, the switch must determine whether this request is for a corner path or part of the communication taking place because the horizontal path is also set. In this situation the switch must determine whether the horizontal path is set before it responds.

This difference is easily identified in the interface net. The transition LO is conditioned by node A. A token at node A means the horizontal path is not set and hence a not-acknowledge is given on LN.

The interface net for the horizontal path is very similar; the difference lies in the manner of path setting. When the horizontal path is set by a request on LZ, the request is acknowledged on LA but no output is generated on the right. This gives rise to the initial part of the net between nodes A and BB and CC. After nodes BB and CC, this interface net is identical to the other two interface nets.

3.6 Need for an Arbiter

At this stage we look at some of the assumptions made while constructing the switch interface net. Referring back to Figure 1, the first assumption is that LO, LZ and TO cannot occur simultaneously. While this is true for LO and LZ, it is not true when TO is also considered. Next consider the nodes F and P in the corner net of Figure 3. By ordering the transition TO at this position we are assuming that LO, LZ and TO do not occur

simultaneously. Also, we are assuming that LO, LZ and TO can only occur after LA. While this is true for LO and LZ, it is not necessarily true for TO. A similar set of assumptions have been made while constructing the vertical and the horizontal interface nets.

Since this switch interface net assumes that certain restrictions apply to the sequence in which events can happen, there must be some agency which maintains these restrictions. The inclusion of an arbiter between the environment and the switch therefore is necessary. The arbiter maintains the assumed order in the sequence of events.

There is one additional basic question that needs to be answered: Is the arbiter necessary or has it been made necessary because of the way the switch interface net was constructed? To answer this question we must recognize that the left and the top side are not synchronized, that is, requests at the left and the top side can arrive at any time independent of each other. Now consider the case where no paths are set and requests arrive simultaneously on LO and TO (simultaneous requests for the corner and the vertical paths). The particular course of action is not unique here and the situation necessitates a decision. In the construction of the switch interface net we have removed all situations involving decisions. The arbiter therefore is inherently necessary for this problem. It can be implemented explicitly as has been done here or implicitly by incorporating it within the switch itself.

3.7 Interface Net for the Arbiter

The input and output signals of the arbiter and the manner in which it is interfaced with the switch and its environment is shown in Figure 6. The requests from the environment now arrive to the arbiter; the arbiter

imparts the proper sequence to the requests before sending them to the switch.

As mentioned earlier, the requests to the switch should be such that the following order is maintained:

LO can arrive only after TA or TN and before TO or TZ.

TO can arrive only after LA or LN and before LO or LZ.

LO, LZ and TO cannot arrive simultaneously.

The arbiter interface net can now be constructed. The construction as shown in Figure 6 follows this rule: A new request is sent to the switch only if the transactions of the previous request have been completed.

The interface net has three initial tokens at nodes AA, BA and BZ. Note that requests ATO or ATZ and ALO or ALZ can arrive at the arbiter simultaneously, but only one of TO, TZ, LO and LZ will go through at a time as the arbiter outputs TO, TZ, LO and LZ are conditioned by the same node BZ.

3.8 Liveness and Safety of the Interface Nets

The interface nets constructed so far were tested by the program PETRI available on the DEC-20 system and were found to be live and safe.

4.0 Circuit Design

Once the problem has been specified by means of the interface net, we proceed to the next step which involves the generation of the interface state graph (ISG). The ISG is a representation showing all possible conditions of the interface variables and the corresponding successor condition of the interface variables.

The interface net that was constructed for the switch was extremely complex and the PETRI program could not be run with the whole interface net as the input because the maximum state space obtainable with the

DEC-20 system was exceeded. Therefore the corner net was tested separately from the vertical and the horizontal nets because the corner net does not interact with the other two nets. Two stroke signalling was chosen in which the occurrence of a signal is due to a transition from 0 to 1 or from 1 to 0. Using this scheme results in about 800 firings of transitions for the corner net and about the same number of firings for the vertical and the horizontal nets. The total number of firings for the whole net would then be about 800*800=640,000.

In an attempt to simplify the specification, the four stroke signalling scheme was adopted. In this signalling scheme the interface variables always return to zero before the next transaction is started.

4.1 Arbiter Circuit Realization

Consider the arbiter interface net. The conversion of the initial arbiter interface net to an interface net incorporating the four stroke signalling scheme is almost automatic and the resultant interface net is shown in Figure 7.

This interface net was checked by the program PETRI and all possible node markings together with their successor node markings were generated. In going from this stage to the ISG one important difference between the output of program PETRI and the ISG should be observed. The output of PETRI is a list of a set of node markings and its successor where there is change in only one interface variable. The ISG also provides this information except that it does not have any information about the internal nodes of the interface net; it only lists the condition of the interface variables and the successor condition of the interface variables. Consider a case where for two different node markings the condition of the interface

variables are the same and in which the successor conditions of the interface variables are different. When this is translated to a ISG there will be two different successor conditions of the interface variables for the same parent condition. Obviously such situations represent ambiguity and there are ways of getting around them. Although such cases occur in the general design process, they are not present in the arbiter circuit design.

Next we examine the output of PETRI and determine whether any parent node marking has more than one son node markings where the difference between the parent and the sons is due to a change in an output variable. If such a situation is found, we then determine whether this represents concurrency or conflict. By concurrency we mean that more than one output variable change can occur simultaneously. Consider a specific case; assume that the present marking is (Note: 'C' at the end of an interface variable indicates the complement of the variable):

(ALZ, ALOC, ATZC, ATOC, LNC, LAC, TNC, TAC, LOC, LZC, TOC, TZC, ALNC, ALAC, ATN, ATA, AC, BZ, BJ)

Then the two successor markings are:

(ALZ, ALOC, ATZC, ATOC, LNC, LAC, TNC, TAC, LOC, LZ, TOC, TZC, ALNC, ALAC, ATN, ATA, AD, BJ)

(ALZ, ALOC, ATZC, ATOC, LNC, LAC, TNC, TAC, LOC, LZC, TOC, TZC, ALNC, ALAC, ATNC, ATA, AC, BZ, BA)

In this case both outputs can change concurrently. The successor node marking therefore becomes:

(ALZ, ALOC, ATZC, ATOC, LNC, LAC, TNC, TAC, LOC, LZ, TOC, TZC, ALNC, ALAC, ATNC, ATA, AD, BA)

All such cases of concurrency are handled in the same manner. Now consider the situation where a conflict is present. Assume that the present marking is:

(ALZ, ALOC, ATZ, ATOC, LNC, LAC, TNC, TAC, LOC, LZC, TOC, TZC, ALNC, ALAC, ATNC, ATAC, AC, BZ, BC)

Then the two successor node markings are:

(ALZ, ALOC, ATZ, ATOC, LNC, LAC, TNC, TAC, LOC, LZ, TOC, TZC, ALNC, ALAC, ATNC, ATAC, AD, BC)

(ALZ, ALOC, ATZ, ATOC, LNC, LAC, TNC, TAC, LOC, LZC, TOC, TZ, ALNC, ALAC, ATNC, ATAC, AC, BD)

In this situation only one of the output variables LZ or TZ can occur. This is

a conflict situation.

The problem arising due to this conflict situation is not solved by choosing one of the two successor markings as the only successor marking. Because of the delay involved in the combinational logic this might result in the occurrence of a pulse of unknown duration in one of the output lines LZ or TZ (depending on which successor node marking was chosen) Obviously this must be avoided as this might lead to circuit malfunction.

The way to solve this problem is to allow both the output variables to change. The two output variables go to a two input, two output circuit called a synchronizer. The synchronizer allows only one of its outputs to be high or active at any time. Depending on the condition of its inputs, there is a possibility that the synchronizer will go to a metastable state where none of its outputs are defined; the time it stays in the metastable state is unknown and could be of considerable duration. However when it comes out of the metastable state only one of its outputs will be active. The synchronizer output then goes to a threshold detector where the threshold voltage is high. This can be achieved by two inverters whose inverter threshold voltages are higher than Vdd/2. Thus only when the synchronizer is well out of its metastable state will one of its outputs be recognized as being high or active.

The synchronizer is modelled by means of the interface net, the ISG is obtained, and from this the synchronizer circuit is determined. The synchronizer interface net, the corresponding ISG and the final circuit realization are shown in Figure 8.

Notice that the arbiter has 16 inputs and 8 outputs. Examination of the 4 outputs ALA, ALN, ATA and ATN shows that they are the same as the 4 inputs LA, LN, TA and TN respectively; that is, whwnever LA ia 1 ALA is 1 and whenever LA is 0 so is ALA. The same is true for the other three

pairs of variables. This was also verified from the PETRI generated output. Hence the 4 output variables can be removed from the input; the arbiter then becomes a 12 input, 4 output logic minimization problem. A logic minimization algorithm was used to obtain the minimized functions.

4.2 The Switch Circuit Realization

The switch interface net is too large to directly generate the ISG. One solution is to partition the net into several smaller interface nets. In order to do this, use was made of the fact that the switch functions in different ways depending on the particular path that is set. Also note that the switch interface net is functionally partitioned into the corner, vertical and the horizontal nets. Since the interaction among these three nets is not large, we partition the switch interface net into the corner, vertical and horizontal interface nets where each is independent of the others. The communication among these three nets is the responsibility of a fourth net, which will be called the supervisor interface net. Introduction of the supervisor introduces new interface variables which will modify the corner, vertical and the horizontal nets. We begin by constructing the supervisor interface net.

4.3 The Supervisor Interface Net

The interface between the supervisor net and the other three nets must be arranged so that the supervisor is able to indicate, without ambiguity, which of the three nets, the corner, vertical and the horizontal, should be active. This indication should be present at all times because the three nets have common interface variables. Thus for example, if the corner path is set, then any change in the input variables LO or LZ is relevant to the corner net and not to the horizontal net.

Two interface nets for the supervisor can be formed as shown in Figures 9 and 10. The interface variables COR, VER and HOR are output variables that go to the corner, vertical and the horizontal nets respectively indicating that the particular path has been set while CCL, VCL and HCL are input variables that arrive from the corner, vertical and the horizontal nets respectively that indicate that the particular path has been cleared. Finally CCLA, VCLA and HCLA are acknowledge signals from the supervisor.

There are two constraints that must be satisfied. First, the interface net complexity should not exceed the maximum program PETRI can handle. Second, the number of interface variables should not be greater than 12 so that a logic minimization program available here can be used.

The first interface net (Figure 9) has 13 interface variables. Although the determination of the ISG for this net is not difficult, the determination of the ISG's of the corner, vertical and the horizontal nets present a problem. Observe that this supervisor net does not provide a constant signal to indicate which of the three nets should be active at any time. Although this problem can be solved by having internal variables in each of the three nets to indicate whether they should be active or not, this increases the number of variables in each of the three nets (exceeding the limit of 12).

The second net shown in Figure 10 avoids this problem since it has only 10 interface variables. Note the transitions marked COR and $\overline{\text{COR}}$ after nodes V and X. There is no change of any interface variable associated with these transitions as the variable has already changed before.

From this net and program PETRI the logic functions are obtained in a manner similar to the arbiter circuit realization. No conflict situations were present and all concurrent situations were handled in the same

manner.

There is one important comment on the way in which the supervisor interacts with the other three nets. In the corner, vertical and horizontal nets the final $\overline{LN}$ transition takes place before the corresponding $\overline{CCL}$, $\overline{VCL}$ or $\overline{HCL}$ transitions. The $\overline{LN}$ transition enables the environment to send in a new request which might arrive at the supervisor before the $\overline{CCL}$, $\overline{VCL}$ or $\overline{HCL}$ transitions arrive. In such a situation the supervisor must not generate the COR, VER or HOR signal until it receives the $\overline{CCL}$, $\overline{VCL}$ or $\overline{HCL}$ transition. This constraint has been added to the COR, VER and HOR functions. Interface of the various switch building blocks is shown in Figure 11.

4.4 The Corner Interface Net

The corner interface net was reconstructed incorporating the four stroke signalling scheme and is shown in Figure 12. Some outputs have been scheduled to occur after other outputs. This does not violate the interface net construction rules. This was done to avoid ambiguous situations where for the same condition of the interface state variables two different output conditions appeared to be necessary. There is a considerable reduction in the complexity of the net when compared with the net presented in Figure 4. The total number of firings have been reduced to 59.

4.5 The Vertical and the Horizonatal Interface Nets

The vertical and the horizontal interface nets are very similar to that of the corner and are shown in Figures 13 and 14 respectively. The only major difference between these two interface nets and the corner interface net is that in the horizontal net the same input condition of the interface variables necessitates two different output conditions. Situations of this nature were encountered in the corner net but by putting restrictions

on the sequence in which the outputs could change, we were able to circumvent those situations. Here that is not possible because in the horizontal path, when the first LZ is received there is no output to the right on RZ; for all subsequent LZ's an output on RZ takes place. In order to differentiate between the first LZ and any other LZ we had to create an internal variable INT. This was the only major difference .

4.6 Logic Minimization

The functions were minimized with the help of a logic minimization program. A 12 variable function (like the arbiter and the corner net) took between 2 and 3 hours of TI-980 CPU time. The minimized functions are:

Arbiter

ALA = LA

ALN = LN

ATA = TA

ATN = TN

$$TO = ATO.\overline{LA}.\overline{LN}$$

$$TZ = ATZ.\overline{LA}.\overline{LN}$$

$$LO = ALO.\overline{TA}.\overline{TN}$$

$$LZ = ALZ.\overline{TA}.\overline{TN}$$

Supervisor

$$COR = LO.\overline{CCL}.\overline{HCL}.\overline{VCL}.\overline{VER}.\overline{HOR} + \overline{CCL}.COR$$

$$HOR = LZ.\ HCL.\ CCL.\ COR + HCL.\ HOR$$

$$VER = TO.\ VCL.\ CCL.\ COR + VCL.\ VER$$

$$LN = LO.\ VER$$

Corner

$$BZ = COR.\ (\ LZ.\ \overline{BO} + BZ.\ \overline{BA}\ )$$

$$BO = COR.\ (\ LO.\ \overline{BZ} + BO.\ \overline{BA}\ )$$

$$LA = COR.\ (\ LZ.\ LA + LO.\ LA + \overline{BN}.\ BZ + \overline{BN}.\ BO\ )$$

$$LN = COR.\ (\ LZ.\ \overline{LA}.\ BN + LO.\ \overline{LA}.\ BN + BN.\ LN\ ) + CCL.\ BN.\ LN$$

$$CCL = \overline{LZ}.\ \overline{LO}.\ LN.\ (\ COR + CCL\ )$$

$$TN = COR.\ TO$$

Horizontal

$$RZ = HOR.\ (\ LZ.\ \overline{RO}.\ INT + RZ.\ \overline{RA}\ )$$

$$RO = HOR.\ (\ LO.\ \overline{RZ} + RO.\ \overline{RA}\ )$$

$$LA = HOR.\ (\ LZ.\ LA + LO.\ LA + \overline{INT} + \overline{RN}.\ RZ + \overline{RN}.\ RO\ )$$

$$LN = HOR.\ (\ LZ.\ \overline{LA}.\ RN + LO.\ \overline{LA}.\ RN + RN.\ LN\ ) + HCL.\ RN.\ LN$$

$$HCL = \overline{LZ}.\ \overline{LO}.\ LN.\ (\ HOR + HCL\ )$$

$$INT = HOR.\ (\ \overline{LZ} + INT\ )$$

Vertical

$$BZ = VER.\ (\ TZ.\ \overline{BO} + BZ.\ \overline{BA}\ )$$

$$BO = VER.\ (\ TO.\ \overline{BZ} + \overline{BO}.\ BA\ )$$

$$TA = VER.\ (\ TZ.\ TA + TO.\ TA + \overline{BN}.\ BZ + \overline{BN}.\ BO\ )$$

$$TN = VER.\ (\ TZ.\ \overline{TA}.\ BN + TO.\ \overline{TA}.\ BN + BN.\ TN\ ) + VCL.\ BN.\ TN$$

$$VCL = \overline{TZ}.\ \overline{TO}.\ TN.\ (\ VER + VCL\ )$$

A summary of this design effort is shown in Figure 15 where interconnection of the arbiter, supervisor, corner, vertical and horizontal circuits are shown.

5.0 Summary and Conclusion

This paper presented the results of the design of an asynchronous crosspoint switch. The switch designed is the basic building block of a Crossbar network. A number of such switches can be interconnected to implement a full, asynchronous Crossbar network which has provisions for path establishment, data transfer, indication of a blocked path and path clearing associated with end of transmission. The asynchronous nature of the network makes it truly modular in that it can be expanded without encountering any synchronozing problems.

References

SWAN77
Swan, R. J. et. al. 'Cm* A Modular Multi-Microprocessor', AFIPS Proc. Nat. Comp. Conf. (1977).

SEJN80
Sejnowski, M. C., et. al. 'An overview of the Texas Reconfigurable Computer', AFIPS Proc., Nat. Comp. Conf. (1980)

SULL77
Sullivan, H. and Bashkow, T. R. 'A Large Scale Homogeneous, Fully Distributed Parallel Machine I', Proc. 4th Ann. Symp. on Comp. Arch. (March 1977)

PADM81
Padmanabhan, K. 'Multiprocessor Interconnection Networks in a VLSI Environment', M. S Thesis, Dept. of Elec. Engr., Washington Univ., St. Louis (Dec. 1981)

FRAN81A
Franklin, M. A., Wann, D. F. ' Asynchronous and Clocked Control Structures for VLSI Based Interconnection Networks'

FRAN81B
Franklin, M. A., Wann, D. F., Thomas, W. J. 'Word Inconsistency in Partitioned VLSI Interconnection Networks', Center for Computer Systems Design, T. M. 81FR-B, Washington Univ., St. Louis (1981)

FRAN79
Franklin, M. A., Kahn, S. A. and Stucki, M. J. , 'Design Issues in the Development of a Modular Multiprocessor Communication Network', Proc. 1979 Int. Symp. on Comp. Architectur (April 1979)

KAHN78
Kahn, S. A., 'Design Issues of a modular Crossbar Network for a Multiprocessor' M. S. Thesis, Washington University, St. Louis, Mo (Dec 1978)

MOLN81
Molnar, C. E., Fang, T. P. "An Asynchronous Design Methodology", Computer Systems Lab., Tech. Mem. 287, Washington University, St. Louis, Mo (Nov. 81)

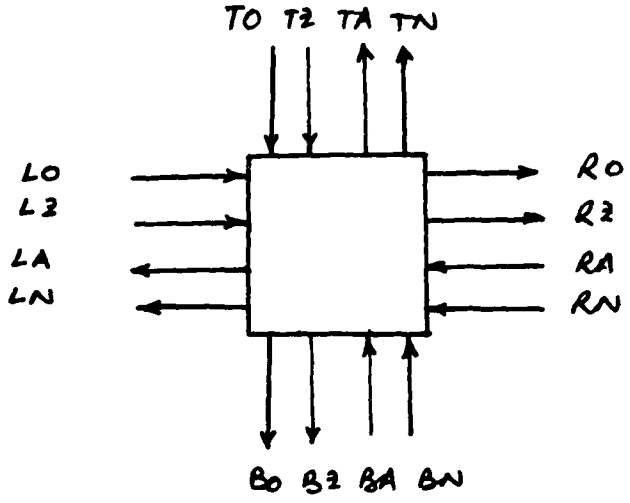


Figure 1: The asynchronous crosspoint switch.

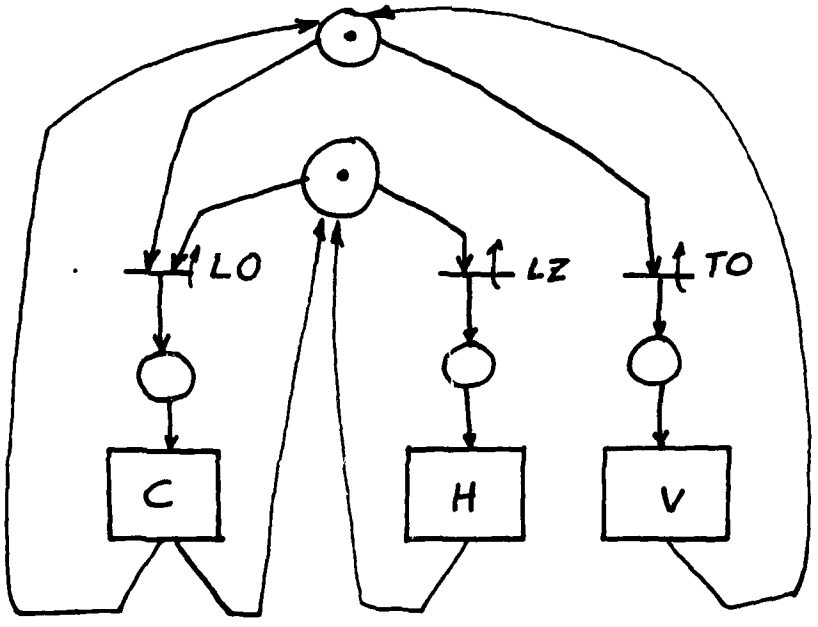


Figure 2: Basic structure of the switch interface net.

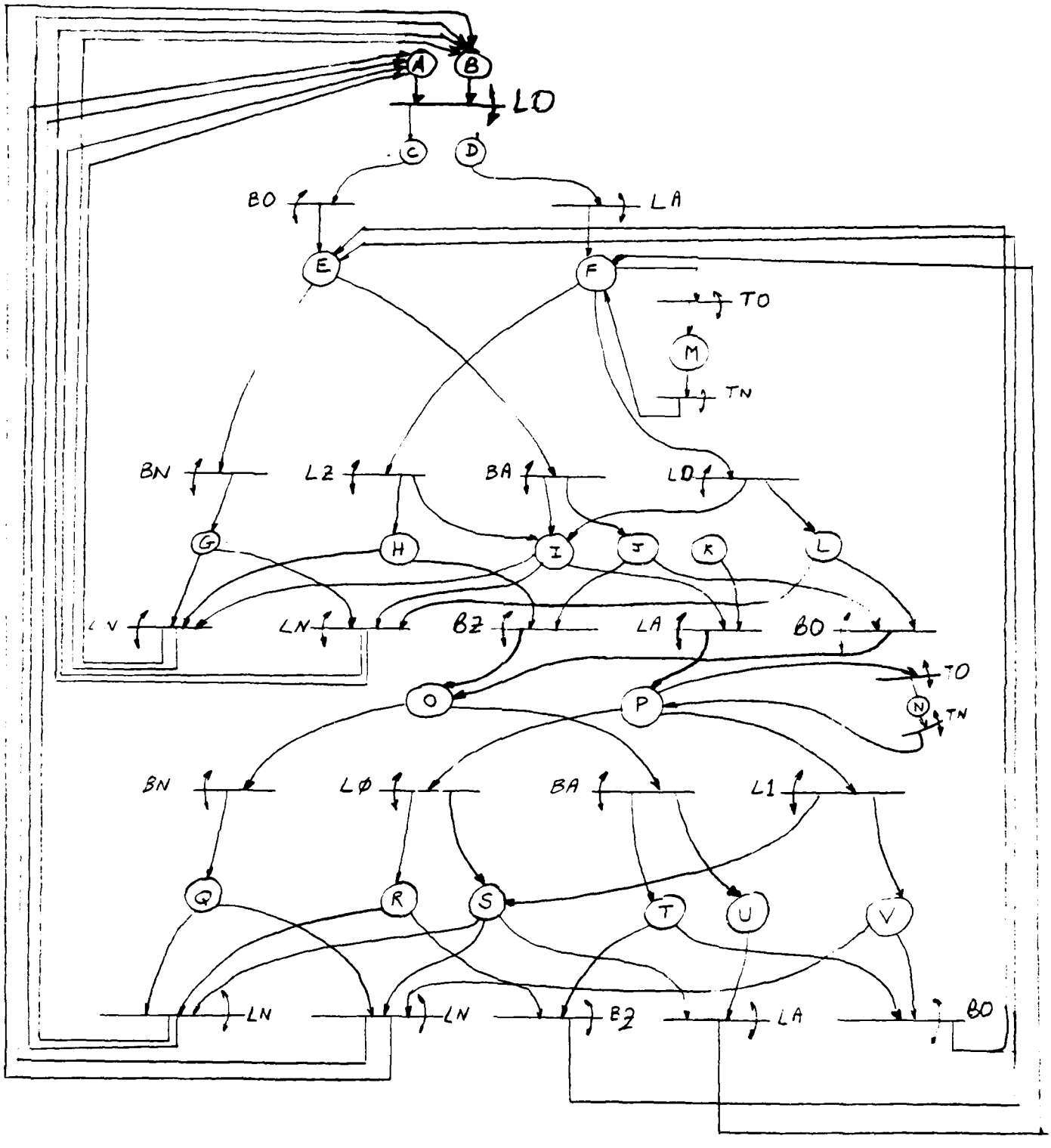


Figure 3: Interface net for the corner path.

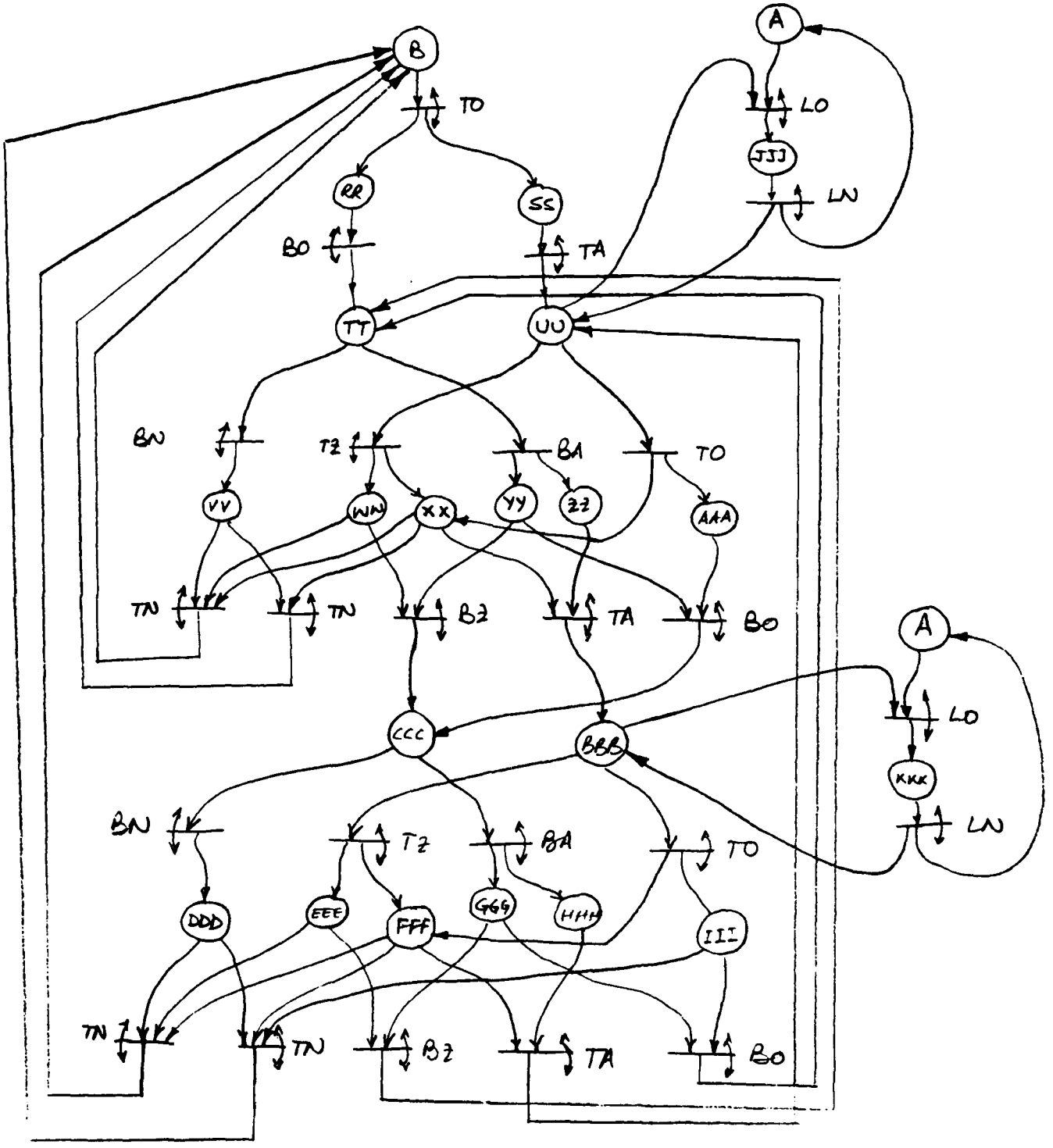


Figure 4: Interface net for the vertical path.

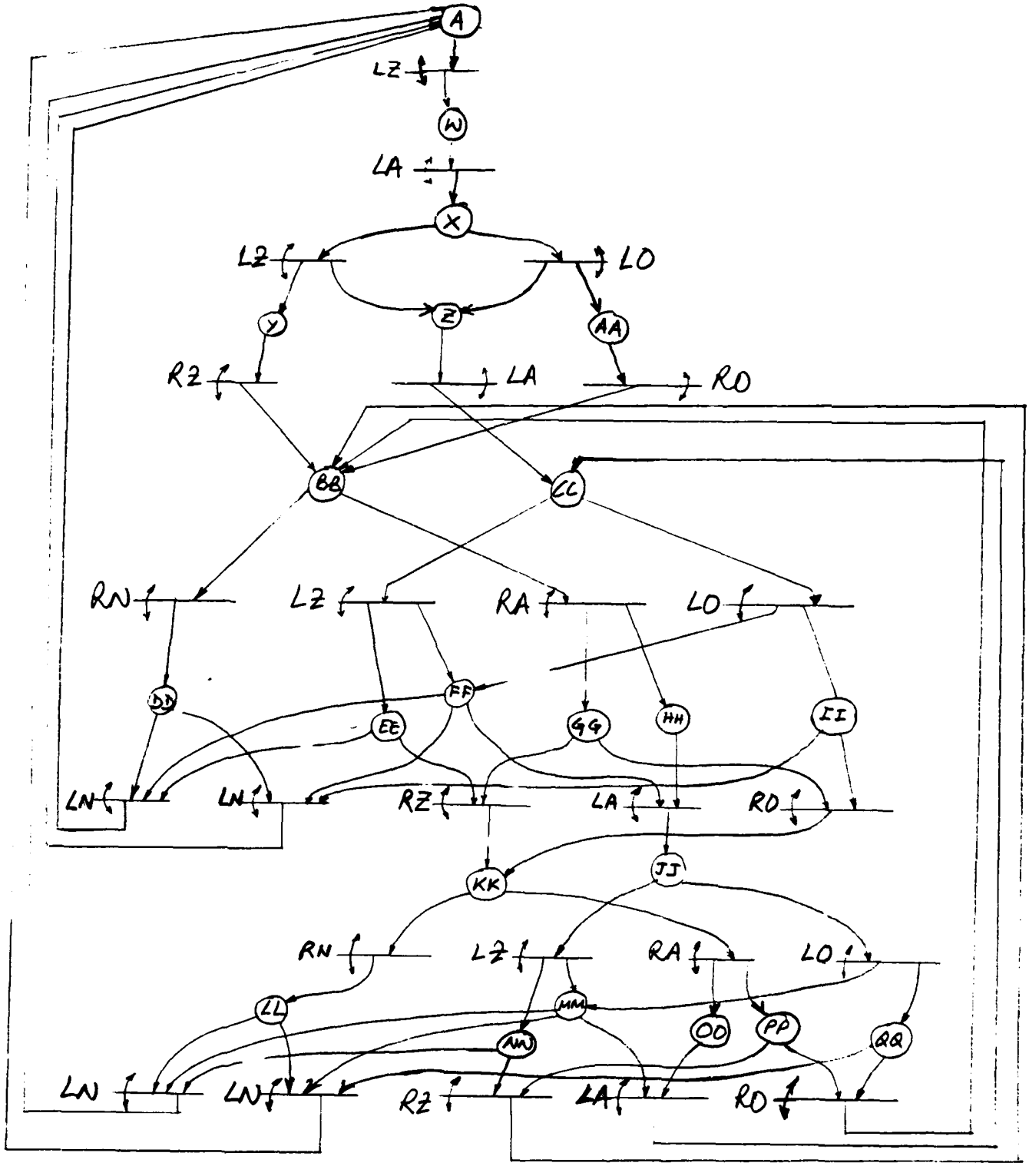
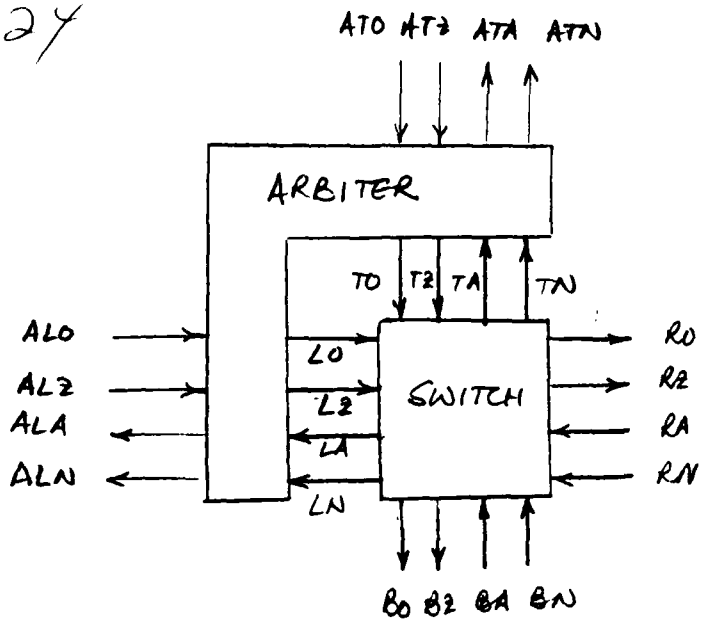


Figure 5: Interface net for the horizontal path.



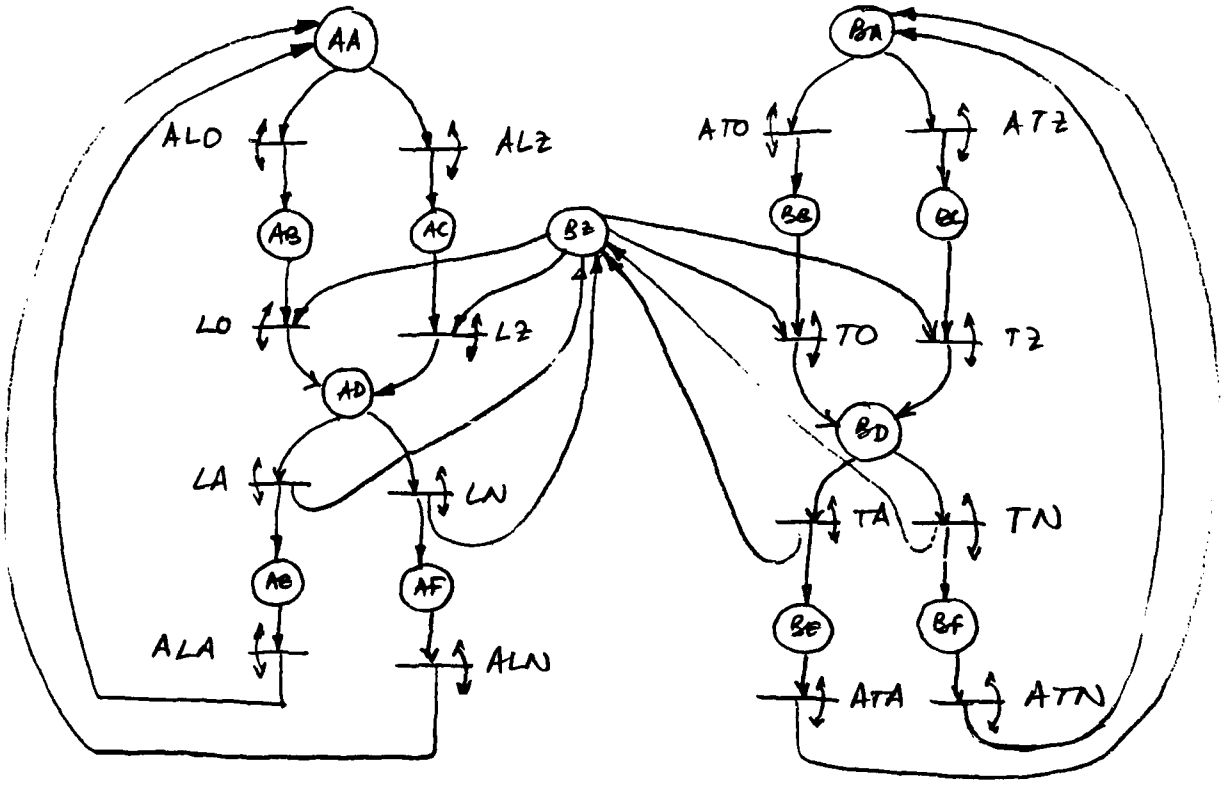


Figure 6: The arbiter-switch interface and the arbiter interface net.

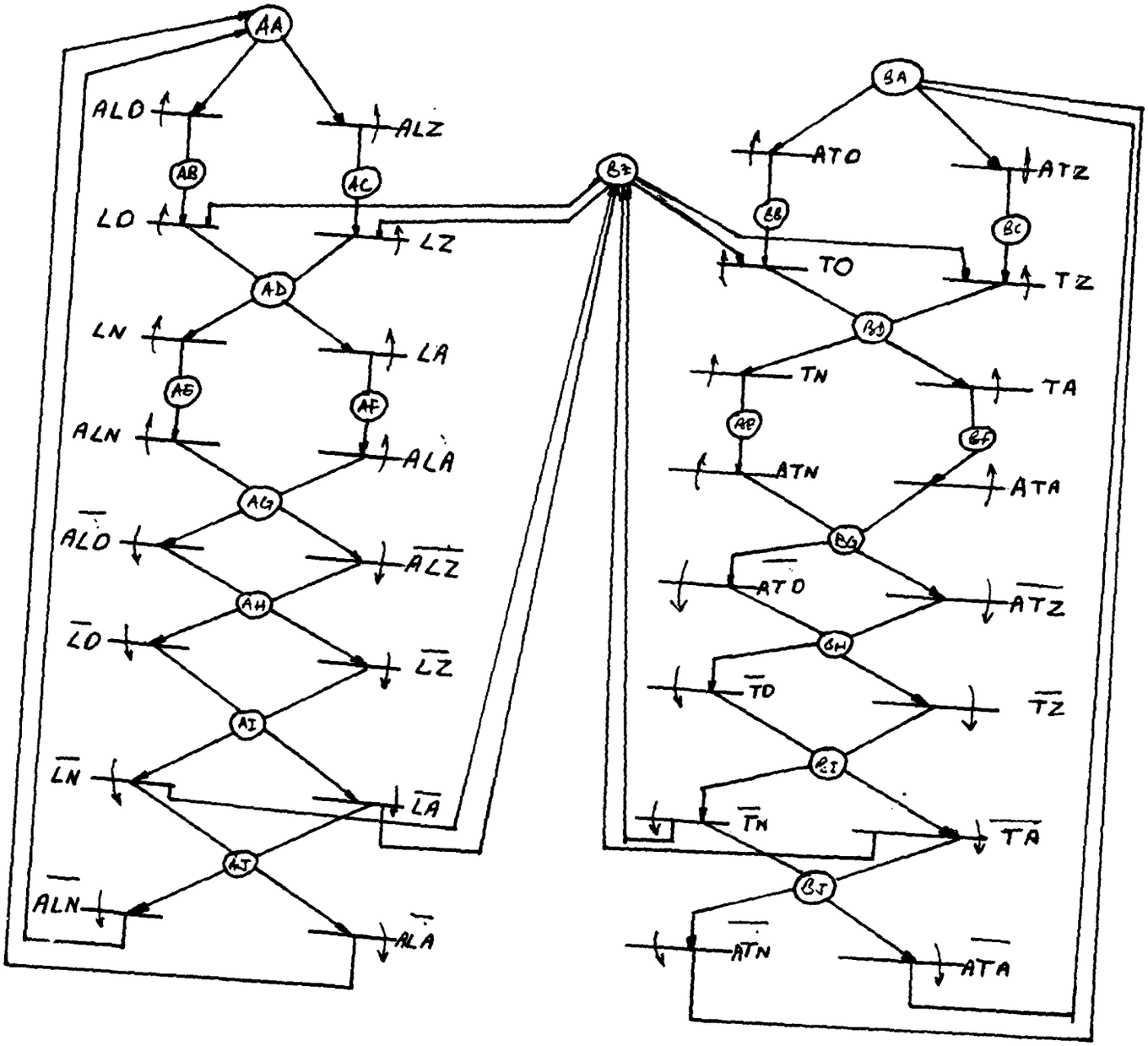
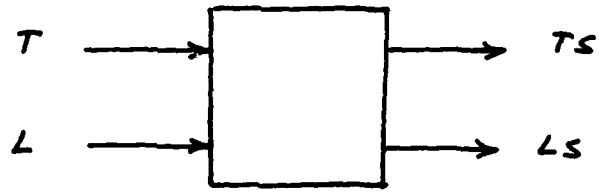
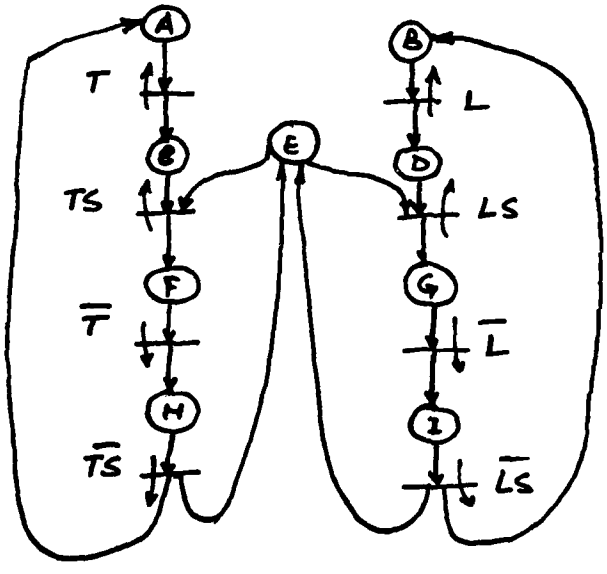


Figure 7: The arbiter interface net with four stroke signalling.

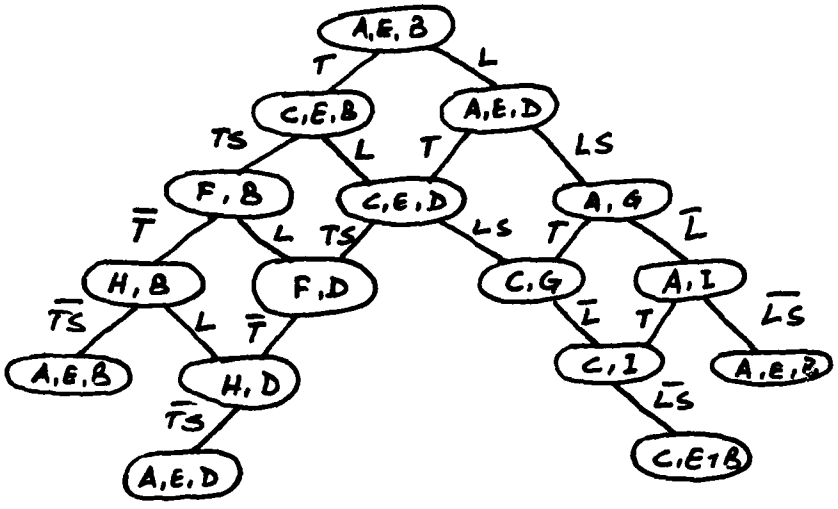


(i) The synchronizer.



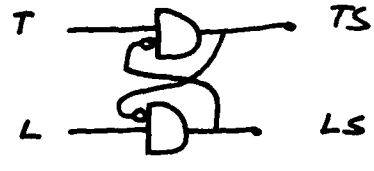
(ii) The synchronizer interface net.

Figure 8(a): The synchronizer and its interface net.



(i) The synchronizer interface state graph.

$$TS = T.\overline{LS}$$
$$LS = L.\overline{TS}$$



(ii) The synchronizer circuit.

Figure 8(b): The synchronizer interface state graph (ISG) and the circuit realization.

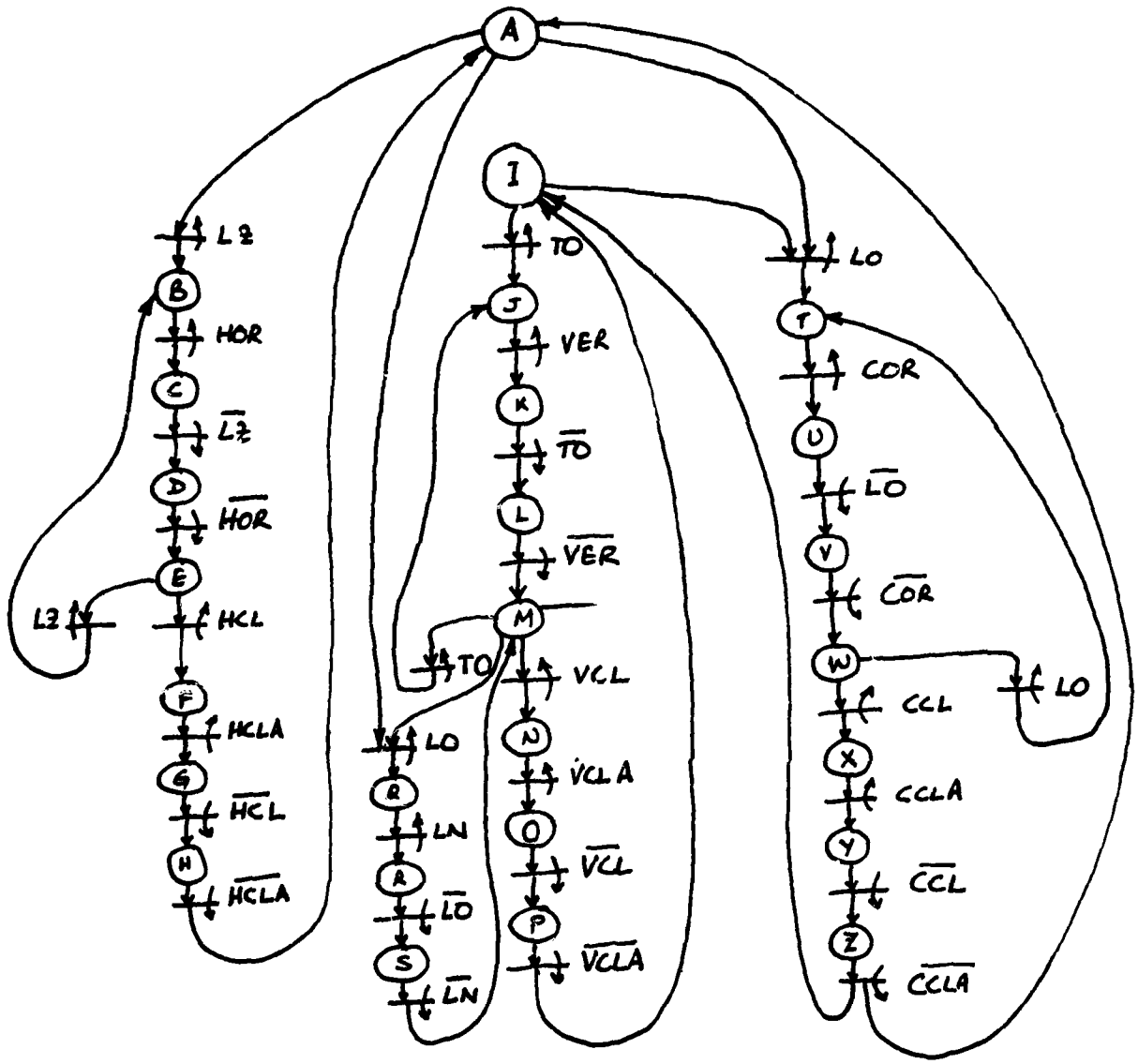


Figure 9: A tentative interface net for the supervisor.

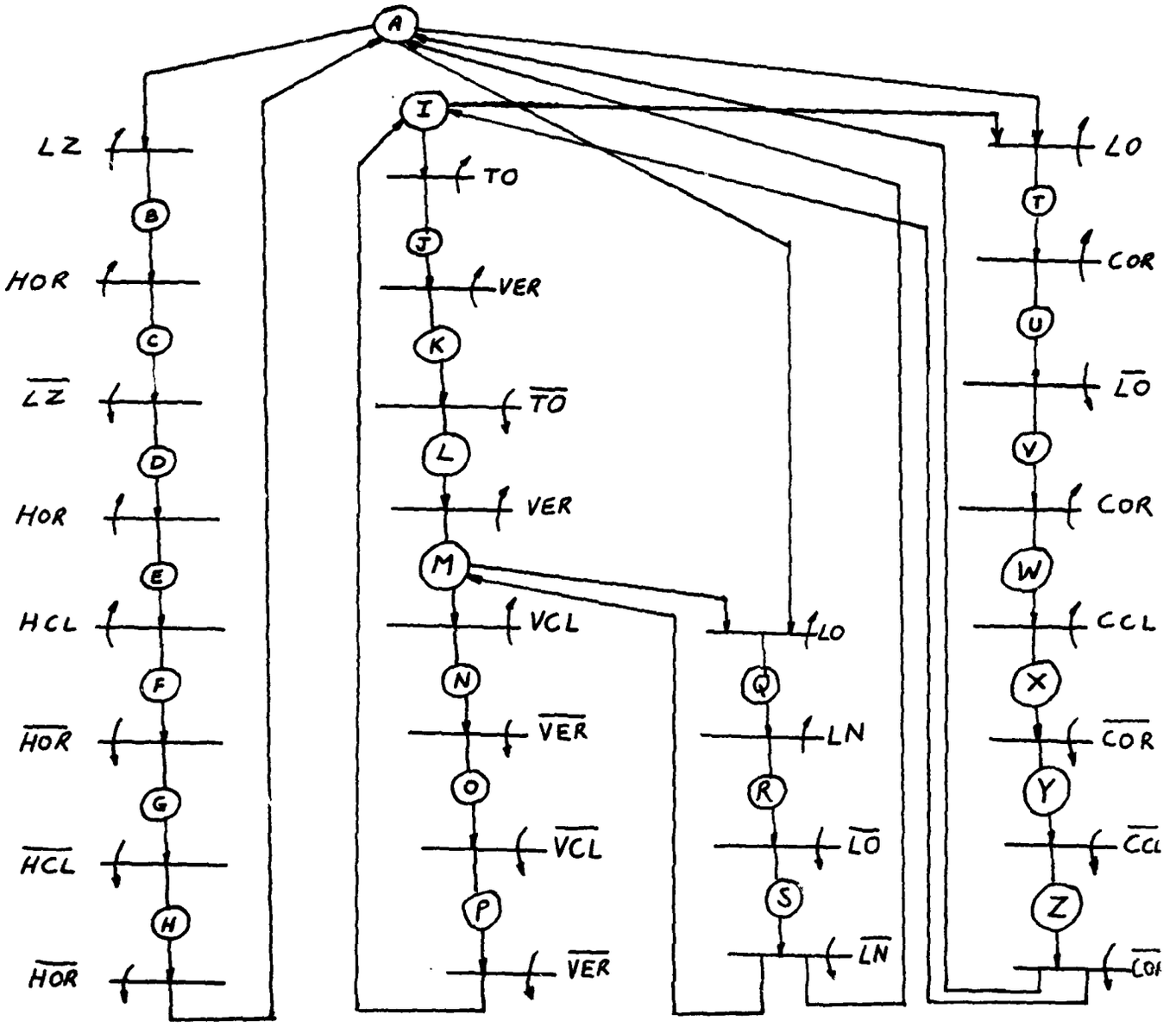


Figure 10: The supervisor interface net that was used to generate the ISG.

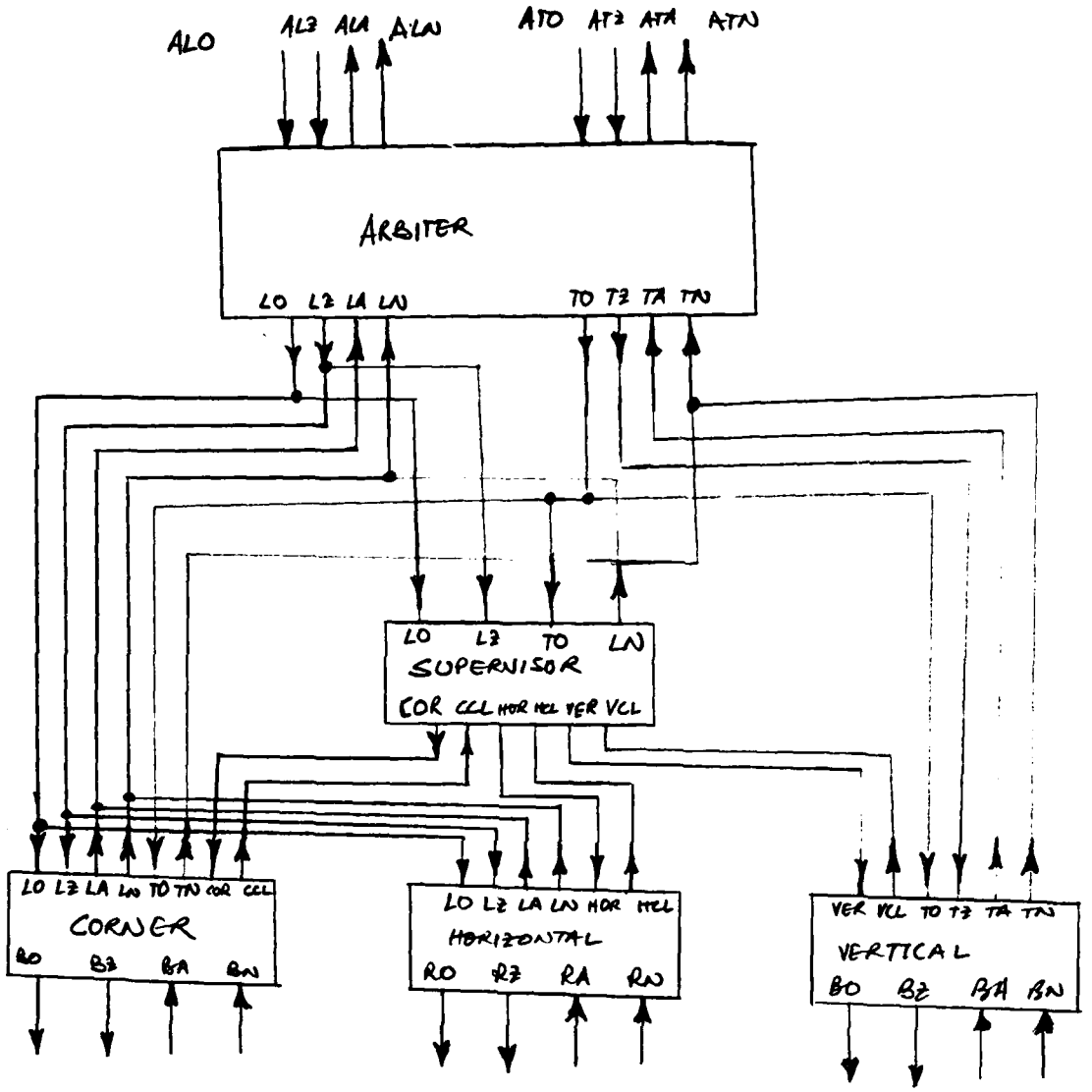


Figure 11: Interconnection of the various building blocks of the switch.

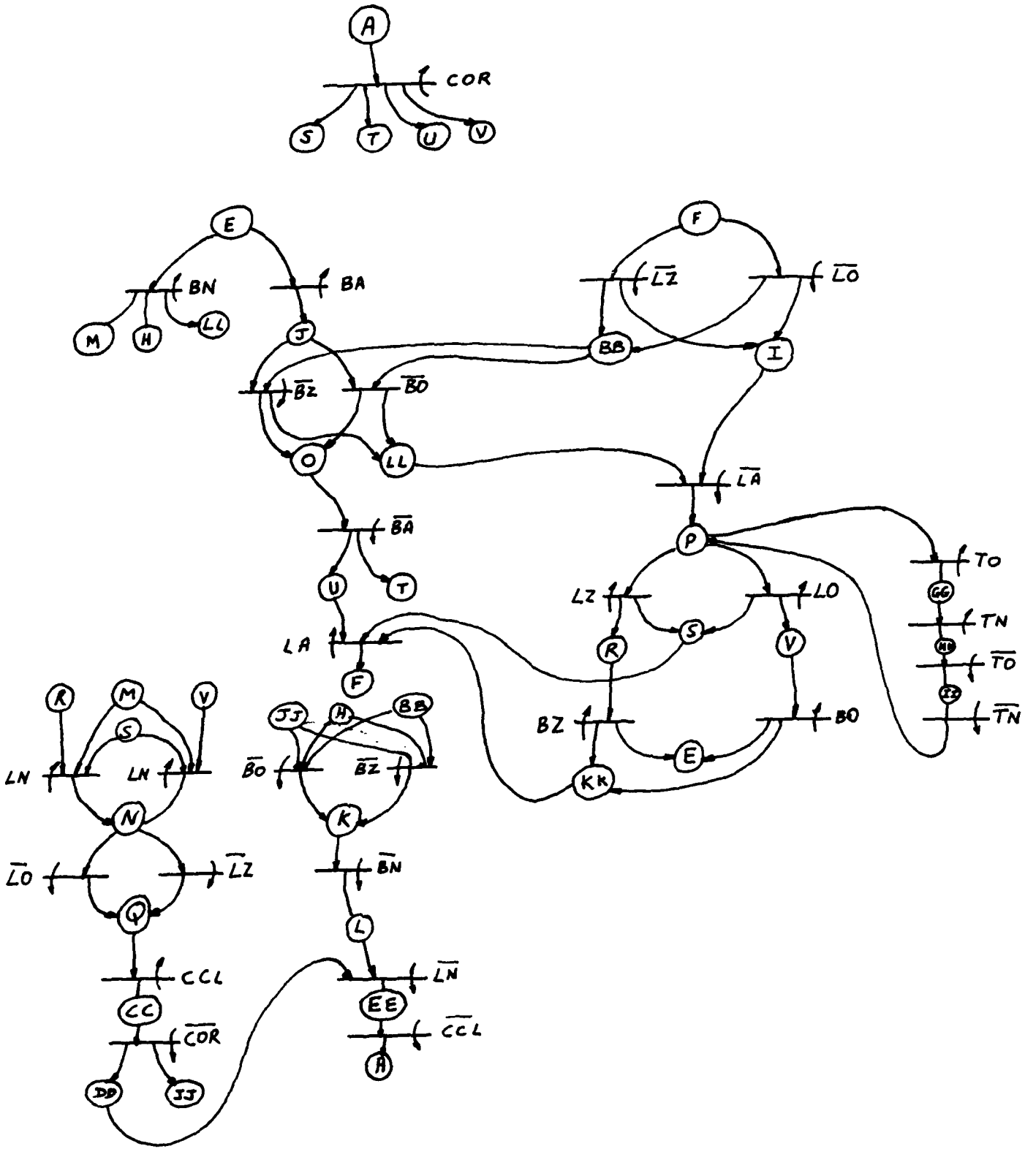


Figure 12: The corner interface net with four stroke signalling.

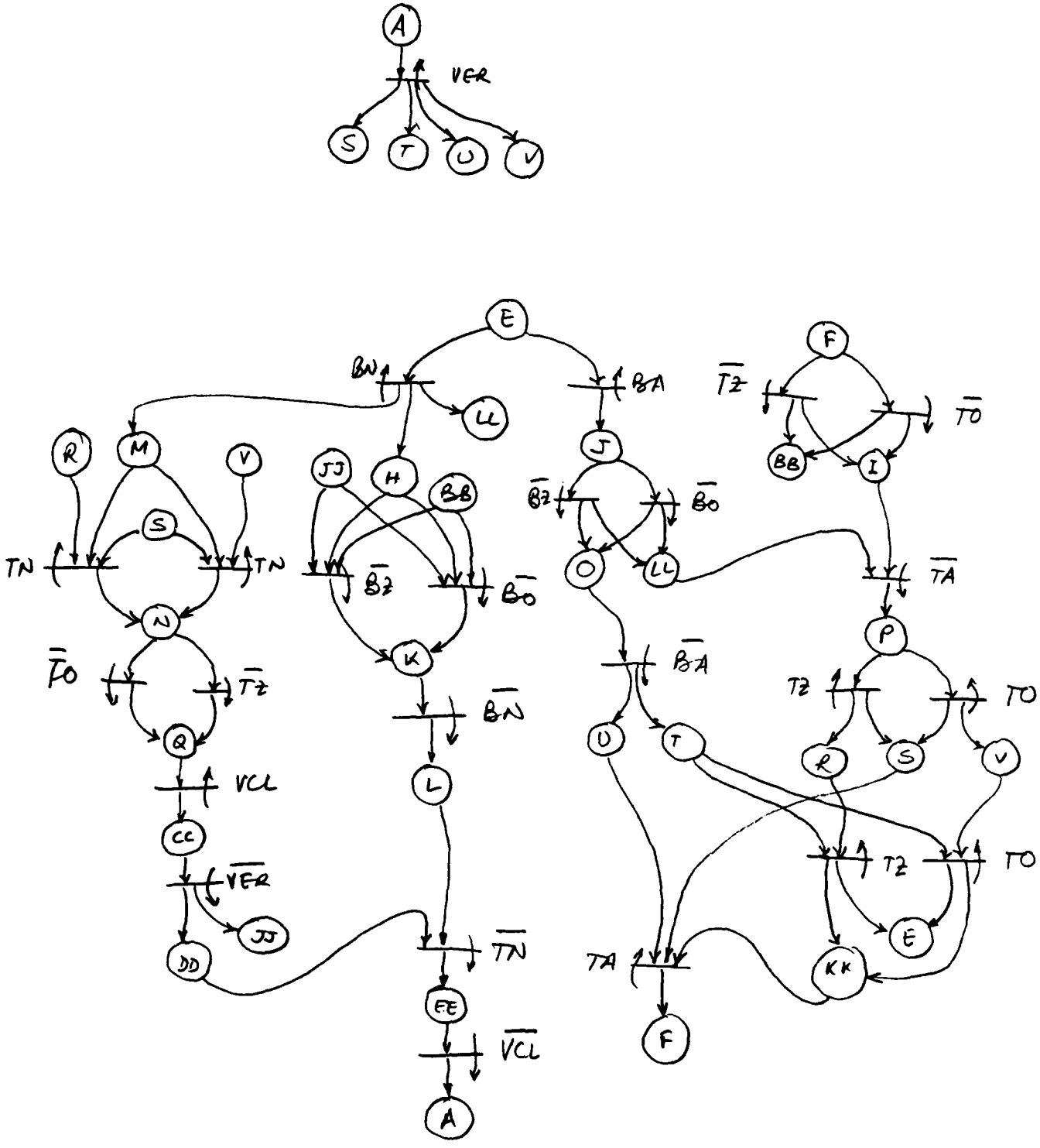


Figure 13: The vertical interface net with four stroke signalling.

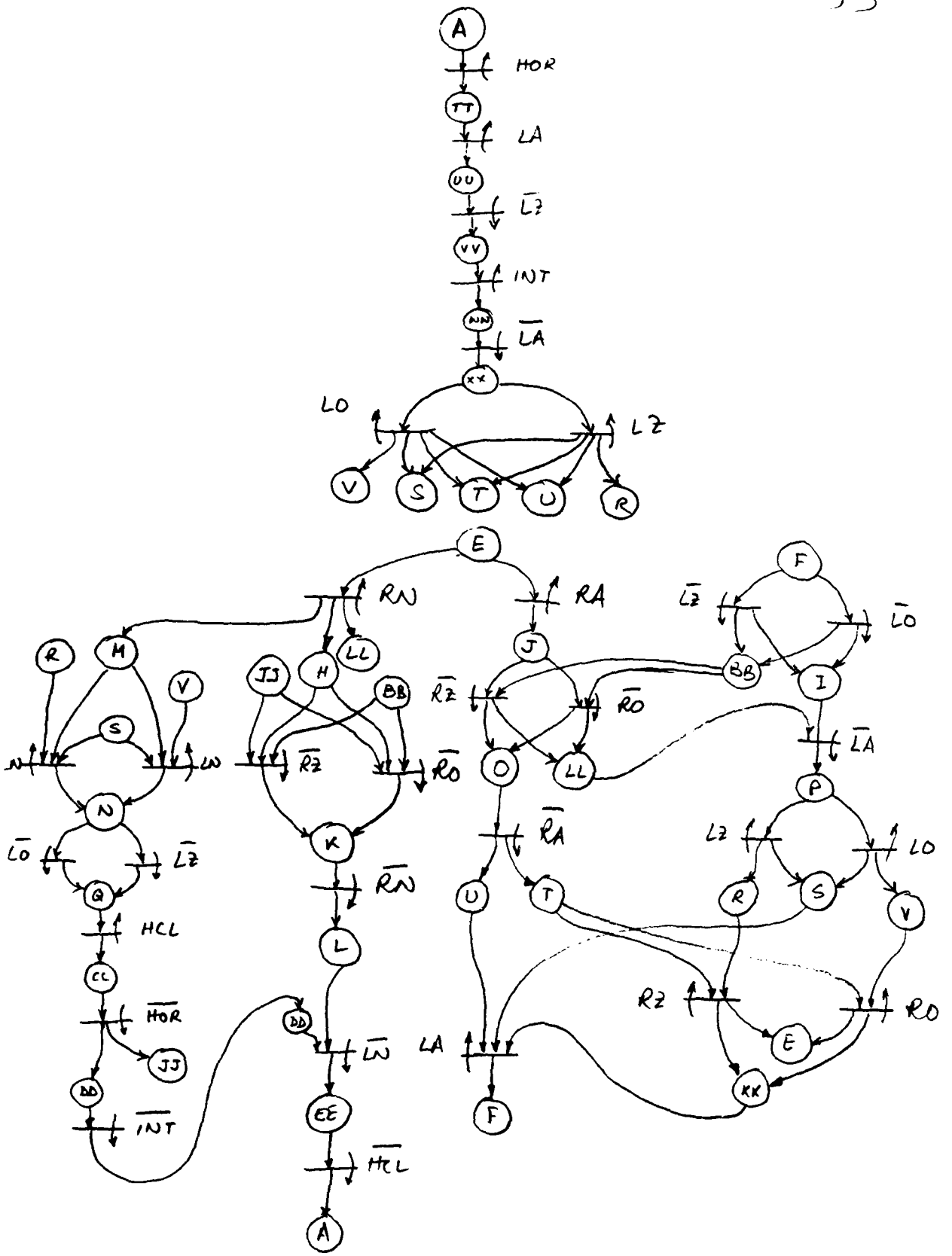


Figure 14: The horizontal interface net with four stroke signalling.

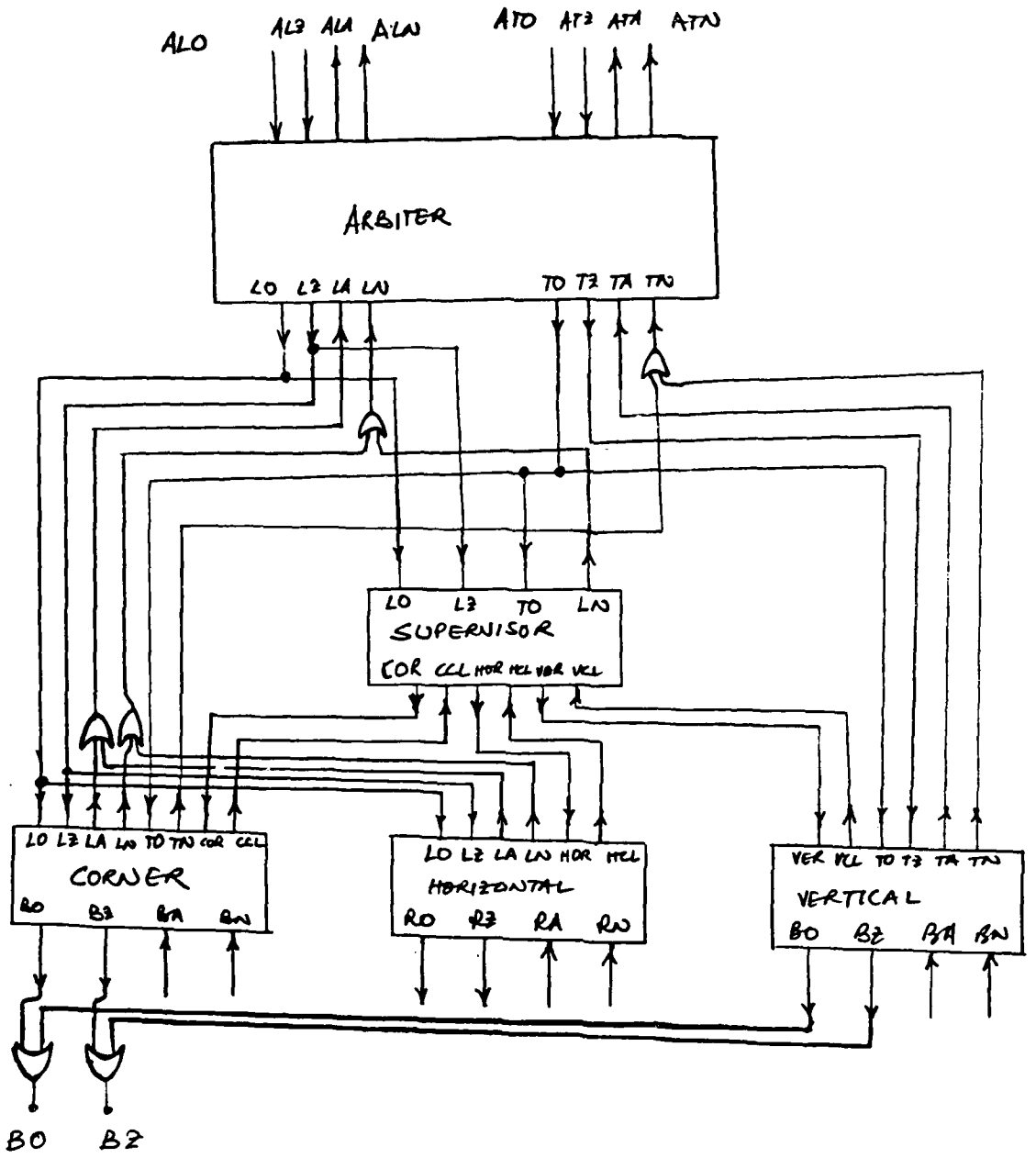


Figure 15: Interconnection of the switch components.

Systolic Processing Architectures: Asynchronous versus Clocked Control

Donald F. Wann and Sanjay Dhar

1.0 Introduction

Although increases in processing power can be achieved using multiple processors interconnected via a switching network, another architectural style that seems to offer similar performance increases has recently received considerable attention. This is the systolic array. This architecture has the desirable properties of being modular, can be pipelined, and has the potential for high speed concurrrent processing. It also appears to be quite attractive for VLSI implementation because of its regular structure. That is, it only requires the design of a single basic module for each of the linear, rectangular or hexagonal arrays.

There are numerous papers describing the applications of such arrays, particularly in the signal processing field [see references 1,2,3,4,5]. However, most of this literature is concerned with the presentation of algorithms and there is only a limited discussion of how such arrays can be controlled. We also are not aware of any reports on how one determines the actual performance of these arrays in terms of bandwidth or data rate. As with the multiprocessor interconnection networks, both synchronous and asynchronous control can be used.

In this Appendix we present some preliminary studies aimed at determining accurate delay based models for both types of control strategies. These models allow us to make comparisons of the control strategies and also permit us to predict the performance (e.g. data rates) of both structures.

The single dimensional linear systolic array is examined and delay models are constructed. From these models the worst case computational periods are extracted and the data rates obtained. The results for the synchronous and asynchronous structures are :

$$DRs = 1/(dcompute + dpath + alpha + delta)$$

$$DRa = 1/(dcompute + 2dpath)$$

where dcompute is the processing delay associated with the module, dpath is the propagation path delay associated with the transfer of data from module to module, and alpha and delta are related to the skew of the clock in the synchronous system.

It should be emphasized here that the systolic array architecture is substantially different than multiprocessor architecture and this has important fabrication implications. For a moderate number of processors an interconnection network can probably be placed on a single chip or a bit slice approach can be used. The interconnection network, being simple, is therefore not chip area limited, but pin limited. On the other hand, with the systolic array, each module contains a processor. This increased logical complexity will require chip area and thus more of a balance may be achieved between area and pin constraints.

2.0 Control Techniques for Systolic Arrays

There are two fundamental types of control that can be used for the systolic array: synchronous (or clocked) and asynchronous (or self-timed). Each of these will now be examined and data rates obtained as a function of the delays of the computations, the intermodule paths, and the clock lines.

2.1 Synchronous Control

Figure 1 illustrates a one dimensional systolic array. Each module in

this array is configured to compute the relation

$$Yj = Yk + A*Xi \qquad (1)$$

where Yj, Yk, and Xi are nx1 column matrices and A is a square nxn matrix. A register/processor model for module j of Figure 1 is shown in Figure 2. Notice that there are five registers and one processor, which consists of a multiplier and an adder. Module j accepts Xi from module i on its left, accepts A from the top, and Yk from module k on its right. Module j then computes a new Y, (Yj), according to (1) and sends it to module i. Module j also takes Xi, and passes it unchanged to module k as Xj. The synchronization of these steps can be accomplished by using a central clock in which standard two-phase techniques are employed. A typical clock waveform that might be employed for this purpose is sketched in Figure 3. The two phases are identifed as $\phi 1$ and $\phi 2$. The asserted time for phase-one is $\phi 1$ and for phase-two is $\phi 2$. The interphase times are denoted as $\phi 12$ and $\phi 21$.

One systolic array control procedure is to capture A, Xi, and Yk on the assertion of phase one and then during $\phi 1$, to make the computation of Yj. The length of $\phi 1$ would be adjusted so that this computation would be completed before the end of $\phi 1$. Then, on the assertion of phase two, Xj and Yj would be transferred to modules k and i. This is the control procedure depicted in Figure 2. We are interested in evaluating the data rate for this process, so our models must include the major delays encountered in the above processing. A model illustrating the delays in the clock lines is illustrated in Figure 4. In addition to the eight clock delays shown in this figure, there are three other delays that are important. Two of these are intermodule delays associated with the transfer of Yk from module k to module j, which we will call dYkj, and the transfer of Xi from module i to module j , which we shall call dXij. The third delay is associated with the delay in obtaining the coefficients from A and transferring them to module j. This will be called dAj.

The time necessary for module j to complete its processing task is constrained by its interactions with module i, module k and the coefficients contained in A. Equations for these three constraints can be obtained by constructing three timing diagrams corresponding to the transfer of

1) Yk to module j
2) Xi to module j
3) A to module j

Timing diagrams for these three cases are depicted in Figures 5, 6 and 7 respectively. As an example of how these constraints are determined, consider Figure 5 which shows the timing for transferring Yk to module j. Assume that Xi, Yk and A are available and stable when the phase-one clock is asserted. We can determine the first constraint on the value of the clock period T needed for correct behavior of module j, by tracing a path from one assertion of the phase-one clock through the various signal paths that include both processsing and propagation delays, and concluding on the next assertion of the phase-one clock. This path is depicted in Figure 5 and yields the following relation:

$$T1 > \phi 1 + \phi 12 + dYkj + (dc2k - dc1j) \qquad (2)$$

Observe that this relation is found by starting at the assertion of phase-one of the master clock, proceeding through the delay $\phi 1$, which is the time necessary to capture A, Xi, and Yk and to perform the computation of the new Yj, then through the delay between the assertion of phase-two of the master clock and the assertion of phase-two of the clock at module k which is equal to dc2k, then through the delay necessary to transfer the new Yk to module j, which is related to the assertion of the master phase-two clock by the relation dYkj - dc1j.

In a similar fashion another constraint on the clock period that is associated with the transfer of Xi to module j can be obtained from Figure 6

and gives:

$$T2 \geq \phi 1 + \phi 12 + dXij + (dc2i - dc1j) \quad (3)$$

Finally a third constraint on T associated with the transfer of A to module j is depicted in Figure 7 and yields

$$T3 \geq \phi 1 + \phi 12 + dAj + (dc2A - dc1j) \quad (4)$$

It is useful to rewrite each of the three period constraints. We can express T1 as

$$T1 \geq \phi 1 + \phi 12 + dYkj + (dc1k - dc1j) + (dc2k - dc1k) \quad (5)$$

The last term of this equation is the difference in delay of the two clock lines that reach module k. Let us call this alpha.kk. Thus

$$alpha.kk = dc2k - dc1k \quad (6)$$

The next to last term is the difference in the arrival of the phase-one clock at modules k and j. This is commonly referred to as clock skew and we will denote it as delta.kj. Hence

$$delta.kj = dc1k - dc1j \quad (7)$$

Each of the remaining two clock period constraints, T2 and T3, also have similar expressions, therefore

$$T2 \geq \phi 1 + \phi 12 + dXij + delta.ij + alpha.ii \quad (8)$$

$$T3 \geq \phi 1 + \phi 12 + dAj + delta.Aj + alpha.jj \quad (9)$$

where

$$\begin{aligned} alpha.ii &= dc2i - dc1i \\ alpha.jj &= dc2j - dc1j \\ delta.ij &= dc1i - dc1j \\ delta.Aj &= dc1A - dc1j \end{aligned} \quad (10)$$

Now $\phi 1$ represents the amount of time necessary for the capture of the X, Y and A values plus the time to compute the new value of Y. Let us represent this as a single delay, dcompute. Furthermore, the terms dA, dX, and dY represent intermodule path propagation delays, and we will identify them as dpath.

Then the three constraints on the clock period can be rewritten as

$$T1 > dcompute + dpathY + delta.kj + alpha.kk \quad (11)$$

$$T2 > dcompute + dpathX + delta.ij + alpha.ii \quad (12)$$

$$T3 > dcompute + dpathA + delta.Aj + alpha.jj \quad (13)$$

As a consequence the constraint on the clock period can be specified as

$$T > \max_{\text{all modules}}(T1, T2, T3) \quad (14)$$

In addition to the constraint shown by equation 14 we must also guarantee that the two clock phases do not overlap at the input to any module. That is we must insure

$$\phi 12 > 0 \quad (15)$$

$$\phi 21 > 0$$

at each module clock input. A timing diagram for module j is shown in Figure 8. From this diagram we see that

$$O21j = \phi 21 + dc1j - dc2j = \phi 21 - delta.jj > 0 \quad (16)$$

$$O12j = \phi 12 + dc1j - dc2j = \phi 12 - delta.jj > 0 \quad (17)$$

So ,

$$\phi 21 > delta.jj \quad \text{and} \quad \phi 12 > delta.jj \quad (18)$$

Similar expressions can be obtained for the interphase times at other module inputs, hence

$$\phi 12 > \max(delta.ii, delta.jj, delta.kk)$$
$$\phi 21 > \max(delta.ii, delta.jj, delta.kk) \quad (19)$$

Each of the parameters in these constraint equations has a range of values which depend on the variations in the fabrication process, the computation time, and in the interconnection pathways. Let us assume that we have a large systolic array and that the delays take on their full range of values. Further, let us assume that, because the array is large, there exists an adjacent set of three modules i, j, and k and a coefficient array, A, in which all the

parameters have their worst case values. Then the identifying subscripts can be removed from the constraint equations and they can be rewritten as

$$\begin{aligned} T &> dcompute + dpath + delta + alpha \\ \emptyset 12 &> delta \\ \emptyset 21 &> delta \end{aligned} \tag{20}$$

These equations then show how clock skew, computation time, and intermodule path delays affect the clock period and thereby the data rate. Recalling that the data rate is the inverse of the clock period, we have the data rate for the synchronously controlled systolic array as

$$DRs < 1/(dcompute + dpath + delta + alpha) \tag{21}$$

2.2 Asynchronous Control

The one dimensional systolic array shown in Figure 1 can be adapted to incorporate the asynchronous or self-timed control structure. In the synchronous control structure, all events are globally synchronized with the clock; in other words, every event has a fixed position in the time domain. In a self-timed system, on the other hand, events can be thought of as having a fixed position in the sequence domain with no event having to occur at a particular or fixed time. Self-timed systems are an interconnection of components where each component performs a step in the desired computation. In order to maintain the order in the sequence domain necessary to perform the computation, a signal is necessary at the input of a component to initiate the computation step, and a signal is necessary at the output of the component to mark the completion of the computation step.

Considering a particular module j in the one dimensional systolic array, the computation performed by the module is the innner product step

$$Yj = Yk + A*Xi \tag{1}$$

As has been mentioned earlier, each module must have three data input lines,

one each for the X, Y and A data and two data output lines, one each for the X and Y data. The control lines necessary for each data input line to achieve self-timed operation are:

(i) An input line which signals the availability of data on the particular data line

(ii) An output line which signals the completion of use of data on the data line.

The control lines necessary for each data output line are:

(i) An output line which signals the availability of data on the particular data line

(ii) An input line which signals the completion of use of data on the data line.

The first signal in each set of control lines (i) will be referred to as the Data Available lines (identified by DA) and the second signal in each set (ii) will be referred to as the Acknowledge lines (identified by A). The asynchronous systolic array module and associated control signals is shown in Figure 9.

2.2.1 A Petri Net Specification of the Systolic Array Module

Because of the concurrency in the control and data flow in the systolic array module, it is not possible to specify formally the behaviour of such a system by a state table. Instead a Petri net is used for the formal specification of the module. In obtaining this net for the module, we would like to preserve all the concurrency in the module, as this should maximize the data rate through the module.

The following characteristics of the asynchronous module (Figure 9) are important in order to construct the Petri net:

(i) New data can be made available to the module on the right on XDAR

if the acknowledge of the previous data is available on XAR and data is available from the module on the left on XDAL.

(2) The computation process in the module consists of two distinct steps, a multiplication followed by an addition. The multiplication step can begin as soon as data on XDAL and ADAT are available and the previous addition step is complete.

(3) The acknowledge signal AAT for ADAT can be sent after the multiplication step is complete. The acknowledge XAL for XDAL is sent only after the multiplication step is complete and the acknowledge to the data available signal XDAR on XAR has been received.

(4) The addition step can begin as soon as the data required for this step is available. That is, after the multiplication step is complete and data is available on YDAR.

(5) New data can be made available to the module on the left when the addition step is complete and when the acknowledge to the previous data on YDAL has been received on YAL.

(6) The acknowledge signal on YAR is sent after the addition step is complete and the acknowledge signal YAL has been received.

Ordering YAR after YAL and XAL after XAR becomes necessary to avoid generating a new output before the old output has been used. For example, if we send the YAR signal independent of the YAL signal, new data will become available for the addition step and a new result could be generated before the old data has been used, that is, the addition step could end before the acknowledge signal is received on YAL. Similar reasons necessitate the ordering of XAL after XAR.

The Petri net for the systolic array module j is presented in Figures 10(a) and 10(b). A note on the new signals used in the Petri net:

ME Multiplication Enable: initiates the multiplication step.

MC Multiplication Complete: indicates end of the multiplication step.

AE Addition Enable: initiates the addition step.

AC Addition Complete: indicates end of the addition step.

Notice that the Petri net can be divided into two halves (upper and lower in Figure 10) which are essentially identical. In the one dimensional systolic array, the module k is one step ahead in the computation process than module j because module k starts the computation process before module j. The Petri net therefore not only represents the interaction between two adjacent modules but also represents the interaction between two adjacent computation steps. This results in the two identical halves of the Petri net.

The delay between successive computations in the systolic array module j can be obtained by analyzing the paths in the module and determining the "loop" delays. To be as general as possible, this analysis should take into account the concurrency in the module. Such an analysis, however, becomes quite involved and is not consistent with the derivations for the synchronous control structure. Therefore we will remove the concurrency from module j. In this case, the path having the largest delay is shown in Figures 10(a) and 10(b) in dashed line. This path along with the delays in the path is shown in Figure 11. The path involves propagation delays that are internal to a module (identified by the subscript i), propagation delays that are external to the module, that is, intermodule propagation delays (identified by the subscript e) and the delay due to the computation, that is, the time necessary to perform the multiplication and addition.

The intermodule path delays will, in general, be much larger than the internal propagation delays. We will therefore assume that the internal delays have been included in the external path delays. The delay through the module is then given by

$$dasync = dcompute + 2dpath \quad (23)$$

where dpath is the intermodule path delay and dcompute is the delay in the computation. The data rate for the one dimensional systolic array is then given by the inverse of the dasync and becomes

$$DRa = 1/(dcompute + 2dpath) \quad (24)$$

3.0 Observations

The results of this work provide very useful insight into the performance of these two control structures. Both of the data rates are inversely proportitional to the computation time. The asynchronous structure data rate is related to two times the path delay while the synchronous date rate involves only a single occurrence of the path delay. The synchronous control is naturally dependent on the clock skew, while the asynchronous is not. The further intrepretation of the results depends on the relative magnitudes of the three parameters. For example, if the path delay is small compared to the computation delay (which often would be the case) then only if the clock skew were zero would the synchronously controlled array perform as well as the asynchronously controlled array.

Our next efforts will be directed toward obtaining realistic estimates for these delays and making such comparisons. We also plan on developing models and data rate equations for both the square and hexagonal systolic arrays.

4.0 References

1. Cappello, P.R and Steiglitz, K. "Digital Signal Processing Applications of Systolic Arrays". in VLSI Systems and Computations, Ed. by Kung and Steele, Computer Science Press, pp 245-254, 1981

2. Kung, H.T. , Ruane, L.W. and Yen, , D.W. "A Two-Level Pipelined Systolic Array for Convolutions". in VLSI Systems and Computations, Ed. by Kung and Steele, Computer Science Press, pp 255-264, 1981

3. Kung, H.T. "Special Purpose Devices for Signal and Image Processing: An Opportunity in VLSI", in Proc. SPIE, pp 76-84, July, 1980

4. Kung, H. T. and Leiserson, C. E. "Algorithms for VLSI Processor Arrays", Section 8.3 of Introduction to VLSI Systems, C. Mead and L. Conway, Addison-Wesley, 1980.

5. Yen, D. and Kulkarni, A. "The ESL Systolic Processor for Signal and Image Processing", Proc. Computer Soc. Workshop on Computer Arch. and Image Database Management, November, 1981

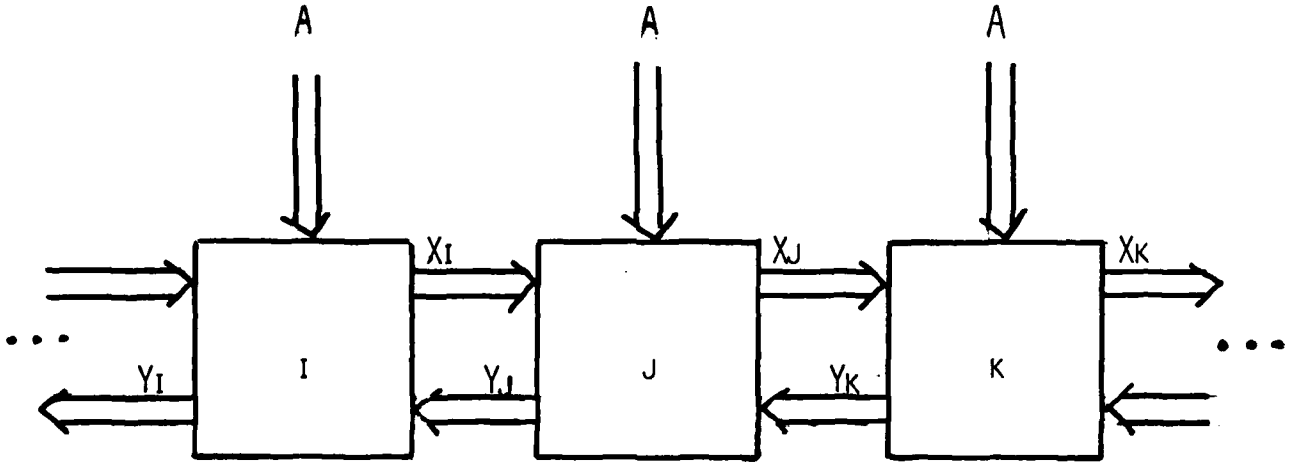


FIGURE 1

ONE DIMENSIONAL SYSTOLIC ARRAY

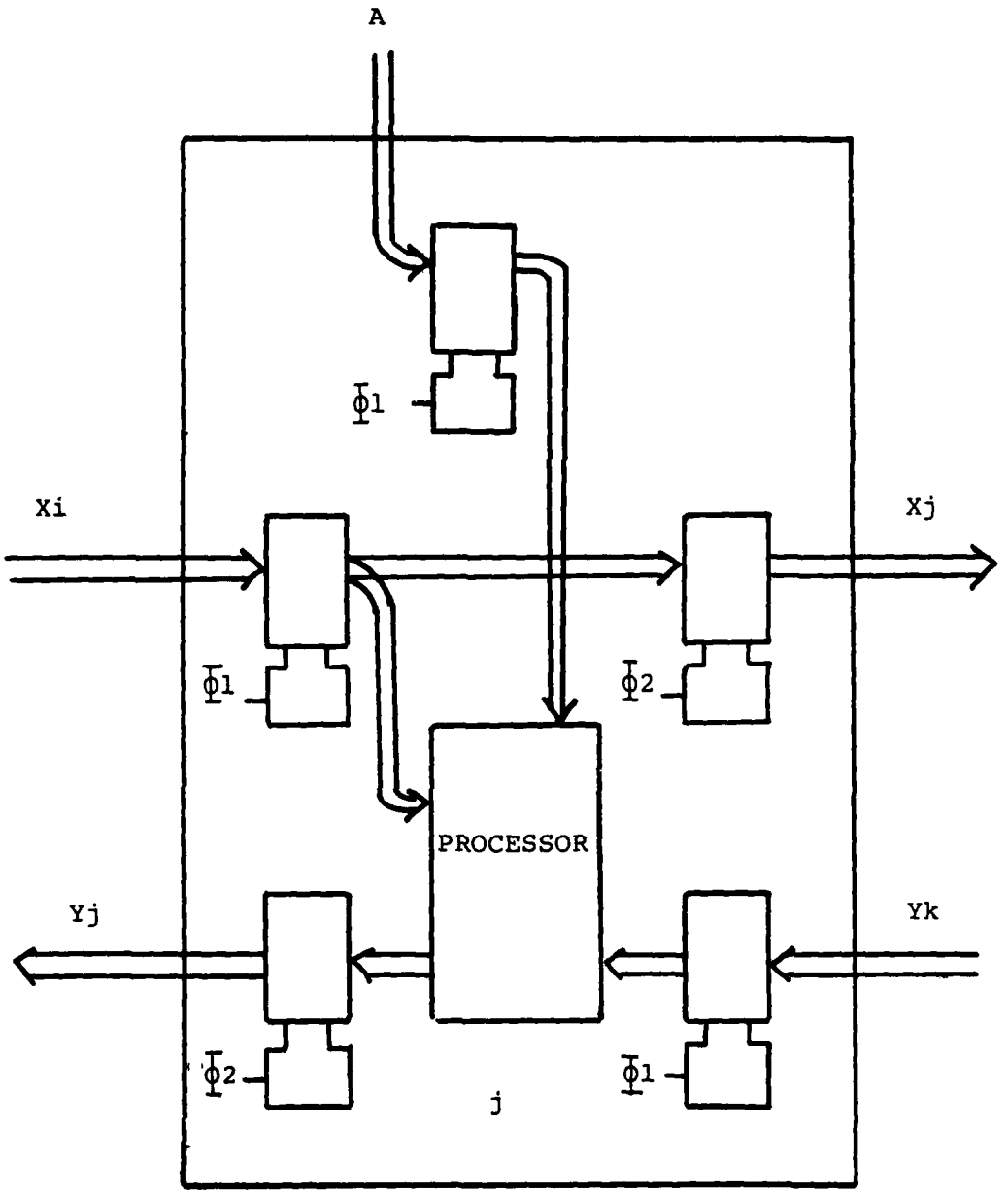


FIGURE 2

REGISTER/PROCESSOR ARRANGEMENT FOR MODULE j

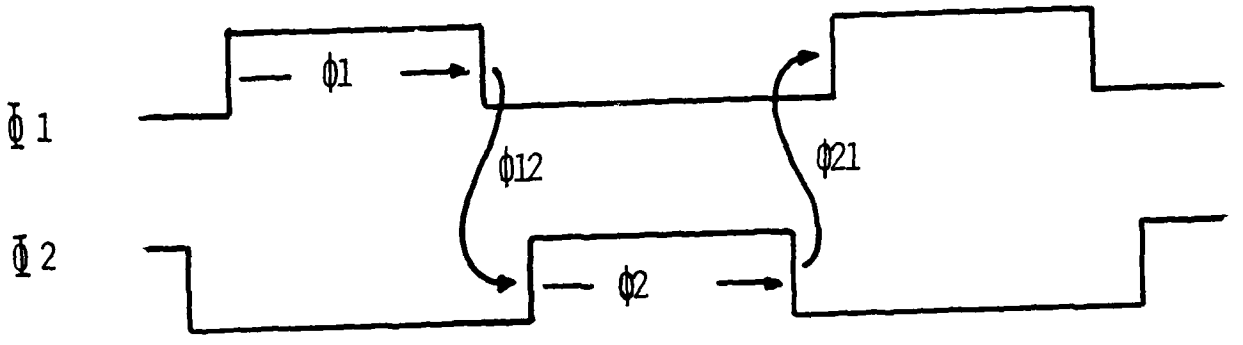


FIGURE 3

WAVE FORMS FOR TWO-PHASE CLOCK

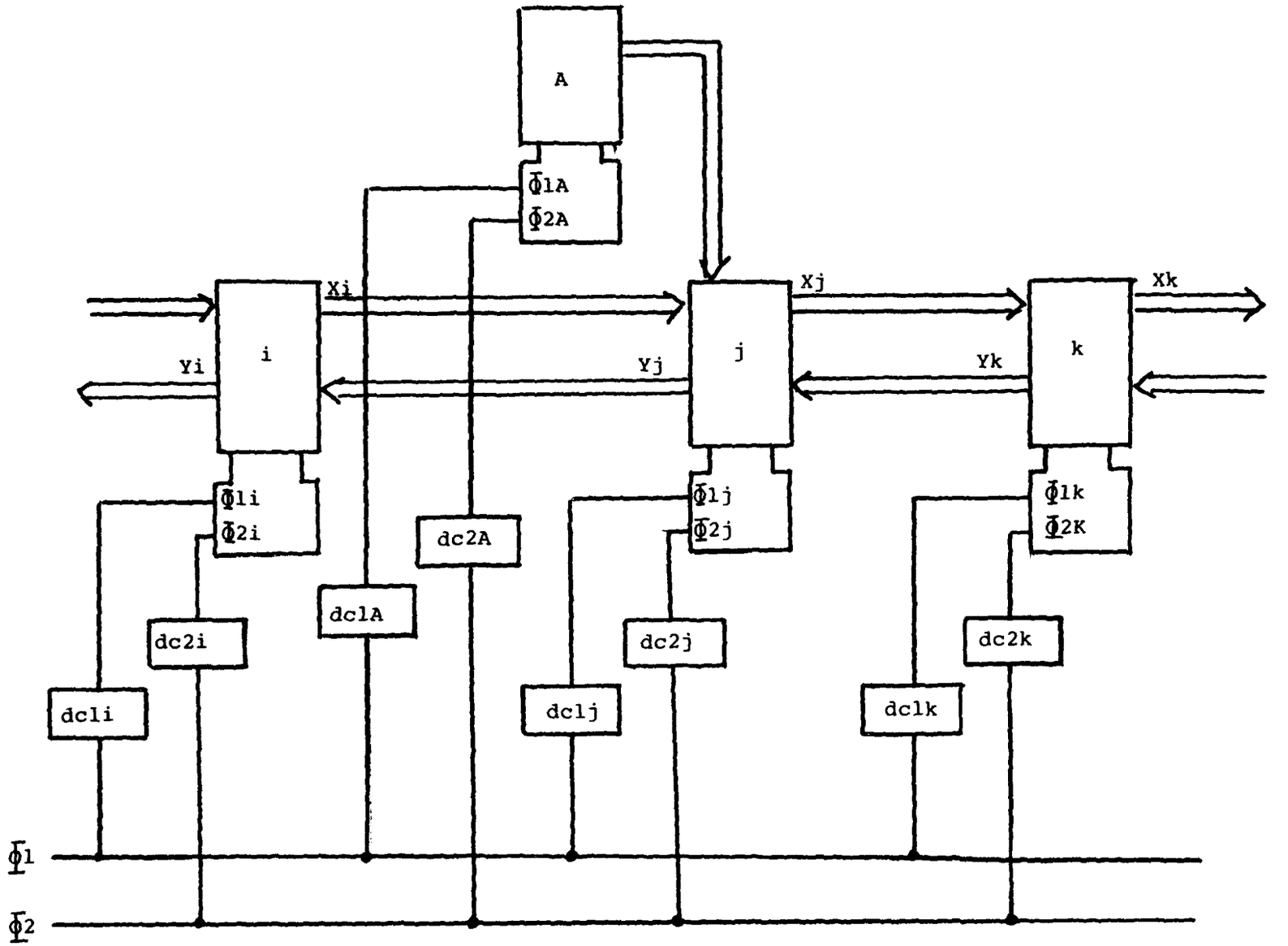


FIGURE 4

CLOCK DISTRIBUTION TO i,j,k, AND A

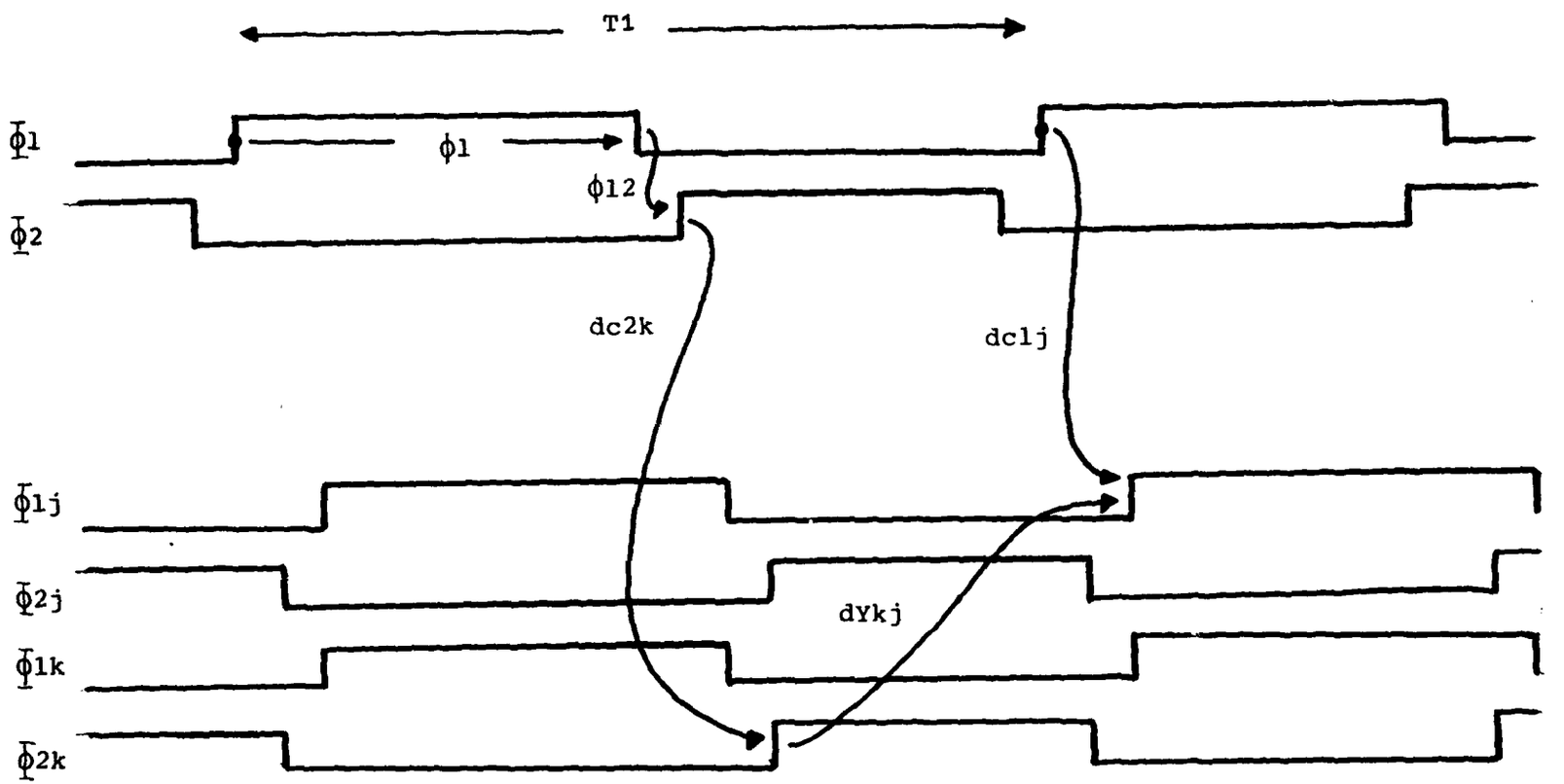


FIGURE 5

TIMING DIAGRAM FOR Yk TO j

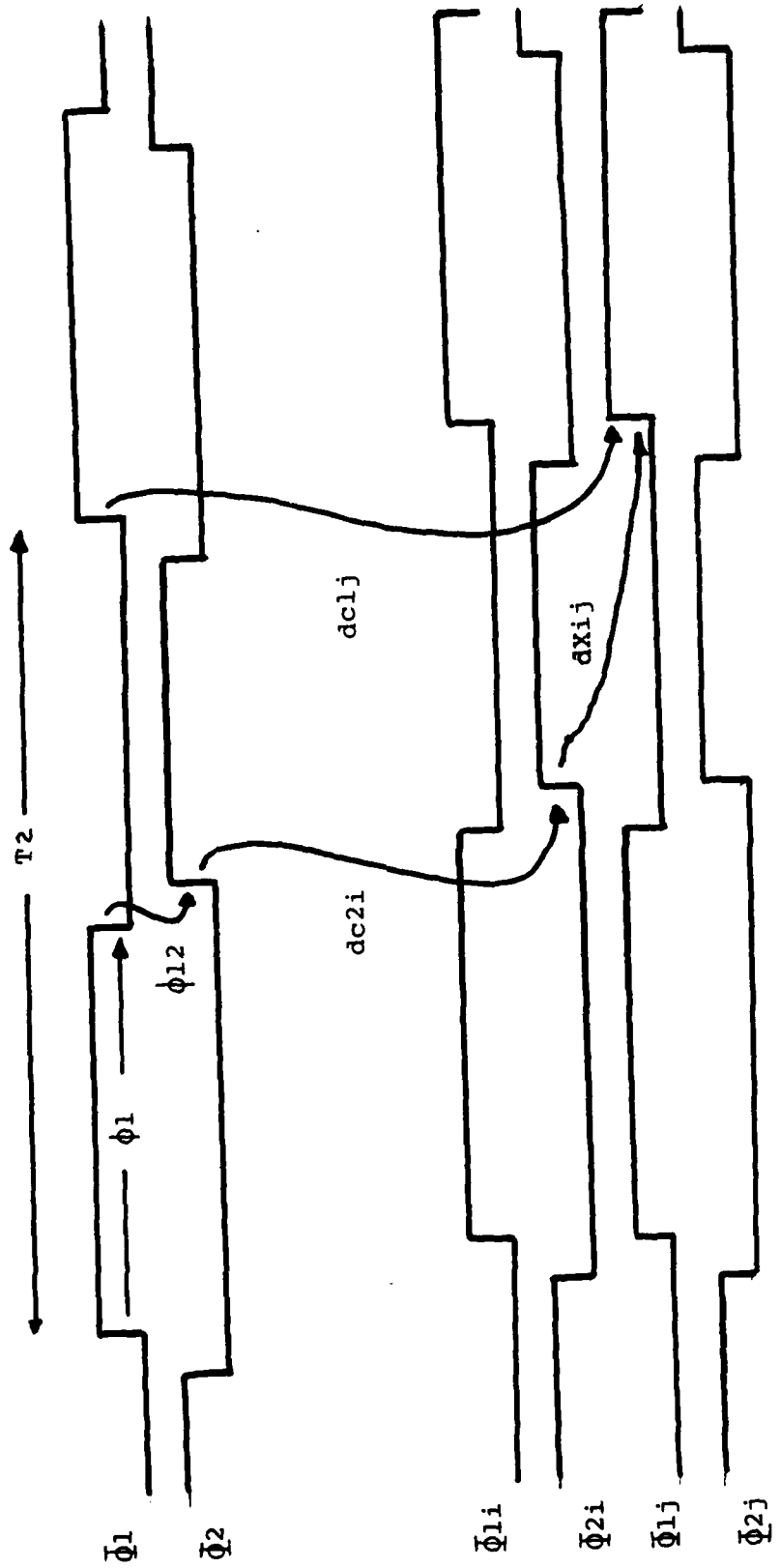


FIGURE 6

TIMING DIAGRAM FOR Xi TO j

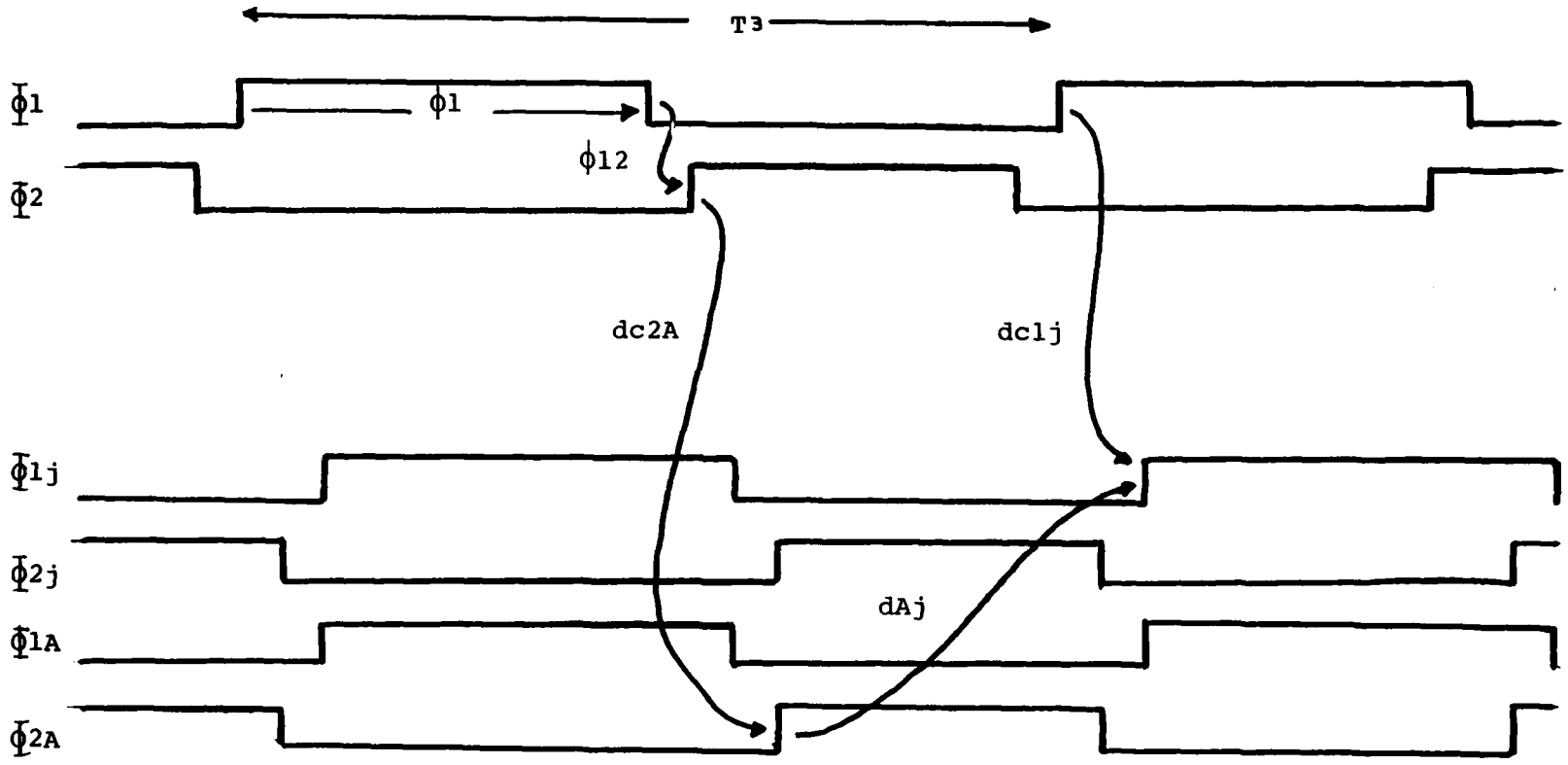


FIGURE 7

TIMING DIAGRAM FOR A TO j

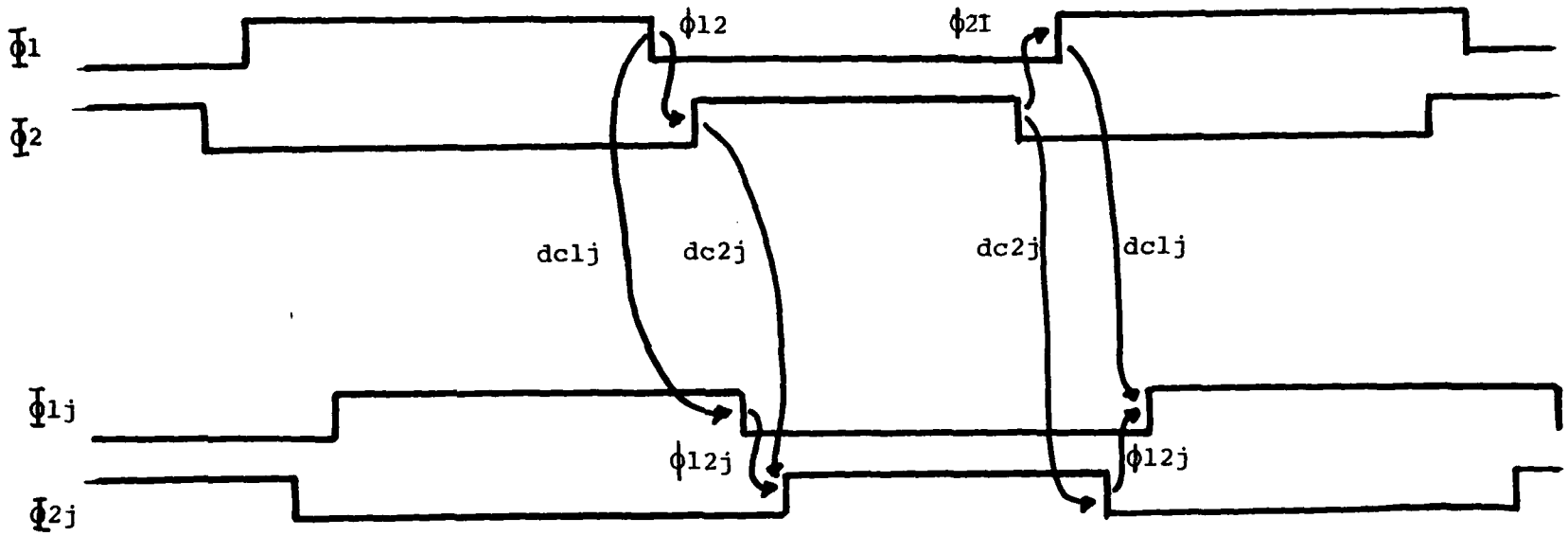


FIGURE 8

TIMING DIAGRAM FOR INTERPHASE CONSTRAINTS

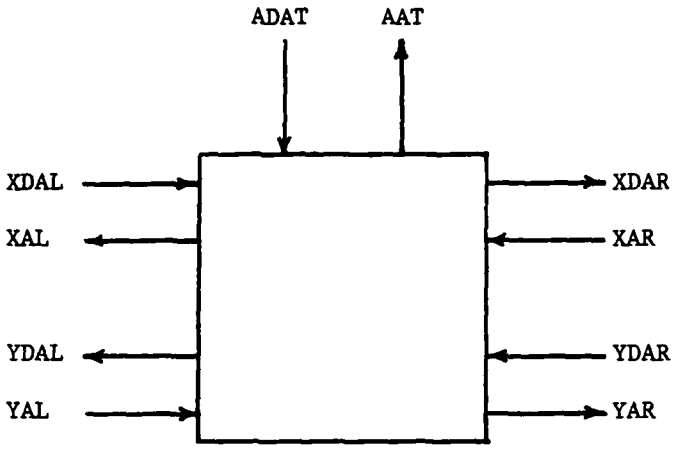


Figure 9: Control signals of the asynchronous systolic array module.

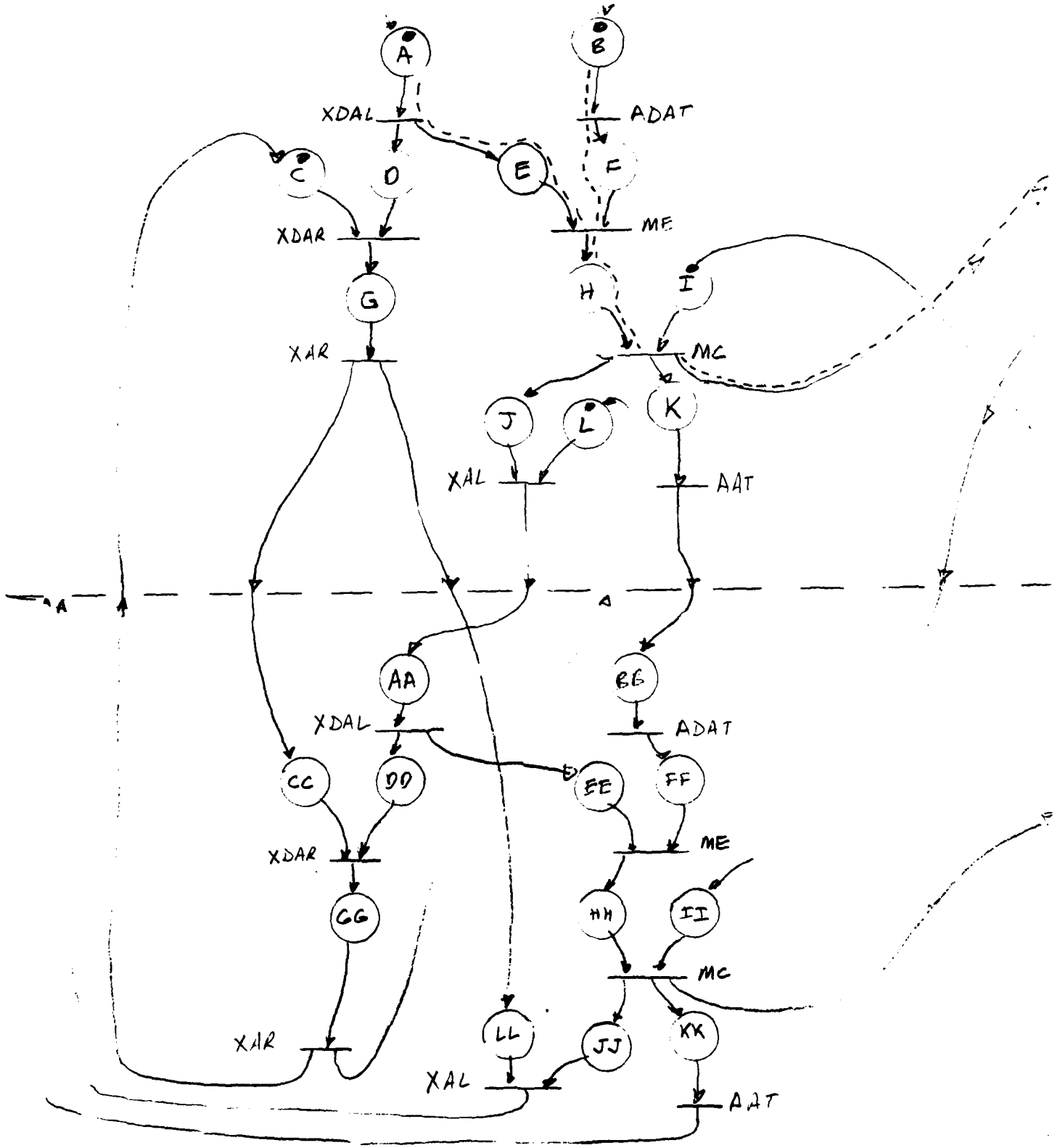


Figure 10(a): Petri net for the one dimensional, asynchronous systolic array module.

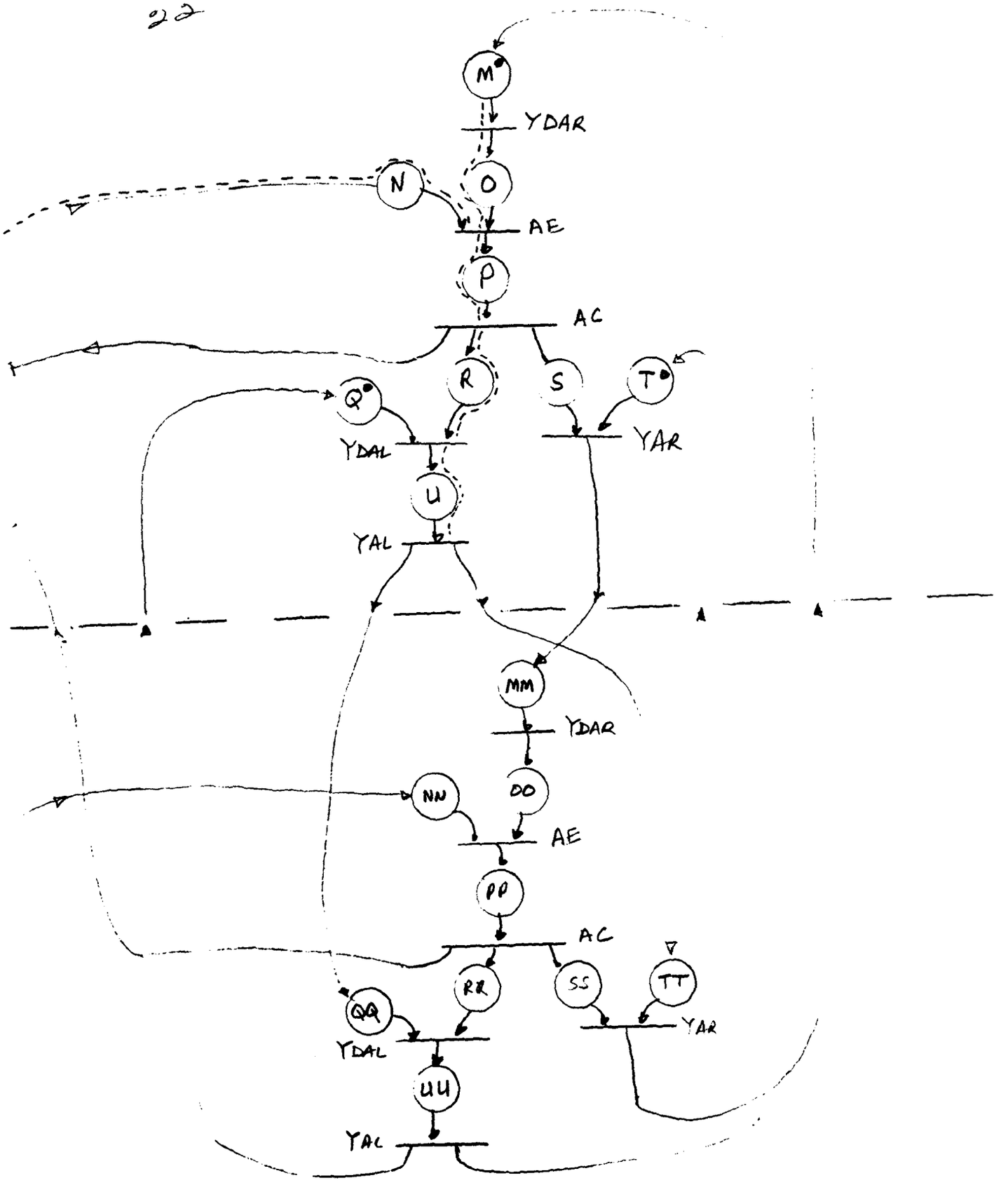
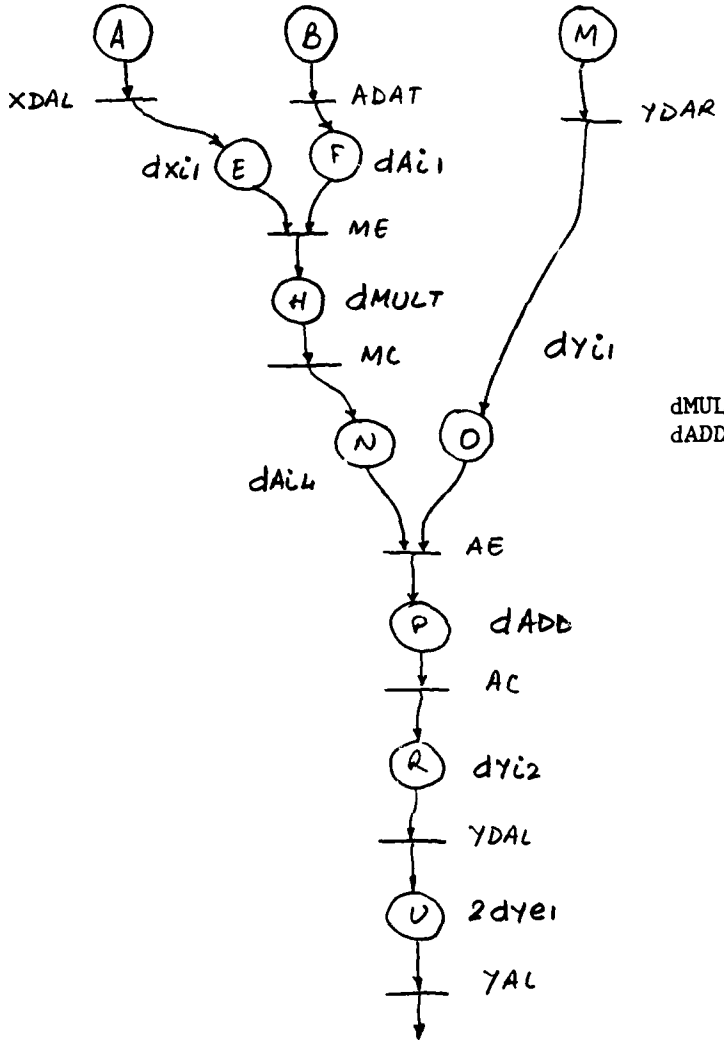


Figure 10(b): Petri net for the one dimensional, asynchronous systolic array module.



dMULT: Time for multiplication
dADD : Time for addition.

Figure 11: Path with the maximum delay in figure 10.

REINFORCED DELTA NETWORKS AND THEIR BLOCKING CHARACTERISTICS

William J. Thomas

1.0 Introduction

Computer architects are now considering large scale MIMD-type multiprocessor systems as a way of increasing computational power. A typical configuration for such a system is shown in Figure 1A, where the boxes labelled with P's represent processing elements and the boxes labeled with M's represent memory units. The interconnection network allows each processor to communicate with any of the memory units.

Since the processors in an MIMD-type processing system are unsynchronized and are following separate instructions streams, memory accesses occur in a more or less random fashion. A commonly studied problem for such a system is determining its memory bandwidth, i.e. the average number of memory units that are active at any given time. The memory bandwidth of multiprocessor systems in which the interconnection network is a crossbar has been studied extensively [1]-[3]. For large scale systems requiring connection networks with thousands of ports, however, a crossbar network appears to require too many discrete components to actually implement. A study by Franklin, et al. [4] indicates that a network having 1024 input ports and output ports and a pathwidth of 16 would require about 7,000 IC packages to implement (using 240 pin packages). Other interconnection alternatives must therefore be considered and several efforts in this direction have been made.

Barnes and Lundstrum [5] and Thanawastien and Nelson [6] have both considered the use of the shuffle-exchange type of network in these systems. This type of network is composed of a series of stages, each stage consisting of numerous 2 input x 2 output switching elements. The overall structure of

the network may be said to have a rectangular banyan topology [7]. Some specific examples of shuffle-exchange networks are the baseline network, the indirect binary n-cube, and the flip network. Such networks have been examined mainly in the context of SIMD-type processing, where their cost effectiveness has been demonstrated [8].

Shuffle-exchange type networks have several characteristics which make them attractive for large scale systems. Chiefly, these are: 1) they require a relatively moderate number of components to implement, and 2) control for establishing and clearing connections can be decentralized. This last characteristic enables highly modular designs, as well as potentially short set-up times for the connections.

The shuffle-exchange structure, however, exhibits internal blocking. That is, certain connections prohibit certain other connections from being established. This is due to the fact that paths within the network are shared. As a consequence of internal blocking, the memory bandwidth of a system using the shuffle-exchange type of network for random access will be less than that of a system using a non-blocking type of network (such as a crossbar). The shuffle-exchange type of network therefore represents a tradeoff between cost and performance.

Patel [9] has demonstrated that a better cost/performance tradeoff can be achieved if larger switching elements (e.g. 4*4 instead of 2*2) are used to construct the network. He has defined a class of networks called Delta networks and has presented performance models for their operation in the random access type of environment.

The scope of Patel's work, however, is somewhat limited since it only

considers networks which are constructed homogeneously, i.e. with only one size switching element. The goal of this paper is to extend Patel's work.

The sections of this paper can be described as follows. Section 2 presents the general concept of delta networks and reviews Patel's method for analyzing their blocking characteristics. Section 3 extends the concept of the delta network to include non-homogeneous topologies, and evaluates the blocking within these topologies using the model discussed in Section 2. Section 4 then demonstrates how these non-homogeneous delta networks can be converted into networks which are referred to as reinforced delta networks for the purpose of better fault tolerance and a reduction in blocking. The results are summarized in Section 5.

2.0 Standard Delta Networks

This section first defines the topology of delta networks and demonstrates their ability for decentralized control. Several properties of delta networks are then given, followed by a review of Patel's methodology for analyzing their blocking characteristics.

2.1 Topology

The general structure of an $N^n \times N^n$ delta network is shown in Figure 1. The network has N^n network input ports and N^n network output ports. It is composed of n stages ($S_0, S_1, \ldots, S_{n-1}$), each stage consisting of N^{n-1} N*N crossbar switches. These switches are capable of connecting any subset of their input terminals to an equinumerous subset of their output terminals in any desired one-to-one combination. The N different output terminals of a switch are distinguished for control purposes using the labels $0, 1, \ldots, N-1$.

The stages are sequentially interconnected via links. The link pattern between any two stages may be viewed abstractly as a permutation, P. Together, the set of permutations $\langle P_0, P_1, \ldots, P_{n-2} \rangle$ must satisfy the requirement that there exists a transmission path between any network input port and any network output port. An additional constraint on each link pattern is that only output terminals with identical control labels can be linked to a given switch in the next stage.

There are many sets of permutations which meet these constraints. Patel has shown that one such set consists of a single type of permutation, called an N-shuffle, which is defined as follows:

> An N-shuffle of Nr objects, denoted $\mathcal{S}_{N,r}$, corresponds to the following permutation of the Nr indices $\langle 0, 1, \ldots, (Nr-1) \rangle$:

$$\mathcal{S}_{N,r}(i) = \left(Ni + \left\lfloor \frac{i}{r} \right\rfloor \right) \bmod (Nr) \qquad 0 \leq i \leq Nr-1$$

Figure 2(a) shows an example of a 2-shuffle of 8 indices, and Figure 2(b) shows a $2^3 \times 2^3$ delta network which uses this permutation for the link patterns.

Patel has also shown that all of the permutation sets available for interconnecting the stages of an $N^n \times N^n$ delta network are essentially equivalent (in the sense that they all exhibit the same amount of blocking) when the networks are used in random access environments.

2.2 Properties

Delta networks have a number of fairly evident properties. These are

listed below without proof.

1. Delta networks are homogeneous, i.e., all of the switches used to construct a given network are the same size.

2. Delta networks are uniform, i.e., the number of transmission links between each pair of stages is constant.

3. In all delta networks, there is a unique path from each network input port to each network output port.

4. In a delta network, each link connecting two switches is a constituent part of at least 2 paths that connect different network input and output ports.

5. In a delta network, at most one link interconnects any two switches in adjacent stages.

6. Let I_k be the set of network input ports with paths to input terminal i_k of arbitrary switch m in stage S_j $(0 \leq j \leq n-1)$ of an $N^n \times N^n$ delta network, and let O_k be the set of network output ports with paths from output terminal o_k of switch m for $k = 0, 1, \ldots, N-1$, then

   (a) $I_0, I_1, \ldots, I_{N-1}$ are all disjoint, and

   $O_0, O_1, \ldots, O_{N-1}$ are all disjoint.

   (b) $|I_k| = N^j$ for all $k = 0, 1, \ldots, N-1$, and

   $|O_k| = N^{n-1-j}$ for all $k = 0, 1, \ldots, N-1$.

2.3 Decentralized Routing Scheme

We now illustrate how a connection between two ports can be established without the use of a centralized controller. We note first that a link connecting two switching elements actually represents a collection of wires. Some of these wires are for sending data while others are for sending control

signals. The control wires associated with the i'th input 'terminal' of a switch are used (possibly in conjunction with the data wires) to tell the switch which output terminal the i'th input terminal should be connected with. There are many ways of actually transmitting this information. Logically, the output terminals of an N*N switch can be distinguished using base N digits as suggested by the labeling scheme above.

When a control 'digit' arrives at a terminal of a switch, the switch connects the wires associated with that terminal to the respective wires of the indicated output terminal (if the output terminal is not already in use) Future control signals arriving at the terminal (except maybe a reset signal) are simply passed through the switch without having any effect. In this way, a switch in the following stage can be controlled. It should be clear, then, that a connection between two ports of a network can be established by sending into the network a string of control digits, each one of which is used to set the connection through a particular switch. This string of digits is called the connection 'header'.

In an $N^n \times N^n$ delta network, the connection headers all consist of n base-N digits. The topology of the delta network is such that all of the input ports use the same header for establishing a connection to a given output port. Thus, each output port has associated with it a particular header, and we call this the address of the port.

2.3 Blocking

The operation of a multiprocessor system is extremely complex. Typically, a number of assumptions have to be made when analyzing aspects of its performance.

The model that Patel uses to evaluate network bandwidth assumes that the network operates in a cyclic fashion. This operation is described as follows:

At the beginning of a cycle, connection requests are presented to the network. Using header information supplied by each request, the conflicts are resolved and the connections are established. After a time period alotted for data transfer, the connections are cleared and a new cycle begins.

Included in the model is a parameter (p) which can be used to specify the probability that each processor has of making a request at the beginning of a cycle ($0 \leq p \leq 1$). The model assumes that the requests occur independently, and that a given request is equally likely to be directed to any one of the network output ports. The bandwidth of the network is defined as the average number of connections that are established per cycle.

In this paper, we choose to measure the amount of blocking within a network in terms of the probability (P_B) that a path request is not accepted by the network rather than in terms of bandwidth (BW). These two measures are related in the following way: $P_B = 1 - BW/T$, where T represents the average number of requests submitted per cycle, (i.e., $T = N*p$). Because a similar analysis of a more general nature is given in section 4.3, we will now go directly to a discussion of Patel's results.

Figure 3 shows the probability of blocking for delta networks ranging in overall size from 4*4 to 4096*4096 (by powers of two). The values are shown also as a function of the switch sizes that can be used to construct each overall network, eg. three values are shownfor a 64*64 delta network corresponding to the three different switch sizes that can be used for its construction (2*2, 4*4, and 8*8). The values given pertain to a system in saturation (i.e., p=1).

Also shown for each network size is the blocking probability when a single switch (i.e., of size N'*N') is used to realize the entire network. Since a single switch is in itself "non-blocking", these values represent the blocking which occurs in the form of output port conflicts, and offer a lower bound for the analysis to which the other values can be compared.

The figure illustrates two major trends. First, the probability of blocking increases as the size of the network increases for a constant switch size. Secondly, for a given size network, the probability of blocking decreases as the size of the switch used to construct the network increases, which is as one would expect.

2.5 Construction Limitations

Figure 3 also illustrates the basic limitation in the construction of delta networks; that is, very few switch sizes can be used to construct a given size network. For example, 128*128 and 2048*2048 delta networks are defined for only one size switch (2*2). In the next section, we extend the concept of the delta network to include other topologies. Heretofore, we refer to the networks that Patel has considered as Standard delta networks.

3.0 Non-Homogeneous Delta Networks

The ideas discussed here on how to construct versatile, yet digit-controllable networks are not new. Similar concepts have been discussed by Lawrie in his treatment of omega networks [9]. We will limit the scope of our discussion to network sizes which are powers of two.

3.1 Topology

Consider how we might construct a 32*32 size digit-controllable network

given that the semiconductor industry has supplied us with both 2*2 and 4*4 crossbar switching modules. It is clear that we would like to use the larger module as much as possible in our design.

The same technique that underlies the construction of standard delta networks can be applied to this problem as well. That is, we first create a demultiplexer tree that fans out an input port to the required number of output ports. The tree is obtained by first factoring the total fanout that is needed into a list of sub- fanout requirements, where each of these requirements can be met by one of the available switch modules. The factorization should have as few terms as possible, thus utilizing the largest modules to their greatest extent. This stategy also yields the network with the fewest number of stages.

In our example, '32' factors into 4x4x2. The three different ways in which these terms can be ordered correspond to three different fanout trees. These are shown in Figure 4(a-c).

To construct an overall network, all that needs to be done is to superimpose numerous fanout trees until the input port requirement is met, sharing switch modules wherever possible. For all network sizes which are powers of two, this procedure will result in every switch module being used to its fullest capacity, i.e., no input or output terminals on any switch module will be left unconnected. The procedure also results in a network that is *uniform*, and has a single path between each network input and putput port.

The link pattern between each of the network stages will depend on the way in which the trees are superimposed. As with standard delta networks, all of the link patterns obtained for a given fanout tree will result in networks

which are equivalent in terms of their blocking characteristics. Two networks which use different trees, however, will most likely exhibit different amounts of blocking. Figure 4(d-f) shows completed networks for each of the trees given in Figure 4(a-c).

The procedure above for constructing an N'*N' delta network (N' a power of two) can be reduced to the following form:

1) Factor N', i.e., $N' = N_1 \times N_2 \times \ldots$,

2) Compose stage i with N'/N_i switch modules of size $N_i * N_i$,

3) Interconnect stages i and i+1 using an N_{i-1}-shuffle link pattern.

Note that this formulation provides for the construction of standard delta networks as well.

3.2 Digit-Control

The non-homogeneous delta networks described above are digit-controllable in much the same way that standard delta networks are except that all of the digits in the header will not be the same base numbers. Recall that each digit in the header is used to control switches in a different stage, and that the number base of a control digit is dependent on the size of the switches that it controls -- a base N number is used to control a switch that has N output terminals. Since the switch sizes vary from stage to stage, so will the bases of the control digits.

3.3 Blocking

Patel's analysis is general enough so that the blocking probability of non-homogeneous delta networks can be calculated directly using his equations. Figure 5 presents these results. The values shown for each network size

correspond to non-homogeneous constructions in which various switch sizes were taken to be the maximum size available. The given values are for the optimal fanout structure for each network, eg. the value given for the 32*32 network composed with only 4*4 and 2*2 switches (our continuing example) corresponds to the fanout structure of Figure 4(a). We note that these optimal values always correspond to the structure which employs the smallest size switch in the last stage.

4.0 Reinforced Delta Networks

We now propose what we call the *reinforced delta network*. The defining characteristic of these networks is that they have more than one path connecting each input port to each output port. The effects of this are twofold. First, these networks are better resistent to faults than the networks discussed earlier since numerous types of faults will still leave at least one backup path available between ports which would have elsewise been isolated. Secondly, the additional paths between ports are capable of being exploited to give these networks better blocking characteristics than their counterparts. In both of these senses, the network may be considered to be "reinforced", and hence the name.

4.1 Topology

In section 3.0, we considered how 'non-standard' delta networks could be constructed. Our method was to employ a nonhomogeneous set of switches. There is in fact no reason why we can not construct these networks using only one size switch. For example, let us replace adjacent pairs of 2*2 switches

which appear in the final stage of the 32*32 network shown in Figure 4(f) with single 4*4 switches. The resulting network is shown in Figure 6. Note that now switches in stages two and three that are connected are connected via two links. We could therefore simply leave off one of these connections, but this would clearly decrease the concurrent connection capabilities of the network since only $N'/2$ transmission links would connect stage two to stage three, and thus at most $N'/2$ connections could ever be simultaneously established (as opposed to a maximum of N' for all of the networks previously considered).

The fact that there are now two physical paths between each pair of network ports does not pose a problem for the digit-controlled routing scheme -- the device at a network input port simply chooses one or the other of the paths and specifies the appropriate routing header. In essence, each output port now simply has *two addresses*. Note that since the network is once again homogeneous, the digits composing the output port addresses are all once again base N numbers.

Although we will not discuss the fault tolerant properties of reinforced delta networks in this paper, we note that different interconnection patterns have different degrees of fault tolerance. For instance, the network shown in Figure 7 is only fault tolerant to those types of faults which affect a single link or terminal of a switch. If an entire switch fails, certain pairs of network ports will be isolated. The network shown in Figure 8, however, which uses a different interconnection scheme (i.e., a pair of 4-shuffles) is tolerant to whole switch failures for any of the switches in the middle stage.

4.3 Blocking

We now derive an approximate expression for the blocking probability of a reinforced delta network under the assumption that the network is employed in a distributed proccessing system and is used for random access purposes. Specifically, we assume the following system operating charateristics.

1) Attached to each input of the network is a source of path requests, eg. a processor, and attached to each output is a destination, eg. a memory unit. All requests for paths are source initiated.

2) Requests enter the network in a synchronized fashion, i.e. as a batch. All previous connections are cleared before a new batch is entered. The entering of a batch of requests corresponds to the beginning of a network cycle and the clearing of the old paths corresponds to the end of a cycle.

3) Any request which is blocked is ignored; i.e., the requests entered with the next batch are independent of the requests that were blocked.

4) At the beginning of each cycle, each source has generated a new request with probability p.

5) The requests generated by each source are random, independent, and uniformly distributed over all of the destinations.

6) The requests are self-routing through the network: when a request is accepted by a switching element, it is randomly assigned to one of the L links of the requested output port; when a conflict is incurred at a switching element, all possible outcomes are equally likely.

These operating assumptions provide a simple framework for evaluating the blocking characteristics of a network. The measure used to characterize blocking is the probability that a path request is not accepted by the network.

Requests can be blocked for two reasons. First, if two or more sources request the same destination, only one of these requests can be accepted, while the others are said to be blocked. Second, since the networks that are being considered use a system of shared links, it is possible for two or more

requests to require a common link even though their final destinations may be distinct. Again, only one of the requests can be accepted, while the others are considered blocked.

We have assumed a self-routing control scheme. That is, a path request from a source enters the network with enough information to set each switch that it encounters to the required switch-setting. We also assume that the network is self-arbitrating. That is, both types of conflicts discussed above are resolved within the network. Specifically, they are resolved at the point in the network where the conflict occurs. A conflict materializes when more than L requests arrive at a switch module requiring the same module output port. When such an event occurs, the module arbitrates among these. L requests are randomly selected to be passed on via the L available output links, while a blocked signal is returned to the sources of the other requests. Hence, the elimination of requests occurs at the switches in terms of local output conflicts. The probability that a request encounters a local output conflict and is not selected while transversing the network is thus the probability that it gets blocked.

Combining this technique for resolving conflicts with the assumptions that have been made about the distribution of network traffic (assumptions 4 and 5), a probabilistic model can be derived to evaluate this blocking probability. We first analyze the affects of local output contention at a switch module.

4.3.1 Switch Module Blocking Analysis

Assume that the following four traffic conditions exist at a switch module. We will later show that these conditions approximately coincide with

those at an arbitrary switching element in the network.

1) The request rate at each of the module's N inputs is p; that is, each input contains a request with probability p on any given cycle.

2) The existence of a request at an input is independent of the requests at the other inputs.

3) A request is equally likely to require any one of the module's f output ports (f = N/L).

4) The requests at each cycle are independent of the requests of the proceeding trials.

Let us label the output links of a module as follows: each output is labeled as Oij, where i indicates an output port (i = 1,2,...,f) and j indicates a particular link of that port (j = 1,2,...,L). After the conflicts for a given trial are resolved, an output link may or may not contain a request. Let Xij be a indicator random variable such that

$$Xij = \begin{cases} 1 & \text{if output link Oij contains a request} \\ 0 & \text{if output link Oij does not contain a request.} \end{cases}$$

Since the arbitration procedure is independent of time, then under the above conditions, the Xij's will have the following stationary probability distibutions:

$$P(Xij = 1) = p'ij \quad \text{and} \quad P(Xij = 0) = 1 - p'ij \quad \text{for all } i,j.$$

Hence, each Xij is a Bernoulli random variable. Further consideration of the facts that each incoming request is equally likely to require any one of the module's output ports and each link of an output port is equally likely to be selected to pass on an accepted request indicates that every output link will contain a request with equal probability. Thus

$$P(Xij = 1) = p' \quad \text{and} \quad P(Xij = 0) = 1 - p' \quad \text{for all } i,j.$$

Note, however, that the Xij's are not independent. That is, given that link Oij contains a request, this influences the probability that link Oik will contain a request.

An expression relating p' to the parameters p, N, and L has been derived in the appendix and is given in equation (1).

$$p' = \sum_{0 \le r \le N} \left\{ \left[1 - \sum_{0 \le k_i \le L-1} \left(1 - \frac{k_i}{L}\right) \binom{r}{k} \left(\frac{1}{f}\right)^{r} \left(\frac{f-1}{f}\right)^{r-k} \right] \cdot \binom{N}{k} p^{r} (1-p)^{N-r} \right\} \quad (1)$$

For the special configuration where L=1, the expression for p' reduces (see appendix) to

$$p' = 1 - (1 - p/N)^{N} \qquad \text{for } L=1 \qquad (2)$$

This expression was first obtained by Strecker [10] as an approximation to the memory access rate of a multiprocessor system in which N processors are connected to N memories via an N*N non-blocking switch. This is also the expression on which Patel bases his blocking analysis of delta networks. Recall that all of the modules in a delta network are configured as full N*N switches.

Let us define the probability of acceptance for a module, Pa, to be the ratio of the expected number of requests that are accepted to the expected number of requests that arrive. Then

$$Pa = N \cdot p' / N \cdot p = p'/p \qquad (3)$$

Figure 9 shows the probability of acceptance of various modular configuations and sizes for p equal to 1.

From the above analysis, we may conclude that under the given traffic

conditions, the resolution of conflicts in the prescribed manner will yield a request rate at an output link which is uniquely determined by the configuration of the module and the request rate at the input links; and that the request rate at each output link will be the same, although not independent. This rate is given by expression (1).

4.3.2 Network Blocking Analysis

In order for a request to be accepted by the network, it must be accepted at each switch module that it encounters. In transversing the network, a request will encounter n switch modules, one at each stage. If we can show that the traffic at each module encountered adheres to the conditions specified in the last section, we can then apply equation (2) to determine the probability that a request survives each encounter. The probability that it survives all of the encounters is the probability that it is accepted by the network. Since the paths between every input-output pair are equivalent, this probability characterizes the acceptance probability for the entire network.

Alternatively, we can define the probability of acceptance for the network in the same fashion that we defined the probability of acceptance for a module; i.e., let the acceptance probability, PA, be the ratio of the expected number of requests which are accepted by the network to the expected number of requests which arrive at the network. Since the network has an equal number of inputs and outputs, this again reduces to the ratio of the average request rate at the output links, po, to the average request rate at the input links, pi.

$$PA = po/pi \qquad (4)$$

Under the assumption that each source has a probability p of generating a

request on a given cycle, then p is the average input rate of requests. By recursively applying the results of equation (1) at each stage of the network, we can obtain the average output rate of requests.

Let us now show that the traffic conditions under which equation (1) was obtained approximately correspond to those at an arbitrary switch module.

Consider first a module in stage one. Each of its inputs is directly linked to a source. Therefore, by assumption 4 of the system operating characteristics, the request rate at every input is p. By assumption 5, these requests arrive independently, and by assumption 3, the requests are independent from cycle to cycle.

Recall from section 2.2 (property 6) that the topology of the networks under consideration is such that each output port of a switch module leads to a distinct set of destinations, each set being of the same cardinality. Therefore, given that a generated request is equally likely to be directed to any of the destinations (assumption 5), then the request is equally likely to require any one of the module's output ports. Hence, the four assumed traffic conditions hold true for a module in stage one.

Consider now the modules in stage two. Since the incoming requests to the network are independent from cycle to cycle, and the network is cleared between cycles (assumption 4), it is clear that the requests at all stage two modules are independent of the requests of preceeding cycles. By a similar arguement to the one used for the stage one modules, it is also clear that an incoming request is equally likely to require any one of a module's f output ports.

All of the inputs to a stage two module are directly linked to modules in

stage one. Since all of the stage one modules are configured alike (a topological rule), and each has the same input rate, then all of the links between stages one and two will contain a request with the same probability (determined by equation 1). Thus all of the inputs to a module in stage two will have the same request rate. However, if the modules in stage one are configured with L > 1, then all of the incoming requests will not be independent. Specifically, only those requests arriving from different modules will be independent because these requests will have evolved along independent paths from distinct sets of sources. There will be some correlation between the requests arriving from a single module via multiple link connections. This correlation, however, is small and we will assume that the arrival of requests on these links is independent as well. Therefore, equation (1) can be used to approximate the result of the conflict resolutions at stage two. Similar arguements can be applied at the remaining stages. It is clear then that under the assumption of independent requests, equation (1) can be applied to every module in the network.

4.3.2 Results

Figure 10 illustrates the reduction in blocking of reinforced delta networks with respect to their non-homogeous counterparts. This reduction is most significant for small networks.

5.0 Concluding Remarks

This paper has presented the concept of the reinforced delta network. Like the standard delta network, reinforced delta networks have moderate component counts and are able to be decentrally controlled. Moreover, they are versatile in their constructions and have slightly better bandwidths than standard delta networks.

Appendix 1

Derivation of equations [1] and [2]

The general configuration of a switch module is shown in Figure A.1. The traffic conditions for the analysis are:

1) Each input contains a request with probability p,

2) Requests at the inputs occur independently,

3) A particular request is equally likely to require any one of the f output ports

Requests are accepted and assigned to specific output links as follows. Let k_i be the number of requests that require output port i, $(k_i = 0, 1, \ldots, N)$ Then,

1) if $k_i <= L$, all k_i requests are accepted;

2) if $k_i > L$, L of the k_i requests -- chosen randomly -- are accepted

Accepted requests are then randomly assigned to output links of the requested port.

We are to determine p', the probability that an arbitrary output link contains a request. Let Xij be defined as in section 3.1, i.e., Xij is an indicator random variable such that:

$$Xij = \begin{cases} 1 & \text{when output link Oij contains a request} \\ 0 & \text{when output link Oij does not contain a request} \end{cases} \qquad [A1]$$

Then, $p'ij = P(Xij = 1)$.

Let Er be the event that r requests arrive at the module, $(r = 0, 1, \ldots, N)$ Then, by the law of total probability,

$$p'ij = \sum_{0<r<N} P(Xij = 1 \mid Er)\ P(Er). \qquad [A2]$$

For the given traffic conditions,

$$P(Er) = \binom{N}{r} p^r (1-p)^{N-r} \qquad [A3]$$

Consider now P(Xij = 1 | Er), the probability that output link Oij contains a request given that r requests arrived. Let Ek_i be the event that k_i requests require output port i, (i = 1, 2, ..., f). Then, once again by the law of total probability,

$$P(Xij = 1 \mid Er) = \sum_{0<k<r} P(Xij = 1 \mid Ek_i \cap Er)\ P(Ek_i \mid Er). \qquad [A4]$$

$P(Ek_i \mid Er)$ is the probability that k_i of the r requests require port i. Since each request has probability 1/f of requiring port i, then

$$P(Ek_i \mid Er) = \binom{r}{k}\left(\frac{1}{f}\right)^{k}\left(\frac{f-1}{f}\right)^{r-k} \qquad [A5]$$

Given that k_i requests require port i, the probability that arbitrary output link Oij is assigned a request, $P(Xij=1 \mid Ek_i \cap Er)$, is:

$$\begin{cases} k_i/L & \text{for } k_i < L \\ 1 & \text{for } k_i > L \end{cases} \qquad [A6]$$

Hence,

$$P(Xij = 1 \mid Er) = \sum_{0<k_i<L-1} (k_i/L)\ P(Ek_i \mid Er) + \sum_{L<k_i<r} (1)\ P(Ek_i \mid Er)$$

$$= \sum_{0<k_i<L-1} (k_i/L)\ P(Ek_i \mid Er) + \left[\sum_{0<k_i<r}\binom{r}{k}\left(\frac{1}{f}\right)^{k}\left(\frac{f-1}{f}\right)^{r-k} - \sum_{0<k_i<L-1}\binom{r}{k}\left(\frac{1}{f}\right)^{k}\left(\frac{f-1}{f}\right)^{r-k}\right] \qquad [A8]$$

$$= 1 - \sum_{0<k_i<L-1} (1 - k_i/L)\cdot\binom{r}{k}\left(\frac{1}{f}\right)^{k}\left(\frac{f-1}{f}\right)^{r-k} \qquad [A9]$$

Substituting [A9] and [A3] into [A2], then

$$p'ij = \sum_{0<r<N}\left\{\left[1 - \sum_{0<k_i<L-1}\left(1-\frac{k_i}{L}\right)\binom{r}{k}\left(\frac{1}{f}\right)^{r}\left(\frac{f-1}{f}\right)^{r-k}\right]\cdot\binom{N}{r}p^{r}(1-p)^{N-r}\right\} \qquad [A10]$$

To show that the standard configuration of L=1 is simply just a special case of this general analysis, we can substitute L=1 into [A10]. This expression then reduces as follows:

$$p'ij\Big|_{L=1} = \sum_{0<r<N}\left\{\left[1 - \sum_{k_i=0}\left(1-\frac{k_i}{L}\right)\binom{r}{k}\left(\frac{1}{f}\right)^{r}\left(\frac{f-1}{f}\right)^{r-k}\right]\cdot\binom{N}{r}p^{r}(1-p)^{N-r}\right\} \qquad [A11]$$

$$= \sum_{0<r<N}\left\{\left[1-\left(1-\frac{1}{N}\right)^{r}\right]\binom{N}{r}p^{r}(1-p)^{N-r}\right\} \qquad [A12]$$

$$= \sum_{0<r<N} \binom{N}{r} p^r (1-p)^{N-r} - \sum_{0<r<N} \left(1 - \frac{1}{N}\right)^r \binom{N}{r} p^r (1-p)^{N-r} \quad [A13]$$

$$= 1 - \sum_{0<r<N} \left(p - \frac{p}{N}\right)^N (1-p)^{N-r} \quad [A14]$$

which by the binomial theorem,

$$= 1 - \left[\left(p - \frac{p}{N}\right) + (1 - p)\right]^N \quad [A15]$$

$$= 1 - \left(1 - \frac{p}{N}\right)^N ; \quad [A16]$$

which is what we got in section 2.4 during the analysis of standard delta networks.

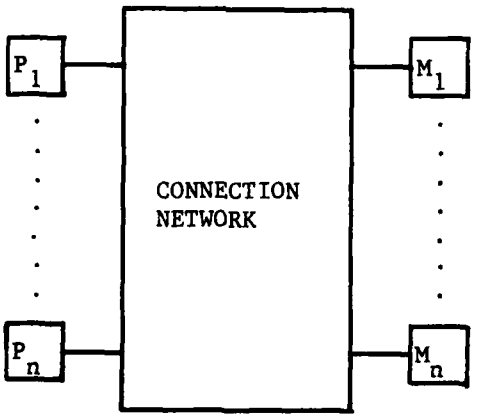


FIGURE 1A: A TYPICAL CONFIGURATION OF A MULTIPROCESSOR SYSTEM.

REFERENCES

[1] B. Rau, "Interleaved Memory Bandwidth in a Model of a Multiprocessor Computer System," IEEE Trans. Comput., vol. C-28, pp. 678-681, September 1979

[2] D. Y. Chang, D. J. Kuck, and D. H. Lawrie, "On the Effective Bandwidth of Parallel Memories," IEEE Trans. Comput., vol. C-24, pp. 480-490, May 1977

[3] D. P. Bhandarkar, "Analysis of Memory Interference in Multiprocessors," IEEE Trans. Comput., vol C-24, pp. 897-908, September 1975

[4] M. A. Franklin and D. F. Wann, "Pin limitations and VLSI interconnection networks," in Proc. 1981 Int. Conf. on Parallel Processing, 1981, pp. 253-258.

[5] G. H. Barnes and S. F. Lundstrom, "Design and Validation of a Connection Network for Many-Processor Multiprocessor Systems," Computer, vol. 14, no. 12, pp. 31-41, December 1981.

[6] S. Thanawastien and V. Nelson, "Interference Analysis of Shuffle-Exchange Networks," IEEE Trans. Comput., vol. C-30, pp. 545-556, August 1981.

[7] D. H. Lawrie, "Access and alignment of data in an array processor," IEEE Trans. Comput., vol. C-24, pp. 1145-1155, Dec. 1975

[8] J. H. Patel, "Performance of processor-memory interconnection networks for multiprocessors," IEEE Trans. Comput., vol. C-30, pp. 771-780, Oct. 1981.

[8] L. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," in Proc. 1st Annu. Symp. Comput. Arch., 1973, pp. 21-28.

[9] D. H. Lawrie, "Memory-Processor Connection Networks," Univ. of Ill., Urbana-Champaign, Dep. Comput. Sci., Rep. 557, Feb 1973

[10] W. D. Steccker, "AnAnalysis of the Instruction Execution Rate in Certain Computer Structures," Ph. D. dissertation, Dep. Comput. Sci., Carnegie-Mellon Univ., Pittsburg, Pa., 1970.

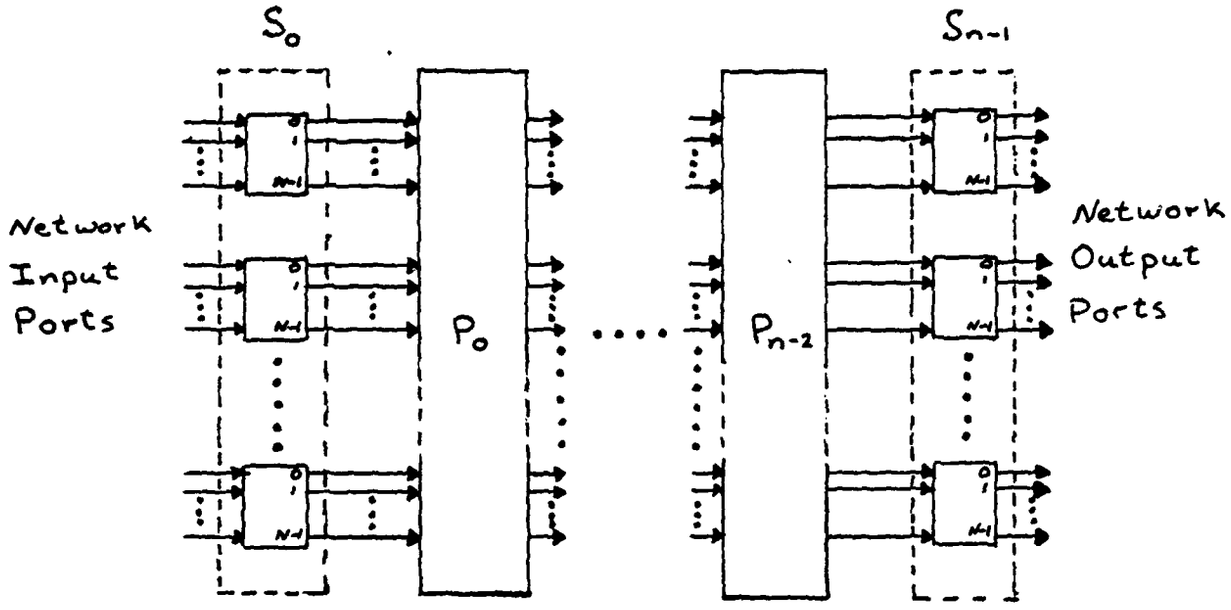


FIGURE 1: GENERAL TOPOLOGY OF AN N^n x N^n DELTA NETWORK

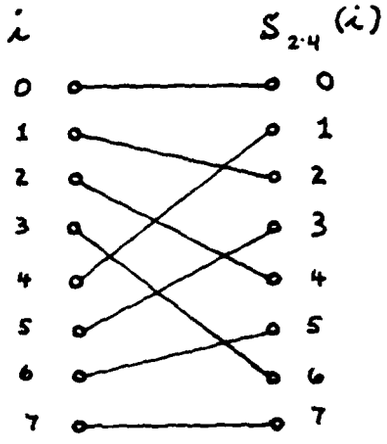


FIGURE 2(a): A 2-SHUFFLE OF 8 INDICES

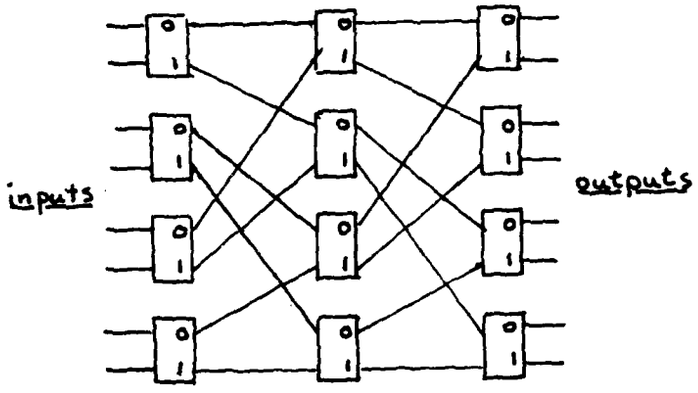


FIGURE 2(b): A 2^3 x 2^3 DELTA NETWORK

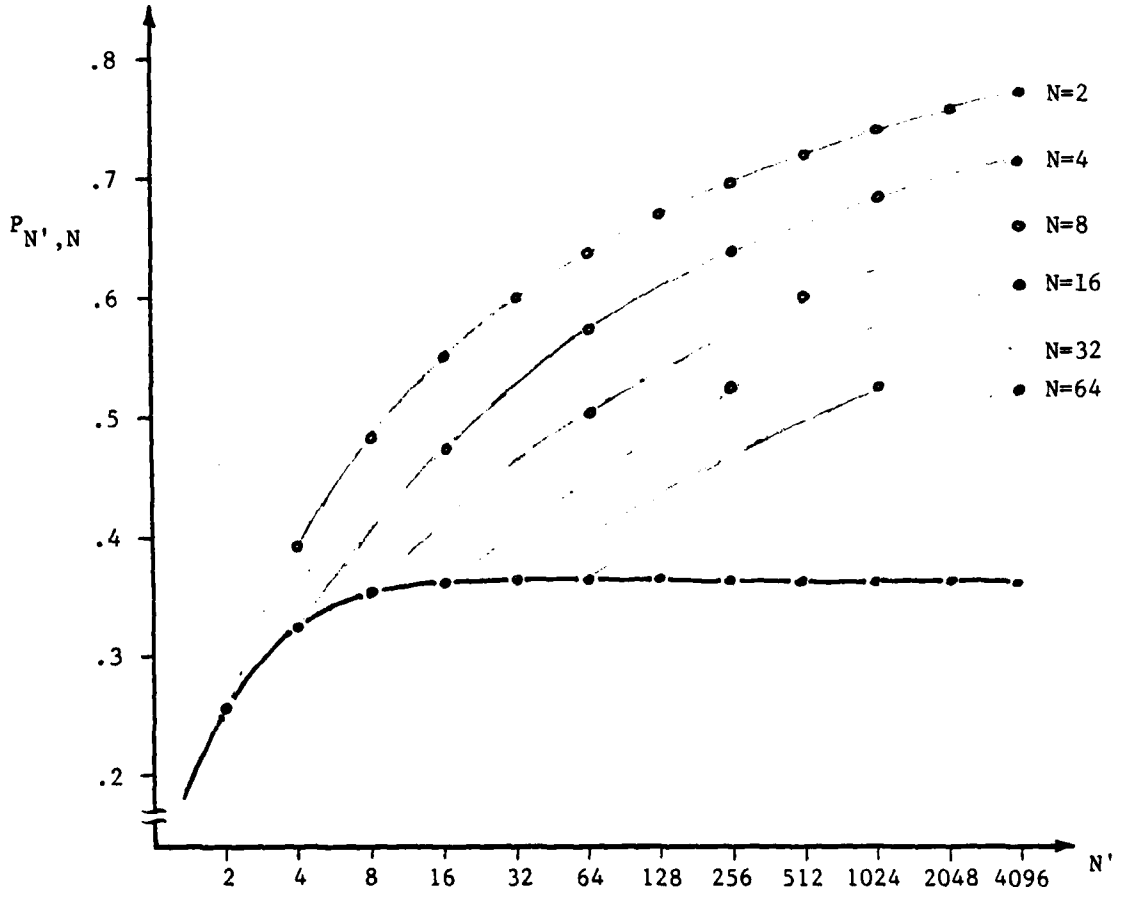


FIGURE 3: PROBABILITY OF BLOCKING $P_{N',N}$ VERSUS NETWORK SIZE N' FOR STANDARD DELTA NETWORKS

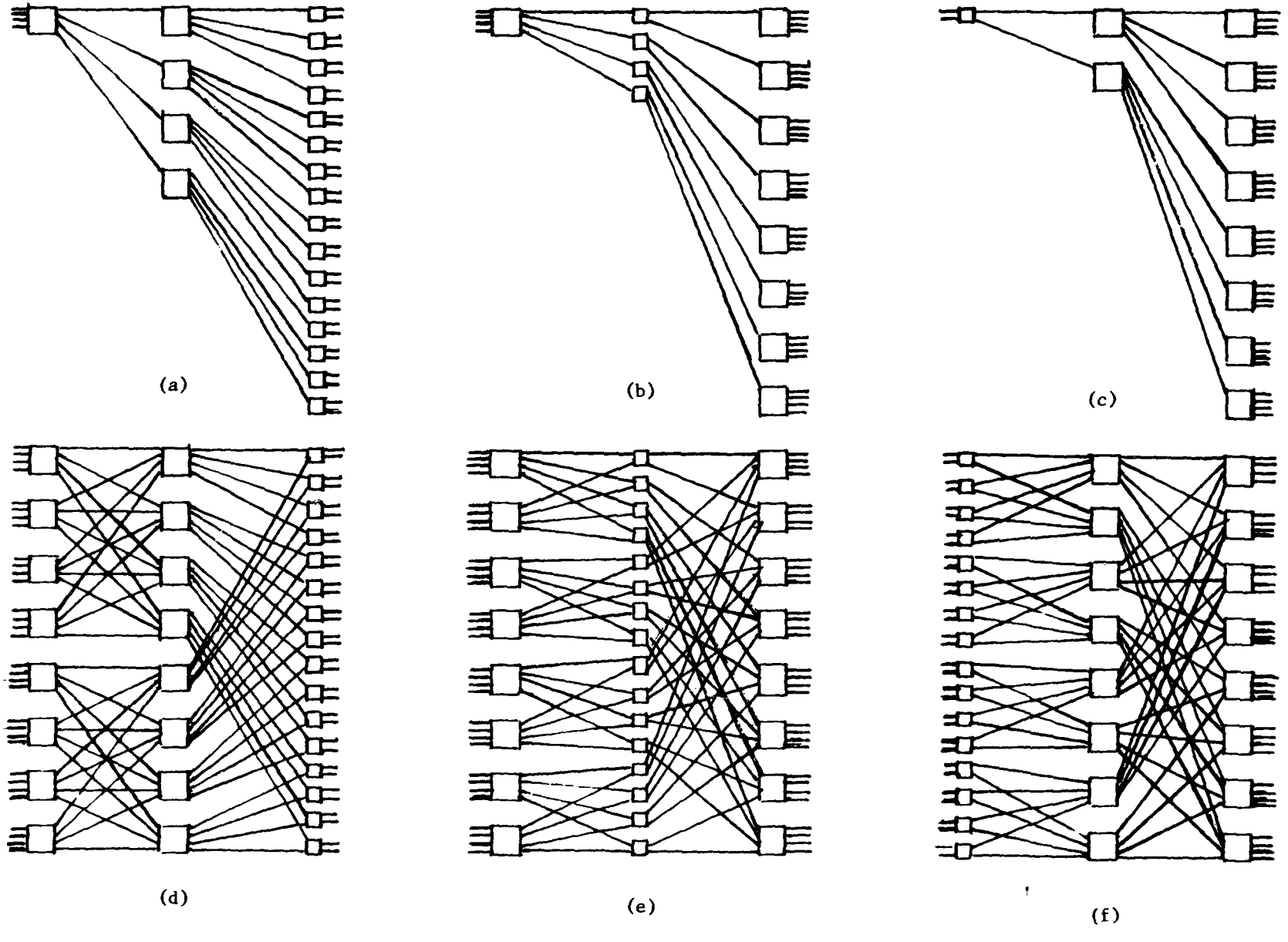


FIGURE 4: (a) - (c) THREE DIFFERENT 1 x 32 DEMULTIPLEXER (FANOUT) TREES.
(d) - (f) COMPLETED 32*32 NETWORKS FOR EACH OF THE TREES.

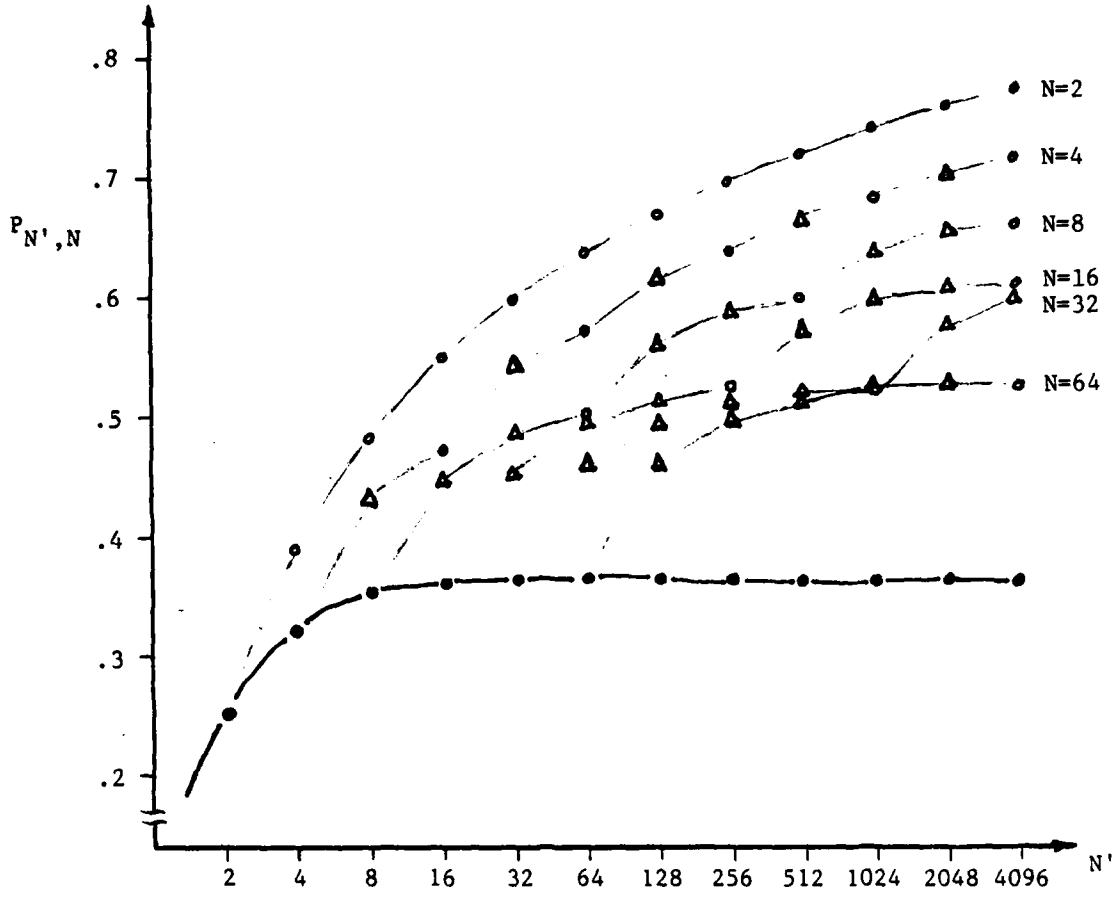


FIGURE 5: PROBABILITY OF BLOCKING $P_{N',N}$ VERSUS NETWORK SIZE N'

FOR BOTH STANDARD AND NON-HOMOGENEOUS DELTA NETWORKS

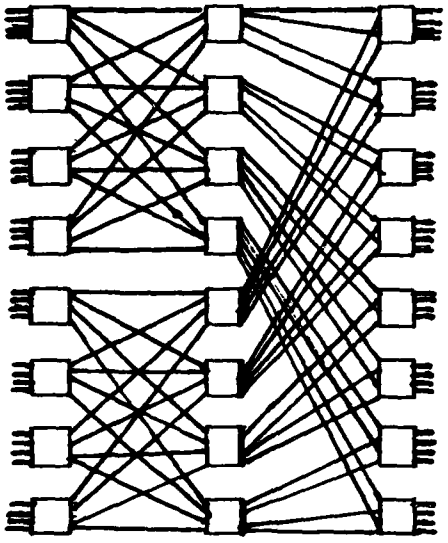


FIGURE 6: 32*32 NETWORK OF FIGURE 4(f) WITH 2*2 SWITCHES REPLACED WITH 4*4 SWITCHES

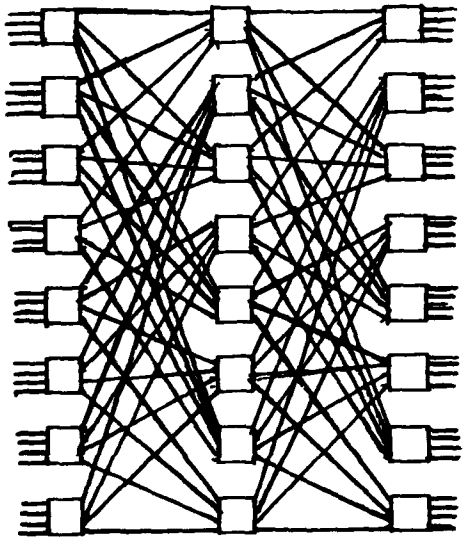


FIGURE 7: 32*32 NETWORK WITH DIFFERENT INTERCONNECTION SCHEME (4-SHUFFLES)

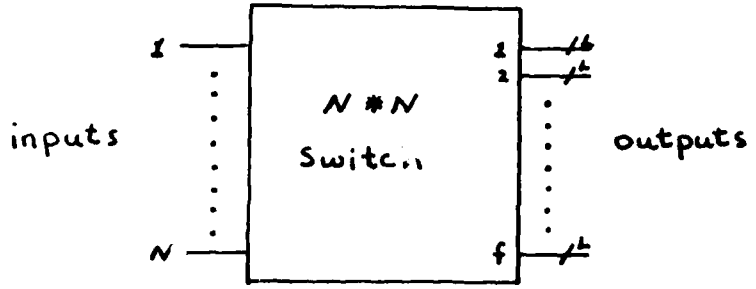


FIGURE 8: MULTIPLE-LINK CONFIGURATION OF AN N*N SWITCH MODULE

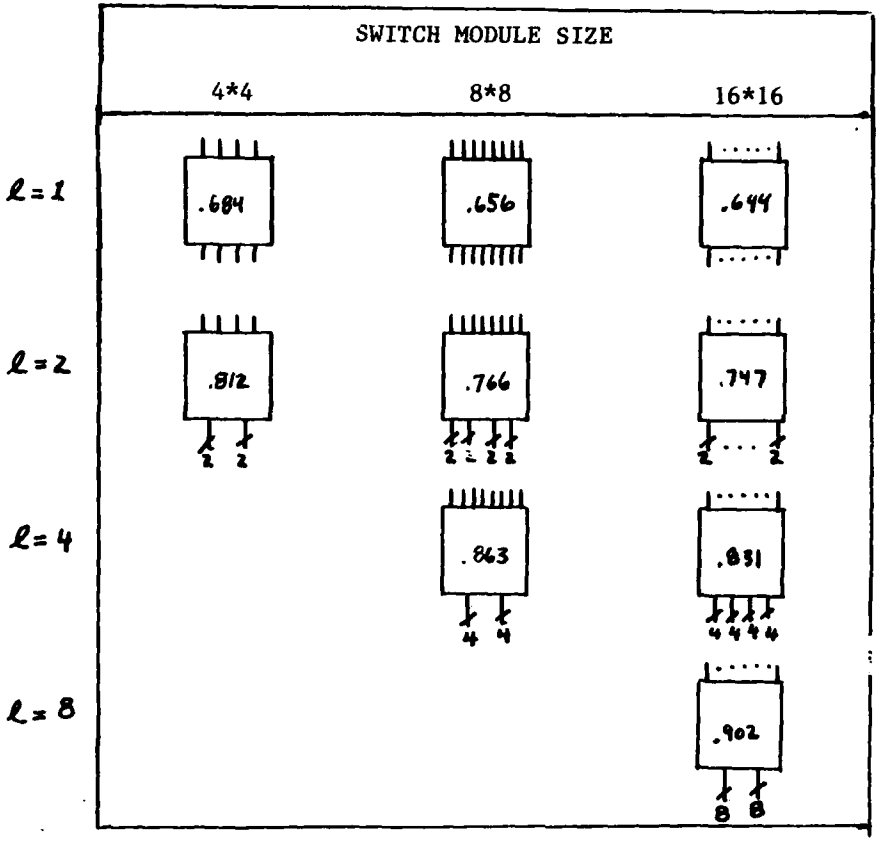


FIGURE 9: ACCEPTANCE PROBABILITIES FOR VARIOUS SIZE SWITCH MODULES IN VARIOUS CONFIGURATIONS

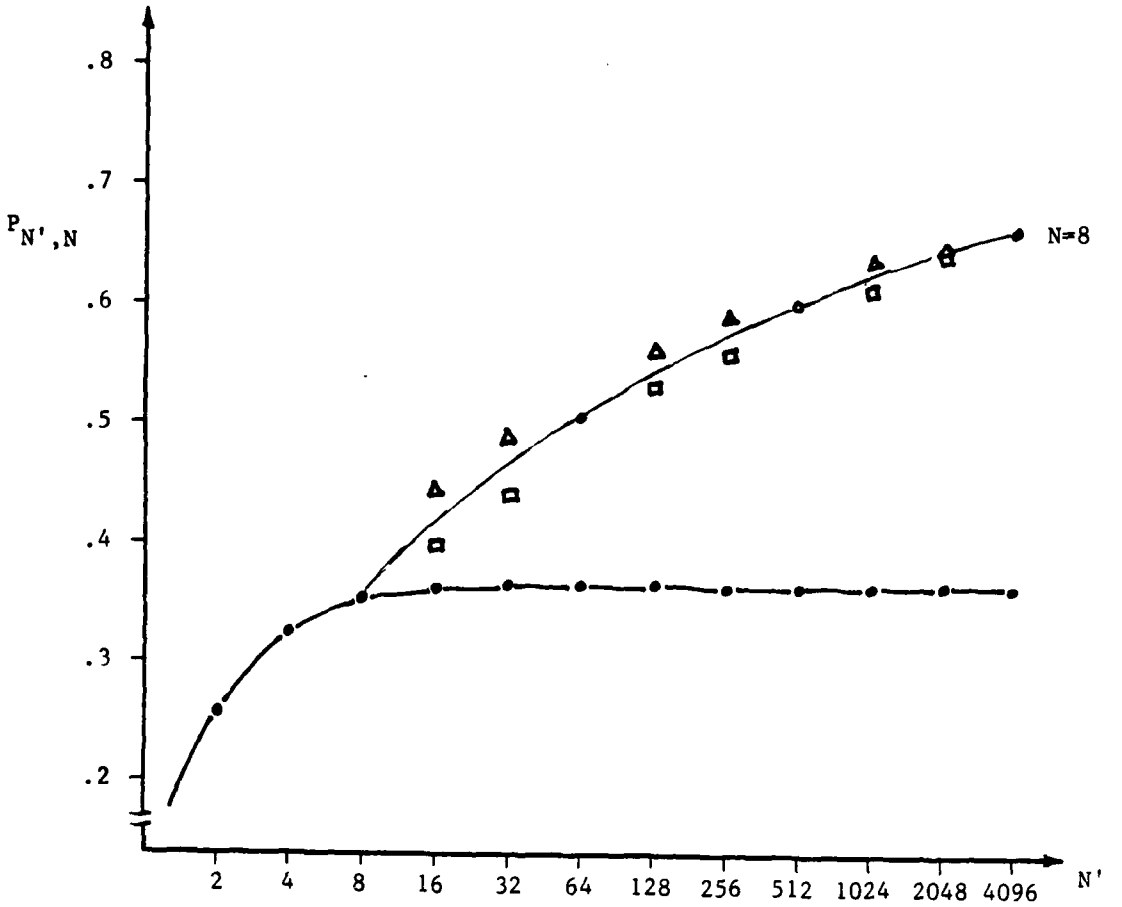


FIGURE 10: A COMPARISON OF BLOCKING PROBABILITIES BETWEEN REINFORCED (□) AND NON-HOMOGENEOUS (△) DELTA NETWORKS

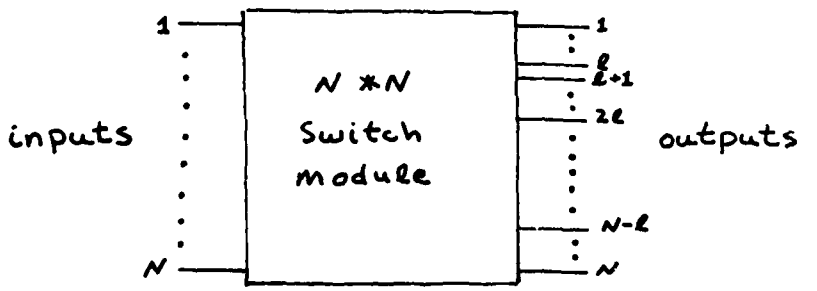


FIGURE A1: MULTIPLE-LINK CONFIGURATION OF AN N*N SWITCH MODULE

END

DATE
FILMED

5 -83

DTIC