MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Technical Report No. 1445 September, 1993

Robust, High-Speed Network Design for Large-Scale Multiprocessing

André DeHon
andre@ai.mit.edu

Abstract: Large-scale multiprocessing remains an elusive, yet promising paradigm for achieving high-performance computation. As machine size scales upward, there are two important aspects of multiprocessor systems which will generally get worse rather than better: (1) interprocessor communication latency will increase and (2) the probability that some component in the system will fail will increase. Both of these problems can prevent us from realizing the potential benefits of large-scale multiprocessing. In this document we consider the problem of designing networks which simultaneously minimize communication latency while maximizing fault tolerance for large-scale multiprocessors. Using a synergy of techniques including connection topologies, routing protocols, signalling techniques, and packaging technologies we assemble integrated, system-level solutions to this network design problem. In particular, we recommend the use of multipath, multistage networks, simple, source-responsible routing protocols, stochastic fault-avoidance, dense three-dimensional packaging, low-voltage, series-terminated transmission line signalling, and scan based diagnostic and reconfiguration.

Acknowledgements: This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's Artificial Intelligence Research is provided in part by the Advanced Research Projects Agency under Office of Naval Research contract N00014-91-J-1698. This material is based upon work supported under a National Science Foundation Graduate Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

Acknowledgments

The ideas presented herein are the collaborative product of the MIT Transit Project under the direction of Dr. Thomas Knight, Jr. The viewpoint is my own and, as such, the description reflects my perspective and biases on the subject matter. Nonetheless, the whole picture presented here was shaped by the effort of the group and would definitely have been less complete without such a collective effort.

At the expense of slighting some who's influences and efforts I may not, yet, fully appreciated, I feel it only appropriate to acknowledge many by name. Knight provided many of the seed ideas which have formed the basis of our works. Preliminary work with the MIT Connection Machine Project and with the Symbolics VLSI group raised many of the issues and ideas which the Transit Project inherited. Collaboration with Tom Leighton, Bruce Maggs, and Charles Leiserson on interconnection topologies has also been instrumental in developing many of the ideas presented here.

Knight, of course, provided the project with the oversight and encouragement to make this kind of effort possible. Henry Minsky has been a key player in the Transit effort since its inception. Minsky has helped shape ideas on topology, routing, and packaging. Without his VLSI efforts, RN1 would not have existed and all of us would have suffered greatly on other VLSI projects. Along with Knight, Alex Ishii and Thomas Simon have been invaluable in the effort to understand and develop high-performance signalling strategies. While Simon has a very different viewpoint than myself on most issues, the alternative perspective has generally been healthy and produced greater understanding. The packaging effort would remain arrested in the conceptual stage without the efforts of Fred Drenckhahn. Frederic Chong and Eran Egozy provided the cornerstone for our network organization evaluations and developments. Their efforts took vague notions and turned them into quantifiable entities which allowed us greater insight. Eran Egozy spearheaded the original METRO effort which also included Minsky, Simon, Samuel Peretz, and Matthew Becker. Simon helped shape the MBTA effort. The MBTA effort has also benefitted from contributions by Minsky, Timothy Kutscha, David Warren, and Ian Eslick. Patrick Sobalvarro, Michael Bolotski, Neil Brock, and Saed Younis have also offered notable help during the course of this effort.

This work has also been shaped by numerous discussions with people in "rival" research groups here at MIT. Notably, discussions with Anant Agarwal, John Kubiatowicz, David Chaiken, and Kirk Johnson in the MIT Alewife Project have been quite useful. William Dally, Michael Noakes, Ellen Spertus, Larry Dennison, and Deborah Wallach of the Concurrent VLSI Architecture group have all provided many useful comments and feedback. Noakes and George Andy Boughton have provided much valuable technical support during the life of this project. I am also indebted to William Weihl, Gregory Papadopoulos, and Donald Troxel for their direction.

Our productivity during this effort has been enhanced by the availability of software in source form. This has allowed us to customize existing tools to suit our novel needs and correct deficiencies. Notably, we have benefitted from software from the GNU Project, the Berkeley CAD group, and the Active Messages group at Berkeley.

We have also been fortunate that several companies offer generous university programs through which they have made tools available to us. Intel has been most helpful in providing a rich suite of tools for working with the 80960 microprocessor series in source form. Actel provided sufficient discounts on their FPGA tools to make their FPGAs useful for our prototyping efforts. Exemplar has also provided generous discounts on their synthesis tools. Cadence provided us with Verilog-XL for simulation. Logic Modeling Corporation has made their library of models for Verilog available. MetaSoftware provided us with HSpice. Many of our efforts could not have succeeded without the support of these industrial strength tools.

This research is supported in part by the Defense Advanced Research Projects Agency under contracts N00014-87-K-0825 and N00014-91-J-1698. This material is based upon work supported under a National Science Foundation Graduate Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

Fabrication of printed-circuit boards and integrated circuits was provided by the MOSIS Project. We are greatly indebted to the careful handling and attention they have given to our projects. Particularly we are grateful for the help provided by Terry Dosek, Wes Hansford, Sam DelaTorre, and Sam Reynolds.

Contents

List of Figures

List of Tables

Part I

Introduction and Background

1. Introduction

The high capabilities and low costs of modern microprocessors have made it attractive from both economic and performance viewpoints to design and construct large-scale multiprocessors based on commodity processor technologies. Nonetheless, many challenges remain to effectively realize the potential performance promised by large-scale multiprocessing on a wide-range of applications. One key challenge is to provide sufficient inter-processor communication performance to allow efficient multiprocessor operation – and to provide such performance at a reasonable cost.

In order for processors to work effectively together in a computation, they must be able to communicate data with each other in a timely fashion. The exact nature and role of communication varies with the particular programming model, but the need is pervasive. Virtually all paradigms for parallel processing depend critically on low communication latency to effectively exploit parallel execution to reduce total execution time. Communication latency is a critical determinant of the amount of exploitable parallelism and the cost of synchronization. For shared-memory algorithms, latency affects the speed of cache-replacement and coherency operations. In message-passing programs, latency affects the delay between message transmission and reception. In dataflow programs, latency determines the delay between the computation of a data value and the time when the value can actually be used. Data parallel operations are limited by the rate at which processors can obtain access to the data on which they need to operate.

Multithreaded ([Smi78] [Jor83] [ALKK90] [SBCvE90] [CSS$^+$91] [NPA92]) and dataflow ([ACM88] [AI87] [PC90]) architectures have been developed to mitigate communication latency by hiding its effects. These techniques all rely on an abundance of parallelism to provide useful processing to perform while waiting on slow communications. The limit to the usable parallelism then, can be determined by the nature of the problem and the algorithm used to solve it, the rate of computation on each processor, and the communication latency. Our challenge today is to provide sufficiently low-latency communications to match the computation rate provided by commodity processors while allowing the most effective use of the parallelism inherent in each problem.

Regardless of the exact network topology used for communications, both the number of switching components and the amount of wiring inside the network are at least linear in the number of processors supported by the network. The single component failure rate is also linear in the network size. If we do not engineer the network to operate properly when faults exist, the acceptable failure rate for any system will directly fix a ceiling on the maximum machine size. To avoid this ceiling we consider network designs which can operate properly in the presence of faults.

In this document, we examine a class of processor interconnection networks which are designed to simultaneously minimize network latency while maximizing fault tolerance. A combination of organizational techniques, protocols, circuit techniques, and packaging technologies are employed to realize a class of integrated solutions to these problems.

1.1 Goals

Our goals in designing a high-performance network for large-scale multiprocessing are to optimize for:

- Low Latency
- High Bandwidth
- High Reliability
- Testability/Repairability
- Scalability
- Flexibility/Versatility
- Reasonable Cost
- Practical Implementation

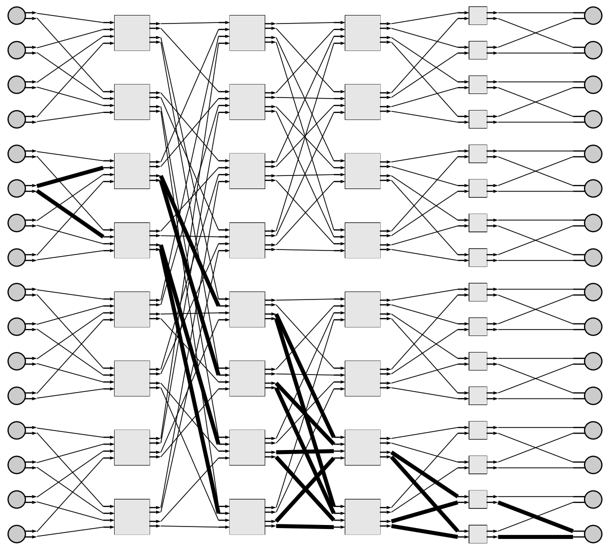
As suggested above and developed further in Sections 2.3 and 2.5, latency and reliability are key properties which must be considered when designing a large-scale, high-performance multiprocessor network. Insufficient bandwidth will have a detrimental impact on latency (Section 2.4). Fault diagnosis and repair are key to limiting the impact of any faults in the network (Section 2.6). Scalability of the solution is important to maximize the longevity with which the solutions are effective. Flexibility in the solutions allow the class of networks to remain applicable across a wide range of specific needs (Section 2.8).

1.2 Scope

This work only attempts to address issues directly related to the network for a large-scale multiprocessor. Attention is paid to providing efficient and robust interfaces between processing nodes and the network. Attention is also given to how the node interacts with the network. However, the fault-tolerance schemes presented here do not guard against failures of the processing nodes or in the memory system. The scheme detailed here may be suitable for a reliable network substrate for future work in processor and memory fault recovery.

1.3 Overview

In this section, we provide a quick overview of the network design at several levels. This section should give the reader a basic picture of the class of networks and technologies being considered. Part II develops everything introduced here in detail.



A multibutterfly style interconnection network constructed from 4×2 (inputs$\times$radix) dilation-2 crossbars and 2×2 dilation-1 crossbars. Each of the 16 endpoints has two inputs and outputs for fault tolerance. Similarly, the routers each have two outputs in each of their two logical output directions. As a result, there are many paths between each pair of network endpoints. Paths between endpoint 6 and endpoint 16 are shown in bold.

Figure 1.1: 16×16 Multibutterfly Network

1.3.1 Topology

A suitable network topology is the first essential ingredient to producing a reliable, high-performance network. The network topology will ultimately dictate:

- **Switching Latency** – the number of switches, and to some extent the length of the wires, which must be traversed between nodes in the network
- **Underlying Reliability** – the redundancy available to make fault-tolerant operation possible
- **Scalability** – the characteristic growth of resource requirements with system size
- **Versatility** – the extent to which the network can be adapted to a wide-range of applications.

To simultaneously optimize these characteristics, we utilize multipath, multistage interconnection networks based on several key ideas from the theoretical community including multibutterflies [Upf89] [LM92] and fat trees [Lei85].

Using multibutterfly (See Figure 1.1) and fat-tree networks (See Figure 1.2), we minimize the number of routing switches which must be traversed in the network between any pair of nodes. Using bounded degree routing nodes, the least possible number of switches between endpoints is logarithmic in the size of the network, a lower bound which these networks achieve. For small machine configurations the multibutterfly networks achieve the logarithmic lower bound with a multiplicative constant of one (*e.g.* routing switches traversed = $\log_r N$; where N is

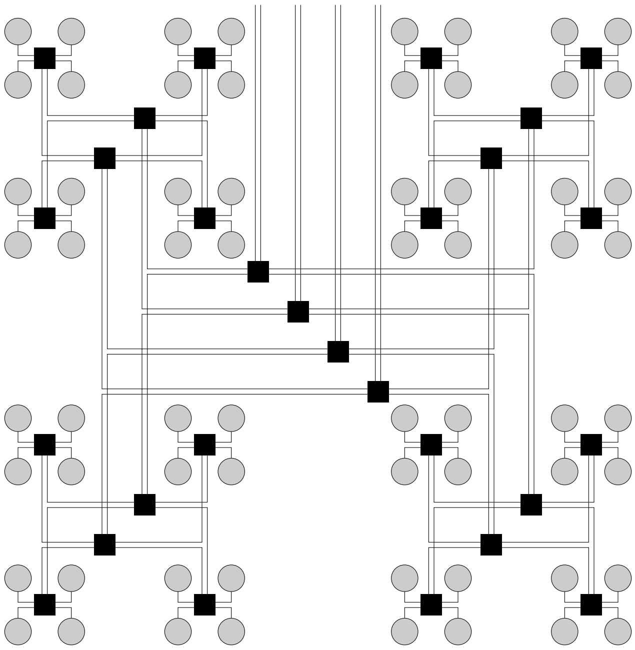


Figure 1.2: Area-Universal Fat-Tree with Constant Size Switches (Greenberg and Leiserson)

the number of processing nodes in the network and r is the radix of the routing component used for switching). For larger machine configurations, fat trees provide lower latency for local communication. Applications can take advantage of the locality inherent in the fat-tree topology to realize lower average communication latencies. To further minimize switching latency, our fat-tree networks make use of short-cut paths, keeping the worst-case switching latency down to $\frac{4}{3}\log_4 N$ when using radix-four routing components.

The multipath nature of these routing networks provides a basis for fault-tolerant operation, as well as providing high bandwidth operation. The multipath networks provide multiple, redundant paths between every pair of processing nodes. The alternative paths are also available for minimizing congestion within the network, resulting in increased effective bandwidth and decreased effective latency. When faults occur, the availability of alternative paths between endpoints makes it possible to route around faulty components in the network.

A high-degree of scalability is achieved by using fat-tree organizations for large networks. The scalable properties of fat trees allow construction of arbitrarily large machines using the same basic network architecture. When organized properly, these large fat trees can be shown to minimize the total length of time that any message spends traversing wires within the routing network as compared to any other network. The hardware resources required for the fat-tree network grow linearly in the number of processors supported.

Further, these networks provide considerable versatility allowing them to be adapted to meet the specific needs of a particular application. By selecting the number of network ports into each processing node, we can customize the bandwidth and reliability within the network to meet the needs of the application. By controlling the width of the basic data channel, we can provide varying amounts of latency and bandwidth into a node. This flexibility makes it possible to use the same basic network solutions across a broad range of machines from low-cost workstations to high-bandwidth supercomputers by selecting the network parameters appropriately.

1.3.2 Routing

While a good network topology is necessary for reliable, high-performance communications, it is by no means sufficient. We must also have a routing scheme capable of efficiently exploiting the features of the network. In developing a routing strategy for use with multiprocessor communications networks, we focussed on achieving a routing framework with the following properties:

1. **Low-overhead routing** – Low-overhead routing attempts to minimize the fraction of potential bandwidth consumed by protocol overhead and similarly minimize the latency associated with protocol processing.
2. **Fault identification and localization with minimal overhead** – To achieve fault tolerance, we must be able to detect when faults corrupt data in our system. Further to minimize the impact of faults on system performance, we must be able to efficiently identify the source of any faults in the system.
3. **Flexible protocol** – To be suitable for use in a wide range of applications and environments, the protocol must be flexible allowing efficient layering of the required data transfer on top of the underlying communications.
4. **Dynamic fault tolerance** – For the network to scale robustly to very large implementations, it is critical that the network and routing components continue to operate properly as new faults arise in the system.
5. **Distributed routing** – In order to avoid single-points of failure in the system, routing must proceed in a distributed fashion, requiring the correct operation of no central resources.

To this end, we have developed the METRO Routing Protocol, MRP, a simple, reliable, source-responsible router protocol suitable for use with multipath networks. MRP provides half-duplex, bidirectional data transmission over pipelined, circuit-switched routing channels. The simple protocol coupled with pipelined routing allows for high-bandwidth, low-latency implementations. The circuit-switched nature avoids the issues associated with buffering inside the network. Each routing component makes local routing decisions among equivalent outputs based on channel utilization, using randomization to choose among equivalent alternatives. Routing components further provide connection information and checksums back to the source node to allow error localization within the network. When errors or blocking occurs, the source can retry data transmission. The randomization in path selection guarantees that any existing non-faulty path can eventually be found without global information.

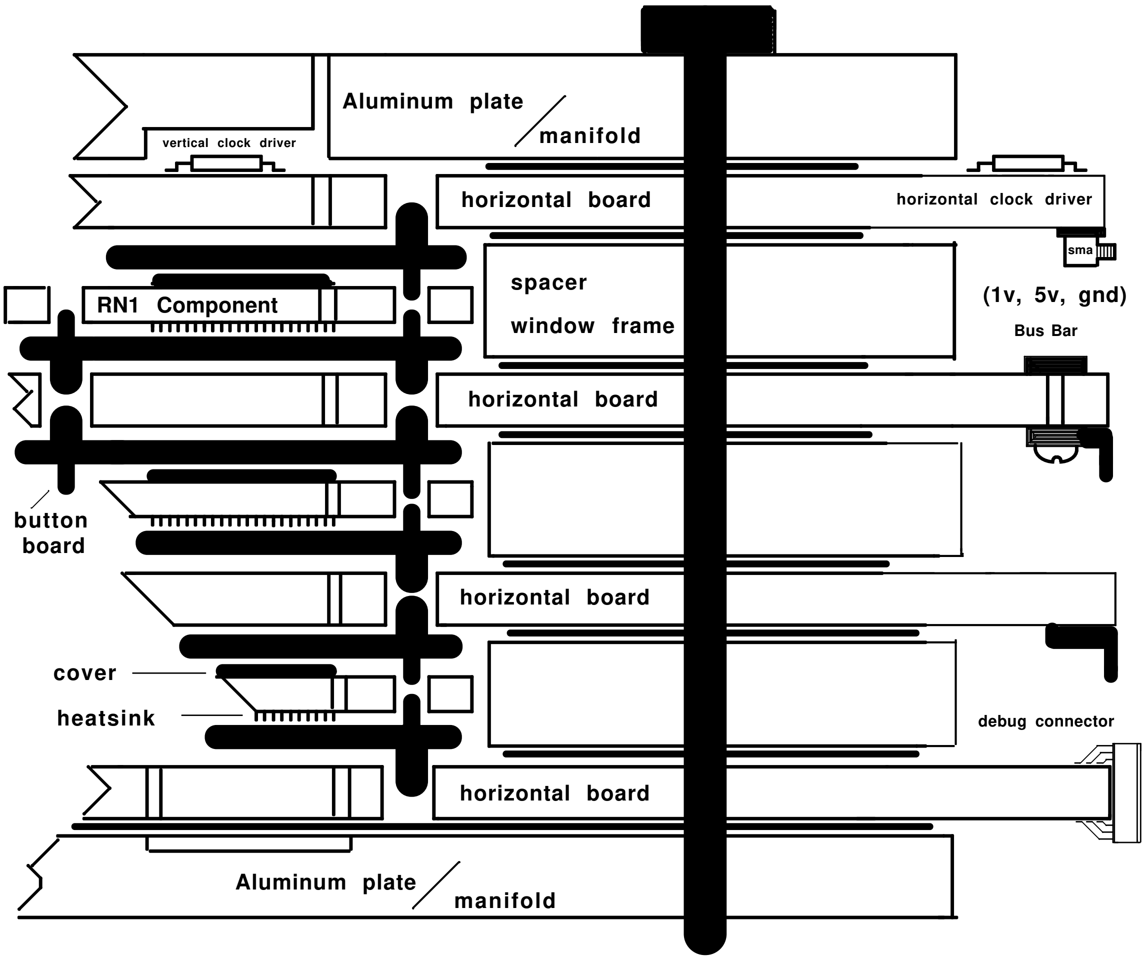


Figure 1.3: Cross-Section of Stack Packaging (Diagram courtesy of Fred Drenckhahn)

1.3.3 Technology

Regardless of the advances we make in topology and routing, the ultimate performance of an implementation is limited by the implementation technology. Packaging density constrains the minimum lengths for interconnect and hence the minimum latency between routing components and nodes. Once our interconnection distances are fixed, data transmission latency is limited by the time taken to traverse the interconnect and to traverse component i/o pads.

Packaging

Our goal in packaging these networks is to minimize the interconnection distances between components. At the same time, we aim to utilize economical technologies and provide efficient cooling and repair of densely packaged components. The basic packaging unit is a three-dimensional *stack* of components and printed-circuit boards (See Figure 1.3). Computational, memory, and routing components are housed in dual-sided land-grid arrays and sandwiched between layers of conventional PCBs. The land-grid arrays, with pads on both sides of the package, serve to both house VLSI components and provide vertical interconnect in the stack structure. Button boards are used to provide reliable, solderless connection between land-grid array packages and adjacent PCBs. The land-grid array and button board packages provide channels for coolant flow. The composite stack structure is compatible with both air and liquid cooling. The stack structure provides the necessary dense interconnection in all three physical dimensions allowing for minimal wiring distances between components. Using this technology, we can package an entire 64-node multiprocessor including the network and nodes in roughly $1' \times 1' \times 5''$.

Signalling

To minimize wire transit and component i/o time, we utilize series-terminated, matched-impedance, point-to-point transmission line signalling. Further, to reduce power consumption the i/o structures use low-voltage signal swings. By integrating a series-terminated transmission line driver into the i/o pads, we avoid the need to wait for reflections to settle on the PCB traces without requiring additional external components. The low-voltage, series-terminated drivers can switch much faster than conventional 5V-swing drivers. Initial experience with this technology indicates we can drive a signal through an output pad, across 30 cm of wire, and into an input pad in less than 5 ns.

1.3.4 Fault Management

Performance in the presence of faulty components and wires can be further improved by hiding the effects of faulty components. Using some novel, fault-tolerant additions to baseline IEEE 1149.1-1990 JTAG scan functionality, we can realize an effective scan-based testing strategy. By configuring components with multiple test-access ports, the architecture is resilient to faults in the test system itself. With port-by-port deselection and scan capabilities, it is possible to diagnose potentially faulty network components online; *i.e.* , while the rest of the system remains fully operational. Furthermore, these facilities allow faulty wires and components to be configured out of the system so that they do not degrade system performance. Once localized using boundary scan, the system can log faulty components for later repair and make an accurate assessment of the system integrity. For larger systems, these facilities allow online replacement of faulty subsystems.

1.4 Organization

Before developing strategies for addressing these problems, Chapter 2 develops the problems and issues in further detail. Part II takes a detailed look at the key components of robust, low-latency networks. Chapter 3 leads off by examining the network topology. Chapter 4 addresses the issue of low-latency, high-speed, reliable routing on the networks introduced in Chapter‘3. Chapter 5 considers fault identification and system reconfiguration. Chapter 6 develops suitable, high-speed signalling techniques compatible with the router-to-router communications required by networks the routing protocol. Finally, Chapter 7 looks at packaging technologies for practical, high-performance networks. Part III contains a brief series of case-studies from our experience designing and building reliable, low-latency networks. Chapter 8 reviews the RN1 routing component. Chapter 9 discusses RN1's successor, the METRO router series. Chapter 11 describes METRO-LINK, a network interface suitable for connecting a processing node into a METRO based network. Finally, Chapters 10 and 12 discuss MBTA, an experimental multiprocessor which puts most of the technology described in Part II and the components detailed in Part III together in a complete multiprocessor system. Chapter 13 concludes by reviewing the techniques introduced in Part II and showing how they come together to achieve low-latency and fault-tolerant operation.

2. Background

This chapter provides background material to prepare the reader for the development in Parts II and III. Section 2.1 describes the fault model and multiprocessor model assumed throughout this document. Section 2.2 provides a brief review of standard scan based testing practices. Section 2.3 and 2.5 point out the importance of low latency and fault tolerance to large-scale multiprocessor systems. Section 2.4 reviews the composition of network latency. Section 2.6 looks at the requirements for fault tolerance. Finally, Sections 2.7 and 2.8 introduce several other key issues in the practical design of interconnection networks.

2.1 Models

2.1.1 Fault Model

Faults occurring in a network may be either static or dynamic and may be transient faults or permanent faults. While a *permanent* fault occurs and remains a fault, a *transient* fault may only persist for a short period of time. Transient faults which recur with notable frequency are termed *intermittent*. [SS92] indicate that transient and intermittent faults account for the vast majority of faults which occur in computer systems. For the purposes of this presentation, *static* faults are permanent or intermittent faults which have occurred at some point in the past and are known to the system as a whole. *Dynamic* faults are transient faults or any faults which the system has not yet detected.

Throughout this work, we assume that faults manifest themselves as:

1. **Stuck-Values** – a data or control line appears to be held exclusively high or low

2. **Random bit flips** – a data or control line has some incorrect, but random value

Faults may appear and disappear at any point in time. They may become permanent and remain in the system, they may be transient and disappear, or they may be intermittent and recurring. Stuck-value errors may take on an arbitrary, but constant, logic value. Bit flips are assumed to take on random values. Specifically, we are not assuming an adversarial fault model (*e.g.* [MR91]) in which faulty portions of the system are allowed to take on arbitrary erroneous values.

These fault-manifestations are chosen to be consistent with fault expectations in digital hardware systems. *Structural* faults in the interconnect between components may give rise to floating or shorted nodes. With proper electrical design, floating i/o's can appear as stuck-values to internal logic. Shorted nodes will depend on the values present on the shorted nodes and may appear as random bit flips when the values differ. Clocking, timing, and noise problems which cause incorrect data to be sampled by a component will also appear as random bit errors. Opens and bridging faults within an IC may also leave nodes shorted or floating. For a good survey of physical faults and their manifestations see Chapter 2 in [SS92].

The manner in which we handle dynamic faults in this work relies on end-to-end checksums to make the likelihood that a corrupted message looks like a good message arbitrarily small. As long as faults produce random data, we can select a checksum which has the desired property. However, if we allow arbitrary, malicious intervention as in an adversarial fault model, the adversary could remove a corrupted message from the network and replace it with one which looks good or remove a good message from the network and fake an acknowledgment. In order to handle this stronger fault-model, one would have to replace our practice of guarding data with checksums with an end-to-end data encryption scheme. A properly chosen encryption scheme could make the chances that an adversary could fake any message sufficiently remote for any particular application.

For the sake of the presentation here, we limit our concern to faults within the network itself. The processing nodes are presumed to function correctly, if at all. A processing node may cease to function, but it may not provide erroneous data to the network. All network transactions requested by the node are presumed to be intentional. The computational implications of losing access to an ongoing computation or the memory stored at a failing node are important but beyond the scope of this work.

Without knowing the reliability design of the computational system as a whole, it is not clear whether a fault-tolerant network should be designed to optimize for harvest or yield. *Yield* is the term used to describe the likelihood that the system can be used to complete a given task. If we require that all nodes be fully connected to the network, then designing the network is a yield problem in which the network is only considered good when it provides full connectivity. In this case, we want to optimize for the highest yield at the fault levels of interest. *Harvest Rate* is the term used to refer to the fraction of total functional unit which are usable in a system. If the computational model can cope with the node loss, then designing the network is a harvest problem in which we attempt to optimize for the most connectivity at any fault level.

2.1.2 Multiprocessor Model

For the purpose of discussion, we assume a homogenous, distributed memory, multiprocessor model as shown in Figure 2.1. Each node is composed of a processor, some memory, and a network interface. In a hardware-supported shared-memory machine, this network interface might be the cache-controller [LLG$^+$91] [ACD$^+$91]; in a message-passing machine, it would be the network message interface [Cor91] [Thi91]. Increasingly, the network interface may be tightly-integrated with the processor [D$^+$92] [NPA91]. We explicitly assume the network interface has multiple connections both into the network and out of the network. Multiple connections are necessary to avoid having a potential single point of failure at the connection between each node and the network.

2.2 IEEE-1149.1-1990 TAP

In Part II, we introduce extensions to standard, scan-based testing practices to make them suitable for use in large-scale systems. This section reviews the major points of the existing standard upon which we are building.

The IEEE Standard Test-Access Port (TAP) [Com90] defines a serial test interface requiring four dedicated I/O pins on each component. The standard allows components to be daisy-chained

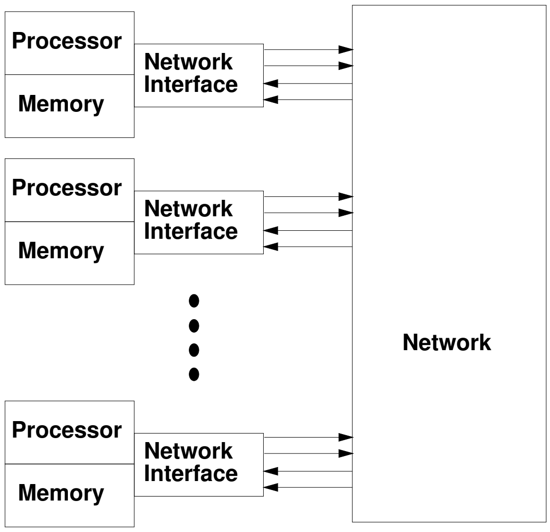


Figure 2.1: Multiprocessor Model

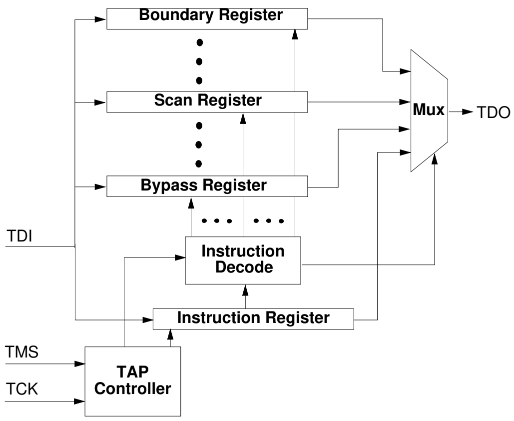


Figure 2.2: Standard IEEE TAP and Scan Architecture

so that a single test path can provide access to many or all components in a system. The standard provides facilities for external boundary-scan testing, internal component functional testing, and internal scan testing. Additionally, the TAP provides access to component-specific testing and configuration facilities. Figure 2.2 shows the basic architecture for an IEEE scan-based TAP.

In a system in which all components comply with the standard, boundary-scan testing allows complete structural testing. Using the serial scan path, every I/O pin in the system can be configured to drive a logic value or act as a receiver. Using the same serial scan path, the value of every receiver can be sampled and recovered. This mechanism allows the TAP to verify the complete connectivity of the components in the system. All connectivity faults, shorted wires, stuck drivers or receivers, or open-circuits can be identified in this manner [GM82] [Wag87].

The scan path allows data to be driven into a component independent of the values present on the component's external I/O pins. The resultant values generated by the component in response to the driven data can similarly be sampled and recovered via the serial scan path. This facility permits functional, in-circuit verification of any such component.

The standard allows additional instructions which may function in a component-specific manner. These instructions provide uniform access to internal-component scan-paths. Such internal paths are commonly used to allow a small number of test-patterns to achieve high-fault coverage in components with significant internal state. Other common additions are configuration registers and Built-In-Self-Test (BIST) facilities [KMZ79] [LeB84] [Lak86].

2.3 Effects of Latency[1]

For the sake of understanding the role of latency in multiprocessor communications, we consider a very simple model of parallel computation. To solve our problem we need to execute a total number of operations, c. Let us assume our problem is characterized by a constant amount of parallelism, p. During each clock cycle, we can perform p operations. Parallelism is limited because each set of p operations depends on the results of the previous p operations. After a set of operations complete, they must communicate their results with the processors which need those results for the next set of p operations. Let us assume that communicating between processors requires l clock cycles of latency.

If we executed our program on a multiprocessor with more than p nodes, it would take time $T_{multiproc}$ cycles to solve the problem.

$$T_{multiproc} = \frac{c \cdot (l+1)}{p} \tag{2.1}$$

At clock cycle 1, we can execute p operations on the nodes. We then require l cycles to communicate the results. The next p operations can then be executed in cycle $l+2$. Computation continues in this manner executing p operations every $(l+1)$ cycles. Thus $\frac{p}{l+1}$ operations are executed, on average, each cycle giving us Equation 2.1.

We see immediately that the exploitable parallelism is limited by the latency of communication. If our problem allows much more parallelism than we have nodes in our multiprocessor, we can hide the effects of latency by performing other operations in a set while waiting for the communication associated with the earlier operations to complete. However, if we wish to use large-scale multiprocessors to solve big problems, latency directly acts to limit the extent to which we can exploit parallel execution to solve our problem quickly.

In most parallel programs, the number of operations which can be executed in parallel varies throughout the program's execution. Hence p is not a constant. Researchers have characterized this parallelism for particular programs and computational models using a *parallelism profile* which

[1]The basic argument presented here is drawn from an unpublished manuscript by Professor Michael Dertouzos.

shows the number of operations which may be executed simultaneously at each time-step assuming an unbounded number of processors (*e.g.* [AI87]). The available parallelism will be a function of the compiler and run-time system in addition to being dependent on the problem being solved and the algorithm used to solve it.

Communication latency is also, generally, not constant. Section 2.4 looks at the factors that affect latency in a multiprocessor network.

Despite the fact that our model used above is overly simplistic, it does gives us insight into the role which latency plays in parallel computing. When our algorithm, compiler, and run-time system can discover much more parallelism than we have processing elements to support, with good engineering we can hide some or all of the effects of latency. On the other hand, when we are unable to find such a surplus of parallelism, latency further derates the exploitable parallelism in a linear fashion.

2.4 Latency Issues

In this section, we consider in further detail many of the issues relevant to achieving low-latency communications.

2.4.1 Network Latency

Ignoring protocol overhead at the destination or receiving ends of a network, the latency in an interconnection network comes from four basic factors:

1. *Transit Latency* (T_t): The amount of time the message spends traversing the interconnection media within the network
2. *Switching Latency* (T_s): The amount of time the message spends being switched or routed by switching elements inside the network
3. *Transmission Time* ($T_{transmit}$): The time required to transmit the entire contents of a message into or out-of the network
4. *Contention Latency* (γ): The degradation in network latency due to resource contention in the network

Transit latency is generally dictated by physics and geometry. Transit latency is the quotient of the physical distance and the rate of signal propagation.

$$T_t = \frac{d}{v} \tag{2.2}$$

Basic physics limits the amount of time that it takes for a signal to traverse a given distance. Materials will affect the actual rate of signal propagation, but regardless of the material, the propagation speed will always be below the speed of light, $c \approx 3 \times 10^{10}$cm/s. The rate of propagation is given by:

$$v = \frac{1}{\sqrt{\mu\epsilon}} \tag{2.3}$$

For most materials $\mu \approx \mu_0$, where μ_0 is the permittivity of free space. Conventional printed-circuit boards (PCBs) have $\epsilon = \epsilon_r \epsilon_0$, where $\epsilon_r \approx 4$ and ϵ_0 is the dielectric constant of free space; thus, $v \approx \frac{c}{2}$. High performance substrates have lower values for ϵ_r. The physical geometry for the network determines the physical interconnection distances, d. Physical geometry is partially in the domain of packaging (Chapter 7), but is also determined by the network topology (Chapter 3). All networks are limited to exploiting, at most, three-dimensional space. Even in the best case, the total transit distance between two nodes in a network is at least limited by the physical distance between them in three-space. Additionally, since physical interconnection channels (*e.g.* wires, PCB traces, silicon) occupy physical space, the volume these channels consume within the network often affects the physical space into which the network and nodes may be packed.

For networks with uniform switching nodes, *switching latency* is the product of the number of switching stages between endpoints, s_n, and the latency of each switching node, t_{nl}.

$$T_s = s_n \cdot t_{nl} \tag{2.4}$$

The network topology dictates the number of switching stages. The latency of each switching node is the sum of the signal i/o latency, t_{io}, and the switching node functional latency, t_{switch}.

$$t_{nl} = t_{io} + t_{switch} \tag{2.5}$$

The signal i/o latency, or the amount of time required to move signals into and out-of the switching node, is generally determined by the signalling discipline and the technologies used for the switching node (Chapter 6). The switch functional latency accounts for the time required to arbitrate for an appropriate output channel and move message data from the input channel to the output channel. In addition to technology, the switch functional latency will depend on the complexity of the routing and arbitration schemes and the complexity of the switching function (Chapter 4). Larger switches generally require more complicated arbitration and switching, resulting in larger inherent switching latencies.

The *transmission time* accounts for the amount of time required to move the entire message data into or out-of the network. In many networks, the amount of data transmitted in a message is larger than the width of a data channel. In these case, the data is generally transmitted as a sequence of data where each piece is limited to the width of the channel. Assuming we have a message of length L to send over a channel w bits wide which can accept new data every t_c time units, we have the transmission time, $T_{transmit}$, given by:

$$T_{transmit} = \left\lceil \frac{L}{w} \right\rceil \cdot t_c \tag{2.6}$$

Here we see one of the places where low bandwidth has a detrimental effect on network latency. $T_{transmit}$ increases as the channel bandwidth decreases.

Contention latency arises when resource conflicts occur and a message must wait until the necessary resources are available before it can be delivered to its designated destination. Such conflicts result when the network has insufficient bandwidth or the network bandwidth is inefficiently used. In packet-switched networks, contention latency manifests itself in the form of queuing which must occur within switches when output channels are blocked. In circuit-switched networks, contention latency is incurred when multiple messages require the same channel(s) in the network and some messages must wait for others to complete. Contention latency is the effect which differentiates an architecture's theoretical minimum latency from its realized latency. The amount of contention latency is highly dependent on the manner in which an application utilizes the network. Contention latency is also affected by the routing protocol (Chapter 4) and network organization (Chapter 3). One can think of contention latency as a derating factor on the unloaded network latency.

$$T_{unloaded} = T_s + T_t \tag{2.7}$$

$$T_{net} = \gamma(\text{application}, \text{topology}) \cdot T_{unloaded} + T_{transmit} \tag{2.8}$$

One of the easiest ways to see this derating effect is when an application requires more bandwidth between two sets of processors than the network topology provides. In such a case, the effective latency will be increased by a factor equal to the ratio of the desired application bandwidth to the available network bandwidth. *e.g.* if A_{bw} is the bandwidth needed by an application, and N_{bw} is the bandwidth provided by the network for the required communication, we have:

$$\gamma \approx \frac{A_{bw}}{N_{bw}}$$

In practice, the derating factor is generally larger than a simple ratio due to the fact that the resource conflicts themselves may consume bandwidth. For example, on most local-area networks, when contention results in collisions, the time lost during the collision adds to the network latency as well as the time to finally transmit the message.

The effects of contention latency make it clear why a bus is inefficient for multiprocessor operation. The bus provides a fixed bandwidth, N_{bw}. There is no switching latency and generally a small transit latency over the bus. However, as we add processors to the bus, the bandwidth potentially usable by the application, A_{bw}, generally increases while the network bandwidth stays fixed. This translates into a large contention derating factor, γ, and consequently high network latency.

Unfortunately, it is hard to quantify the contention latency factor as cleanly as we can quantify other network latency factors. The bandwidth required between any pair of processors is highly dependent on the application, the computational model in use, and the run-time system. Further, it depends not just on the available bandwidth between a pair of processors, but between any sets of processors which may wish to communicate simultaneously.

2.4.2 Locality

Often physical and logical locality within a network can be exploited to minimize the average communication latency. In many networks, nodes are not equidistant. The transit latency and switching latency between a pair of nodes may vary greatly based on the choice of the pair of nodes. *Logical distance* is used to refer to the amount of switching required between two nodes (T_s), and *physical distance* is used refer to the transit latency (T_d) required between two nodes. Thus, two nodes which are closer, or more local, to each other logically and physically may communicate with lower latency than two nodes which are further apart. Additionally, when logically close nodes communicate they use less switching resources and hence contribute less to resource contention in the network.

The extent to which locality can be exploited is highly dependent upon the application being run over the network. The exploitation of network locality to minimize the effective communication latency in a multiprocessor system is an active area of current research [KLS90] [ACD$^+$91] [Wal92]. Exploiting network locality is of particular interest when designing scalable computer systems since the latency of the interconnect will necessarily increase with network size. Assuming the physical and logical composition of the network remains unchanged when the network is grown, for networks without locality, the physical distance between all nodes grows as the system grows due to spatial constraints. For networks with locality the physical distance between the farthest separated nodes grows. Additionally, as long as bounded-degree switches (Section 2.7.1) are used to construct the network, the logical distance between nodes increases as well. Locality exploitation is one hope for mitigating the effects of this increase in latency.

It is necessary to keep the benefits due to locality in proper perspective with respect to the entire system. A small gain due to locality can often be dwarfed by the fixed overheads associated with communication over a multiprocessor network. Locality optimizations yield negligible rewards when the transmission latency benefit is small compared to the latency associated with launching and handling the message. Johnson demonstrated upper bounds on the benefits of locality exploitation using a simple mathematical model [Joh92]. For a specific system ([ACD$^+$91]), he shows that even for machines as large as 1000 processors, the upper bound on the performance benefit due to locality exploitation is a factor of two.

2.4.3 Node Handling Latency

This document concentrates on designing the network for a high-performance multiprocessor. Nonetheless, it is worthwhile to point out that the effective latency seen by the processors is also dependent on the latency associated with getting messages from the computation into the network, out-of the network, and back into the computation. *Network input latency*, T_p, is the amount of time after a processor decides to issue a transaction over the network, before the message can be launched into the network, assuming network contention does not prevent the transaction from entering the network. Similarly, *network output latency*, T_w, is the amount of time between the arrival of the a complete message at the destination node and the time the processor may begin actually processing the message. If not implemented carefully, large network input and output latency can limit the extent to which low-latency networks can facilitate low-latency communication between nodes. Combining these effects with our network latency we have the total processor to processor message latency:

$$T_{message} = T_p + T_{net} + T_w \tag{2.9}$$

This document will not attempt to directly address how one minimizes node input and output latency. Node latencies such as these are highly dependent on the programming model, processor, controller, and memory system in use. [NPA92] and [D$^+$92] describe processors which were designed to minimize these latencies. [E$^+$92] and [CSS$^+$91] describe a computational model intended to minimize these latencies. Here, we will devote some attention to assuring that the network itself does not impose limitations which require large node input and output latencies.

2.5 Faults in Large-Systems

In this section we review a first-order model of system failure rates. We use this simple model to underscore the importance of fault tolerance in large-scale systems.

For the sake of simplicity, let us begin by considering a simple discrete model of component failures. A single component fails with some probability, P_c in time T. This gives us a failure rate:

$$\lambda_p = \frac{P_c}{T} \tag{2.10}$$

The probability that the component survives a period of time T, is then:

$$P_{cs} = 1 - P_c \tag{2.11}$$

If we have a system with N components which fails if any of the individual component fail, then the system survives a period of time T only if all components survive the period of time T. Thus:

$$P_{ss} = P_{cs}^N = (1 - P_c)^N \tag{2.12}$$

For any reasonable component and a small time period, T, $P_c << 1$. To first order, Equation 2.12 can be reasonably be approximated as:

$$P_{ss} = (1 - N \cdot P_c) \tag{2.13}$$

Which tells us the probability that the system fails during time T is simple:

$$P_s = N \cdot P_c \tag{2.14}$$

Which corresponds to a failure rate:

$$\lambda_s = \frac{N \cdot P_c}{T} = N \cdot \lambda_p \tag{2.15}$$

From Equation 2.15 we see that the failure rate increases linearly with the number of components in the system, to first order.

Example A moderate complexity, modern component has a failure rate of ten failures per million hours ($\lambda_c \approx 10^{-5}\text{hr}^{-1}$) (See [oD86] for estimating component failure rates). A million component machine which depended on all million components working correctly, would have:

$$\lambda_s = N \cdot \lambda_p = 10^6 \times 10^{-5}\text{hr}^{-1} = 10/\text{hr} \tag{2.16}$$

This gives the machine a Mean Time To Failure (MTTF) of 6 minutes.

If we can relax the requirement that all components and interconnect function correctly in order for the system to be operational, we can improve the MTTF. If we can sustain k faults before the system is rendered inoperative, the MTTF will be longer. As long as $k << N$, we can assume a constant failure rate for components given by Equation 2.15. Assuming the faults are independent, the rate of occurrence of k failures is:

$$\lambda_k = \frac{\lambda_s}{k} \tag{2.17}$$

That is, the MTTF increases linearly with the number of tolerated faults. Revisiting our example, if we design the system to sustain 1000 faults ($k = 1000$), or just 0.1% of the total components in the system, the MTTF increases by a factor of 1000 to 6000 minutes or 100 hours.

For sufficiently large systems, we cannot achieve an adequately low system failure rate by requiring that every component in the system function properly. Rather, we must design sufficient redundancy into our system to achieve the reliability desired.

2.6 Fault Tolerance

In order to achieve fault tolerance, we need the ability to detect when faults have occurred and the ability to handle faults which have occurred. Typically, one uses redundancy in some form to satisfy both of these needs. Redundant data transmitted along with the message can be used to identify when portions of a message are damaged. Parity bits and message checksums are common examples of redundant data used to identify data corruption. Once faults are detected, we rely on redundant network hardware to avoid faulty portions of the network. That is, there must be different resources which perform the same function as the faulty portion of the network which can be used in place of the faulty portion. We also need a mechanism for exploiting the redundancy. The network organization (Chapter 3) often provides the resource redundancy. The routing protocol (Chapter 4) provides the redundancy for fault detection and provides mechanisms for exploiting the redundancy in the network.

Designing networks to perform well in the presence of faults is very similar to designing networks to perform well in the presence of contention. Faults in the network look much like contention. Faulty resources are not useful for effectively routing data. In this manner, they have the same effect as resources which are always in use. Faulty resources also cause additional traffic in the network since they may corrupt messages and hence require the messages to be retransmitted. Alternately, we can think of the faulty resources as migrating routing traffic which they would have handled to other resources in the network. These non-faulty resources now see more traffic as a result. Appropriate design can yield solutions which improve both the performance of the system in the face of faults and the performance of the system in the face of heavy traffic.

2.7 Pragmatic Considerations

We must also consider several pragmatic considerations associated with building any systems. When building a system, such as a network, we are constrained by the economics of currently available technology, issues of design complexity, and fundamental physical constraints.

2.7.1 Physical Constraints

For instance, we have already observed that the speed of signal propagation is largely fixed by the speed of light and the dielectric constant of readily available materials. Materials with notably lower dielectrics do exist, but the cost and reliability of these materials currently relegates their use to small, high-end systems. As technology improves, we can expect these or other materials with lower dielectric constants to be available at prices which make their use more worthwhile.

One might consider using light, itself to achieve the maximum transmission rate ($v = c$). In some situations, this makes the most sense. However, the fact that signals must be converted from propagating electrons to propagating photons and back again, often defeats any potential gains. The latency associated with converting from electrons to photons and back is currently large. Even assuming 100% power effeciency, modern optical modulator/detectors have at least an order of magnitude more i/o latency, t_{io}, than purely electrical i/o pads (*e.g.* 40 ns versus 3 ns) at comparable power levels [LHM$^+$89]. Since optical detection latency is inversely proportional to the incident power level, the optical conversion would require an order of magnitude greater power than the electrical pads to make the optical i/o latency comparable to electrical i/o latency. It only makes sense to make this optical conversion when the distance traversed is sufficiently large that the reduction in physical transit latency due to faster propagation is larger than the conversion latencies.

Current VLSI technology limits the bonding of i/o pads to the periphery of the integrated circuit die. This forces the number of i/o channels into an integrated circuit (IC) to be proportional to the perimeter of the die. Due to external bonding requirements, i/o pads are shrinking more slowly than other IC features. Consequently, ICs have a fairly fixed, limited number of i/o pads and this number is not scaling comparable to the rate of scaling of useful silicon area inside the die. Available technology, thus, limits the number of i/o channels into an IC and hence the size of the primitive switching elements we can build.

We must always take account of the fact that wires and components consume space. The finite thickness of wires limits the physical compactness of our multiprocessor. The space between nodes and routers must be large enough to accommodate the wires necessary to provide interconnect. In some topologies, the growth rate of the machine is dictated by the growth rate for the interconnect as much as the number and size of components. Additionally, space must be provided for adequate component cooling and access for repair.

2.7.2 Design Complexity

Each different component in a system requires separate:

- Engineering effort to design and verify
- Non-recurring engineering (NRE) costs to produce
- Testing to select good components and diagnose potentially faulty components
- Shelf-space to stock the components

Consequently, it is beneficial to minimize the number of different components used in constructing any system.

2.8 Flexibility Concerns

Just as engineering more types of components is costly in terms of development, NRE, and testing, designing a new network for each new application or specific machine is also costly. We look for solutions which provide a wide range of flexibility so they can easily be extended or re-parameterized to solve a variety of problems. When building a network for a large-scale multiprocessor, our desire for flexibility leads us to be concerned about the following:

- How do we provide additional bandwidth for each node at a given level of semiconductor and packaging technology?
- How do we get more/less fault tolerance for applications which have a higher/lower premium for faults
- How do we build larger (smaller) machines?
- How can we decrease latency? at what costs?

Part II

Engineering Reliable, Low-Latency Networks

3. Network Organization

In this chapter we survey potential low-latency networks and identify a family of networks which is most suitable for use in large-scale, fault-tolerant multiprocessors given practical considerations. After determining the basic network structure, we examine the issues involved in optimizing a particular network for a given application.

3.1 Low-Latency Networks

3.1.1 Fully Connected Network

From the standpoint of latency, the optimal network is a fully-connected network in which every processor has a direct connection to every other processor (See Figure 3.1). Here, there is no switching latency (*i.e.* $T_s = 0$). The problem with this network, of course, is that the processor node size grows linearly with the size of the system. This is not practical for several reasons. We cannot build very large networks with bounded pin-out components, and a different component size is needed for each different network size. Using techniques from [Tho80] and [LR86], we find the interwiring resources will grow as $\Theta(N^3)$. Wiring constraints alone require that the best packaging volume grows as $\Theta(N^3)$, making, in the best case, the wiring distances, d, grow as $\Theta(N)$. Such an organization is not very practical.

3.1.2 Full Crossbar

Next, we consider a full crossbar arrangement (See Figure 3.2). If we could build a large enough crossbar, we only traverse on switching node between any source-destination pair. Unfortunately,

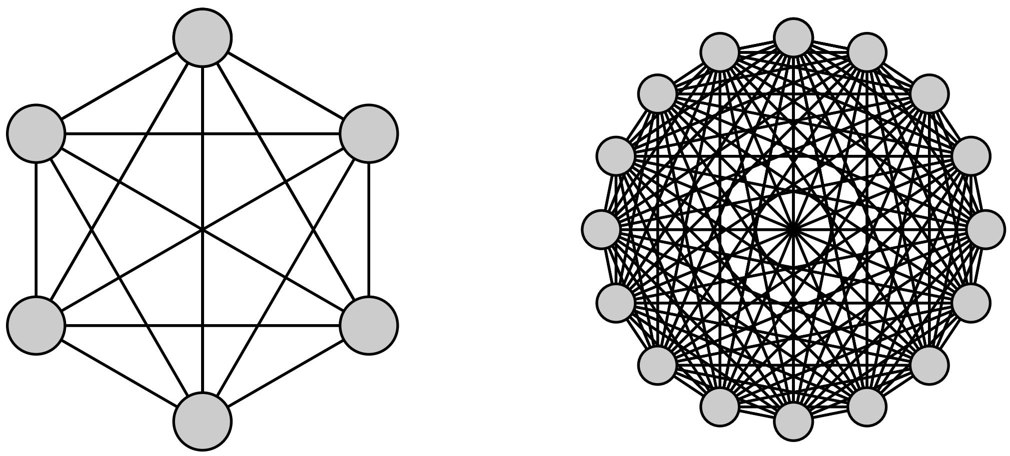


Figure 3.1: Fully Connected Networks

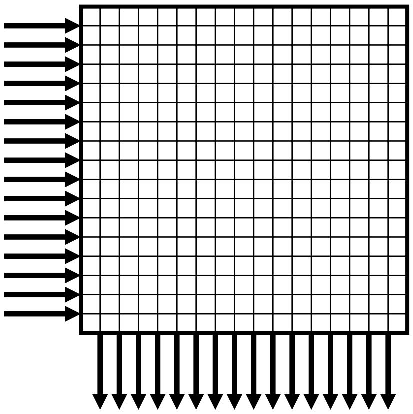


Figure 3.2: Full 16×16 Crossbar

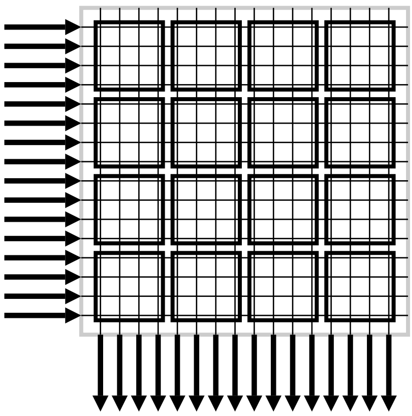
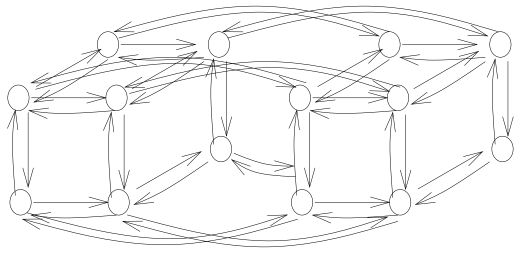


Figure 3.3: Distributed 16×16 Crossbar

our pin limitations (Section 2.7.1), will not allow us to build a single crossbar of arbitrary size. In practice, we would have to distribute the function across many different components as shown in Figure 3.3. This would incur $O(n)$ switching latency and require $O(n^2)$ such switches.

3.1.3 Hypercube

We might consider building a hypercube network to exploit locality and distributed routing control. The switching latency is $\log_2(N)$ as we need traverse at most one switching link in each dimension of the hypercube. Unfortunately, to maintain this characteristic, the switching node degree grows as $\Theta(\log(N))$. Node size soon runs into our pin limitations (Section 2.7.1) and a different size node is needed for each size of the machine constructed. Additionally, when implemented in three-dimensional space, the interconnection requirements cause the machine volume to grow as $\Theta(N^{\frac{3}{2}})$. This result is also derivable from the techinques presented in [Tho80] and [LR86] by considering the number of wires which must cross through the middle of the machine



Shown above is a 16 processor hypercube. (Drawing by Frederic Chong)

Figure 3.4: Hypercube

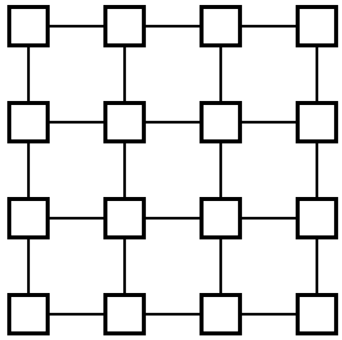


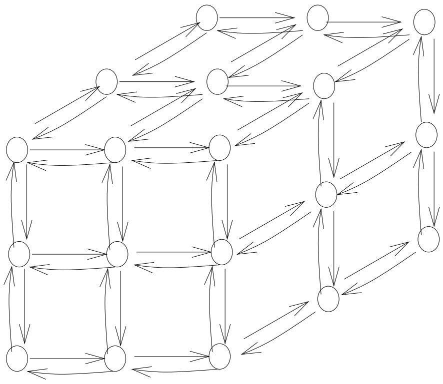
Figure 3.5: Mesh – k-ary-n-cube with $k = 2$

in any decomposition. If we divide an N-processor machine in half, the number of wires crossing the bisecting plane will be $\Theta(N)$. If we distribute these wires in the two-dimensional plane dividing the two halves, then the plane is $\Theta(\sqrt{N})$ wire widths wide in each dimension. Considering that we get the same effect if we divide the machine via an orthogonal plane which also bisects the machine, we see that the machine is $\Theta(\sqrt{N})$ long in each dimension and hence the volume is $\Theta(N^{\frac{3}{2}})$. From this we can see that the transit distance, d, will generally grow as $\Theta(\sqrt[2]{N})$.

Making some compromises for practicality on the basic hypercube structure, a number of derivative networks result. The next two sections cover two major classes, multistage networks and k-ary-n-cubes.

3.1.4 k-ary-n-cube

For k-ary-n-cubes, we fix the dimension (k) to avoid the switching node size growth problem associated with the pure hypercube. We still get the locality and distributed routing. The switching latency grows as $O(\sqrt[k]{N})$ since there are at most $\sqrt[k]{N}$ routers which must be traversed in each



Shown above is a 27 processor cube network. (Drawing by Frederic Chong)

Figure 3.6: Cube – k-ary-n-cube with $k = 3$

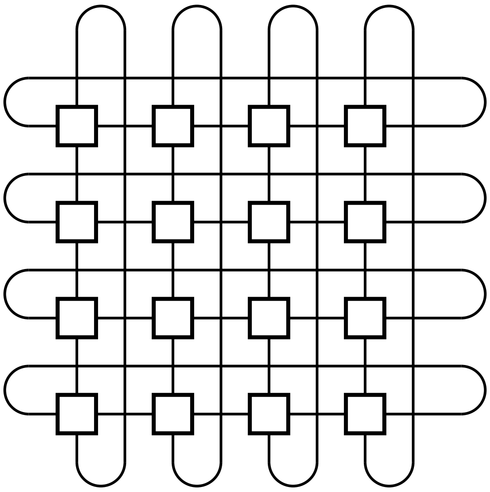


Figure 3.7: Torus – k-ary-n-cube with $k = 2$ and Wrap-Around Torus Connections

dimension. Many popular k-ary-n-cubes networks in use today set $k = 2$ or $k = 3$ to build mesh (See Figure 3.5) or cube (See Figure 3.6) structures [Dal87]. For these networks, the distances between components can be made uniformly short such that the switching latency dominates the transit latency. When constrained to three-dimensional space, larger values of k, will tend to have transit latencies which scale as $\Omega(\sqrt[3]{N})$. Toroidal k-ary-n-cubes can be used to cut the worst case switching latency in each dimension in half and avoid hot-spot problems in simple k-ary-n-cubes (See Figure 3.7) [DS86].

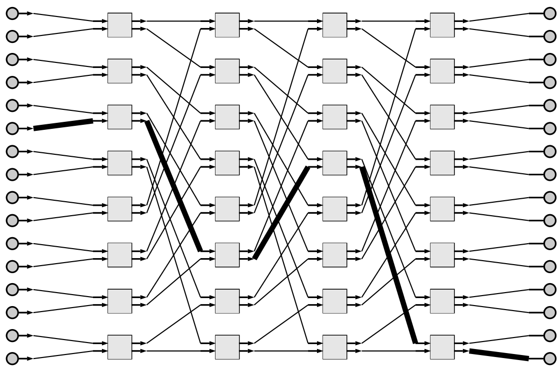


Figure 3.8: 16×16 Omega Network Constructed from 2×2 Crossbars

3.1.5 Flat Multistage Networks

A multistage network distributes each hypercube routing element spatially so that fixed-degree switches can be used for routing. Like the hypercube, routing can occur in a distributed manner requiring only $\log_r(N)$ stages between any pair of nodes in the network. Here r is a constant known as the *radix* which denotes the number of distinct directions to which each routing switch can route. Unlike the hypercube and k-ary-n-cube, the multistage network does not provide any locality. The number of switches required by a multistage network grows as $O(N \log(N))$. The best-case packaging volume grows as $\Theta(N^{\frac{3}{2}})$ and the transit latency grows as $\Theta(\sqrt{N})$ like the hypercube [LR86].

Quite a variety of networks can be classified as multistage networks including: Butterfly networks, Banyan networks, Bidelta networks [KS86], Benes networks, and Multibutterfly networks. Figures 3.8 through 3.11 show some popular multistage networks. Each stage in these networks routes by successively subdividing the set of possible destinations into a number of *equivalence classes* equal to the radix of the routing components. For example, consider a radix-2 network. When connections enter the network, any input can reach any destination. The first stage of routing components divides this class into two different equivalence classes based on desired destination. Each succeeding network stage further subdivides a previous stage's equivalence classes into two more equivalence classes. When there is a single destination in each equivalence class, the network has uniquely determined the desired destination and can connect to the destination endpoints. This successive subdivision can be easily seen in the network shown in Figure 3.9.

3.1.6 Tree Based Networks

Properly constructed, a tree-based, multistage network avoid the major liabilities associated with the standard multistage networks. Specifically, we consider fat-tree networks as described in [Lei85] and [GL85] and shown in Figure 1.2. The switching delay remains $O(\log(N))$ as

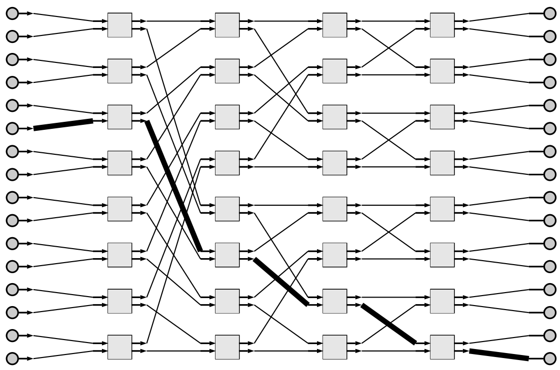


Figure 3.9: 16×16 Bidelta Network

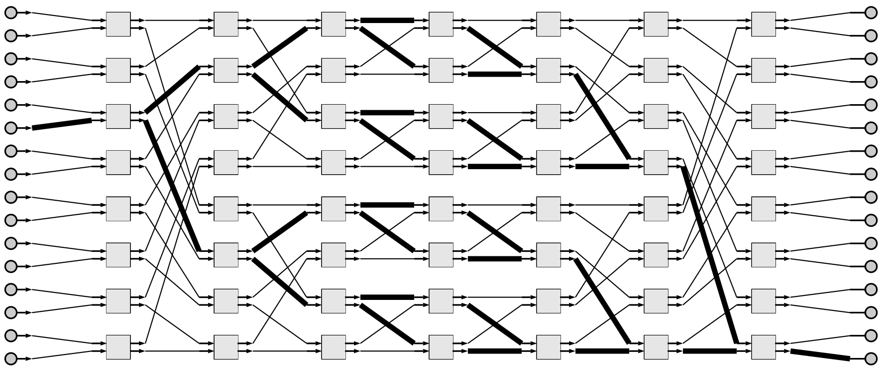


Figure 3.10: Benes Network

with hypercubes and multistage networks. Routing may occur in a distributed fashion. Unlike the multistage networks described above, the tree-based networks do allow locality exploitation. When the bandwidth between successive stages of the tree is chosen appropriately, the tree structures can be arranged efficiently in three-dimensional space; switching and wiring resources grow as $\Theta(N)$ and transit latency will grow as $\Theta(\sqrt[3]{N})$. While a tree-based network may have less cross-machine bandwidth than a hypercube with the same number of nodes, the tree-based machine requires $O(\log(N))$ less interconnect hardware. As a result, if one were to compare machines of the same size, taking into account three-dimensional space restrictions, the tree machine provides at least as much bandwidth while supporting $O(\log(N))$ more nodes. Leiserson shows that properly sized fat trees can efficiently perform any communication performed by any other similarly sized network

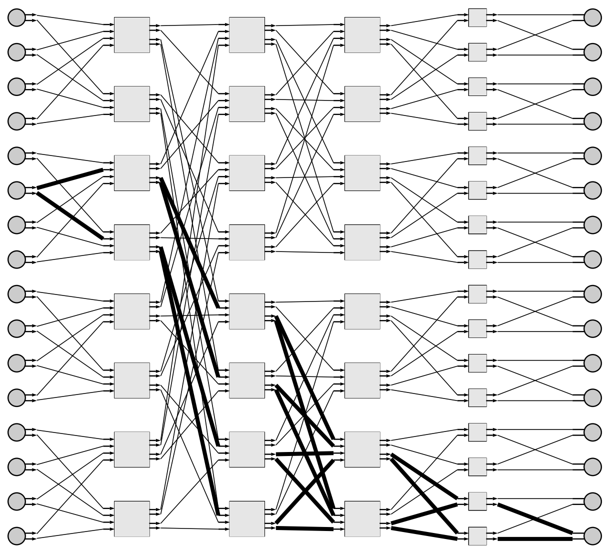


Figure 3.11: 16×16 Multibutterfly Network

[Lei85].

3.1.7 Express Cubes

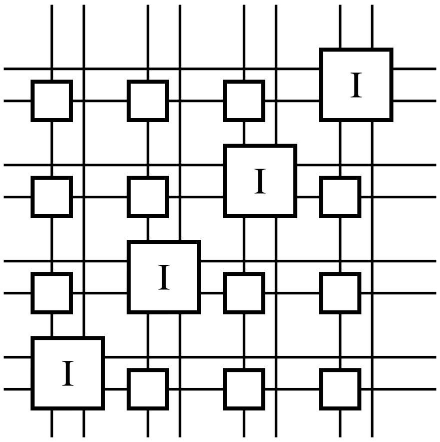
Express cubes [Dal91] are a hybrid between a tree-structure and a k-ary-n-cube (See Figure 3.12). By placing interchange switches periodically in a k-ary-n-cube, the switching delay can be reduced from $\Theta(\sqrt[k]{N})$ to $\Theta(\log(N))$. Done properly, the transit latency remains $\Theta(\sqrt[3]{N})$. If we allow several different kinds of switching elements in the network, the size of each switching element can be limited to a fixed size.

3.1.8 Summary

Table 3.1 summarizes the major characteristics of the networks reviewed here. Asymptotically, at least, we see that fat trees and express cubes have the slowest growing transit and switching latencies while maintaining the slowest resource growth. For a limited range of network sizes, flat multistage networks and k-ary-n-cubes may offer reasonable, or even superior, performance at reasonable hardware costs.

3.2 Wire Length

In this chapter, we have introduced many networks which have wires whose length is a function of the network size. We call a *long wire* any single run of wire between two switches which has a transit time in excess of the rate at which we could otherwise clock data between the switches. If we required the data to traverse any such wires in a single clock cycle, we would have to increase the clock period to accommodate the longest wire in the system. The longest wires in many of these network will be $\Omega(\sqrt[3]{N})$ due to spatial constraints in three-dimensions. Requiring data to



Shown above is a portion of an express mesh after [Dal91]. The components labelled with an *I* are interchange units which allow connections to be routed along express channels, thereby bypassing intermediate switching nodes.

Figure 3.12: Express Cube Network – $k = 2$

Network	T_s	T_t	**Locality**	**Resources**	**Practical Drawbacks**
Fully Connected	0	$\Theta(N)$	–	$\Theta(N^3)$	Node size $\sim \Theta(N)$
Distributed Crossbar	$\Theta(N)$	$\Theta(N)$	some	$\Theta(N^2)$	
Hypercube	$\Theta(\log(N))$	$\Theta(\sqrt[2]{N})$	yes	$\Theta(N^{\frac{3}{2}})$	Node size $\sim \Theta(\log(N))$
k-ary-n-cube	$\Theta(\sqrt[k]{N})$	$\Theta(\sqrt[3]{N})$	yes	$\Theta(N)$	
Flat Multistage	$\Theta(\log(N))$	$\Theta(\sqrt[2]{N})$	no	$\Theta(N^{\frac{3}{2}})$	
Fat-Tree	$\Theta(\log(N))$	$\Theta(\sqrt[3]{N})$	yes	$\Theta(N)$	
Express Cube	$\Theta(\log(N))$	$\Theta(\sqrt[3]{N})$	yes	$\Theta(N)$	

Table 3.1: Network Comparison

traverse these wires in a single clock cycle would require our clock period to increase comparably with network size. However, if we pipeline multiple bits on the long wires, we do not have to adjust the clock frequency to accommodate long wires. Our notion of transit latency as proportional to interconnection distance (Equation 2.2), will still hold. Instead of being a continuous equation as given, it becomes discretized in units of the clock period, t_c.

$$T_t = \sum_i \left\lceil \frac{\frac{d_i}{v}}{t_c} \right\rceil \cdot t_c \tag{3.1}$$

Equation 3.1 explicitly breaks the total distance into segments (d_i) between each pair of switching elements in the path between the source and destination nodes to properly account for the effects of this discretization. Techniques for ensuring correct operation when bits are pipelined on the wires are detailed in Section 6.9.

3.3 Fault Tolerance

In order to achieve fault tolerance in the network, we need multiple, distinct paths between any pair of nodes. The more distinct paths our network supports, the more robust the network will be to faults occurring in the network. In this section, we look at the multipath nature of the practical low-latency networks identified in the previous section.

3.3.1 Indirect Routing

If we allow *indirect routing*, all of the networks examined in this chapter have multiple paths. With indirect routing, a message may be routed from a source to a destination node by first routing the message through one or more intermediate nodes in the network. That is, when the source cannot reach the destination directly through the network, it is often possible for it to reach another processing node in the network which can, in turn, reach the desired destination node. If we allow arbitrary indirect hops through the network, any message can eventually be routed as long as the transitive closure of the non-faulty direct interconnect covers all the nodes in use in the network.

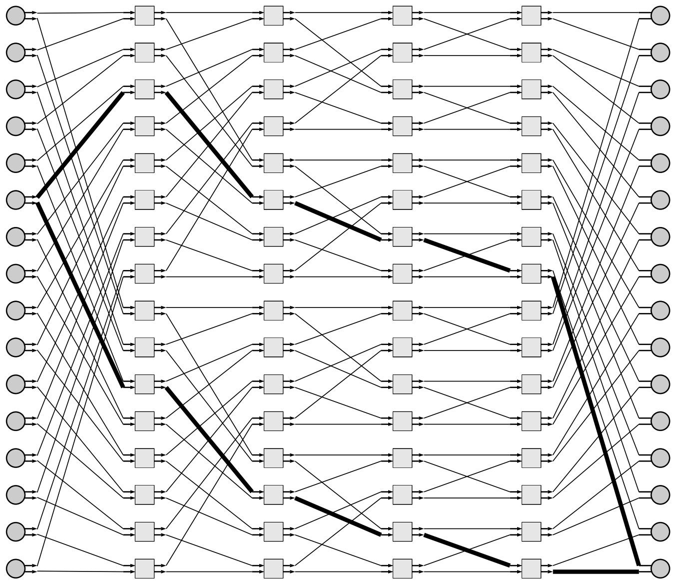
While indirect routing will allow messages to eventually reach their destination, they do so at an increase in latency. Latency increases due to several effects. First, since messages must cross the network multiple times. Additional overhead is generally required to allow indirection and process messages requiring re-routing. Also, contention latency is increased since each indirected message consumes network bandwidth on each hop through the network.

3.3.2 k-ary-n-cubes and Express Cubes

Direct, cube-based networks, like the k-ary-n-cube or the express cube, function by indirect routing. Each node is connected to $O(k)$ neighbors in a regular pattern and all routing is achieved by sending the message to a neighbor node which, generally, moves the message closer to the desired destination. At every hop, the message has a choice of paths to take to the destination, many of which would require the same transit and switching latency. The underlying network thus provides the requisite multiple paths. It is then up to the routing algorithm to efficiently utilize them. If our routing algorithm is omniscient about faults in the network, it can always find the shortest path between points in a faulty network. For many faults, the length of the shortest paths between close nodes will increase. However nodes which are further apart will see no increase in transit or switching latency. The more distant two nodes are from each other, the more minimum length paths there will be between them.

3.3.3 Multiple Networks

A simple technique for adding adding fault tolerance to a network which works for all kinds of networks is to simply replicate a base network. We give each node a connection to each of the networks. As long as there is a non-faulty path on some network between any pair of nodes which must communicate, normal communication may occur with no degradation in switching or transit latency. The originating node need only choose which network to use for each message it needs to deliver. Additionally, the existence of multiple networks increases the bandwidth available in the network and hence can reduce contention latency if utilized efficiently. Unfortunately, the gain in



Two four-stage networks connecting 16 endpoints are attached together at the endpoints. Each component is a 2×2, dilation-1 crossbar.

Figure 3.13: Replicated Multistage Network

fault-tolerance is small compared to the costs. Each additional path through the network requires that we construct a complete copy of the original network. Multiple, multistage style networks are used in the telecommunications field to minimize contention and increase available bandwidth over single-path networks [Hui90]. Figure 3.13 shows a 2-replicated bidelta network.

Replicated networks do have one advantage over pure indirect routing schemes including most cube style networks. With multiple networks, each node does have multiple connections both to and from the network. As noted in Section 2.1.2 multiple network i/o connections are key to avoiding a single point of failure which may sever a node completely from the interconnection network.

3.3.4 Extra-Stage, Multistage Networks

When using multistage interconnection networks one can construct *extra-stage* networks with more switching stages than are actually required to uniquely specify a destination ([LP83], [CYH84] *et. al.*) (See Figures 3.10 and 3.14). The set of routing specifications that reach the same physical destination defines a class of equivalent paths. So long as one path of each such class remains intact in a faulty extra-stage network, any endpoint will be able to successfully route to its destination. The extra stages in these schemes result in larger switching and transit latencies than the corresponding baseline network, even in the absence of faults.

If extra stages are added, but the single connection into and out-of each node is retained, extra-stage networks retain a single-point of failure where the nodes connect to the network. To eliminate

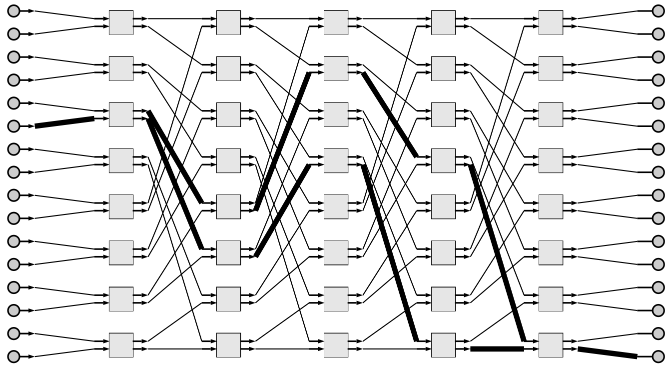


Figure 3.14: Extra Stage Network

this problem, the extra-stage network should be constructed such that multiple network endpoints can be assigned to each node of the network.

3.3.5 Interwired, Multipath, Multistage Networks

Multibutterfly style, multipath networks are multistage networks which use dilated crossbar routing components. In addition to being characterized by the radix of the switching element, each dilated crossbar router is characterized by its *dilation*, d. The dilation is the number of logically equivalent outputs in each distinct direction. With a dilation greater than one, redundant routing is provided in each routing direction. Figure 3.11 shows an example of such a network. Figure 3.15 shows some configurations for the dilated routing elements used in Figure 3.11.

This class of multipath networks has a large number of distinct paths between each pair of nodes. The number of different switches in a stage which can be used to route between any pair of routers increases toward the center of the network. Up to the center of the network, the number of routers in any path grows by a factor of the dilation with each successive stage. Past the center of the network, the sorting function performed by the network limits the number of routers in the path to the desired destination. For those later stages, all routers which are in the path to the destination are candidates for use in routing any connection.

For a given number of node connections, the multibutterfly style networks generally have more paths than the comparable replicated network. Consider a k-replicated network. A multipath network can be constructed from the k-replicated network by taking each of the k routers in the same location in each of the k-replicated network and creating one dilated router out of them with dilation, $d = k$. This will give us a multibutterfly style network. Note that in the replicated network, we were only able to chose which resources to use when the message entered the network. In the multibutterfly network we have the option of switching between networks at each routing stage. Thus, there are many more paths through the multibutterfly networks. The fine details of how one wires these redundant paths are discussed in Section 3.5.

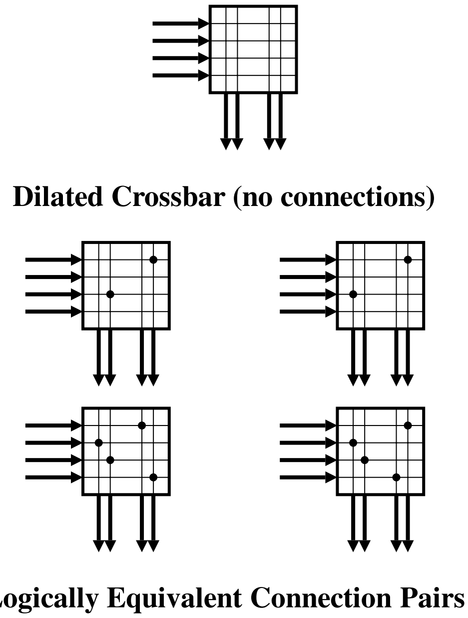


Figure 3.15: 4×2 Crossbar with a dilation of 2

By constructing fat-tree networks using dilated crossbar routers, it is possible to build multipath, fat-tree networks which exhibit the same basic properties. The tree networks will need multiple connections into and out-of the network to avoid single points of failure. Connections made through higher tree-levels have more paths between the source and the destination as they traverse more dilated routers in the network.

3.4 Robust Networks for Low-Latency Communications

Given our need for fault tolerance and low latency, the classes of networks which are most attractive are express cubes and multipath, fat-tree networks. For smaller networks, k-ary-n-cubes and flat multipath, multistage networks are also worth considering. Because of the acyclic nature of multistage routing networks, it is easier to devise robust and efficient routing schemes for this class of networks. Consequently, we will focus on multistage networks for the remainder of this document.

3.5 Network Design

This section discusses many of the issues relevant to designing a high-performance, robust, multipath, multistage routing network. The space of possible multipath networks is quite large, and some of the decisions made when selecting a particular network can make a significant difference in the fault tolerance and performance of the network. In addition to the basic parameter selection,

N	total number of nodes on the network
ni	input ports from each node to the network
no	output ports from each node to the network
i	input ports per router
o	output ports per router
r	router radix
d	router dilation
w	channel width

Table 3.2: Network Construction Parameters

the detailed network wiring scheme can have a notable affect on the performance of the resulting network. Many of the wiring issues are easier to describe and understand using small, flat, multipath, multistage interconnection networks. As a result, the examples and development which follow are given in terms of this class of networks. Nonetheless, the same design principles apply when developing multipath, fat-tree networks.

3.5.1 Parameters in Network Construction

Table 3.2 summarizes several parameters which will be used in this section when characterizing a network. Radix and dilation were introduced in Sections 3.1.5 and 3.3.5. ni and no quantify the number of connections between each node and the network. i and o are the number of connections in and out of each router. Generally, $i = o = r \cdot d$. Since the number of inputs and the number of outputs on the routing components are the same, we say the routers are *square*. When we use square routers, the aggregate bandwidth between stages in flat, multistage networks remains constant.

3.5.2 Endpoints

The network endpoints are the weakest link in the network. If we are designing a network with a yield model in mind, in the worst case, we can sustain only min(ni, no) faults. If we are designing a network with a harvest model in mind, in the worst case each min(ni, no) faults will remove an additional node from the operational set.

Once ni and no are chosen, we must also ensure that these connections are utilized effectively. Particularly, to maximize robustness, each must link connect to a distinct routing component in the network. Note, for instance, in the network shown in Figure 3.11, that dilation-1 routers are used in the final stage of the network. These dilation-1 routers are used to achieve maximal fault tolerance by ensuring that the maximum number, $no = 2$, of distinct routers provide output connections from the network to each node. Figure 3.16 shows another alternative for using dilation-1 routers in the final stage. Rather than using d times as many routers with dilation-1 and the base radix unchanged, the network in Figure 3.16 uses routers which increase the radix by a factor equal to the dilation (*i.e.* $r_{final\_stage} = o = r \cdot d$).

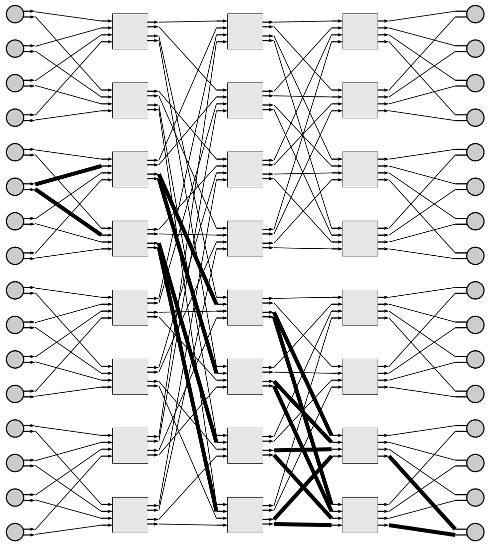


Figure 3.16: 16×16 Multibutterfly Network with Radix-4 Routers in Final Stage

3.5.3 Internal Wiring

Inside a multipath network, we have considerable freedom as to how we wire the multiple paths between stages. As described in Section 3.1.5, multistage networks operate by successively subdividing the set of potential destinations at each stage. All inputs to routing components in the same equivalence class at some intermediate network stage are logically equivalent since the same set of destinations can be reached by routing through those components. If we exercise this freedom judiciously, we can maximize the fault-tolerance and minimize the congestion within the network, and hence minimize the effects of congestion latency.

Path Expansion

A simple heuristic for achieving a high degree of fault tolerance is to wire the network to maximize the *path expansion* within the network. That is, we want to select a wiring which allows the connection between any two endpoints to traverse the maximum number of distinct routing components in each stage. Maximizing path expansion improves fault-tolerance by maximizing the redundancy available at each stage of the network.

Let S be the total number of routing stages in the network. The number of paths between a single source-destination pair expands from the source into the network at the rate of dilation, d. Thus, we have $p_{in}(s)$, the number of paths to stage s given by Equation 3.2.

$$p_{in}(s) = ni \times d^{[s-1]} \tag{3.2}$$

After a stage in the network, the paths will have to diminish in order to connect to the proper destination. Looking backward from the destination node, we see that the paths must grow as the network radix r. This constraint is expressed as follows:

$$p_{out}(s) = no \times r^{[(S+1)-s]} \tag{3.3}$$

s	1	2	3	4	5
$p(s)$	2	4	8	4	2

Table 3.3: Connections into Each Stage

These two expansions must, of course, meet at some point inside the network. This occurs when p_{in} and p_{out} are equal. Let us call this turning point stage s'. s' can be determined as follows:

$$\begin{aligned}
p_{out}(s') &= p_{in}(s') \\
ni \times d^{[s'-1]} &= no \times r^{[(S+1)-s']} \\
s' &= \frac{(S+1) \cdot \ln(r) + \ln(no) + \ln(d) - \ln(ni)}{\ln(d) + \ln(r)} \\
s' &= \frac{(S+1) \cdot \ln(r) + \ln\left(\frac{no \cdot d}{ni}\right)}{\ln(d \cdot r)}
\end{aligned} \tag{3.4}$$

Once Equation 3.4 is solved for s', we can quantify the number of connections into each stage of the network by Equation 3.5.

$$p(s) = \begin{cases} ni \times d^{[s-1]} & s < s' \\ \min(ni \cdot d^{[s-1]}, no \cdot r^{[(S+1)-s]}) & s = s' \\ no \times r^{[(S+1)-s]} & s > s' \end{cases} \tag{3.5}$$

Note that Equation 3.5 expresses the maximum achievable number of paths between stages for a single source-destination pair. This is effectively an upper bound on the path expansion in any dilated multipath network. The total number of distinct paths between each source and destination simply grows as Equation 3.2 and is thus given by Equation 3.6.

$$p_{total}(s) = ni \times d^{[S-1]} \tag{3.6}$$

For example, consider the network in Figure 3.11 ($ni = no = r = d = 2$, $S = 4$). Solving Equation 3.4 for s', we find $s' = 3$. The number of connections into each stage can then be calculated as shown in Table 3.3. The total number of paths is simply $2 \times 2^3 = 16$. Noting Figure 3.11, we see it does achieve this maximum path expansion for the highlighted path; the paths between all other source and destination pairs in Figure 3.11 also achieve this path expansion.

α-β Expansion

Unfortunately, path expansion can be a naive metric when optimizing the aggregate fault-tolerance and performance of a network. Path expansion looks at a single source-destination pair and tries to maximize the number of paths between them. If we only considered path expansion in selecting a network design, many nodes could share the same sets of routers and connections in their paths through the network. This sharing would lead to a higher-degree of contention. Additionally, when faults accumulate in the network, a larger number of nodes are generally isolated from the

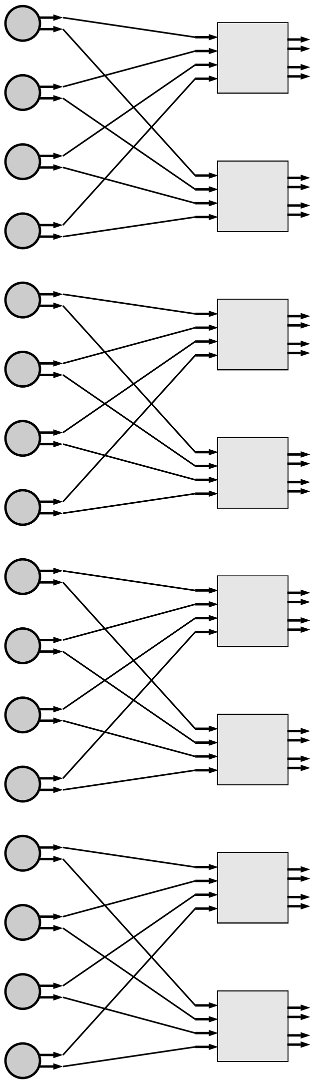
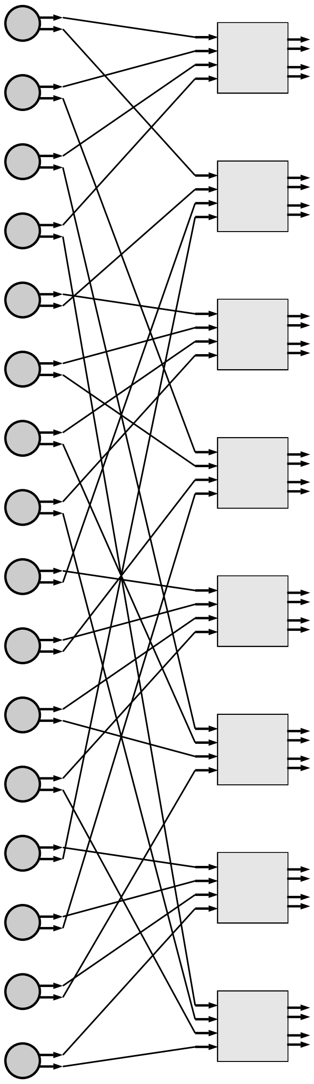



Figure 3.17: Left: Non-expansive Wiring of Processors to First Stage Routing Elements

Figure 3.18: Right: Expansive Wiring of Processors to First Stage Routing Elements

rest of the network at once. Consider, for instance, the two first stage network wirings shown in Figure 3.17 and 3.18. Both wirings are arranged such that each processor connects to two distinct processors in the first stage of routing. However, the wiring shown in Figure 3.17 has four processors which share a pair of routers, whereas any group of four processors in the wiring shown in Figure 3.18 is connected to five routers in the first stage. As a result, there will generally be less contention for connections through the first stage of routers in the latter wiring than in the former.

Leighton and Maggs introduced α-β *expansion* to formalize the desirable expansion properties as they pertain to groups of nodes which may wish to communicate simultaneously [LM89]. Informally, α-β expansion is a metric of the degree to which any subset of components in one stage will fan out into the next stage. More formally, we say a stage has α-β expansion (α, β) if any subset of α components from one stage must connect to at least $\alpha \times \beta$ components in the next stage. β is thus an expansion factor which is guaranteed for any set of size α. Networks with favorable α-β expansion are networks for which the α-β expansion property holds with higher β for each value of α. The more favorable the α-β expansion, the more messages can be simultaneously routed between any sets of communicating processors, and hence the lower the contention latency.

Networks Optimized for Yield

If we cannot tolerate node loss, and hence wish to optimize the fault-tolerance of the network as a yield problem, then it makes sense to focus on achieving the maximal path expansion first, then achieving as large a degree of α-β expansion as possible. Unfortunately, there is presently no known algorithm for achieving a maximum amount of α-β expansion, so the techniques presented here are heuristic in nature.

To achieve maximum path expansion, we connect the network with the algorithm listed in Figure 3.19 [CED92]. The paths from any input to any output may fanout by no more than a factor of *d*, the dilation of the routers, at each stage. This fanout may also become no larger than the size of the routing equivalence classes at that stage. The routine *groupsz* returns the maximum fanout size allowed by both of these factors. Each stage is partitioned into *fanout classes* of this size, which are then used to calculate network wiring. The maximum path fanout described in Equation 3.5 is achieved by this algorithm for all pairs of components.

As introduced above, the last stage is composed of dilation-1 routers to increase fault tolerance. Figure 3.20 shows a deterministically-interwired network composed of radix-2 routers.

Networks Optimized for Harvest

To achieve a high harvest rate and maximize performance, we want to wire networks with a high degree of α-β expansion. As introduced above, there are no known deterministic algorithms for achieving an optimal expansion. In practice, randomized wiring schemes produce higher expansion than any known deterministic methods. [Kah91] presents some of the most recent work on the deterministic construction of expansion graphs. [Upf89] and [LM89] show that randomly wired multibutterflies have good expansion properties. The high expansion generally means there will be less congestion in the network. Additionally, Leighton and Maggs show that after k faults have occurred on a N node machine, it is always possible to harvest $N - O(k)$ nodes [LM89].

As introduced in Section 3.1.5, multistage networks operate by successively subdividing the set of potential destinations at each stage. All the inputs to routing components in the same equivalence class at some intermediate stage in the network, are logically equivalent. After the routing structure determines which set of outputs in one stage must be connected to which set of inputs in the following stage, we randomly assign individual input-output pairs within the corresponding sets. Figure 3.21 shows the core of an algorithm for randomly wiring a multibutterfly. The algorithm was first introduced in [CED92] and is based on the wiring scheme described in [LM89]. In practice,

▷ Returns the next-stage router to which to wire for maximum path expansion

wire_to_port(n,d_p,s)

▷ n=router number, d_p=dilated port number, s=router stage

$outgrpsz \leftarrow$ **groupsz**(s)

$ingrpsz \leftarrow$ **groupsz**($s+1$)

$eq\_start \leftarrow ingrpsz \times \lfloor n/(r \times d \times outgrpsz) \rfloor$

▷ offset to beginning of fanout class

$eq\_router \leftarrow ((n \times d + d_p) \bmod ingrpsz)$

▷ offset to specific chip within fanout class

return($eq\_start + eq\_router$)

▷ Calculates size of fan-out class

groupsz(s)

$expansion \leftarrow ni \times d^{s+1}$ ▷ maximum fanout due to dilation

$eq\_class \leftarrow no \times r^{S+1-s}$ ▷ equivalence class size

return(min($expansion$, eq_class))

This algorithm generates a network designed to maximize path expansion. Each endpoint will have the maximum number of redundant paths possible through this type of network (boundary cases omitted for clarity).

Figure 3.19: Pseudo-code for Deterministic Interwiring

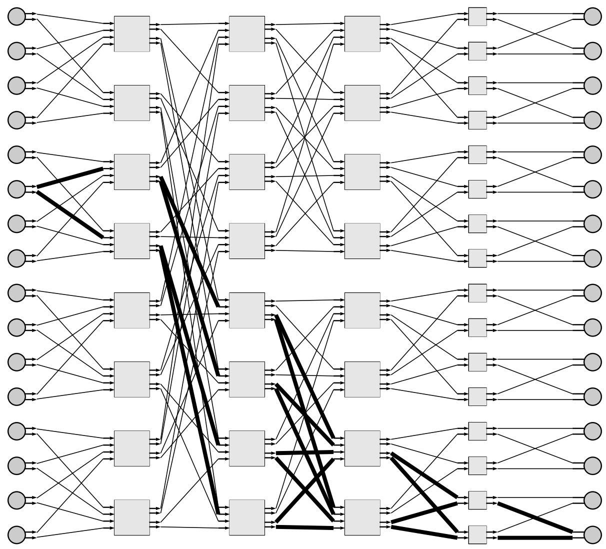


Figure 3.20: 16×16 Path Expansion Multibutterfly Network

```
▷ in_set contains all the input ports of a single equivalence class in the next stage.
▷ connections is an array matching in_ports and out_ports, initially empty.
▷ out_ports_list lists the output ports of a single equivalence class in the
▷     current stage.
wire_eq_class(in_set, connections, out_ports_list)
  foreach out_port
      in_port ← choose and remove a random input port from in_set
      while(connected(router#(in_port), router#(out_port), connections))
          put in_port back in in_set
          in_port ← choose and remove a random input port from in_set
      connect(in_port, out_port, connections)
  return(connections)

connected(in_router, out_router, connections_array)
  if in_router is already connected to out_router
      return(true)
  else return(false)
```

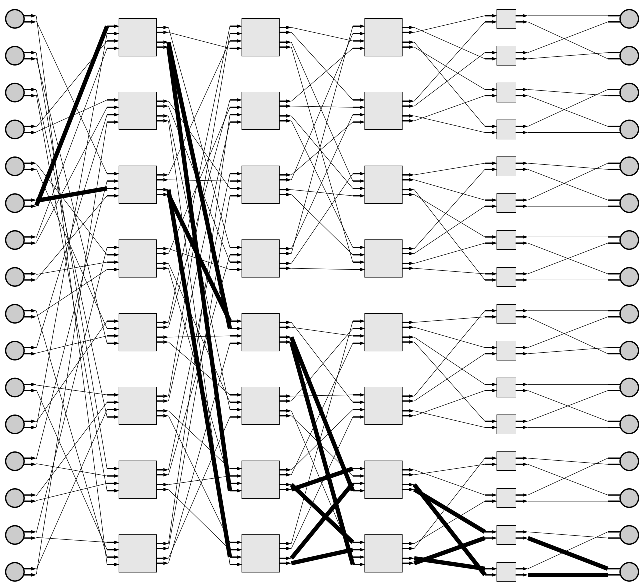
This algorithm randomly interwires an equivalence class. To interwire a whole stage, the algorithm is repeated for each class (boundary cases omitted for clarity).

Figure 3.21: Pseudo-code for Random Interwiring

one would generate many such networks, compare their performance as described in Sections 3.5.4 and 3.5.5, and pick the best one. Experience indicates that most such networks perform equivalently. The testing, however, assures that one avoids the unlikely, but possible, case in which a network with poor expansion was generated. Figure 3.22 shows a network constructed with this algorithm.

Hybrid Network Compromise

Chong observed in [CK92] that one can achieve maximum path expansion while introducing some randomized expansion to minimize congestion. The result is a network which is a hybrid between the two described above. The basic strategy used in wiring such, *randomized, maximal-fanout* networks is to further subdivide each routing equivalence class into fanout classes. Instead of randomly wiring from all outputs destined for a given equivalence class to the inputs on all routers in that equivalence class in the subsequent stage, the dilated outputs from each router are each sent to different fanout classes within the appropriate routing equivalence class (See Figure 3.23). Figure 3.24 sketches the algorithm used for wiring up these networks. Figure 3.25 shows an example of such a network.



A randomly-interwired, four-stage network connecting 16 endpoints. Each component in the first three stages is a 4×2, dilation-2 crossbar. To prevent any single component from being in an endpoint's critical path, the last stage is composed of 2×2, dilation-1 crossbars.

Figure 3.22: Randomly-Interwired Network

3.5.4 Network Yield Evaluation

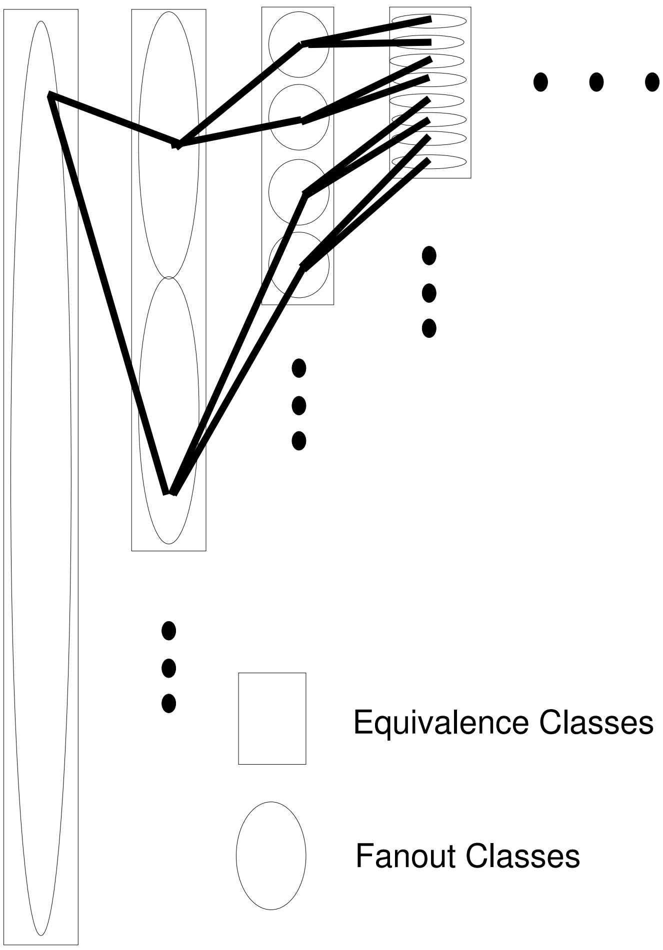
Yield

As a simple metric for evaluating the yield characteristics of these multipath networks, we consider the probability that a network remains completely connected given a certain number of randomly chosen router faults. These Monte Carlo experiments model only complete router faults to show the relative fault-tolerant characteristics of these networks while containing the size of the fault-space which must be explored.

The experiment proceeds by placing one randomly chosen fault at a time until the network becomes incomplete. The basic process is repeated on the same network for enough trials to achieve statistically significant results. Results are tabulated to approximate the probability of network completeness for each fault level. We also derive the expected number of faults each network can tolerate.

Because the routing components in the final stage of our multipath networks are half the size of routers in the previous stages, we assign two such routers to one physical component package and label both routers faulty if the physical component is chosen to be faulty. Furthermore, the two routers are assigned so that removing any such pair will not cut off an endpoint. We make this assignment so that fault increments will be of constant hardware size. This assignment also simulates how the pair of 4×4, dilation-1 routers in an RN1 routing component (See Chapter 8) may be assigned.

We generated three-stage and four-stage networks for each of the types of networks described



The above figure shows how to achieve maximal fanout while avoiding regularity. The routers shown are radix-2 and dilation-2. At stage s, we divide each routing equivalence class into $ni \cdot d^{(s-1)}$ fanout classes until each fanout class contains a single router. Random wirings are chosen between appropriate fanout classes to form fanout trees. The disjoint nature of fanout classes ensures that fanout-trees will have physically distinct components.

Figure 3.23: Randomized Maximal-Fanout (diagram from [CK92])

above, each connecting 64 and 256 endpoint nodes respectively. Each endpoint has two connections to and from the network ($ni = no = 2$) to provide for the minimal amount of redundancy necessary to achieve fault tolerance. Every network uses radix-4 routers of dilation-2 and dilation-1 and hence could be implemented using the RN1 component. All the networks with a given number of stages contain the same number of components. Network wiring is solely accountable for the fault tolerance and performance differences of these networks.

For each network, the yield probability of the network is plotted against the number of uni-

wire_stage(s) ▷ s=routing stage

$prev\_expansion \leftarrow ni \times d^{s+1}$ ▷ maximum fanout to stage s due to dilation

$prev\_eq\_class \leftarrow no \times r^{((S+1)-s)}$ ▷ equivalence class size at stage s

$prev\_fanout\_class \leftarrow \frac{prev\_eq\_class}{prev\_expansion}$ ▷ fanout class size at stage s

$expansion \leftarrow ni \times d^{(s+2)}$ ▷ maximum fanout to stage $s+1$ due to dilation

$eq\_class \leftarrow no \times r^{((S+1)-(s+1))}$ ▷ equivalence class size at stage $s+1$

$fanout\_class \leftarrow \frac{eq\_class}{expansion}$ ▷ fanout class size at stage $s+1$

if $(fanout\_class \geq 1)$
    foreach fanout equivalence class in stage s
        create $(r \times d)$ different output-port lists,
            one for each output from a routing switch
            ▷ each of these lists will contain $prev\_fanout\_class$ ports
        foreach output-port list identified, identify the $fanout\_class$
            routers in stage $s+1$ to which these ports should be
            connected – the inputs on these routers make up the
            corresponding in-port list
        Use **wire_eq_class** to randomly interconnect each in-port list
            to each corresponsding output-port list
else
    foreach equivalence class in stage s
        create r different output-port lists,
            one for each logically distinct output direction from a router
            ▷ each of these lists will contain $(prev\_eq\_class \times d)$ ports
        foreach of the output-port lists identified, identify the eq_class
            routers in stage $s+1$ to which the output list should be
            connected – the inputs on these routers make up the
            corresponding in-port list
        Use **wire_eq_class** to randomly interconnect each in-port list
            to each corresponsding output-port list

This algorithm describes how to wire random, maximal-fanout networks using the random interwiring algorithm, **wire_eq_class** shown in Figure 3.21 (boundary cases omitted for clarity).

Figure 3.24: Pseudo-code for Random, Maximal-Fanout Interwiring

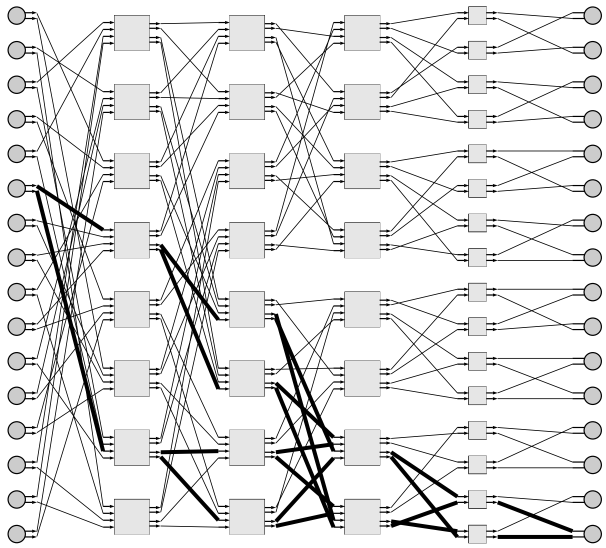
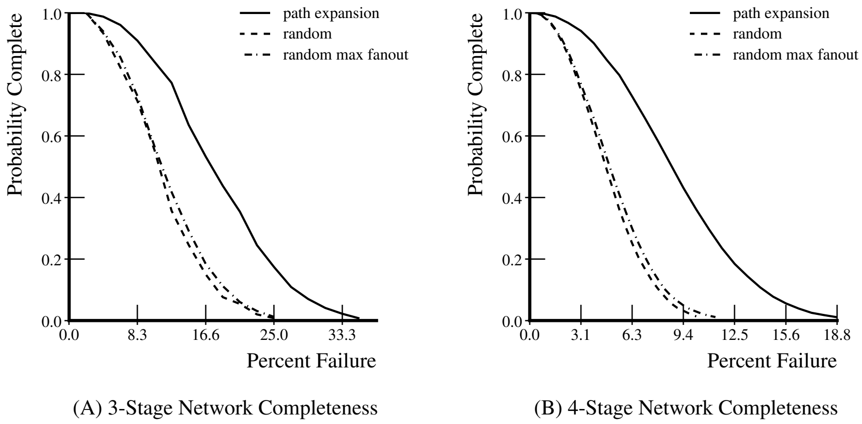


Figure 3.25: 16×16 Randomized, Maximal-Fanout Network



The probability that a network with a given number of faults is complete for the randomly-interwired, path expansion, and random maximal fanout, 3-stage and 4-stage networks. (A) Each 3-stage network uses 48 radix-4 components to interconnect 64 endpoints. (B) Each 4-stage network uses 256 radix-4 components to interconnect 256 endpoints.

Figure 3.26: Completeness of (A) 3-stage and (B) 4-stage Multipath Networks

formly distributed random faults. Results for the three-stage and four-stage networks are shown in Figure 3.26. The expected number of faults that each network can tolerate is summarized in Table 3.4.

Network		Total	Test	Expected Failure Tolerated		Error Bound
Stages	Wiring	# Comp.	Trials	# Faults	% Network	# Faults
3	Random	48	1000	5.0	10%	0.063
3	Path Expansion	48	1000	8.1	16%	0.079
3	Random Max Fanout	48	1000	5.2	11%	0.060
4	Random	256	5000	11.8	4.6%	0.075
4	Path Expansion	256	5000	22.6	8.8%	0.130
4	Random Max Fanout	256	5000	12.5	4.9%	0.069

The above table shows the expected number of faults each network can tolerate while remaining complete. Each network was fault tested as described in section 3.5.4 for the indicated number of trials.

Table 3.4: Fault Tolerance of Multipath Networks

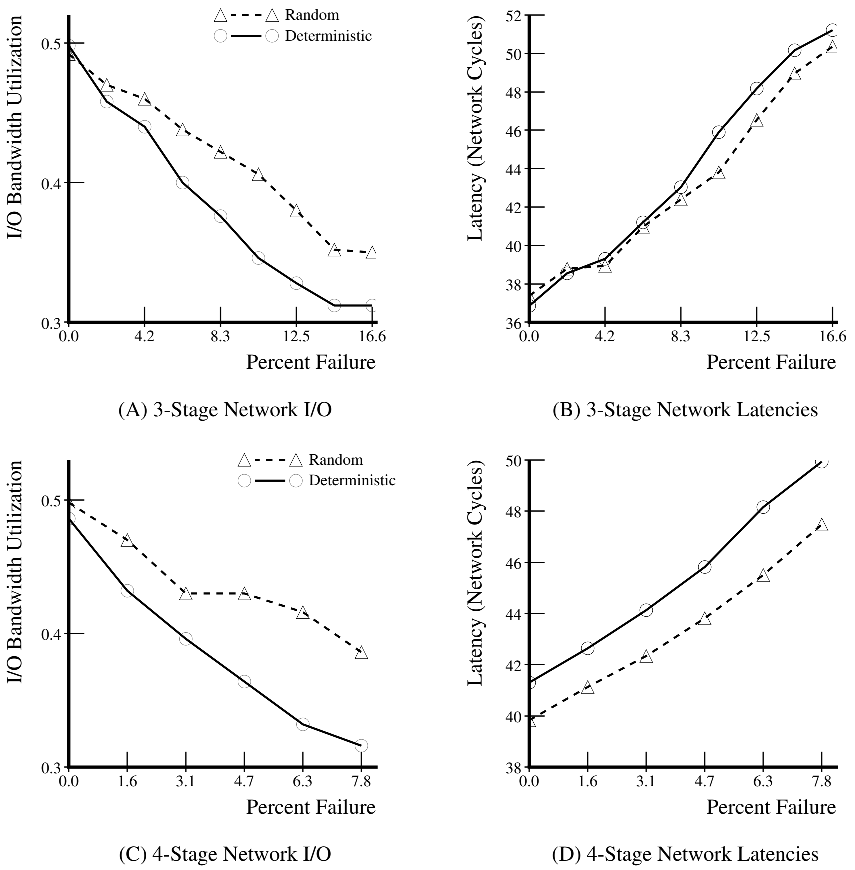
Wiring Extra-Stage Networks for Fault Tolerance

It is worth noting that we can achieve the same fault tolerance as indicated in this section without using dilated routers. Consider replacing each of the dilated routers used in the networks above with an equivalently sized (*i.e.* same number of inputs, i, and same number of outputs o) dilation-1 router (*i.e.* $r = o$, $d = 1$). The network we end up with is an extra-stage network since we have increased the radix while leaving the number of stages the same. Form a fault tolerance perspective, this resulting extra-stage network has the same yield probability as the corresponding dilated network. As a result, the network wiring issues introduced in Section 3.5.3 apply equally well to extra-stage, multistage networks as they did to dilated, multistage networks.

Performance Degradation in the Presence Faults

We are also interested in knowing how robust the network performance is when faults accumulate. To that end, we consider a simple synthetic benchmark on the complete networks at various fault levels. This gives us some idea of the effects of congestion in the network, as well as how the faults affect the overall performance of the network. The routing protocol detailed in Chapter 4 is used for all of these simulations.

Our synthetic benchmark, FLAT24, was designed to be representative of a shared-memory application. FLAT24 uses 24-byte messages with a uniform traffic distribution. FLAT24 generates 0.04 new messages per router cycle based on the assumption that the network is running at twice the clock rate of the processor and a data-cache miss rate of 15%. The application is assumed to barrier synchronize every 10,000 cycles, or every 400 messages. Modeling barrier synchronization exposes the effects of localized degradation. If a small number of nodes have significantly fewer paths through the network than the rest of the nodes, the nodes with less connectivity will fall behind those with more. In a real application, these nodes will tend to hold up the remainder of the application since they are not progressing as rapidly as the rest of the nodes in the network. The periodic barrier synchronization is a simple and pessimistic way of limiting the extent to which



Comparative I/O bandwidth utilization and latencies for 3-stage and 4-stage random and path expansion networks on FLAT24. Recall from Table 3.4 that expected percentages of failure tolerated by random and deterministic networks are, respectively: 10% and 16% for 3-stages; and 4.6% and 8.8% for 4-stages. Note that the performance degradation appears to level off because only complete networks are measured. Although the surviving networks suffer less degradation as percentage of failure increases, the number of surviving networks is becoming substantially smaller.

Figure 3.27: Comparative Performance of 3-Stage and 4-Stage Networks

nodes may get ahead of each other and hence exposing the effects of this localized degradation. This synthetic application and the simulations in general are described in detail in [Cho92]; most relevant details are reprinted in Appendix A.

Figure 3.27 shows the performance degradation of FLAT24 on the surviving networks as various fault levels. Here latency is the average time from when a message is injected into the network until the time its reply and acknowledgment are received. I/O bandwidth utilization measures the average fraction of network outputs which are receiving or replying to successful message transmissions at any point in time. This provides a measure of the useful bandwidth provided by the network.

1 Begin with all nodes live.
2 Determine the *I/O-isolated nodes* and remove them from the set of live nodes.
3 Each faulty chip leading to at least one live node is declared to be blocked. Propagate blockages from the outputs to the inputs according to the definition of blocking given below.
4 If all of a node's connections into the first stage of the network lead to blocked chips, remove the node from the set of live nodes.

This algorithm harvests the nodes in a networks which retain good connectivity in the presence of faults. The algorithm will sacrifice nodes which still retain weak connectivity in order to maximize the performance of the harvested network.
▹ A router is said to be *blocked* if it does not have at least one unused, operational output port in each logical direction which leads to a router which is not blocked.
▹ An *I/O-isolated node* is a node which has lost all of its input connections to the first stage of the network or all of its output connections from the final stage of the network.

Figure 3.28: Chong's Fault-Propagation Algorithm for Reconfiguration

3.5.5 Network Harvest Evaluation

To evaluate the harvest rate of a network with faults, we use the reconfiguration algorithm suggested by Chong in [CK92]. This reconfiguration algorithm identifies all nodes with "good" network connectivity. The algorithm does not necessarily identify all nodes which retain full connectivity in the network as available in the harvested network. Since it is the overall system performance that matters, not simply the number of nodes available for computation, Chong observes that better overall performance is achieved when nodes with low bandwidth into the network are eliminated from the set of nodes used for computation. Chong's algorithm is summarized in Figure 3.28.

Figure 3.29 shows the harvest rate for a 5-stage, radix-4, dilation-2 (1024 node) network. Also shown is the degradation in application performance assuming that the application can be efficiently repartitioned to run on the surviving processors.

3.5.6 Trees

Fat-trees have the same basic multipath, multistage structure as the multistage networks described so far in this section. It is easiest to think of each fat-tree network as two sub-networks. One sub-network routes from the root of the tree down to the leaves. This portion looks almost identical to the routing performed by the multistage networks that have been discussed. Particularly, this downward routing network performs the same recursive subdivision of possible destinations at each successive routing stage. The other sub-network allows connections to be routed up to the appropriate intermediate tree level and then cross over into the down routing sub-network. In fact, we could think of the flat, multistage networks as a tree which had a degenerate up and crossover sub-network. In these networks, the up network is simply set of wires which connect all

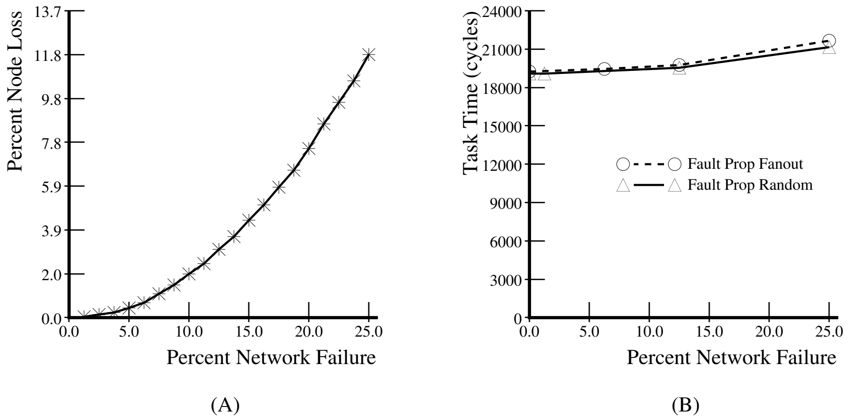


Figure (A) shows the percentage of node loss under the criterion of fault-propagation. Figure (B) compares the performance of the randomly wired multibutterfly with the randomized maximal fanout network.

Figure 3.29: Fault-Propagation Node Loss and Performance for 1024-Node Systems (from [CK92])

network input connections directly into the root of the tree. It is the upward routing portion of the tree networks which give them their ability to exploit locality. Two nodes close to each other can cross over low in the tree structure and avoid traversing a large number of routers or consuming bandwidth near the root of the tree.

Fat-Trees

Fat-trees are distinguished from arbitrary tree based networks in that the interconnection bandwidth increases towards the root of the tree. The internal tree connections closer to the root require more bandwidth because they service a larger number of nodes below them. For instance, in a binary fat-tree the root of the tree will see all traffic that is not constrained solely to either half of the machine. The property that makes fat-tree structures most attractive is their *universality* property. Leiserson shows that, when the rate of bandwidth growth in the fat-tree is chosen properly, fat-trees can be *volume universal*. That is, a properly constructed volume $\Theta(V)$ fat-tree network can simulate any volume $\Theta(V)$ network in polylogarthmic time [Lei85] [Lei89] [GL85].

The key observation in demonstrating the universality of various fat-tree structures, is that the physical world places constraints on the ratio between the volume of a region and the wire channel capacity, and hence bandwidth, which can efficiently enter or leave that volume. The channel capacity into a volume is limited by the surface area surrounding that volume. As we scale up to larger systems and hence larger volumes, the surface area of a given volume, V, grows only as $\Theta(V^{\frac{2}{3}})$. To remain volume efficient, the channel capacity can only grow as $\Theta(V^{\frac{2}{3}})$. If the channel capacity grows faster than this, then the size of the system packaging is limited by the channel capacity between regions rather than the volume of the system being packaged. As the system becomes large, pieces of the system must be placed further apart due to the interconnection bandwidth constraints. As a result, the universality property will not hold because the number of processors per unit volume is decreasing as the system increases in size. If the channel capacity grows slower than this, the universality property does not hold due to insufficient channel capacity to support the potential message traffic. For binary fat-trees Leiserson shows that channel capacity should increase as $\sqrt[3]{4}$ per stage toward the root of the tree in order to achieve volume universality.

Fat-trees also have considerable flexibility. When other pragmatic issues dictate a structure that allows more channel capacity, and consequently more bandwidth, at higher tree levels than is appropriate for volume-universality, the basic fat-tree structure can accommodate the increased interstage capacity. This additional channels will allow additional fault tolerance and lower the network's contention latency, γ.

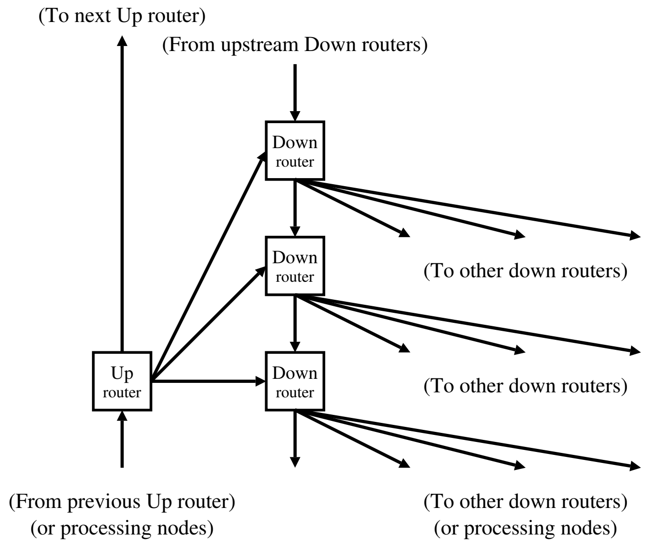
Building Fat-Trees

We can build fat-tree networks with the same fixed-size, dilated routers which we have used to construct flat, multistage networks. The use of such routers in the down sub-network is obvious since the down sub-network performs the same sorting function as in the flat networks. Here, the router radix defines the arity of the fat-tree. The up routing sub-network needs to expand the possible destinations so that a given route may make use of a large portion of the bandwidth at some higher tree stage. The up routing sub-network also needs to provide switching which allows periodic crossover to the down routing network. At the same time, the bandwidth between tree levels needs to be controlled to match the application requirements as described in the previous section. Just as with the flat-multistage networks, the endpoint connections are weak links and one generally wants to organize networks with multiple network connections per endpoint. Similarly, the issues of wiring the internal stages for fanout apply equally well here.

As an example, consider building a fat-tree using radix-4, dilation-2 routing components. The down sub-networks is a quaternary tree. In the up sub-network, we use the routing components to switch between upward routing and crossover connections into the down sub-network. We can take advantage of the radix-4 switching provided by the routing component to route to several crossover connections at a single switching stage. As a result, we effectively create short-cut paths in the up routing tree. Figure 3.30 shows how a radix-4 up router can switch to three successive tree-stages and provide upward connection in the tree. Since each up router in the up sub-tree services three down-tree stages, the route to the root is only $\frac{1}{3}\log_4 N$ long. Figures 3.31 and 3.32 shows the logical connectivity for the up and down sub-trees using the short-cut crossover scheme shown in Figure 3.30.

3.5.7 Hybrid Fat-Tree Networks

Fat-trees allow us to exploit a considerable amount of locality at the expense of lengthening the paths between some processors. Flat, multistage networks fall at the opposite extreme of the locality spectrum where all nodes is uniformly close or distant. Another interesting structure to consider is a *hybrid fat-tree*. A hybrid fat-tree is a compromise between the close uniform connections in the flat, multistage network and the locality and scalability of the fat-tree network. In a hybrid fat-tree, the main tree structure is constructed exactly as described in the previous section. However, the leaves of the hybrid fat-tree are themselves small multibutterfly style networks instead of individual processing nodes. With small multibutterfly networks forming the leaves of the hybrid fat-tree,



Shown above is a cross-sectional view of a fat-tree network showing a switching node in the up routing sub-tree and a down router in each of the three successive tree stages to which this up router can crossover. As shown, each router is a radix-4 routing component. Only a single output is shown in each logical direction for simplicity. With dilated routers, each dilated connection would be connected to different routers in the corresponding destination direction for fault tolerance.

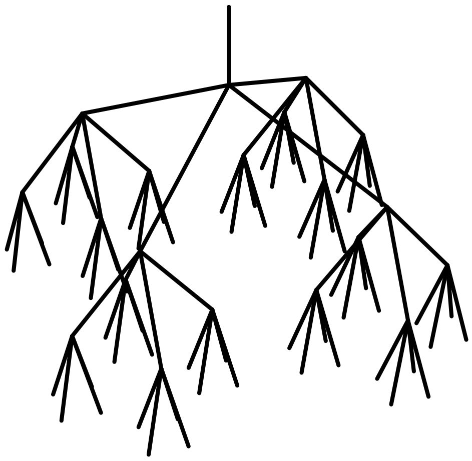
Figure 3.30: Cross-Sectional View of Up Routing Tree and Crossover

small to moderate clusters of processors can efficiently work closely together while still retaining reasonable ability to communicate with the rest of the network.

The flat, leaf portion of the network is composed of several stages of multibutterfly style switching. Each stage switches among r logical directions. The first stage is unique in that only $(r-1)$ of the r logical directions through the first stage route to routers in the next stage of the multibutterfly. The final logical direction through the first routing stage connects to the fat-tree network. The remaining stages in the leaf network perform routing purely within the leaf cluster. To allow connections into the leaf cluster from the fat-tree portion of the network, one r-th of the inputs to the first routing stage come from the fat-tree network rather than from the leaf cluster processing nodes. Figure 3.33 shows a diagram of such a leaf cluster. This hybrid structure was introduced in [DeH90] and is developed in more detail there.

3.6 Flexibility

In Section 2.8, we raised some concerns about how well a network topology can be adapted to solve particular applications. Having reviewed the properties of these networks, we can answer many of the questions raised.



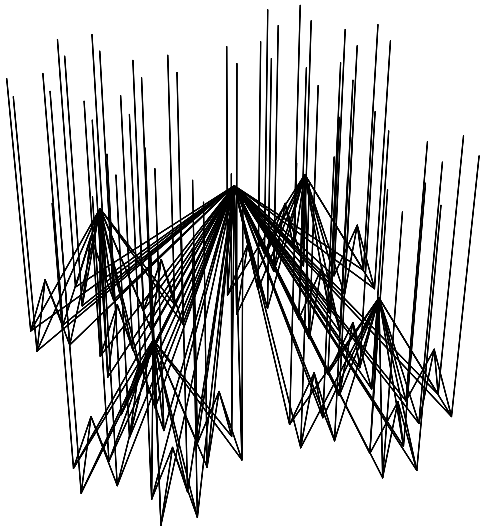


Figure 3.31: Connections in Down Routing Stages (left)

Figure 3.32: Up Routing Stage Connections with Lateral Crossovers (right)

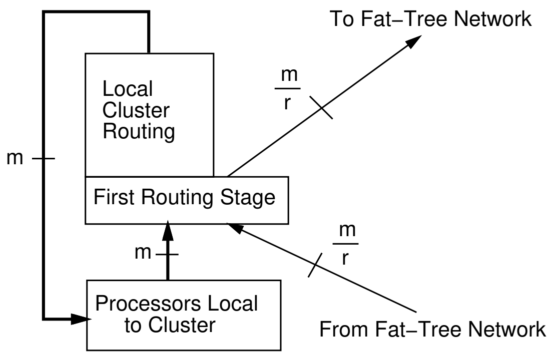


Figure 3.33: Multibutterfly Style Cluster at Leaves of Fat-Tree

- How do we provide additional bandwidth for each node at a given level of semiconductor and packaging technology?

  If we assume that the semiconductor technology limits the interconnect speed, then we are trying to increase the bandwidth in an architectural way. With both flat multipath networks and multibutterflies, we can easily increase the bandwidth into a node by increasing the number of connections to and from the network, (*i.e.* ni and no). This also has the side effect of increasing the network fault tolerance.

- How do we get more/less fault tolerance for applications which have a higher/lower premium for faults

  The simple answer here is to increase the number of connection to and from the network, since this is the biggest limitation to fault tolerance. Using higher dilation routers will provide more potential for expansion and hence better fault-tolerance. Hybrid schemes which use extra-stages in a dilated network will also serve to increase the number of paths and hence the fault-tolerance of the network.

- How do we build larger (smaller) machines?

  The scalability of the schemes presented here, allow the same basic architecture to be used in the construction of large or small machines. For very large machines, we saw that fat-trees or hybrid fat-trees are the best choice. For smaller machines, we saw that multistage networks may provide better performance. In between, the details of the technologies involved as well as other system requirements will determine where the crossover lies.

- How can we decrease latency? at what costs?

  We have control over the latency in several forms. The switching latency (T_s) is directly controlled by the router radix, r. Increasing the radix of the router will lower the number of stages which must be traversed and tend to decrease latency. However, the router radix is limited by the pin limitations of the routing component. Increasing the radix will either require an increase in die-size and package pin count (and hence cost), or a decrease in dilation or data channel width. Decreasing dilation will tend to reduce fault-tolerance and increase congestion. Decreasing the data channel width decreases the bandwidth and thus increases both congestion and the message transmission time ($T_{transmit}$). By increasing the channel width, we can decrease transmission time; again, this will either increase die-size and cost, or require the decrease in radix or dilation. Finally, we can decrease congestion by increasing router dilation or increasing the aggregate network bandwidth. Increasing the dilation, again must be traded off against radix, channel width, and cost. Increasing the number of inputs and outputs to the network will increase the aggregate bandwidth of the network at the cost of more network resources.

3.7 Summary

In this section, we have examined network topologies suitable for implementing robust, low-latency interconnect for large-scale computing. We saw that express-cubes and fat-trees have the best asymptotic characteristics in terms of latency and growth. We also saw how the multipath nature of these networks allows the potential for tolerating faults within the networks. For many networks, we see that architectures which tolerate network fault do not necessarily require additional network latency. The only increase in network latency results from the lower bandwidth available in the faulty network. We examined detailed issues relevant to wiring multistage networks. We found that good performance results from wiring the network to avoid congestion and that randomized techniques provide the best strategy currently known for achieving such network wirings.

3.8 Areas to Explore

We have, by no means, explored all the issues associated with selecting the optimal network for every application. The following is a list of a few interesting areas of pursuit:

1. It is hard to provide a final head-to-head latency comparison between networks without a good quantification of the effects of congestion in various networks. As mentioned in Section 2.4, this is particularly difficult because the effects of congestion are highly dependent upon the network usage pattern needed by the application and the detailed network topology. A good quantification of congestion applicable across a wide range of networks and loading patterns would go a long way toward helping engineers design and evaluate routing networks

2. In Section 3.5.4 we demonstrate that a class of extra-stage networks has the same fault-tolerant properties as dilated networks. These networks will generally have lower performance due to the necessity to make detailed routing decisions at the node rather than inside the network where the freedom can be used to minimize blocking. It would be worthwhile to quantify the magnitude of the performance improvement offered by the dilated routing components.

3. Express-cubes have the same asymptotic network characteristics as fat-trees. We avoid detailed consideration of these networks at this point due to the difficulties associated with efficiently routing on such networks in the presence of faults. In the next chapter, we will show how to route effectively with faults for fat-tree and multistage networks. It would be interesting to see comparable routing solutions for express-cubes.

4. Routing Protocol

In the previous chapter, we saw how to construct multipath networks. The organization of these networks offers considerable potential for low-latency communication and fault-tolerant operation. To make use of this potential, we need a routing scheme which is capable of exploiting the multiple paths with low latency. In this chapter, we develop a suitable routing scheme and show how it meets these needs.

4.1 Problem Statement

As introduced in Chapter 1, we need a routing scheme which provides:

1. Low-overhead routing
2. Protocol Flexibility
3. Distributed routing
4. Dynamic fault tolerance
5. Fault identification and localization with minimal overhead

4.1.1 Low-overhead Routing

Any overhead associated with sending a message will increase end-to-end message latency. There are two primary forms of overhead which we wish to minimize:

1. Overhead data
2. Overhead processing

Overhead data includes message headers and trailers added to the message. Overhead data will diminish the available network bandwidth for conveying actual message data. Overhead processing includes the processing which must be done at each endpoint to interact with the network (*e.g.* T_p, T_w) and the processing each router must perform to properly process each data stream (*e.g.* t_{switch}). Endpoint overhead processing includes:

1. processing necessary to prepare data for presentation to the network
2. processing necessary to use data arriving from the network
3. processing necessary to control network operations

We want a protocol that satisfies the various routing requirements with minimal overhead in terms of both processing time and transmitted data.

4.1.2 Flexiblity

In the interest of providing general, reusable routing solutions, we seek a minimal protocol for reliable end-to-end message transport. Specific applications will need to use the network in many different ways. To allow as large a class of applications as possible the opportunity to use the network efficiently, the restrictions built into the underlying routing protocol should be minimized.

4.1.3 Distributed Routing

In the interest of fault tolerance, scalability, and high-speed operation, we want a distributed, self-routing protocol. A centralized arbiter would provide a potential single point of failure and have poor scalability characteristics. Rather, we need a routing scheme which can allocate routing resources and make connections efficiently in practice using only localized information. A distributed routing scheme operating on local information has the following beneficial properties:

- faults only affect a small, localized area
- routing decisions are simple and hence can be made quickly.

4.1.4 Dynamic Fault Tolerance

To provide continuous, reliable operation, the routing scheme must be capable of handling faults which arise at any point in time during operation. As introduced in Section 2.1.1, transient faults occur much more frequently than permanent faults. Additionally, for sufficiently long computations on any large machine, one or more components are likely to become faulty during the computation (*e.g.* example presented in Section 2.5).

4.1.5 Fault Identification

Although, a routing protocol which can properly handle dynamic faults can tolerate unidentified faults in the system, the performance of the routing protocol can be further improved by identifying the static faults and reconfiguring the network to avoid them. Fault identification also makes it possible to determine the extent of the faults in the system. This allows us to determine how close the system is to becoming inoperable. To the extent possible, the routing scheme should facilitate fault identification with low overhead. The faster that faults can be identified and the system reconfigured, the less impact the faults will have on network performance.

4.2 Protocol Overview

We have designed the METRO Routing Protocol (MRP) to addresses the issues raised in Section 4.1. MRP is a synchronous protocol for circuit-switched, pipelined routing of word-wide data through multipath, multistage networks constructed from crossbar routing components. MRP uses circuit switching to minimize the overhead associated with routing connections while facilitating tight time-bounded, end-to-end, source-responsible message delivery. MRP is composed of two parts: a router-to-router communication protocol, MRP-ROUTER, and a source-responsible node protocol, MRP-ENDPOINT.

In operation, an endpoint will feed a data stream of an arbitrary number of words into the network at the rate of one word per clock cycle. The first few data words are treated as a routing specification and are used for path selection. Subsequent words are pipelined through the connection, if any, opened in response to the leading words. When the data stream ends, the endpoint may signal a request for the open connection to be reversed or dropped. When each router receives a reversal request from the sender, the router returns status and checksum information about the open connection to the source node. Once all routers in the path are reversed, data may flow back from the destination to the source. The connection may be reversed as many times as the source and destination desire before being closed. End-to-end checksums and acknowledgments ensure that data arrives intact at the destination endpoint. When a connection is blocked due to contention or a data stream is corrupted, the source endpoint retries the connection.

4.3 MRP in the Context of the ISO OSI Reference Model

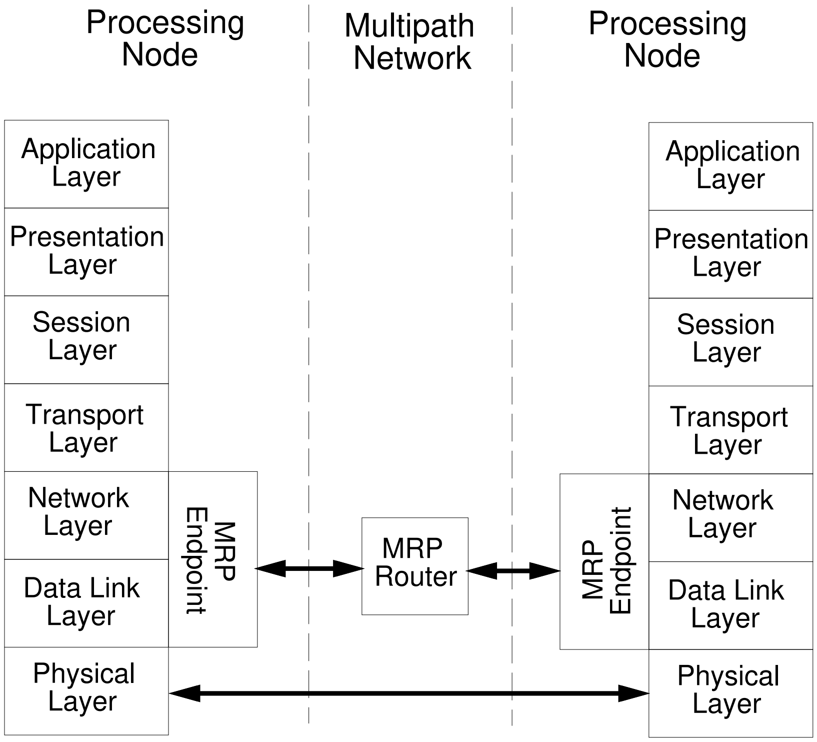
MRP fits into a layered protocol scheme, such as the ISO OSI Reference Model [DZ83] at the *data-link layer* (See Figure 4.1). That is, MRP itself is independent of the underlying physical layer which takes care of raw bit transmissions. MRP is, thus independent of the electrical and mechanical aspects of the interconnection. The protocol is applicable both in situations where the transit time between routers is less than the clock period and in situations where multiple data bits are pipelined over long wires (See Section 3.2). MRP provides mechanisms for controlling the transmission of data packets and the direction of transmission over interconnection lines. It also provides sufficient information back to the source endpoint so the source can determine when a transmission succeeds and when retransmission is necessary. By leaving the retransmission of corrupted packets to the source, MRP allows the source endpoint to dictate the retransmission policy. As such, both the MRP-ROUTER and MRP-ENDPOINT are required to completely fulfill the role of the data-link layer. Since MRP provides dynamic self-routing, the protocol layer identified as the *network layer* by the ISO OSI model is also provided by MRP.

MRP itself is connection oriented, though there is no need for higher-level protocols to be connection oriented. Together, MRP-ROUTER and MRP-ENDPOINT provide a reliable, byte-stream connection from end-to-end through the routing network.

4.4 Terminology

Recall from Chapter 3 that a *crossbar* has a set of inputs and a set of outputs and can connect any of the inputs to any of the outputs with the restriction that only one input can be connected to each output at any point in time. A *dilated crossbar* has groups of outputs which are considered equivalent. We refer to the number of outputs which are equivalent in a particular logical direction as the crossbar's *dilation*, d. We refer to the number of logically distinct outputs which the crossbar can switch among as its *radix*, r.

A *circuit-switched* routing component establishes connections between its input and output ports and forwards the data between inputs and outputs in a deterministic amount of time. Notably, there is no storage of the transmitted data inside the routing component. In a network of circuit-switched routing components, a path from the source to the destination is locked down during



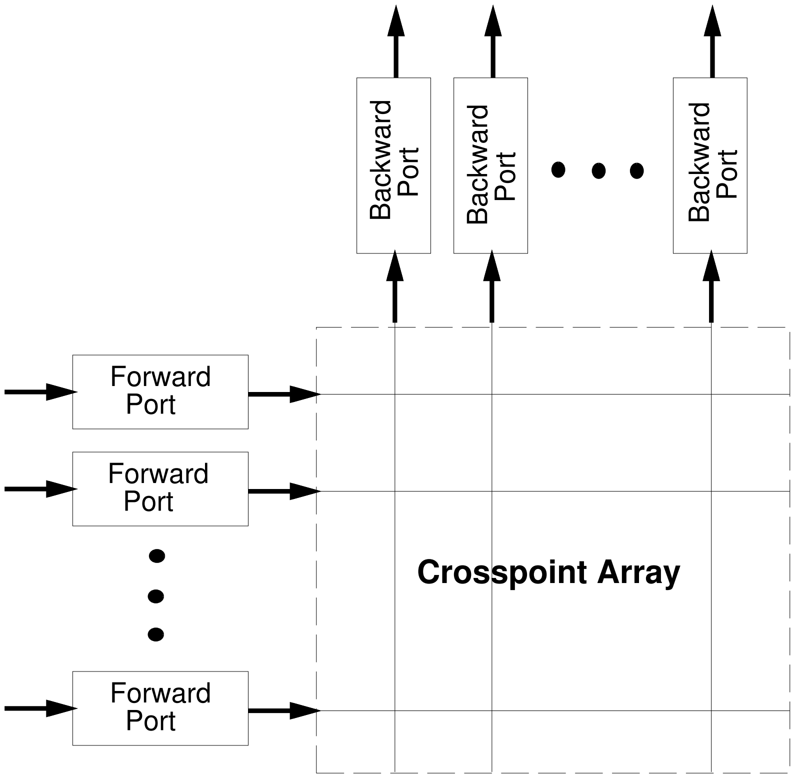
MRP fits into the ISO OSI Reference Model at the data-link layer. The routers in a multipath network use MRP-ROUTER to transfer data through the network. Each endpoint uses MRP-ENDPOINT to facilitate end-to-end data transfers.

Figure 4.1: METRO Routing Protocol in the context of the ISO OSI Reference Model

the connection; the resources along the established path are not available for other connections during the time the connection is established. In a *pipelined, circuit-switched* routing component, all the routing components in a network run synchronously from a central clock and data takes a deterministic number of clock cycles to pass through each routing component.

A crossbar is said to be *self-routing* if it can establish connections through itself based on signalling on its input channels. That is, rather than some external entity setting the crosspoint configuration, the router configures itself in response to requests which arrive via the input channels. A router is said to handle *dynamic message traffic* when it can open and close connections as messages arrive independently from one another at the input ports.

When connections are requested through a router, there is no guarantee that the connections can be made. As long as the dilation of the router is smaller than the number of input channels into a router (*i.e.* $d < i$), it is possible that more connections will want to connect in a given logical direction than there are logically equivalent outputs. When this happens, some of the connections must be denied. When a connection request is rejected for this reason, it is said to be *blocked*. The data from a blocked connection is discarded and the source is informed that the connection was not



The basic router has i forward ports and $o = r \cdot d$ backward ports. Any forward port can be connected through the crosspoint array to any backward port. The arrows indicat the initial direction of data flow.

Figure 4.2: Basic Router Configuration

established.

Once a connection is established through a crossbar, it can be *turned*. That is, the direction of data transmission can be reversed so that data flows from the original destination to the original source. This capability is useful for providing rapid replies between two nodes and is important in effecting reliable communications. MRP provides *half-duplex*, bidirectional data transmission since it can send data in both directions, but only in one direction at a time. When data is flowing between two routers, we call the router sending data the *upstream* router and the router receiving data the *downstream* router.

Since connections can be turned around and data may flow in either direction through the crossbar router, it is confusing to distinguish input and output ports since any port can serve as either an input or an output. Instead, we will consider a set of *forward ports* and a set of *backward ports*. A forward port initiates a route and is initially an input port while a backward port is initially an output port. The basic topology for a crossbar router assumed throughout this chapter is shown in Figure 4.2.

Signal/Datum	Control Bit	Control Field
IDLE/DROP	0	all zeros
ROUTE	1	direction specification
TURN	0	all ones
STATUS	1	port specification
CHECKSUM	1	checksum bits
DATA-IDLE	0	distinguished hold pattern
DATA	1	arbitrary

The words sent over the network links can be classified as data words and signalling words. The use of a single control bit which is separate from the transmitted data bits allows out of band signalling to control the connection state. This table shows how control signals and data are encoded. The control field is a designated $\log_2(\max(r, d))$ bit portion of the data word.

Table 4.1: Control Word Encodings

4.5 Basic Router Protocol

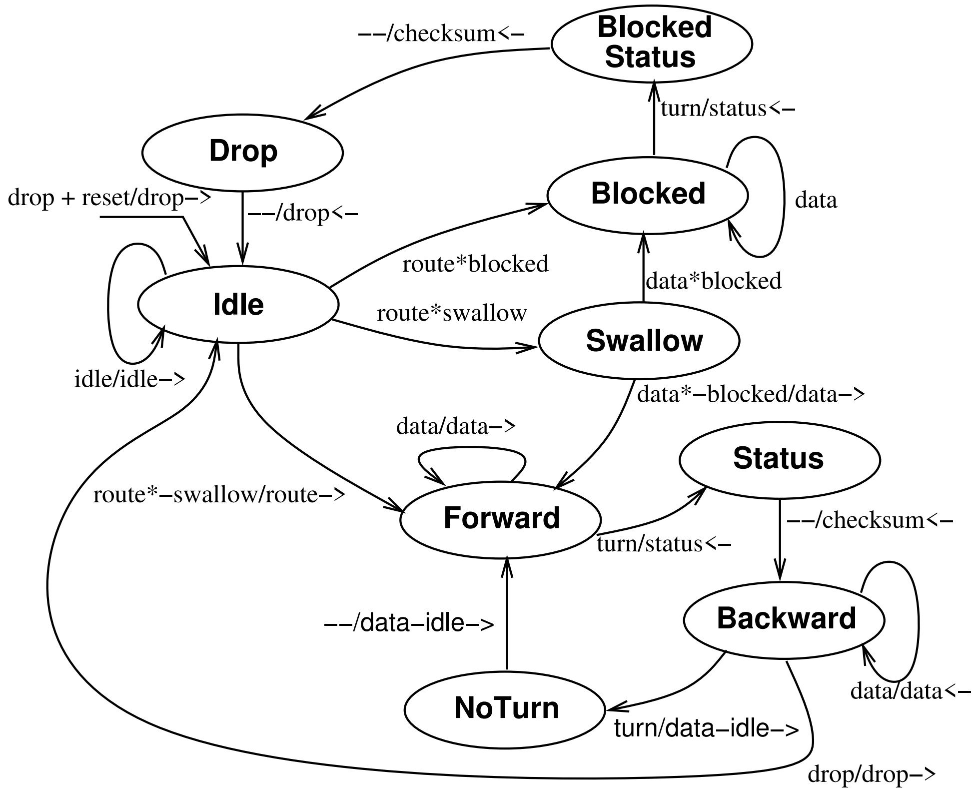
The behavior of MRP-ROUTER is based on the dialog between each backward port of each router and its companion forward port in the following stage of routers. In this section, we describe the core behavior of the router signalling protocol from the point of view of a single pair of routing components.

4.5.1 Signalling

Routing control signalling is performed over data transmission channels. Using simple state machines and one control bit, this signalling can occur *out of band* from the data. That is, the control signals are encoded outside of the space of data encodings. Out of band signalling allows the protocol to pass arbitrary data. Table 4.1 shows the encoding of various signals. The control field is a designated portion of the data word. Due to encoding requirements, it is at least $\log_2(\max(r, d))$ bits long. The remainder of this section explains how these control signals are used to effect routing control.

4.5.2 Connection States

The states of a forward-backward port pair can be described by a simple finite state machine. Figure 4.3 shows a minimal version of this state machine for the purpose of discussion. Each transition is labeled as: *<event>*/*<result><dir>*. Where *<event>* is a logical expression, usually including the reception of a particular kind of control word, *<result>* is an output resulting from the reception, and *<dir>* is an arrow indicating the direction which the *<result>* is sent. For instance, the arrow from **swallow** to **forward** means that when a DATA word is received and the resulting output direction specified is not blocked, forward the DATA out the allocated backward port in the forward direction and change to the **forward** state.



As a connection is opened, released, reversed, and used in the network, the individual links within the network go through a series of connection states as shown above. Transitions are initiated by the receipt of a control word (See Table 4.1) and modified by the local state of the router. Each transition is labeled as: *<event>*/*<result><dir>*. *<event>* describes the control word received along with state modifiers; *<result>* is an output word resulting from the event; *<dir>* is an arrow indicating the direction which the *<result>* is sent.

Figure 4.3: MRP-ROUTER Connection States

4.5.3 Router Behavior

Idle port

When a connection between routers is not in use, the connection is in an **idle** state. While in the idle state, the backward port on a router transmits the IDLE word to its corresponding forward port in the next stage of the network. A forward port interprets the reception of an idle word to mean that it should remain in an idle state and hence should not attempt to open a new connection.

Route

To route a connection through a router, a ROUTE word is fed into a router forward port. The forward port recognizes the transition of the control bit from the zero, which it was receiving when the connection was idle, to a one. The router then uses the control field to determine the desired routing direction. The router forwards the ROUTE word through a backward port in the desired direction, if one is available, and locks down the connection so subsequent DATA words can be passed in the same direction. If no output is available, the connection is blocked.

Data

All words with control set bits which are received following a ROUTE word are forwarded through the allocated backward port of the routing component to the forward port of the next router in the network.

Data Completion

When all of the data words in a message have been passed into a router, the sender has two options. The sender can either drop the connection without getting any response, or turn the direction of the connection around for a reply.

To drop the connection, the input is given a DROP word; the reception of a DROP word causes the router to close down an open connection and free up the output port for reuse. When the output port is freed via a DROP word, it forwards a DROP along to the next router then returns to an idle.

To turn a connection around, the input port is given a TURN word; when the router receives a TURN word, it forwards the turn out the allocated output port, if there is one, and returns several status words. Figure 4.3 shows a version of the protocol which returns two such words. These words fill the pipeline delay associated with informing the subsequent router to turn the connection around and getting backward data from the subsequent router. The filler words differ based on whether the connection is sending data in the forward or reverse direction when the TURN is received.

In the forward direction, the router returns first a STATUS word and then a CHECKSUM word. If the connection was not blocked during the routing cycle, after sending these words, the router will be receiving data in the reverse direction and will forward this data through. If the connection was blocked, the router forwards a DROP word following the CHECKSUM and returns to the idle state.

In the backward direction, the router simply sends several DATA-IDLE words before sending the reverse data. In order for a port to be in the backward direction, there must be a connection through the router so there will always be data to propagate following a backward turn.

Checksum and Status Information

The STATUS and CHECKSUM words form a series of word wide values which serves to inform the source node of the integrity of the connection made through each router in the path obtained through the network. A portion of the STATUS word informs the source about which of the logically equivalent backward ports, if any, through which the connection is routed. When this information arrives uncorrupted at the source endpoint, it allows the source to identify the routers through which the connection was actually routed. When a connection is blocked at some router, this information serves to pinpoint where the connection became blocked. The remaining bits of the STATUS word, together with the bits in the CHECKSUM word or words, can be used to transmit a longitudinal checksum back to the source. This checksum is generated on all the data sent through the connection since the last ROUTE, including the ROUTE word itself. When data is corrupted due to faults in the network, the checksum provides the source endpoint with sufficient information to identify the most likely corruption source .

4.5.4 Making Connections

When a connection is opened through a router, there may or may not be outputs available in the desired logical output direction. If there is no available output, the router discards the remaining bits associated with the data stream. When the connection is later turned around, the router STATUS word returned by the routing node informs the source that the message was blocked at the router. When exactly one output in the desired direction is available, the router switches the connection through that output. When multiple paths are available, the router switches the data to a logically appropriate backward port selected *randomly* from those available.

This random path selection is the key to making the protocol robust against dynamic faults while avoiding the need for centralized information about the network state and keeping the routing protocol simple. When faults develop in the network, the source detects the occurrence of a failed or damaged connection by the acknowledgment from the destination. The source then knows to resend the data. Since the routing components select randomly among equivalent outputs at each stage, it is highly likely that the retry connection will take an alternate path through the network, avoiding the newly exposed fault. Source-responsible retry coupled with randomization in path selection guarantees that the source can eventually find a fault-free path through the network, provided one exists. The random selection also frees the source from knowing the actual details of the redundant paths provided by dilated components in the network. Random selection among equivalent available outputs is an extremely simple selection criterion to implement in silicon and can be implemented with little area and considerable speed. Random selection also requires no state information not already contained on the individual routing component.

4.6 Network Routing

Each router in a path through the network needs to see a different routing specification. Since we require the routing specification to be in a fixed position in the ROUTE word to allow efficient implementation of the protocol, the data seen in this position by each router must be different. Between routing stages, the bits of the datapath can be permuted so that routers in different stages see distinct control fields. This bit reordering allows a single routing word to specify the path through several routers. However, if the network is sufficiently large, all of the bits in a single routing word will eventually be exhausted before the full route through the network can be specified. To deal with this case, MRP allows routing switches to be configured to ignore the first data word in an incoming message by setting a SWALLOW configuration bit. This option allows networks to be arbitrarily large. Every time all of the routing bits in the ROUTE word are exhausted, the ROUTE word can be discarded allowing routing to continue with the fresh routing bits in the subsequent word. In this manner, the first DATA word in the message following the original ROUTE word is promoted to be the ROUTE word after the original is exhausted.

4.7 Basic Endpoint Protocol

Network endpoints use MRP-ENDPOINT to guarantee the delivery of at least one uncorrupted copy of each message stream to the desired destination. An endpoint channel on the source end of a connection is called a *network input*, while a receiving endpoint channel is called a *network output*. At the most primitive level, each network input behaves like a router backward port and each network output behaves like a router forward port. The control of each network input and output, however, is more involved than the simple data stream handling performed by forward and backward ports as described in the Section 4.5.

4.7.1 Initiating a Connection

When a node wants to initiate a connection over the network, the network input adds a routing header and message checksum to the data and send it into the network. The data stream is then followed with either a DROP or TURN to indicate the disposition of the link following data transmission. The initial message thus looks like:

$$\text{(ROUTE)}^* \circ \text{(DATA)}^* \circ (\text{DATA}_{checksum})^* \circ \text{TURN}$$
$$\text{<OR>}$$
$$\text{(ROUTE)}^* \circ \text{(DATA)}^* \circ (\text{DATA}_{checksum})^* \circ \text{DROP}$$

The ROUTE word or words specifies a path to the desired destination. Section 4.6 described how the datapath and routers can be configured so that unique routing bits are available to each routing component in the path between the source and the destination. The route words should be constructed accordingly.

In general, a node has multiple network inputs. In the same way that routers choose randomly among the available logically equivalent outputs, the node should choose randomly among the available network inputs. The benefits in terms of dynamic fault avoidance are the same as for the routers as discussed in Section 4.5.4. When the network in use has multiple, different path specification which reach the same destination node, as would be the case in an extra-stage style network (Section 3.3.4), the source should choose randomly among the available paths through the network. This random selection avoids worst-case congestion of any particular path through the network and gives the routing algorithm the property that it can avoid dynamic faults in the network. In these extra-stage cases, we are simply moving the randomization in path selection from inside the network to the originating endpoint.

Each message should be guarded with a checksum on the message data so that the receiving endpoint can identify when a message has been corrupted. The length of the checksum should be chosen so that the probability of a corrupted message having a good checksum is sufficiently small for the intended application. The checksum should be constructed in such a way that a message mistakenly delivered to an incorrect node will not be accepted as a valid message at that node. One way to ensure this is to include the destination node number in the data portion of the message; another would be to seed the checksum as if the first portion of the data included the node number, but not actually transmit the node number. It is not, in general, possible to include the routing words in the checksum and use them in place of the node-number for assuring that the message arrives at the correct destination. With extra-stage networks or tree networks, it is possible to reach a given destination with many different route-path specifications. In such cases, the set of routing words are not unique to each destination node and some of those routing words will have been stripped from the message in the network (See Section 4.6) before the connection reaches the destination node. If the stripped routing words were used in calculating the checksum, the destination would have no way of knowing what the stripped routing words were and hence their effect on the checksum computation.

If the sending node needs to guarantee that the message was actually received by the destination node, it must turn the network around after sending the data rather than dropping the connection. Unless the node turns the network and gets a reply from the destination endpoint, the originating endpoint will not know what happened to the message inside the network. Since the source node is responsible for message retransmission in the case of network corruption, the source node must store the message for retransmission until a suitable acknowledgment is received from the destination.

4.7.2 Return Data from Network

After sending the TURN into the network, the source endpoint will receive status and checksum information from each router in the path opened by the connection. For the simplified router protocol shown in Figure 4.3 the replies will look like:

$$(\text{STATUS} \circ \text{CHECKSUM})^{s} \circ \text{DROP}$$
$$<\text{OR}>$$
$$(\text{STATUS} \circ \text{CHECKSUM})^{N} \circ (\text{DATA})^{*} \circ (\text{DATA}_{checksum})^{*} \circ \text{DROP}$$
$$<\text{OR}>$$
$$(\text{STATUS} \circ \text{CHECKSUM})^{N} \circ (\text{DATA})^{*} \circ (\text{DATA}_{checksum})^{*} \circ \text{TURN}$$

Here N is the number of stages in the network, and s is the number of stages into the network a connection was routed before it was blocked ($s \leq N$). In the case where a connection is blocked, the source will only receive this status information up to and including the router in which the blocking occurred. As noted in Section 4.5.3, the checksum and status information provides the source endpoint with information which allow it to localize the source of faults in the network.

It is important to note that the checksums coming back from the network cannot be used to determine whether or not the destination endpoint has successfully received the data. Since a dynamic fault may arise at any point in time, a fault may, for example, occur in a router or link after the message data was sent passed the router but before the router passes back a checksum. In which case, a corrupted checksum could seem to appear from a router which passed that data without corruption. For this reason, the checksums from the routers serve only diagnostic purposes. The source endpoint must use information from the destination endpoint's reply to determine whether or not the destination received the data uncorrupted.

When a connection is completed through the network, after the TURN reaches the destination node, the destination has the opportunity to reply. At the very least, this reply should indicate if the data stream arrived uncorrupted. Depending on the application, the destination node may wish to send reply data along with this acknowledgment. When the destination endpoint replies, its reply data should also be guarded with a checksum to protect against dynamic failures in the network. When the destination simply acknowledges the receipt of a message, the acknowledgment encoding should be chosen so that there is sufficiently small probability that a negative acknowledgment can be corrupted into a positive acknowledgment. After its reply, the destination may choose to close down the network connection or turn the connection around for further communication.

4.7.3 Retransmission

We may surmise that a connection has failed to transfer data successfully when any of the following occur:

1. The path is blocked due to resource contention in the network
2. The destination node indicates that data was corrupted upon arrival
3. The return data stream does not adhere to protocol expectations

In any of these events, the source must retransmit the data if it wishes to guarantee the receipt of uncorrupted data. The first event may occur when the network is congested, whereas the later options indicate that there is a fault in the network. Blocking can also be indicative of certain kinds of network failure. The fault in the network could be transient or a dynamically occurring permanent fault. Since the endpoint does not know which kind of fault caused the failure, a single fault occurrence is not conclusive evidence that a particular fault persists in the network. Consequently, the node endpoint may wish to save away the reply data in any of these cases for fault analysis.

When retrying the transmission, the source node has some freedom in the retransmission timing. The source may choose to retry the same message or a message to a different destination, and it may choose to retry immediately or after a wait period. Which technique a node uses depend on the requirements of the application. If the application expects the messages from one node to be delivered to their destinations in the order they were generated, the node will not have the option to choose a different message. Since the path on a retransmission may be very different from the path just taken, it may be beneficial to immediately retry the failed connection even when the failure was due to blocking. While much work has been done on backoff and retry strategies for bus based systems (*e.g.* [HLwn]), retransmission policies for this class of networks remain an open area of research.

If the network continues to retain complete connectivity between all communicating endpoints, the source will eventually be able to deliver its message to its destination. Note that whenever blocking occurs at some stage in the network, some message has been able to reach a further stage in the network. Thus, in order for a connection to be blocked at stage s, a connection must have progressed to stage $s+1$. Following this reasoning, as long as complete connectivity remains in the network, some connection must be reaching its endpoint, and, therefore, forward routing progress is always being made. If all connections are treated equally within the network, each has an equal chance of being routed through the network. Thus, we can expect that any particular connection will eventually complete as long as a path exists to the desired endpoint.

However, if the network has lost full connectivity due to newly arising faults, some destinations may no longer be reachable. Additionally, if there is a large amount of contention for a few destinations, the number of attempts necessary to deliver a message may become very large. As a pragmatic matter, we often limit the number of retries allowed. If a connection cannot be delivered in a fixed number of trials, the connection fails and MRP-ENDPOINT reports this information back to the node. At this point, the node may wish to communicate with other nodes to determine the source and nature of its problem. The node may also wish to initiate network diagnostics to verify if nodes have actually been newly disconnected from the network. If the failure arose purely from contention, then the node can take this opportunity to inform higher-level data and process management protocols of excessive contention.

4.7.4 Receiving Data from Network

To minimize end-to-end network latency, a system may begin processing the head of a data stream in parallel with the reception of the remainder of the data stream. The network output receiving data from the network, however, has no guarantee of the integrity of the data stream it is receiving until it sees a checksum guard. The receiving node may begin processing the data as soon as it arrives only as long as it can guarantee that the processing it does prior to seeing the checksum will have no adverse affects if the data is corrupted.

For example, consider a network operation which is intended to cause data to be remotely written into the destination node's memory. If the only checksum was at the end of the data stream, the destination node could not begin writing data into memory as the data is being received because the address could be corrupted. In such a case, a corrupt address could cause the destination to write data over some arbitrary place in the node's memory. Similarly, the destination has no guarantee about the length of the data it will be receiving. A network fault could cause the network to send what appears as more data than the original, uncorrupted message was transmitting. This would cause data in memory following the intended destination block to be overwritten. To avoid these problems, the message data could start with the destination address and data length which are guarded with their own checksum preceding the actual data transmission. In this case, once the address and length are correctly received, the data may be stored directly to memory. If the corruption occurs in the data itself, the source will retransmit the data. Depending on the way in which the memory is being maintained, it may be necessary for the node to prevent access to the memory being overwritten until the final verification of the integrity of the data is received.

A node may receive a bad data stream for either of the following reasons:

1. Checksum(s) indicate message may be corrupted

2. Data stream does not adhere to protocol expectations

When this happens, the receiving node is only expected to indicate its rejection of the data stream. If the receiving node can give some indication of why the data stream was rejected, the source node may be able to use that information when failure diagnosis is necessary. Nonetheless, the only piece of information the source node requires is the fact the connection failed and must be retried.

4.7.5 Idempotence

In the introduction to this section we said that MRP-ENDPOINT guarantees the uncorrupted delivery of at least one copy of the message to the destination. We might prefer that it guaranteed the delivery of exactly one copy of the message. However, source-responsible retry coupled with the potential for dynamic fault occurrences allows for multiple delivery of a message.

Consider what happens when the source receives a corrupted checksum, corrupted data stream, or other indications that something is wrong. It is safe to assume some fault occurred, but it may not be clear where the problem occurred. Particularly, if the fault only affected the return data from the destination, the source can believe that an operation failed which the destination completed successfully. When the source thinks the operation has been corrupted, the source-responsible nature of the protocol has the source retry the operation since there is some high likelihood the operation has not been accepted by the destination. However, when this retry occurs in the case in which only the return data or acknowledgment was corrupted, the destination node will see the data stream a second time.

The consequence is that all operations performed using MRP must be *idempotent*. That is, performing an operation multiple times cannot produce different results from performing the operation once. Our previous example (Section 4.7.4) of a cross network write operation was idempotent since writing the same data twice will not change the data which is written in memory. However, a network operation which caused a remote counter to be incremented directly would not be idempotent since incrementing the counter more than once would give a different result.

There are a few choices for dealing with the idempotence requirement. Either, we can design all operations which use MRP directly to be idempotent or we can implement a layer of protocol between applications and MRP which guarantees idempotent message delivery.

The Transmission Control Protocol (TCP), in use on many local-area networks, provides "reliable" data streams by using sequence numbers [Pos81] to guarantee idempotent message delivery. When a source needs to communicate with a destination, the source arbitrates with the destination for a valid set of sequence numbers. The source annotates each unique packet of data transmitted to the destination with a different sequence number. The destination node keeps track of all the sequence numbers it has seen so that exactly one copy of each packet arriving at the destination is passed along to higher-level protocols. In this manner, all duplicates arising due to source-responsible retransmission are filtered out, making each message effectively idempotent at the protocol level above TCP.

While one could implement a TCP-style unique sequence number protocol on top of MRP, such a solution is inefficient for high-speed communications in a large-scale multiprocessor context. The overhead in terms of the space and processing time required to track sequence numbers and filter messages based on sequence numbers could easily become many times greater than the time and space required for basic message transmission and hence increase T_p and T_w greatly. Alternatively, designing the lowest level communications primitives to be idempotent seems an attractive way of avoiding this cost.

4.8 Composite Behavior and Examples

Having detailed the basics of MRP in the previous sections, this section reviews the composite behavior and shows several representative examples of protocol operation.

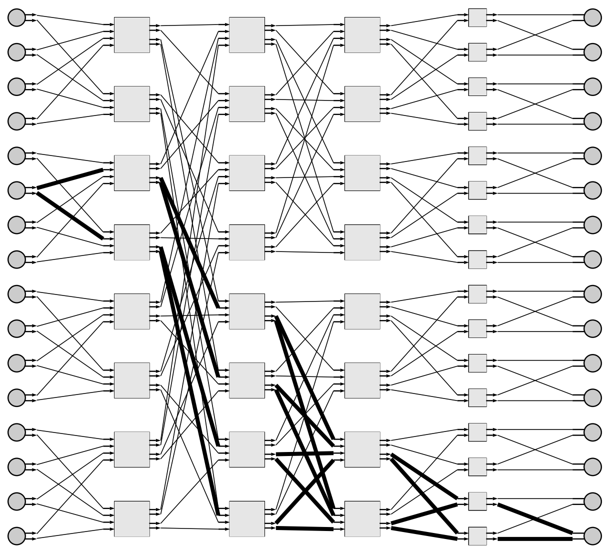


Figure 4.4: 16 $\times$ 16 Multibutterfly Network

4.8.1 Composite Protocol Review

The source endpoint feeds messages into a router in the first stage of the network. The leading word is treated as the ROUTE word. At each stage, the ROUTE is either pipelined through the network or blocked by existing connections. Between router stages, the bits are permuted to present fresh routing bits to each router. When the bits are exhausted the subsequent router will be configured to swallow the ROUTE word and promote the first DATA word to be the new ROUTE word. When the entire message is fed into the network, the source will generally send a TURN word to reverse the connection. The first router in the network returns STATUS and CHECKSUM words and forwards the data received from the second router in the network. The first reverse data from the second router in the network will be the second router's STATUS and CHECKSUM words. The source will, thus successively receive STATUS-CHECKSUM word pairs from each router in the connection. If the connection is blocked at some point, the blocked router will send a DROP following its STATUS-CHECKSUM pair; this DROP will close down the connection as it propagates back to the source. If the connection was not blocked, the source will receive data from the destination following the final STATUS-CHECKSUM pair. The connection may be reversed or closed by the destination once it has completed its reply.

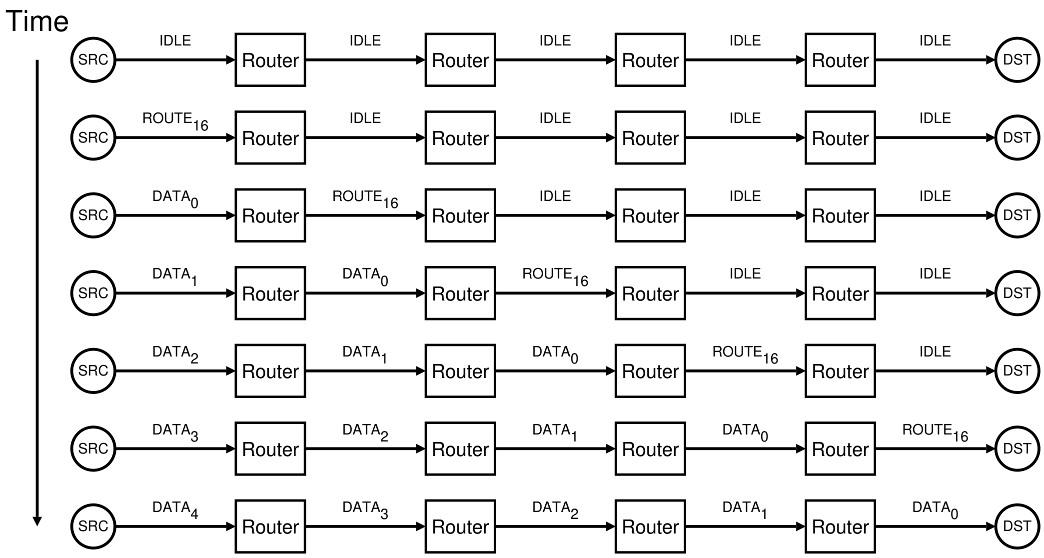
4.8.2 Examples

Consider the network shown in Figure 4.4. The possible paths from input 6 to output 16 are highlighted. For the sake of simplicity in examples, let us consider the routers involved in one of the paths between input 6 and output 16. The same protocol is obeyed between all routers so the examples easily generalize to the complete network.

Each of the following examples (Figures 4.5 through 4.10), shows several cycles of communications over one of the paths indicated in Figure 4.4. Each connection between routers is labeled with the control/data word transmitted during the cycle and annotated with the direction of data flow. Often words of the same type are subscripted so the progression of individual words can be tracked from cycle to cycle.

Opening a Connection

Figures 4.5 and 4.6 show how a connection is opened through the network. The connection in Figure 4.5 succeeds in opening a connection through the network, whereas the one in Figure 4.6 is blocked at the router in the third stage.

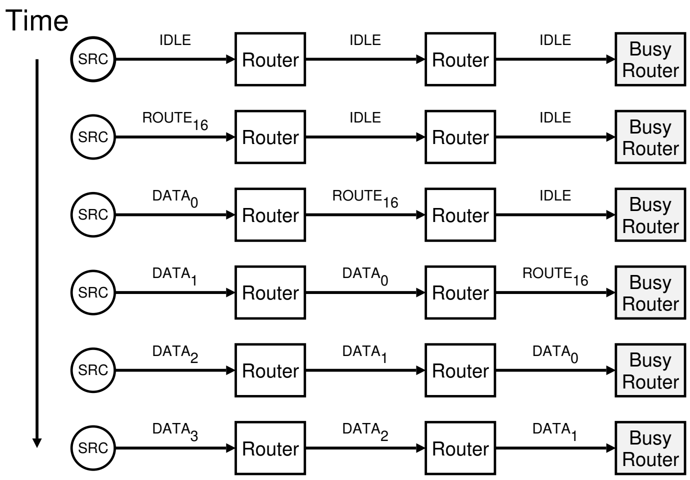


Shown above is a cycle-by-cycle progression of control and data through the network as a connection is successfully opened from one endpoint to another. The message snakes through the network advancing one routing stage on each clock cycle. The first word in the message is the routing word.

Figure 4.5: Successful Route through Network

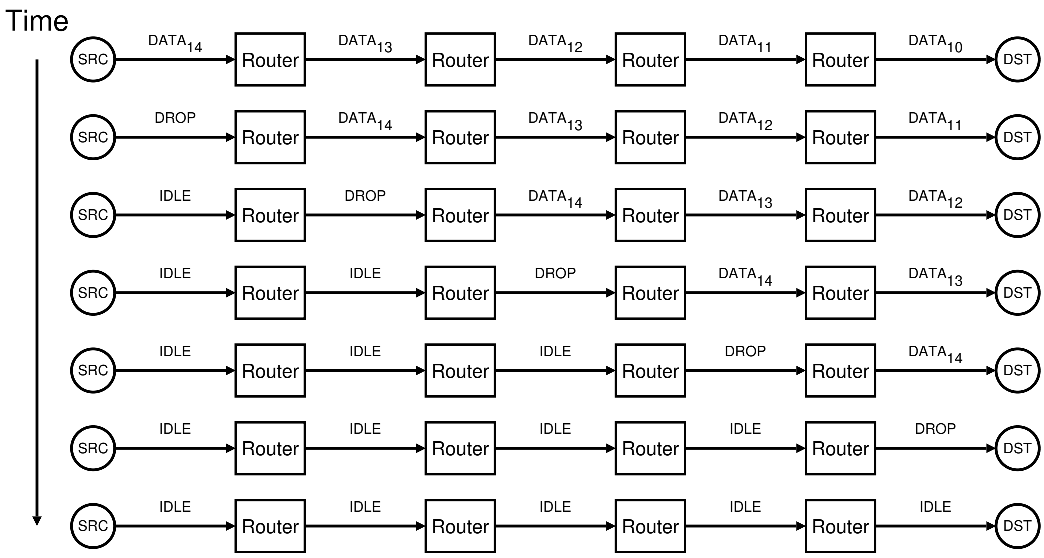
Dropping a Connection

Figure 4.7 shows an open connection being dropped in the forward direction. Dropping a connection from the reverse direction proceeds identically with the roles of the source and destination reversed. If the connection is blocked, the DROP is propagated up to the router at which the connection is blocked.



In the event that a connection cannot be successfully opened through some routing component in the network, the message is discarded word-by-word at that router. The example shown above depicts such blocking a router in the third stage of the network.

Figure 4.6: Connection Blocked in Network

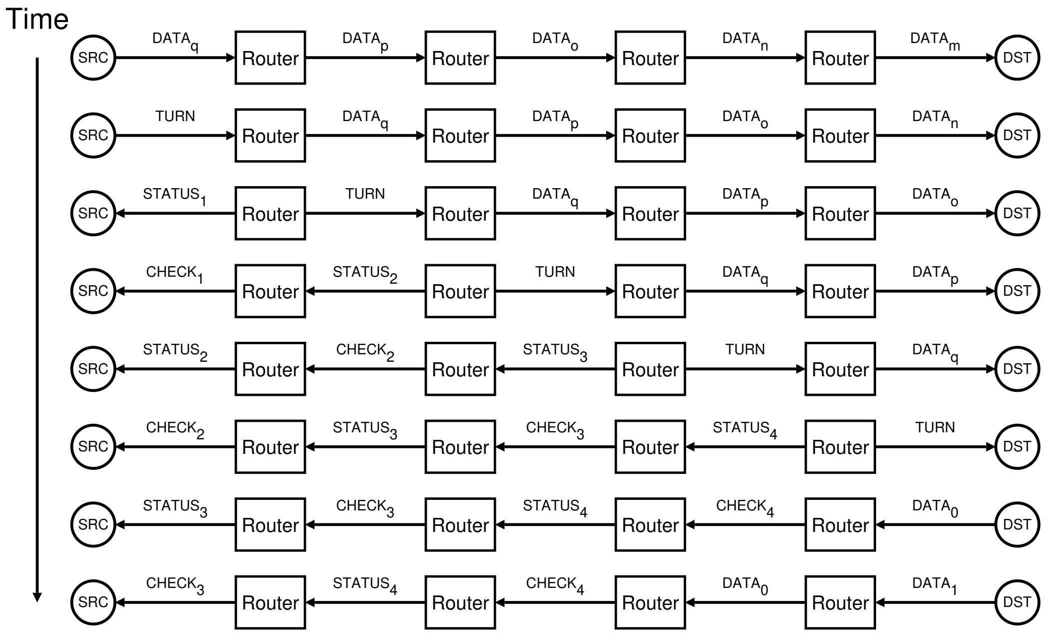


When the transmitting network endpoint decides to terminate a message, it ends the message with a DROP control word. The DROP follows the message through the network resetting each link in the connection to idle after traversing the link.

Figure 4.7: Dropping a Network Connection

Turning a Connection (Forward)

Figure 4.8 shows a successful connection being turned and backward data propagating through the network. Figure 4.9 shows how a blocked connection is collapsed when reversed.

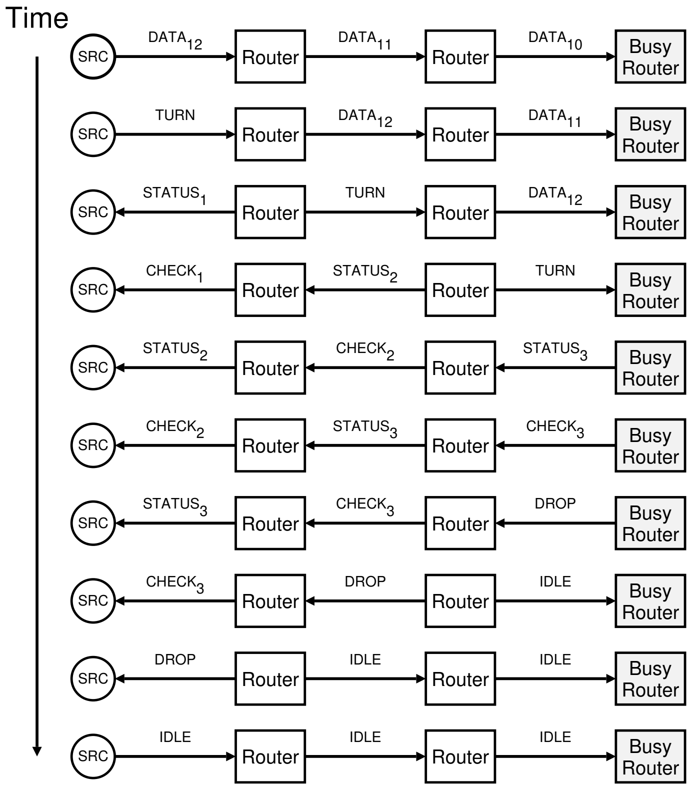


When the source wishes to know the state of its connection and get a reply from the destination, it feeds a TURN into the network following the end of its forward transmitted data. As the TURN works its way through the network, the links it traverses are reversed. In the pipeline delay required for the link to begin receiving data in the reverse direction, each routing component sends status and checksum information to inform the source of the connection state. After the TURN has propagated all the way through the network and all routers along the connection have sent status and checksum words, data flows backward along the connection.

Figure 4.8: Reversing an Open Network Connection

Turning a Connection (Reverse)

Turns from the reverse direction proceed basically as forward turns. Instead of sending STATUS and CHECKSUM words, routers send DATA-IDLE words when turned from the reverse direction. Figure 4.10 shows a turn from the reverse direction.



In the event that the connection was blocked at some router in the network, the blocked router will be unable to provide a reverse connection through the network. Instead, after sending its own checksum and status word, the blocked router will send a DROP back to the source. As the DROP propagates back to the source, it resets the intervening network links.

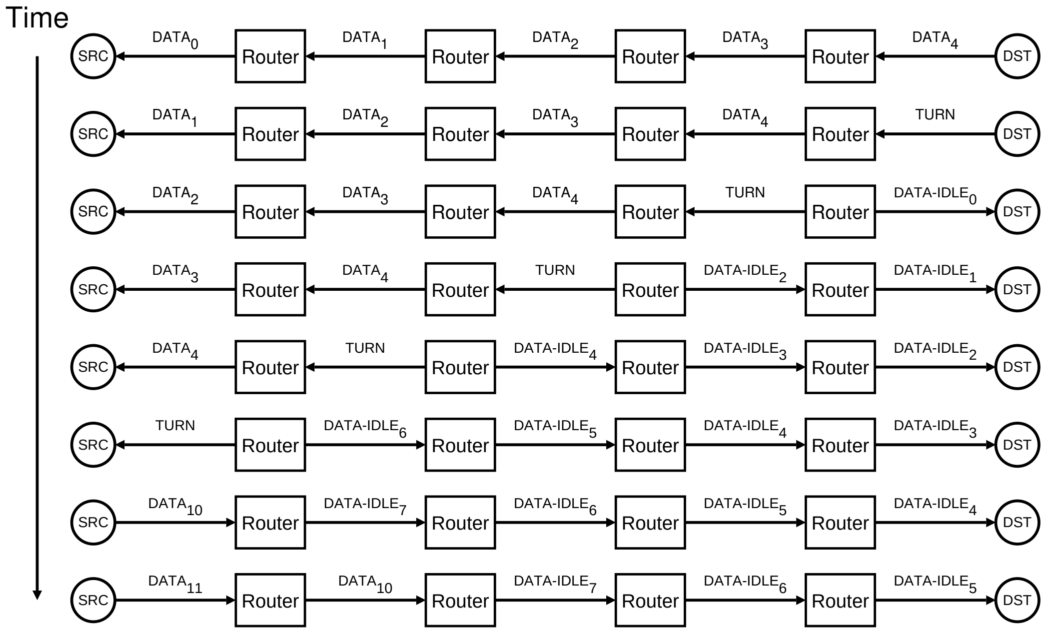
Figure 4.9: Reversing a Blocked Network Connection

4.9 Architectural Enhancements

Beyond the basic routing strategy, there are several protocol enhancements which offer higher performance under certain conditions.

4.9.1 Avoiding Known Faults

As described so far, all router decisions are made purely by random selection. However, when faults have occurred in the network and are known to exist, it would be better, from a performance viewpoint, to deterministically avoid them. In extra-stage style networks, we only need to remove the faulty path(s) from our list of potential paths. When randomly selecting a path, the choice is only made from among the set of paths believed to be non-faulty. In dilated networks, the routers



A connection flowing data in the *backward* direction can be reversed again so that data may flow, once again, from the original source to the original destination. This reversal is similar to the original reversal, except that DATA-IDLE data is returned during the reversal pipeline delay rather than checksum and status information.

Figure 4.10: Reverse Connection Turn

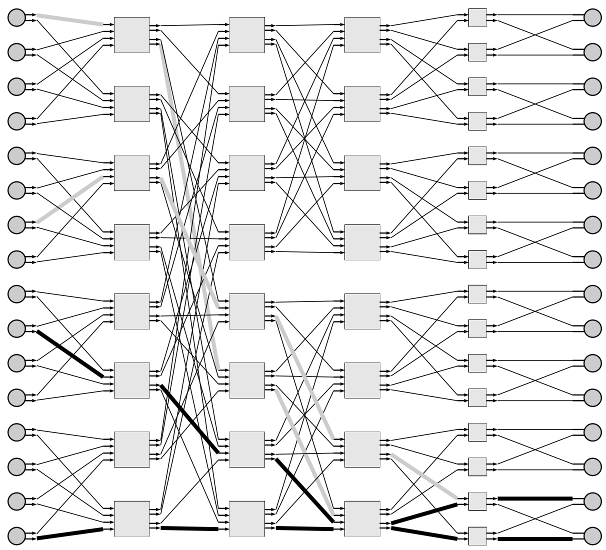
themselves are making the detailed path decisions. For dilated routers we need a way to cause a router to avoid faulty links or routers.

Port deselection is one way to achieve this deterministic fault-avoidance in dilated routing components. That is, if we have a way to deselect, or turn off, each backward port, we can deterministically avoid ever traversing a known faulty link or attempting to use a known faulty router. The semantics of this deselection are such that the deslected backward port is treated as if it is always busy and hence removed from the set of potential backward ports in a given logical direction. When connections are routed through the router, the deselected backward port is simply never used.

For the same reasons, it is useful to be able to deselect forward ports. A forward port attached to a faulty router or faulty interconnection link may see spurious data. To prevent that data from interfering with the normal operation of the rest of the routing component, deselecting the forward port causes the forward port to ignore any connection requests it receives.

A faulty link between routers is thus excised from the network by deselecting the forward and backward port pair attached to the link. A faulty routing component can be effectively removed from the network by deselecting all the backward and forward ports connected to the faulty router.

Chapter 5 addresses the issue of identifying fault sources. Chapter 5 also suggests mechanisms



Shown above is a connection blocking scenario where the successful connections are shown in bold and the blocked connections are shown in thick gray. Connections have been successfully made between nodes 16 and 15 and between nodes 10 and 16. An attempted connection between nodes 7 and 15 was blocked in the third stage since the router had no free output ports in the intended direction. Similarly, a connection attempted between nodes 1 and 15 failed due to blocking in the fourth stage. Each of these blocked connections consumes routing resources up to the stage in which blocking occurs in the same way that non-blocked connections consume resources. New connections which are attempted while these blocked connections continue to utilize network links can, in turn, be blocked by the failed connections.

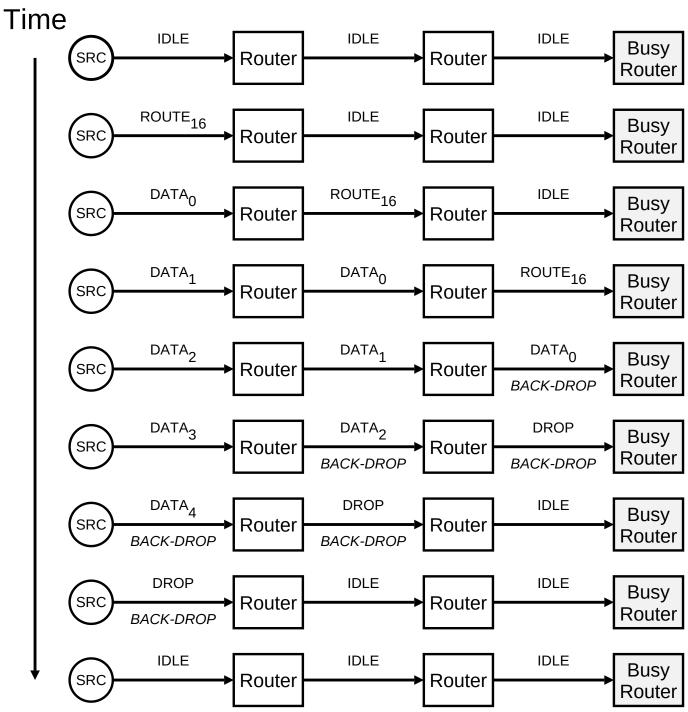
Figure 4.11: Blocked Paths in a Multibutterfly Network

for reconfiguration and considers network reconfiguration in more detail.

4.9.2 Back Drop

As described so far, when a connection becomes blocked at some stage, s, in the network, the path from the source to stage s remains open. The router links which the connection established up to stage s remain allocated to the blocked connection until the TURN or DROP at the tail of the message works it way up to the blocked router. If the network is very large, blocking occurring in later network stages will hold resources in many routing stages and aggravate congestion (See Figure 4.11). Further, the length of time the connection is held open will depend on the length of the initial connection data. Longer data transmissions will exacerbate the effects of a blocked connection on the rest of the network. Since the blocked connections consume router links up to stage s, they may in turn block connections at earlier stages.

To minimize the detrimental effects of a blocked connection on subsequent traffic, each router can be given the ability to shutdown an open connection from the head of the data stream. This requires some way to propagate the information that the connection is blocked backward along the



With fast path collapsing, a blocked connection is collapsed in a pipelined manner from the point in the network where blocking occurs. The example shown above depicts how a connection encounters blocking and is collapsed using a back drop request. Note that routers in the connection closest to the blocked router are freed earlier than routers closer to the source end of the connection.

Figure 4.12: Example of Fast Path Reclamation

open connection to the source. One simple way to achieve this is to add a single extra control line between each pair of backward and forward ports inside the network and between each network input and each network output and their associated backward and forward ports. When a connection becomes blocked, the router which notes the blocking uses this backward control line, the *back-drop line*, to inform the upstream router that the connection is blocked and hence the data is going nowhere. The upstream router may then deallocate the backward port carrying the data, thus freeing it for reuse, and pass along the blocking information to its own upstream router. Continuing in this manner, the connection may be collapsed in a pipelined fashion starting at the head of the message and propagating back to the source. If the source endpoint is signalled of blocking via the back-drop line before it has finished sending the message, it too, can abort the message transmission. The source may then begin to retry the connection. Figure 4.12 depicts how a blocked connection is collapsed using the fast path reclamation.

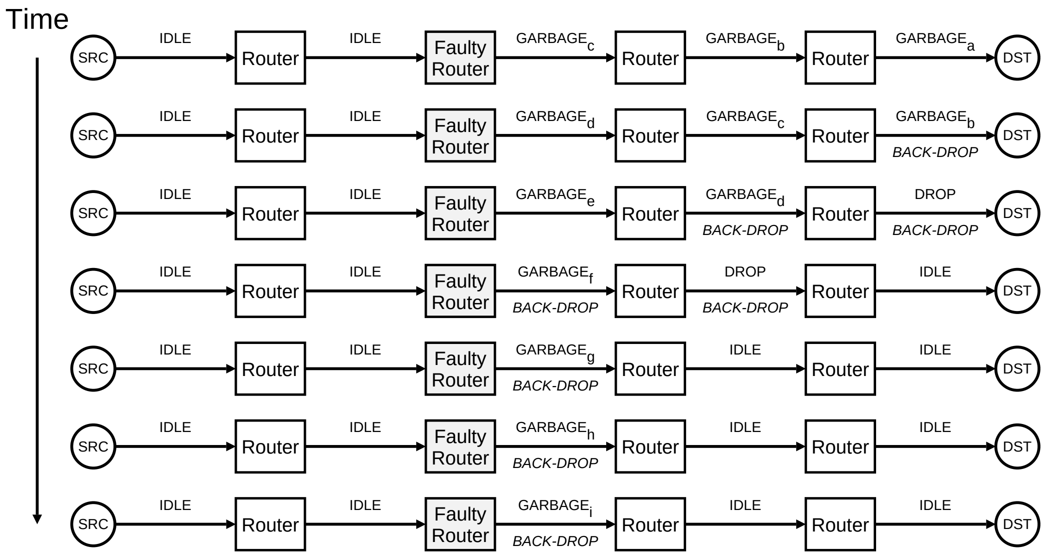
Fast path collapsing has a number of positive effects on performance. It quickly frees up resources which are not being used to transmit successful data. Since the path collapsing is initiated at the head of the message where blocking occurs, the routing resources in later network stages are freed more quickly than those in earlier network stages. This is fortuitous since connections which block in later network stages tie up more resources in the network and hence have a larger detrimental effect on the network than those which block in earlier stages of the network. In Figure 4.12 we see blocking occurring in the third stage. The router in the second stage is freed from transporting data in a few cycles. As a result, the total time this router's backward port is occupied with the blocked connection is small. The router in the first stage, in turn, used its backward port carrying the blocked connection for only a couple of cycles longer.

Fast path collapsing also benefits fault tolerance. Without some form of backward path reclamation, only the sending endpoint has the opportunity to shut down a connection. The receiving endpoint must wait for the connection to be turned or dropped before it can reclaim the network output or network input carrying the data stream. If a fault occurs during a transmission that causes a router to continually send data, the receiving node has no way of shutting down the connection. The routing resources consumed from faulty router up to and including the afflicted receiving network input or output remain unusable as long as the fault persists. The addition of fast path collapsing addresses this problem. If a connection continues to provide data beyond the time when the network endpoint expects the connection to turn or complete, or if the data stream does not conform to the expected protocol, the receiving endpoint may use the back-drop line to initiate a collapse of the path from the downstream end of the connection. Once the path is collapsed, the faulty router, which is continually sending data, will not affect the network any further. The faulty router may continue to send data to its immediate neighbor. However, the neighbor has deallocated the associated backward port and will not forward the data anywhere else. Figure 4.13 shows how a network endpoint can shut down a faulty connection stuck in the open state.

The disadvantage of fast path collapsing is that the source no longer gets status and checksum information back from every router in a blocked path. The source will continue to get this connection information back from every router for connections which are not blocked in the network, however, when connections are blocked, the fast collapse does not allow the source the opportunity to obtain this detailed connection status. Since this information is of interest for diagnostics rather than basic functionality, fast path collapsing should be supported as a configuration option which can be disabled and enabled by the testing system. Fast path collapsing would be enabled for normal operation and disabled as necessary for fault diagnosis.

4.10 Performance

The performance data presented in the previous chapter (*i.e.* Figures 3.27 and 3.29), were gathered on dilated networks using MRP with fast path collapsing and fault masking. We see that MRP allows the performance to degrade gracefully with the network as faults accumulate in the network.



A router (or interconnect) may develop a fault such that it appears to always be sending data. The example shown above depicts how backward path reclamation allows the non-faulty routers in the path to be reclaimed at the request of the receiving endpoint.

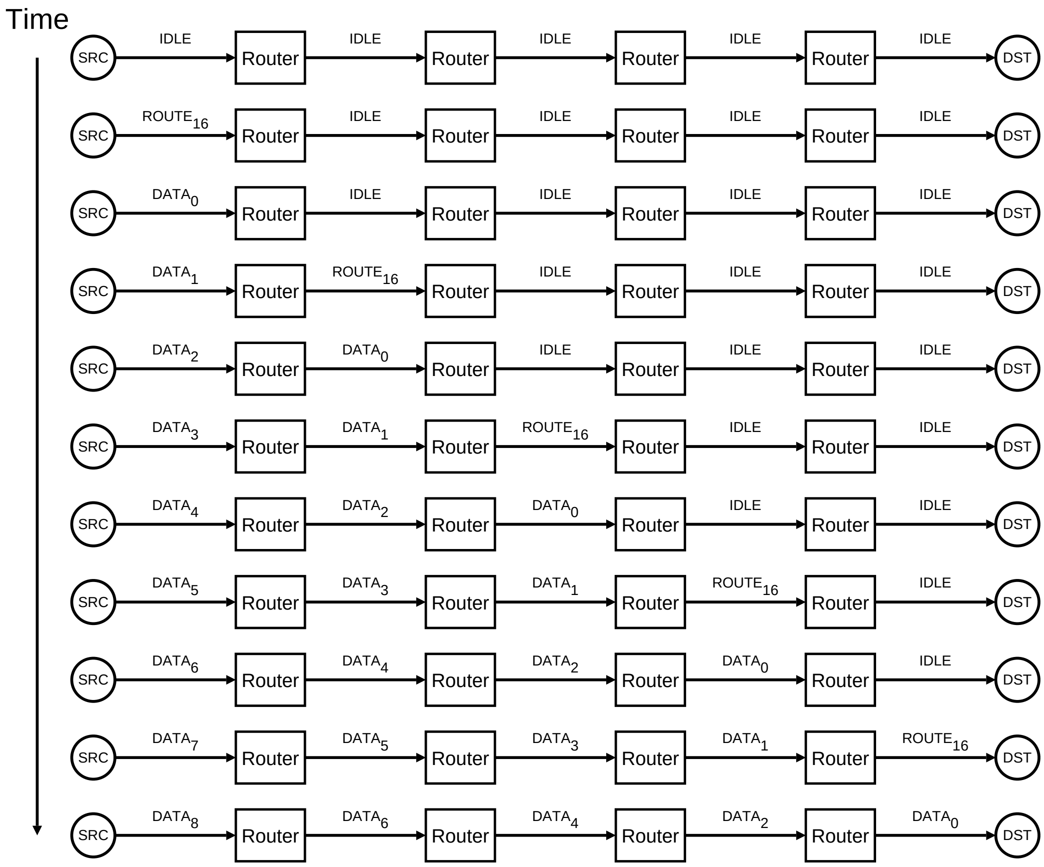
Figure 4.13: Backward Reclamation of Connection Stuck Open

4.11 Pragmatic Variants

There are a number of variants on the basic protocol that arise from implementation considerations. One primary consideration is the difference between the latency involved in performing an operation and the frequency with which we can begin new operations. In this section, we look at several points in the protocol where pipeling the transmission of data may improve the connection bandwidth since the latency involved may be greater than the rate at which new data can be accepted.

4.11.1 Pipelining Data Through Routers

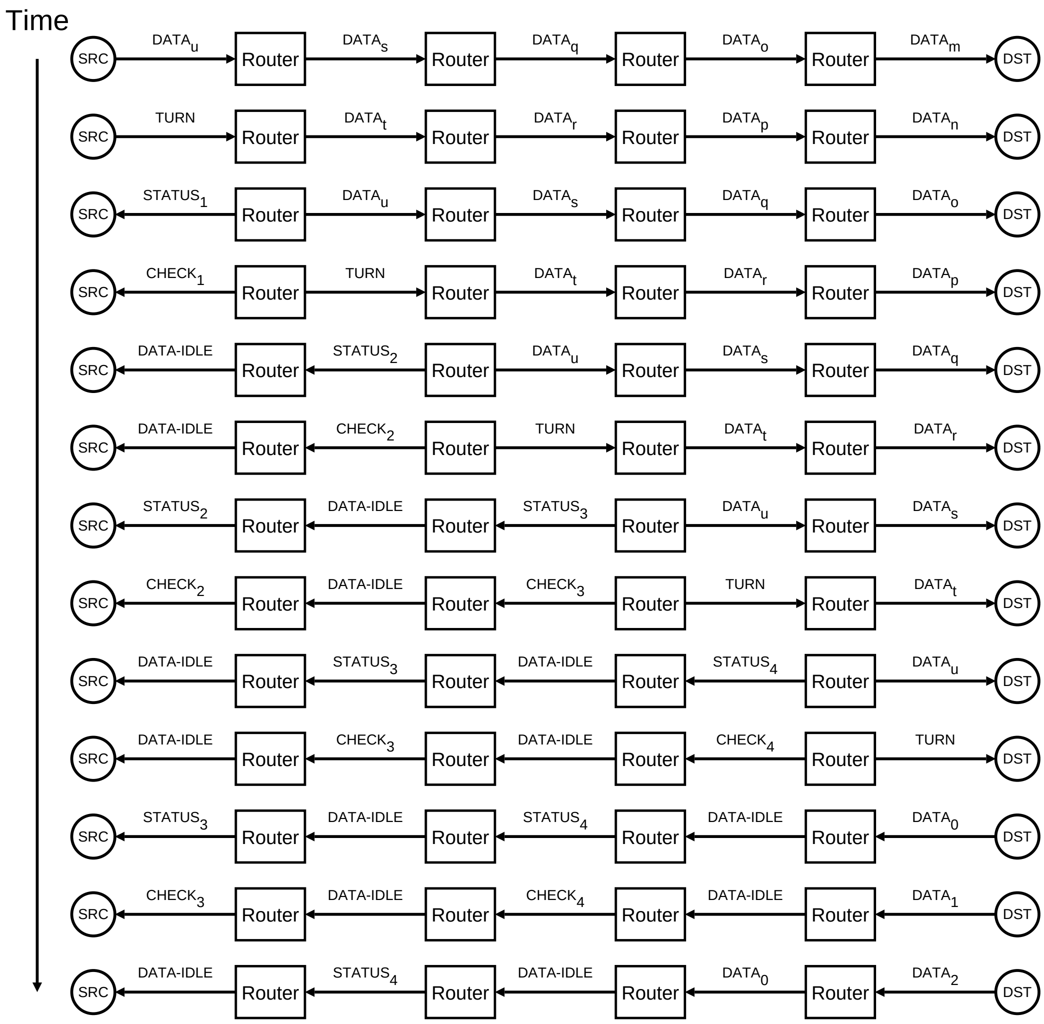
If we can clock data between routing components faster than we can route data through a routing component, we may be able to achieve higher bandwidth by allowing the data to take multiple clock cycles to traverse the routing component. This particularly makes sense when the data can be clocked at a multiple of the switching latency through the routing component. Pipelining data through the routing switches has little impact on the MRP-ROUTER protocol. The one place where it does show up is when connections are reversed. Instead of two clock cycles of pipeline delay before return data is available (See Figure 4.3), there will be an additional two cycles for every additional pipeline stage through the routing switch. These additional delay cycles are due to the fact that it is necessary to flush the router's pipeline in the forward direction, then fill it in the reverse



To increase the network bandwidth, sometimes it makes sense to pipeline the routing component such that multiple clock cycles transpire between the time data enters and the time data exits the routing component. Shown above is a connection establishment in the case where a single additional stage of pipelining is added to each routing component.

Figure 4.14: Example Connection Open with Pipelined Routers

direction before reverse data can be forwarded along. The additional cycles are not completely wasted since they can be used to send additional connection and router status information back to the source endpoint. Additionally, filler data, such as the DATA-IDLE word, can be used to hold the connection open during the pipeline delays. Figures 4.14 and 4.15 show example pipeline routing scenarios. These additional pipeline delay cycles occur when the connection is turned in either direction.



When pipelining the data transfer through routing components, there will be additional delay cycles when the direction of data transmission is reversed. These cycles arise due to the need to flush the router pipeline in the forward direction and refill it in the reverse direction. Shown above is a connection turn when a single additional stage of pipelining is added to each routing component.

Figure 4.15: Example Turn with Pipelined Routers

4.11.2 Pipelined Connection Setup

The longest latency operation inside a routing component is often connection establishment when arbitration must occur for a backward port. If we require that the connection setup occur in the same amount of time, or the same number of pipeline cycles, as the following data is routed through the component, the latency associated with connection setup will be passed along to become the latency associated with data transfer through the router. Alternately, we can allow more time, generally more pipeline cycles, for connection setup than for data transmission. When we allow this, each router will consume a number of words equal to the difference between the setup pipeline latency and the transmission pipeline latency from the head of each data stream.

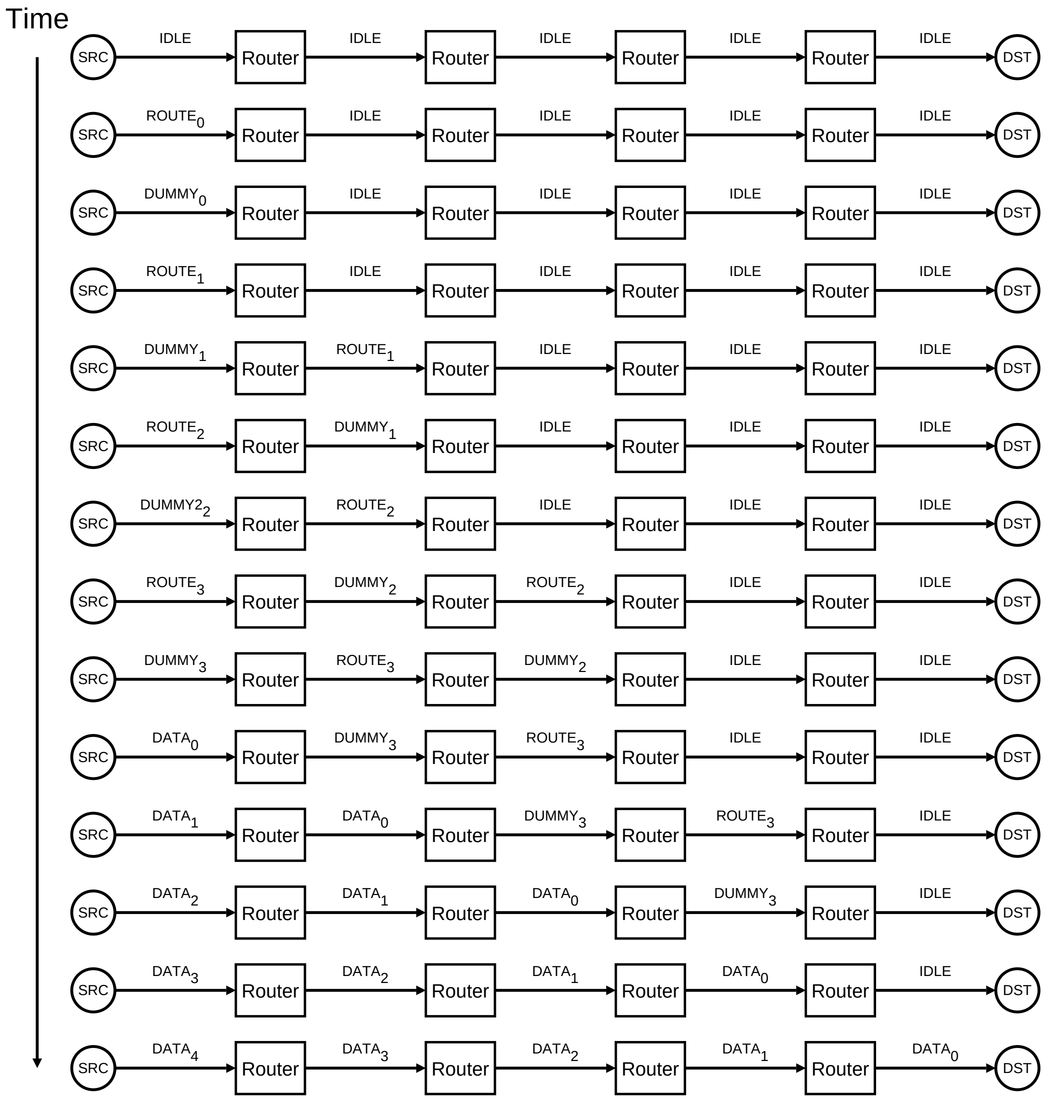
Consider, a router which can route data along an established path in a single cycle, but requires three cycles to establish a new connection. The router would consume the first two words from the head of each routing stream before it was able to establish a connection and forward the remainder of the data out the allocated backward port. Figure 4.16 shows the data flow during connection establishment in this scenario.

When pipelined connection setup is used, there is no need for an explicit swallow capability on each router since each router always consumes one or more words from the head of each data stream it routes. The only other accommodation needed for this kind of setup, is that the endpoint construct the header with the appropriate padding between routing words such that each router in the sees the proper routing specification.

4.11.3 Pipelining Bits on Wires

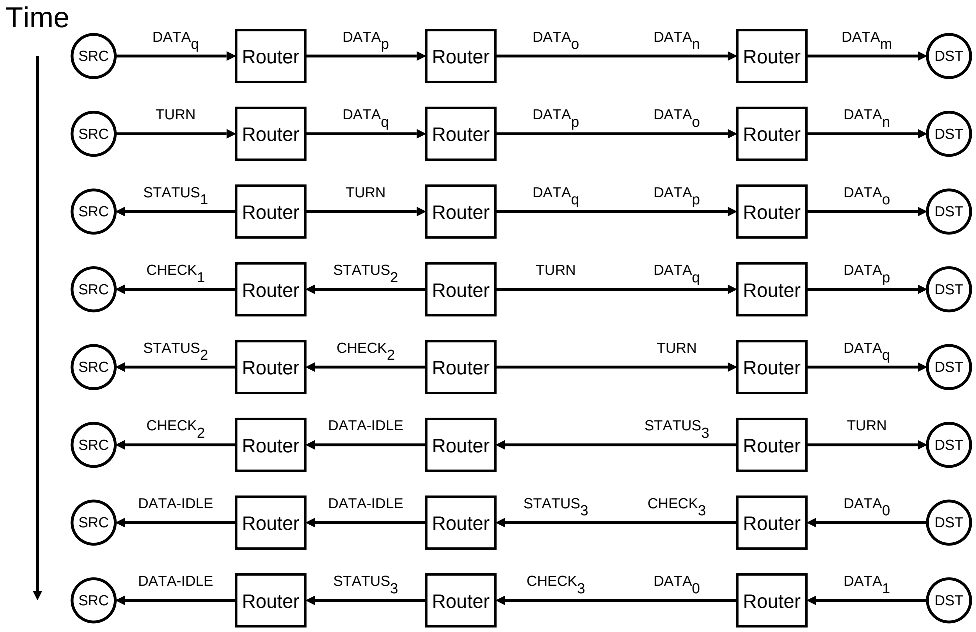
In Section 3.2, we noted that in many of the networks under consideration, the transit time on the wires between routing switches often exceeds the rate at which new data can be clocked. In that section, we suggested that we could pipeline multiple bits on the wires to prevent the bandwidth from being limited by the length of long wires. In many ways, this technique is analogous to pipelining data through a routing component as discussed in Section 4.11.1. With pipelining, we can support a data rate which is higher than the transmission latency through the routing component or across the wire. The effects of wire pipelining on the routing protocol are virtually identical to the effects of pipelining on the routing protocol. Like the router pipeline, when a connection is turned, each wire pipeline must be flushed in the initial direction, then filled in the reverse direction before backward data is returned. Again, filler data, such as the DATA-IDLE word, must be inserted into the return stream to hold the connection open while the wire pipeline is being reversed. Figure 4.17 shows a turn scenario when bits are being pipelined on the wires between components.

One important assumption is that the wire between two components can be modeled as a number of pipeline registers. How one ensures that this assumption is satisfied is, of course, an important implementation detail. With a properly series terminated point-to-point connection between routers, we do not have to worry about reflections and settling time on the wire. The wire will look, for the most part, like a time-delay. The necessary trick is to make the time-delay approximate an integral number of clock cycles so that it does behave like a number of pipeline registers. A brute-force method is to carefully control the length and electrical characteristics of the wires between components. A more sophisticated method is to use adjustable delays in the pad drivers themselves to adjust the chip-to-chip delay sufficiently to meet the pipeline assumption.



In situations where data transfer can occur with significantly less latency than connection setup, it may make sense to pipeline the opening of the connection. The example above shows the case where a connection cannot be established to forward data from the forward port to the backward port until two cycles after the initial route request arrived. As a result, the routing word and the word immediately following it are not forwarded onto the next router. Once the route is setup, data flows through the routers in the normal pipelined fashion with no additional delay.

Figure 4.16: Example of Pipelined Connection Setup



Depicted above is a turn occurring in a 3-stage network where there is a two clock cycle long wire between stages two and three. Since it takes two clock cycles to traverse the long wire in each direction, during a turn the second stage router must fill in the delay cycles with DATA-IDLE words while waiting for return data from the third stage router at the far end of the long wire.

Figure 4.17: Example Turn with Wire Pipelining

Chapter 6 looks at signalling techniques to support this assumption in further detail.

4.12 Width Cascading

In Section 2.7.1 we noted that the number of i/o pins on an IC is limited by perimeter constraints. We also noted that the number of available i/o pins is growing slowly, compared to die area. In Section 3.6 we pointed out that this limitation requires a trade-off between the width of the data channels in the network, the radix of each routing component, and the dilation of each routing component. To first order, the number of i/o pins required for a square router with radix r, dilation d, and data channels of width w is $(2 \cdot r \cdot w \cdot d)$. Since the total number of IC pins at a given technology and cost is a fixed constant, k_{pins}, we must chose r, w, and d as:

$$k_{pins} = 2 \cdot r \cdot w \cdot d \tag{4.1}$$

Alternatively, we can consider how to break the function of a single logical routing component into multiple ICs. If we can split the function of a router across several ICs, we benefit either by being able to build larger primitive routing components or by being able to use smaller and hence cheaper ICs in building our routing components. Of course, this segregation will only be a net benefit if we can do it without incurring the cost of a large number of i/o pins interconnecting the constituent ICs.

Width cascading is a technique for building routing components with wider data channels from components with narrow data channels. This allows us to build a logical routing component with a large width, without directly sacrificing radix and dilation according to Equation 4.1. Rather, we can cascade m routing components to achieve a width $w_{cascade}$ which governed by the following equations:

$$k_{pins} = 2 \cdot r \cdot w_{router} \cdot d \tag{4.2}$$

$$w_{cascade} = m \cdot w_{router} \tag{4.3}$$

Of course, equation 4.1 was just an approximation of the pin requirements and Equation 4.2 holds only to the extent that the overhead, in terms of pins interconnecting cascaded routers, is small compared to the total number of pins on the router.

4.12.1 Width Cascading Problem

Width cascading exploits the basic idea that we can replicate the network and combine corresponding data channels to achieve a wider data path. This replication and channel grouping will only have the desired effect of behaving like a wider data network if the routing performed in the two networks is identical. That is, the data launched at the same time along the same path should arrive at the destination simultaneously or block at the same stage in the network. This problem is complicated by the following aspects:

1. Faults may occur affecting one copy of the network differently from the other
2. The techniques in use for selection among available, equivalent outputs rely on randomization

On a more basic level, then, the problem becomes that of ensuring that the group of routers comprising a single, logical router handle data identically.

Our basic problem is to keep a set of *cascaded routers* in mutually consistent states. That is, the set of primitive routing elements which act as a single logical routing component should connect their crossbars such that the same forward ports are transferring data with the same backward ports. Each crossbar router must, therefore, allocate connection requests in the same way. As long as the following conditions hold, the routers be in mutually consistent states:

1. each router sees the same connection requests
2. each router services the connection requests in the same way
3. each router turns or drops each connection at the same time

The routers may see different connection requests if the routing specification is damaged due to some fault. Once the routers in the cascade see different connection requests for a logical data channel, the routers may handle the route differently. If any of the routers then differ in their assignment of a backward port, the subsequent router stage will not see uniform incoming traffic; that is, the logical data channel will be split and parts of the data stream will go to different logical routers causing these routers, in turn, not to see identical connection requests.

Non-dilated routers in the absence of faults will route identical connection request in identical directions, as long as the routers were already in a consistent state prior to the arrival of new connection requests. However, a fault inside the router IC may cause a router to behave improperly and misroute data or open or close connections in a manner inconsistent with the incoming data stream. Also, with faults the cascaded routers may see different connection requests. With dilation, if the routers utilize their freedom to route connection out different channels in the same logical direction, the subsequent stage will see split traffic.

With faults, it is possible that one portion of the data stream appears to drop or turn the connection when another does not. This can lead to some of the routers in the cascade freeing and reclaiming backward ports while others do not. The routers will then be in inconsistent states since they do not all have the same set of backward ports available to service incoming connection requests

4.12.2 Techniques

We can address the difficulties associated with width cascading by using four simple techniques:

1. Identical control fields
2. Shared randomness
3. Wired-AND in-use indication on backward ports
4. End-to-end checksums across logical data channels

The first thing we need to ensure is that each router in a cascade will see the same sequence of control fields in the absence of faults. This is easy to guarantee by simply constructing the ROUTE, TURN, DROP, and DATA-IDLE values provided to the wide, logical connection into the network from copies of the corresponding control words for the primitive routing elements. We must also construct the wide data such that each routing component sees the same values in its control fields. In the absence of faults and dilation, this will ensure each router sees the same connection requests and the same turn and drop indications.

We want a set of cascaded routers to make the same random decision in response to any incoming connection request. To satisfy this requirement, we provide one or more input pins to each routing component for "random" data. This externalizes the randomness so that it can be shared among the routers in the cascade. We can then make sure that each router makes a deterministic assignment of connections to available backward ports based on the connection requests and the random input values. We call this technique of using shared, external randomization *shared randomness*. To avoid the need for extra components to provide random bit streams to feed the random inputs, we can allow each routing component to generate one pseudo-random bit stream. The random bit streams can then be wired to random inputs as appropriate when the chips are composed in a network.

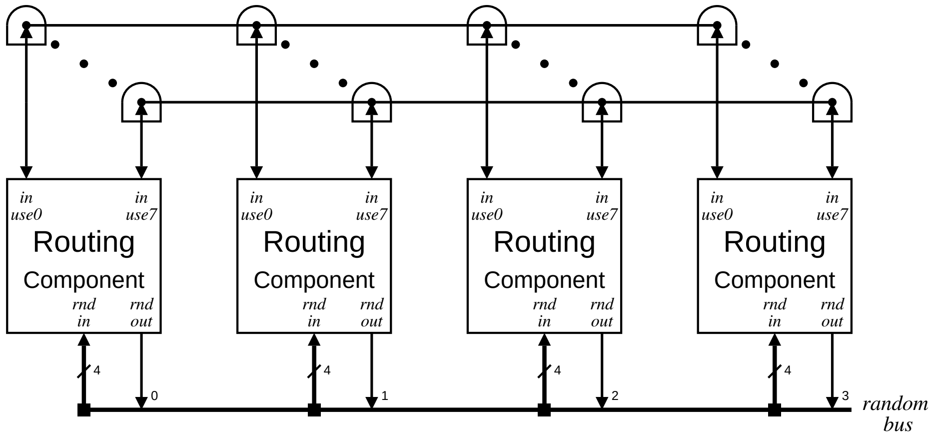


Figure 4.18: Cascaded Router Configuration using Four Routing Elements

The critical piece of state information which needs to be maintained in a consistent manner among routers is which backward ports are being used by which forward ports. Unfortunately, directly comparing this state among routers on every clock cycle would require communicating $(i \cdot o)$ bits among routers each clock cycle. Noting that most connections are many words long and hence connection requests tend to occur infrequently, we can approximate this information by simply comparing which output ports are actually in use on each cycle. This approximation may be imperfect in determining all possible faulty connections, but it will detect most faulty connections immediately. We can arrange for the router to recover from almost all faulty connections quickly.

To make the comparison, we add a single extra pin per backward port. This pin is used by the router to indicate when the backward port is being used. We then externally connect the corresponding backward-port in-use pins of cascaded routers in a wired-AND configuration. When a router begins to use a connection, it signals that the port is in use. The router then sends the data and monitors its in-use pin. The in-use pin will only be asserted if all of the routers in the cascade agree that the port should be allocated. If, after an appropriate settling time, the in-use signal appears unasserted when a router has tried to allocate a backward port, the router knows that it is in an inconsistent state from its peers, deallocates the backward port, and aborts the connection. In this manner, the cascade will only differ in terms of backward port allocation for at most a transient amount of time when a fault occurs. In most cases, if the fault causes the connection to be misrouted, different backward ports are selected and this wired-AND will detect the fault and clear its effects. Figure 4.18 shows how four routing components using this wired-AND and shared randomness scheme might be connected to form a cascaded router.

The only time a fault will fool this approximation is when multiple connections are being opened simultaneously through the router. In this case it is possible that multiple backward ports will be assigned during that same cycle. If the fault causes a set of multiple data streams to be connected to the same set of backward ports, but in a different configuration on each router, the wired-AND will be mistakenly indicate that the connections are valid. Aside from the fact that both faults and connection requests occur infrequently, we do have some reprieve in this case. If this occurs in some stage other than the final stage of the network, it is generally likely that the spliced connections will be going to different destinations. As a consequence, the connections are likely to diverge in a subsequent network stage. When they do diverge, the wired-AND will shut down the connections. If we are using the fast path collapsing described in Section 4.9.2, once dropped the connection can be collapsed quickly from the point of divergence. Consequently, the routers will quickly recover from this inconsistent state.

However, if the fault occurs late in the network or the spliced connections happen to be going to the same destination, it is possible that a network output will see the spliced connection. At the endpoint we can make use of the forward checksum to determine that the data stream arriving at the endpoint is invalid. For this reason, the forward checksum in a cascaded network should be computed across the entire width of the logical data channel rather than on the individual data streams.

4.12.3 Costs and Implementation Issues

The first order cost for supporting this width cascading is:

1. o additional i/o pins on the router (one for each backward port)

2. Several random input and, perhaps, one random output pins

Externally, we will have to place a pull-up on each of the wired-AND pins, and we will have to route the shared random bits and the in-use control lines. In order for the settling time on the in-use wired-AND to be sufficiently small, the primitive routing elements in a cascade must be physically near to each other. Since the state sharing between components is done with the wired-AND structure, the number of components supported by a cascade is limited by the physical locality rather than any architectural features. One reasonable option for large, high-performance cascades is to package a cascade together as bare die on a multi-chip module (See Section 7.6). The multi-chip module will allow the shared interconnect to be very short. Of course, if one is willing to fix a maximum size to the cascade and dedicate input pins on each router for its neighbor's in-use signals, it is possible to avoid the electrical issues associated with the external wired-AND.

4.12.4 Flexibility Benefits

Width cascading allows us to improve the bandwidth without increasing the size of the routing component. Width cascading also improves transmission latency, $T_{transmit}$, by allowing a message to be transferred to and from the network in fewer clock cycles. Additionally, it opens the option to make narrow channel primitive routing components allowing us to take advantage of the limited pin resources for increasing the radix or dilation. Increasing the radix will decrease switching latency, T_s, while increasing the dilation will decrease contention and increase fault-tolerance.

4.13 Protocol Features

In Section 4.1 we identified five key features we hoped to design into a good routing protocol. In this section, we briefly review how the METRO Routing Protocol addresses these issues.

4.13.1 Overhead

MRP keeps overhead low by being minimal. The only overhead data which must accompany data through the network are:

1. Routing words which determine the destination
2. Forward checksum(s) used to insure data integrity

The forward checksums are untouched by the routers in the network, leaving freedom for the endpoints to use checksums of the appropriate length to guarantee sufficiently low likelihood of corrupt message acceptance for the application. The routing words need only constitute sufficient bits to fully specify the desired destination.

Routing words directly specify a desired output direction so that minimal processing is required to handle each incoming connection request. Each routing decision is simple, allowing switching to occur with low overhead. End-to-end acknowledgments allow routers to simply discard unroutable connections.

4.13.2 Flexibility

By being minimal and uncommitted, MRP can be easily adapted to a wide range of higher-level protocol requirements efficiently. The basic protocol is lightweight and unencumbered with detailed packet semantics. This allows applications or custom network interfaces the opportunity to impose the packet structure which best suits them. Transmission length is unlimited by the basic router protocol and a single connection may be reversed any number of times as is appropriate for efficient data transfers.

4.13.3 Distributed Routing

MRP has the desired distributed routing property. No global knowledge is required to route through the network. The incorporated randomization even obviates any requirement for global knowledge of faults. At the same time, local reconfiguration can be used to mask the effects of faults without requiring any single entity to maintain a notion of the global state of the network. Dilated routers allow the network to make more efficient detailed routing decisions based on local information.

4.13.4 Fault Tolerance

End-to-end checksums allow corrupted connections to be detected. Source-responsible retry coupled with end-to-end acknowledgments makes certain that each data stream is successfully transmitted to the desired destination at least once. The combination of randomization in path selection along with multipath networks allows the protocol to eventually find any fault free path which exists between a source and destination pair. The ability to shut down connections from both ends of a connection allows faulty connections to be collapsed by either endpoint. Together, these features combine to guarantee that:

1. Any data corruption is detected

2. Any failed or corrupted connection will be retried until positively accepted by destination
3. As long as the network continues to completely connect all communicating nodes, the protocol can determine a fault-free path which allows any connection to be made

These features are guaranteed regardless of whether the fault is dynamic, static, or transient and even when faults occur during an ongoing transmission. Further, simple reconfiguration options can be integrated into the protocol to make it possible to minimize the detrimental effects of any identified, static faults on network performance.

4.13.5 Fault Identification and Localization

A combination of forward and backward checksums along with connection status information provide MRP with the ability to locate and detect faults. Forward checksums guard all data streams through the network to make sure that no fault which affects data transmission goes undetected. Backward checksums provide an estimate of where faults may have occurred. The backward checksums provide an at-speed indication of the data integrity between a source and some point inside the network. Status indications help localize faulty routing components by detailing the actual path taken by any route in the network. Status information and backward checksums fill pipeline reversal slots which have no data to transmit otherwise. As such, transmission of this data adds no overhead to the routing protocol. Additionally, since end-to-end acknowledgments actually verify the integrity of the received data, this backward status data can be ignored during normal operation.

4.14 Summary

In this chapter, we have described a source-responsible, pipelined circuit-switched routing protocol suitable for use with multipath networks. We saw that randomization, coupled with end-to-end message checksums and source-responsible retry led to a protocol that handles dynamically occurring faults. We also saw that such a protocol could be realized with low overhead and simple, decentralized control. With some simple optimization to the basic protocol, we saw how good performance can be achieved even when faults exist in the network. Finally, we saw how the protocol was easily adapted to accommodate several pragmatic options which arise based on the relative performance we can extract from our implementation technology.

4.15 Areas to Explore

This chapter has demonstrated the existence and details of a protocol suitable for achieving good performance and dynamic fault-tolerance with low overhead. There are several areas which merit further exploration relative to optimizing such a protocol. Additionally, there are many features suggested in this chapter which could be further quantified.

1. In Section 4.7.3, we noted that there are several options for when and how to retransmit failed data. There is much room for understanding the best strategy for retransmission in multipath networks.

2. The routing scheme described in this chapter is *oblivious*, in that no information about downstream network traffic is used when selecting among available paths. The quality of the routes selected could be improved if routers had some information about the relative likelihood of succeeding in routing a connection through a particular backward port. Finding a reasonably simple and cheap way to get timely congestion information back to a routing component remains an interesting problem. Further, utilizing that information in a way that does not destroy the fault-tolerant properties of the protocol or adversely affect the speed with which routing decisions can be made also remains a difficult problem.

3. In Chapter 3 we presented aggregate data for the performance of this protocol on dilated multipath networks. It would be worthwhile to quantify the relative benefits associated with each of the key features. Particularly, it would be desirable to quantify the following:

   (a) The benefit gained by using dilated routers in a network rather than using an equivalently sized non-dilated routers in an extra-stage network with comparable fault-tolerance

   (b) The benefit offered by fast path collapsing for various networks and usage patterns

   (c) The benefit offered by masking faults as opposed to leaving them exposed in the network

   (d) The impact of various pipelining strategies on network performance

   (e) How well this heuristic routing strategy stacks up against a centralized router which does have global knowledge of all connection attempts in the network

4. The strategy presented here is based entirely on circuit switching. It derives many of its end-to-end benefits and timely retries for low-latency data transmission from the circuit-switched nature of the protocol. However, when the network becomes large, in terms of the number of pipeline stages required to get from source to destination, compared to the length of the typical data-stream, it will become less efficient. In such scenarios, a connection is held open longer to hold the connection for the reply than to actually transmit data. We can conjecture that in such a situation packet switching would utilize the interconnect more efficiently, by allowing the router resources to be used by other connections while waiting for the network pipeline to flush and refill in the reverse direction. It would thus be interesting to find a packet switching scheme which offers comparable fault tolerance and low-latency communication benefits as this protocol while allowing more efficient use of routing resources.

5. Testing and Reconfiguration[1]

In this chapter we consider how to deal with failures in the network. In the previous chapters, we noted that the network and routing protocol allow us to continue proper operation oblivious of any knowledge about why connections fail. However, we also noted that there is a performance benefit associated with masking the effects of faults. Further, it is useful to assess the integrity of the network. In this chapter, we develop reliable mechanisms for identifying and masking faults and discuss their utility.

5.1 Dealing with Faults

The primary question to address is: "What do we do about faults in the network?" As noted in the previous chapter, we could do nothing. As long as the network continues to completely connect all communicating processors via some path, correct operation will continue. Unfortunately, if we do nothing, there are a number of important things which we do not know. We do not know:

1. How many faults have occurred in the network.
2. If the complete connectivity requirement has been violated
3. How close the accumulated faults come to violating any connectivity requirements
4. Which components are faulty so that they can be repaired

We established in Section 4.9.1 that masking known faults does have a performance benefit. As introduced in Section 2.1.1 some computing models may allow nodes to be isolated from the network. In such cases it is important to determine when isolation has occurred and, perhaps, guarantee that once isolation has occurred the isolated processor(s) do not rejoin computation in an uncontrolled way.

Even though we can operate oblivious of the detailed nature of faults in the network, we still have reason to be concerned about where faults have occurred and how we can lessen their impact on system performance. We want the ability to:

1. Identify faulty component and interconnect
2. Reconfigure the network to mask known faults

Identifying faulty components and interconnect is useful in several ways. This information is necessary for the system to reconfigure itself to avoid the faulty components. It is also necessary to determine which units need physical replacement when the system is scheduled for manual repair. Having an estimate of the faults in the network is also necessary to identify when isolation occurs or

[1]Portions of this discussion were first presented as [DeH92]

how likely isolation will occur when new faults arise. We have already argued that reconfiguration allows improved performance in the presence of faults. Reconfiguration is also a key mechanism for facilitating on-line repair.

In order for this identification and reconfiguration to work reliably and efficiently for our large-scale multiprocessor, it must function:

1. Without human intervention
2. Without taking the entire machine out of service

In normal operation, we expect the system comprising our multiprocessor to identify each fault some time after it occurs and reconfigure around it. We expect this to happen automatically and continuously while the system runs. For large-scale multiprocessors which may have frequent fault occurrences (*e.g.* MTTF of 6 minutes as in Section 2.5), it is inefficient and often impossible to bring the entire machine out of normal operation in order to perform fault diagnostics and reconfiguration. Further, with networks of this size and potential fault frequency, a person cannot manage the reconfiguration reliably, much less economically.

Finally, we expect our fault identification and reconfiguration to be reliable. If these functions are not robust against faults, they may well be useless when actually needed and could present a liability to system integrity.

5.2 Scan-Based Testing and Reconfiguration

As introduced in Section 2.2, the IEEE standard test-access port and boundary scan architecture [Com90] is emerging as an industry standard for component and board testing. Using moderate overhead, the standard allows functional testing of components and structural testing of interconnect. Additionally, the standard allows component specific registers which can be adapted for use in component configuration.

However, the IEEE standard TAP architecture has drawbacks which make it unsuitable for use in large-scale, fault-tolerant systems. The singular and serial nature of the scan path exposes a critical, single point of failure in the test system. Architects are forced either to use a few long serial scan chains or to use many short scan chains. The former allows a fault in a scan path to affect a large number of components while the latter requires significant wiring for the control of many scan paths. Furthermore, the standard TAP architecture integrates no facilities for bringing small portions of the system into diagnostic mode while leaving the remainder of the system in normal operation. As noted in the previous section, it is inefficient to remove the entire system from normal operation for fault diagnosis.

5.3 Robust and Fine-Grained Scan Techniques

In this section, we present three simple additions to standard scan practices which allow scan techniques to be utilized effectively in a fault-tolerant setting. The basic techniques introduced are:

1. Multi-TAP scan architecture – each component is given multiple test-access ports allowing the component to be accessed from any of several scan paths.

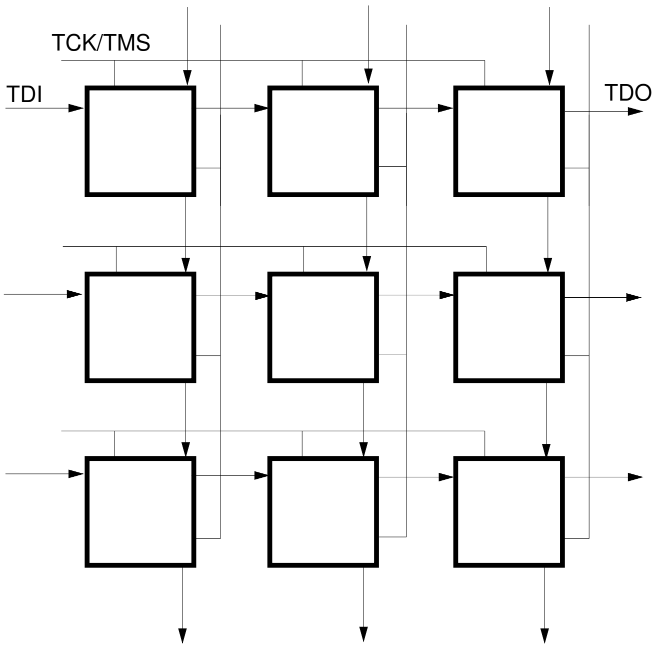


Figure 5.1: Mesh of Gridded Scan Paths

2. Port-by-port selection – each *port* on a component can be independently disabled (See Section 4.9.1).

3. Partial-external scan – each *port* can be scanned in boundary-test mode independently of the operation of other ports on the same component.

With these additions, a scan-based diagnostic and reconfiguration scheme will satisfy our needs for robust, minimally intrusive fault localization and repair.

5.3.1 Multi-TAP

Supporting multiple test-access ports on a single component is a simple extension of the redundant resource and interconnect ideas. With multiple test-access ports, a component's scan capabilities can be accessed through any of multiple, serial scan paths. This allows the component to be tested and reconfigured even when there are faults along some of its scan paths. Further, with multiple TAPs on a single component, scan paths can be arranged so that a minimum number of components are severed from the scan test system by multiple scan-path faults. For instance, we can arrange the scan paths in a system with dual-TAP components such that no two components are on the same pair of scan paths. This guarantees that two faulty scan paths will make at most one component inaccessible. Figure 5.1 shows a gridded topology which has this property.

When adding redundant scan access to a component, there are several issues which must be addressed to assure us that we can realize the potential benefits of having multiple TAPs. We must address the issue of resource contention between the scan paths. For example, two scan paths cannot both perform a boundary scan through the same component at the same time. We must always have the ability to control a component's scan paths from a non-faulty path. This means we must be able to minimize or eliminate any potential for interference from any faulty path(s). We can achieve these goals using two simple techniques:

1. Resource conflict resolution in favor of the most recent requester
2. Sparse encoding of scan instructions

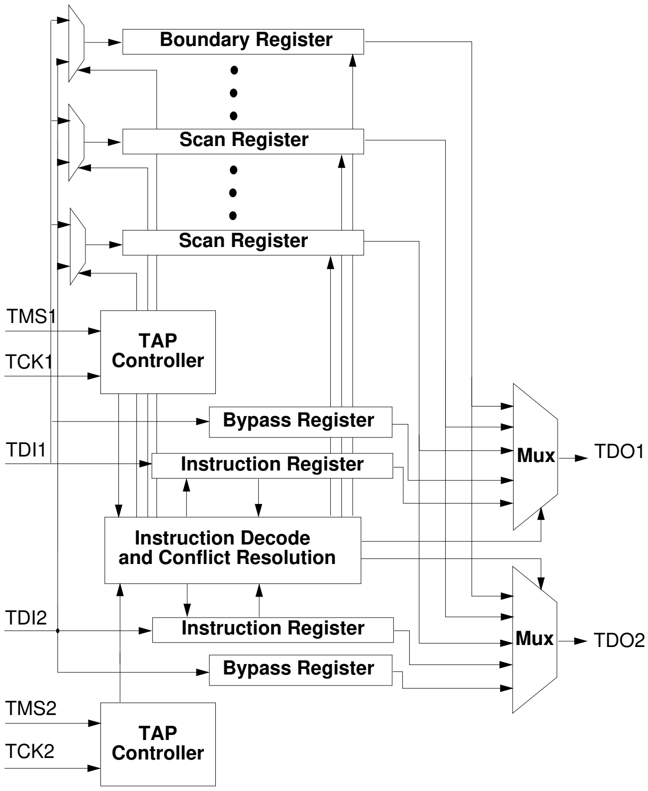
Conflict Resolution

Presumably, access to the scan paths is being coordinated at some level in the system. If everything is working properly, there should never be a resource conflict within a component. However, we are concerned with assuring that reasonable behavior will result even when parts of the system are not behaving properly. We give each TAP its own instruction register and bypass register. These registers behave exactly as in a standard TAP [Com90]. Differences in TAP behavior arise when multiple TAPs attempt to access the same scan registers. This would occur whenever the different TAPs attempted to load instructions that referenced the same scan paths on chip. The simple conflict resolution scheme is to give the TAP loading an instruction most recently access to the path. When the new instruction is loaded, the instruction in any conflicting TAP is reset to the bypass instruction. Since each TAP has its own bypass register, there will be no conflict for access to the bypass register. Assuming we can sufficiently minimize the chances that a faulty scan path can successfully load a non-bypass instruction into its instruction register, this scheme satisfies our fault-tolerance criterion. The scheme allows a non-faulty scan path to wrest a component's scan resources away from a faulty scan path. Figure 5.2 shows a possible architecture for a component with two test-access ports.

Sparse Scan Instruction Encoding

The IEEE TAP protocol for loading instructions is sufficiently involved as to prevent a faulty scan path from successfully loading an instruction in most cases. However, we would like a stronger guarantee that faulty behavior will not interfere with non-faulty access to a component. Simple faults, such as stuck-at faults on the clock (TCK) or mode (TMS) lines will prevent a path from being able to load an instruction. A stuck-at fault in the data lines or data-path of a component (TDI, TDO) will force the downstream component TAPs to see all zeros or ones, making it possible for faults in the data lines to cause instructions with all zeros or ones to be loaded. Of course, stuck-at faults are not the only kind of fault our system must contend with. Sparse instruction encoding is a simple way to make the likelihood that a faulty path can load a valid instruction arbitrarily small.

The basic idea in sparse encoding is to make the number of legal encodings small in comparison to the number of possible encodings. The non-legal instruction encodings all get treated as bypass instructions so that they cannot interfere with the normal operation of the component. Error correcting and detecting codes in common use for data storage and transmission [GC82] [PW72] are common examples of sparse encodings. In this application, we are concerned with detecting errors and preventing them from corrupting non-faulty operation, not correcting errors. If, for



Each scan path has its own instruction register, bypass register, and TAP controller. Each of these is identical to the their counterparts in a single-TAP component. The primary difference in the scan architecture as a result of the multiple TAPs is the instruction decode and conflict resolution unit which replaces the instruction decode unit in a single-TAP component.

Figure 5.2: Scan Architecture for Dual-TAP Component

example, we used a simple instruction encoding scheme which computes an m-bit checksum on an n-bit data word, the space of possible instruction words is $2^{(n+m)}$ whereas the space of legal instruction codes is 2^n. If we assume that the clock and mode bits behave in exactly the correct manner to load in an instruction, but that the data lines hold random data, the chances of a legal code word getting loaded are:

$$P_{random\_load} = \frac{\text{number of legal codes}}{\text{number of possible codes}} = \frac{2^n}{2^{n+m}} = 2^{-m}$$

Of course, when choosing a checksum, one should make sure that the all zero and all one code words are not legal, checksummed instruction encodings.

McHugh and Whetsel propose adding parity to instruction encodings [MT90] to identify corrupted instruction words. Sparse encoding is a more general encoding scheme which allows stronger protection against data corruption.

Costs

Reviewing the dual-TAP example shown in Figure 5.2, we see that the additional costs associated with a multi-TAP component are:

- Four additional i/o pins per additional TAP
- One additional instruction and bypass register per additional TAP
- One additional output MUX per additional TAP
- One conflict resolution unit
- Additional input MUXes for each shared register path

For most modern components, the limited resource is i/o pins (See Section 2.7.1) rather than silicon area. As such, the additional i/o pins will generally be the first order cost associated with a Multi-TAP controller. Note that the size of the conflict resolution unit is proportional to the product of the number of potentially shared resources and the number of TAPs.

Compatibility

As noted above, in the fault-free case, if both scan paths through a component do not attempt to access the same component register, the multi-TAP component will behave identically to a standard single-TAP component. Multi-TAP components place an additional burden on the software to assure that the scan paths through a given component never attempt to load conflicting instructions. In the faulty case, as long as there is a non-faulty path through a component, the fault-free path can be used as a standard TAP as long as the faulty path does not manage to load a conflicting instruction. A standard single-TAP component may be used in a system or scan path with multi-TAP components, but the single-TAP component is susceptible to any faults in its single TAP or TAP control lines.

5.3.2 Port-by-Port Selection

Section 4.9.1 introduced the idea of port deselection for fault-masking. The semantics of disabling a port in this manner imply that the component will ignore the port throughout the time in which the port is disabled. This means the component will not acknowledge any activity on the disabled port, and the component will always choose to avoid the disabled port when seeking service. From the scan path, port selection/deselection is simply accessed as an internal component configuration register.

5.3.3 Partial-External Scan

Once we have a way to selectively remove some ports on a component from normal operation, it makes sense to be able to perform scan testing on each component on a port-by-port basis. This capability gives us finer granularity control over the scan paths allowing us to perform scan tests on subsets of the system while the rest of the system remains in operation.

To support partial-external scan, the component needs to handle additional instructions aimed at selecting the appropriate subset of the normal boundary-scan path. Additional MUXes in the boundary-scan path will be necessary to bypass the portions of the normal boundary path which are not being scanned during a particular partial-scan operation.

5.4 Fault Identification

While describing MRP in Section 4.7.3, we noted that connections could fail when the protocol appeared to be violated, when the destination rejected the incoming data stream, or when a connection was blocked. Any of these cases could be indicative of faults in the network. Blocking occurs even in the absence of faults and is the only case which does not necessarily imply a fault of some sort has occurred. Nonetheless, some faults may manifest themselves as blocking. Additionally, the router checksums, if checked, provide another indication of when and where faults may have occurred.

The fault identification provided by MRP in this way will give us an indication of when faults have occurred and may provide enough clues to narrow the search space for pinpointing the faults. However, MRP's fault identification generally does not tell us exactly what is faulty. Excessive blocking may be due to faults or due to high congestion. Corrupted packets could be due to infrequent, transient faults or due to static faults. It is often useful to distinguish the two, as it makes sense to reconfigure to mask out static faults but it may not make sense to mask out infrequent, transient faults. Since the checksum data travels back along the path of routers, it is always possible that any corruption is inserted by a router between the network input and the actual source of the data. A router closer to the network input could corrupt the checksum and status data of a router further away. Similarly, any router could be responsible for corrupting data returned from the destination. For these reasons, we need a separate mechanism for verifying the integrity of each suspect element in the network.

In the most naive case, we could move the entire system into test mode and use the standard boundary and internal scan facilities to test the integrity of every connection and every component. In this manner, all structural faults in the interconnection can be identified and all functional component faults matching our model (Section 2.1.1) can be determined. Real faulty wires and components can be differentiated from transient faults and congested operation which may trigger false fault theories.

However, if the system is large, the impact of removing the entire system from normal operation for testing can be significant. The larger the system, the higher the rate of single component faults and the larger the amount of hardware that must be removed from service for diagnosis. For sufficiently large systems, it is often neither economical nor practical to remove the entire system from service.

With the additional support described in Section 5.3, we can make the testing significantly less intrusive. The addition of port-by-port selection and partial-external scan provides fine-grain control of scan testing. At a given time, we can isolate a minimal subset of the system that is suspected faulty and perform functional and scan testing. By disabling the ports of all components connected to a physical set of wires and performing scan tests on just those ports on those components, we can quickly determine the integrity of the interconnect in question. Similarly, by disabling all ports on components connected to a given component, we can isolate the single component in question from the network to perform functional testing on that single component. In both cases, the rest of the system may continue normal operation while testing occurs.

This scheme provides a capability for fault-identification and localization which is minimally intrusive. The information gained from this scan testing provides detailed information about the nature and extent of suspected faults. With this information, the system is in a much better position to diagnose the extent of faults, perform reconfiguration to avoid faults, and assess the risks associated with continued operation.

5.5 Reconfiguration

5.5.1 Fault Masking

When faulty components or interconnections are identified, the fault can be masked by reconfiguring the system to avoid the faulty component. Again, the scan-based TAP provides an effective interface to this reconfiguration. The ability to disable a component's usage of a port, introduced in Sections 4.9.1 and 5.3.2, provides one effective means of fault avoidance. If an entire unit is faulty, leaving every port on every component connected to the faulty component in a disabled state will remove the unit from the functional portion of the system so that it cannot interfere with correct operation. Similarly, if faults occur in the wires, drivers, or receivers of an interconnection channel, disabling the port on all affected components will effectively excise the faulty connection from the system.

This mechanism of disabling individual ports works effectively for reconfiguration for exactly the same reasons it was necessary for fine-grained diagnosis. Our multipath network will continue to function correctly as long as there is at least one enabled, non-faulty, path between every pair of nodes in the network which need to communicate. The semantics of disabling a port imply that the component will ignore the port throughout the time in which the port is disabled.

5.5.2 Propagating Reconfiguration

We should note that reconfiguration, both when excising faults and when isolating a region for testing, is best performed accounting for the network structure. If, due to faults or testing, we end up removing all of the backward ports in the same logical direction out of a router, this router becomes a *dead-end* for any connections which are routed through it destined to the logical direction which has no enabled backward ports. We may wish to excise routers with dead-end connections from the network, as well. That is, we use the same basic propagation algorithm used in Figure 3.28 to determine which non-faulty routers should be excised along with the faulty routers to maintain good connectivity in the network. If we do not reconfigure the network this way, the impact falls entirely on performance. As long as paths still exist between communicating nodes, the routing protocol will continue to find the paths through the network. Without this propagating reconfiguration, it is possible for a router to select a backward port which leads to a router which cannot route the data stream any further towards its destination. Propagating the reconfiguration in this manner, prevents a router from ever routing a data stream into such a dead-end situation. It is also worthwhile to note that the impact of not reconfiguring in this manner is lessened when fast path reclamation (Section 4.9.2) is supported. Figure 5.3 shows an example where it may make sense to propagate the reconfiguration and mask additional routing components from the network.

We can also note that it is not always possible to propagate back and mask additional components in the conservative manner suggested above without isolating additional nodes from the network. When the reconfiguration algorithm shown in Figure 3.28 was introduced in Section 3.5.5, it was noted that this reconfiguration was intended for the harvest case where it was permissible to configure some nodes out of the network. In the interest of providing good connectivity for all remaining nodes, the algorithm may end up isolating some nodes for which paths do exist in the network. If we do not wish to isolate nodes from the network, we must verify that the propagation does not leave some nodes disconnected from the network before masking out any additional routers suggested by the reconfiguration propagation. Figure 5.4 shows an example where paths still exist between all endpoints in the network and the algorithm shown in Figure 3.28 would suggest masking components in a manner which isolates nodes from the network.

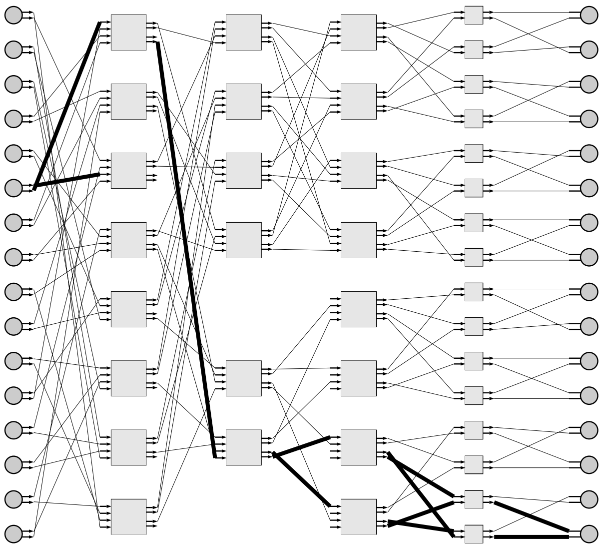
5.5.3 Internal Router Sparing

If a routing component provides sparing within itself, the scan mechanism can be used to reconfigure the unit to swap spares within the router. For some i/o limited components, such as crossbar routers, there may be plenty of additional room for function inside a component whose size is dictated by the pin-limited i/o. In these cases, it may make sense to provide redundant structures on the component. Faults inside some part of the routing component can then be masked by reconfiguring the component to use an alternate portion of the routing component.

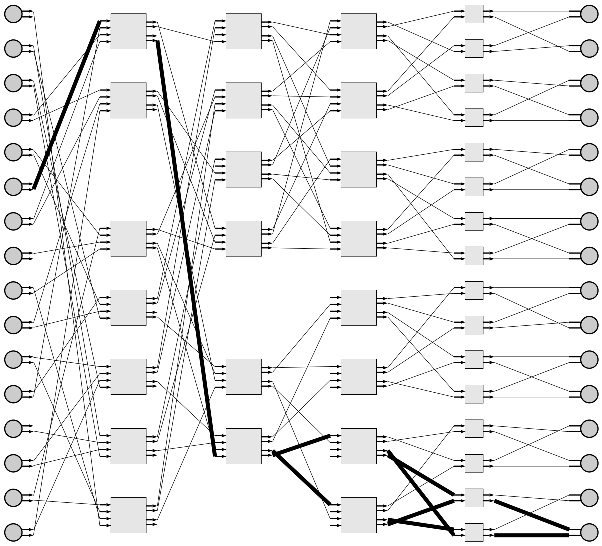
5.6 On-Line Repair

The combination of accurate fault localization coupled with the ability to perform reconfiguration, allows us to realize systems where the fault-repair loop can be closed without human or mechanical intervention, at least up to the fault level provided by the sparing architecture. Programs monitoring the system integrity are empowered to test theories about faults and reconfigure the system to best mask the effects of failures. Further, with a knowledge of the minimal requirements necessary for complete system operation along with an accurate idea of the fault status of the machine, programs can assess overall system integrity.

When outside intervention is necessary to repair the system, these same facilities of port disabling and port-based scan allow for in-operation replacement. If all the ports on all components into a physically replaceable subsystem are disabled, it is possible to replace the physical subsystem without any further interruption of system operation. Of course, the electrical and mechanical design of the system must also be suitable for live replacement (*e.g.* Tandem Non-Stop computer systems [And85], Stratus fault-tolerant computer systems [Web90], Thinking Machines CM5 [Thi91]). Once replaced, scan testing can determine the interconnection and functional integrity of the

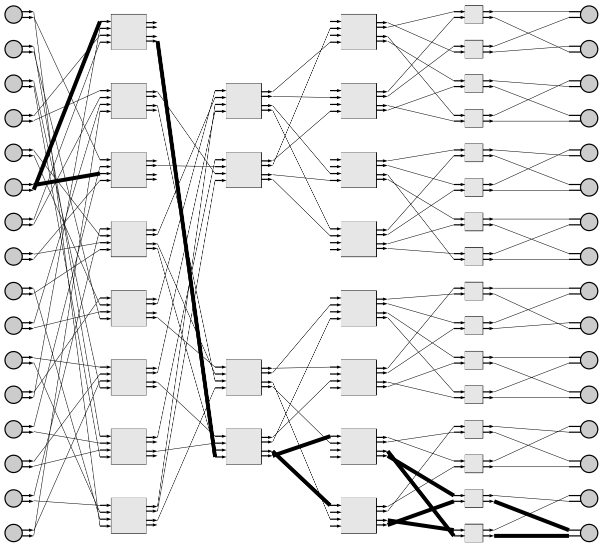


Shown above is a configuration of a multipath network where it may make sense to configure out additional routers (The network without reconfiguration is shown in Figure 3.22). With the loss of the two routers in the second stage of the network, router 3 in the first stage of routers has no outputs in one logical direction. As a result, all connections which need to connect to destinations 8 through 16 which get routed through this router will always block.

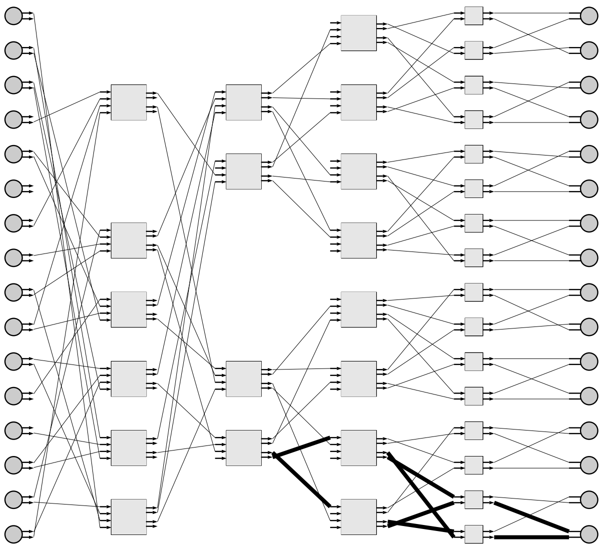


Shown above is the same network with first stage router 3 also configured out of the network. By propagating the reconfiguration, it is possible to avoid the dead-end connections which appeared in the previous configuration.

Figure 5.3: Propagating Reconfiguration Example



Shown above is another reconfiguration of the network first shown in Figure 3.22. This network has similar dead-end path problem to the one shown in Figure 5.3. However, we cannot reconfigure this network to avoid the dead-end paths without isolating a node from the network.



Shown above is the result of propagating reconfiguration to avoid having dead-end paths. As suggested, this propagation results in the isolation of node 6 from the network.

Figure 5.4: Propagating Reconfiguration Example

replaced component. When the replacement is properly installed and identified as functional, the disabled ports into the replaced subsystem can be re-enabled allowing the subsystem to return to full service.

5.7 High-Level Fault and Repair Management

For several reasons, it is desirable to coordinate testing and reconfiguration at a high-level. Particularly, some central coordination is required for:

- Network Integrity Assessment
- Reconfiguration Planning
- Coordinating scan paths
- Collecting fault data from multiple sources

With integrity assessment, we want to determine how safe it is to continue operation. That is, we want to know the likelihood that we will sustain a fault in the near future which will render the machine inoperational or require serious reconfiguration. In a yield based fault-tolerance situation, we simply want to know the likelihood we will retain complete connectivity for a period of time. In a harvest fault-tolerance environment, we may want to know the likelihood of isolating any nodes in a period of time. If we combine the knowledge of the network topology, an estimate of the known faults, and a model of fault-occurrences, we can make these safety assessments.

Integrity assessments can be used for several purposes. In the simplest form, this information allows human operators to assess the danger level associated with continued operation. In such a case, it could be used to schedule down-time for physical repair. The integrity information could feedback into the run-time system and tune operation accordingly. For instance, as the likelihood of complete network failure increases, a system may want to checkpoint the computation more frequently to minimize the impact of the failure. In a harvest situation, the software may choose a certain danger level at which it begins to evacuate the computation and data associated with a given node. If a node can be evacuated before it becomes isolated, the cost of recovering from an isolated node situation can be avoided. If a danger level can be chosen such that nodes can generally be evacuated before isolation, we may be able to get away with simpler strategies for handling node isolation. Simpler strategies generally require less hardware support or have less impact on normal operation.

Actual reconfiguration based on the results of integrity assessment also requires some high-level coordination. As noted, a central notion of faults is necessary to assess their impact and plan reconfiguration as described in Section 5.5.

As noted in Section 5.3.1, multi-TAP scan paths will require central control to avoid potential resource conflicts. High-level central control is necessary to utilize the scan paths for testing and reconfiguration. For large-systems, we would like to avoid a single-point of failure associated with having a single entity responsible for all scan paths. The fact that we have redundant access via scan paths to any component in the network, allows us to distribute the control of these scan paths and tolerate the failure of scan controllers in the same way we tolerate the failure of a scan path. Once we allow the scan path controls to be distributed, their action will need high-level coordination, as well.

When deciding what faults are worth pursuing, it can be useful to collect information from multiple nodes. For instance, when pursuing theories about a faulty network component or interconnect, it may be insightful to integrate fault data from several nodes which may use the component or interconnect in question.

From a fault tolerance perspective, we want to distribute the high-level coordination function. The easiest way to manage this is to choose some subset of the processing nodes in our network, perhaps all of them, and assign these nodes the task of controlling high-level testing and reconfiguration. The control of scan paths could be distributed evenly among all such nodes. Depending on the application, we could either dedicate a set of nodes to performing this function, or we can simply make this part of the work performed by every processor in the system. The network serves to interconnect these coordinators as well as connecting them to the rest of the nodes in the machine.

The coordinating nodes will want to maintain a replicated, distributed database of fault manifestations, system integrity, and current system configuration. Replication is necessary since any of the coordinating nodes may fail or be isolated from the network. The amount of replication will depend on the fault-tolerance requirements for each particular system. When faults are reported or identified, this information needs to be shared among the coordinating nodes in a robust manner.

5.8 Summary

In this chapter, we examined the integration of test and reconfiguration facilities with fault-tolerant networks. We began by reviewing the motivation for fault localizing and masking. With the addition of multiple TAP support, port deselection, and partial-external scan to standard scan-based testing approaches, we developed robust mechanisms to support fault identification and system reconfiguration. We describe how these mechanisms allow scan-based testing and reconfiguration to proceed in a minimally intrusive manner, allowing the portion of the system not being tested or reconfigured to continue normal operation. We also summarized how such a test system facilitates detailed fault identification, system reconfiguration, integrity assessment, and on-line repair. Finally, we sketched how the control and data management associated with fault handling can be integrated with the multiprocessing system, itself.

5.9 Areas To Explore

In Section 5.3.1, we introduce the ability for each component to exist on multiple scan paths. The multipath scan ability gives us the opportunity to wire the scan paths in a way which minimizes the effects of any faults. That is, for a given number of scan paths, we can look for the assignment of components to scan paths which isolates the fewest components at each fault level. Of course, the scan paths are useful to us only because they allows us test and reconfiguration access to the underlying network. The redundancy structure of the network should also be taken into account when selecting an assignment of components to scan paths. For instance, it would probably not be wise to assign all the components in a single stage of the network to the same scan path. Consequently, we also want to optimize the assignment of components to scan paths in a manner which minimizes the effects which scan-path faults may have on network connectivity. That is, if we assume a component which is not accessible via scan paths is faulty, we want to minimize the effects of scan-path introduced faults on network connectivity. Considering the physical attributes of our scan paths, we know that exploiting physical locality when wiring scan paths is beneficial. By keeping scan path connections physically local, we keep the cost of the scan path interconnect down and keep the reliability of the interconnect high. When looking for good assignments of components to scan paths, we would also like to exploit a large degree of physical locality. Determining reliable and practical assignments of scan paths on top of networks with particular redundancy characteristics remains an interesting problem to study.

6. Signalling Technology

In the previous chapters we focussed on the architectural and organizational aspects of achieving robust, low-latency communications. In this chapter and the next we address some of the key physical aspects of high-performance network design. In Chapter 4 where we described the routing protocol at the data link and network levels (See Figure 4.1), we pointed out that MRP could exist on top of any number of physical transport layers. Here, we address the issue of selecting a practical and economical signalling discipline to support such routing protocols with minimum transit latency.

6.1 Signalling Problem

Our primary goal in selecting a signalling discipline is to transmit data between components with minimum latency while affording high bandwidth. The transit latency, T_t, and to some extent the component input/output latency, t_{io}, will depend on the design of our signalling discipline as well as the basic technologies available for implementation. Since we are interested in large-scale computer networks with inter-router delays which grow as the system grows, we are concerned with signalling over potentially long interconnect lines. As a consequence, our interconnection medium will behave like a transmission line, and our signalling problem becomes a transmission line design problem.

6.2 Transmission Line Review

In this section, we review the salient features of transmission line signalling. See [RWD84] for a thorough treatment of transmission lines.

For most physical interconnection media in use in digital systems, the effects of signal attenuation and phase distortion can be ignored. If we further ignore the bandwidth limits of the media, there are two primary characteristics which describe a transmission line interconnect, *impedance* and *propagation velocity*.

The propagation velocity, v, characterizes the speed at which electrical waves traverse the interconnect. When the voltage across a transmission line changes at some point, a voltage wave propagates down the transmission line at the rate v. The propagation velocity is determined by the materials in the interconnect and is given by Equation 6.1.

$$v = \frac{1}{\sqrt{\mu\epsilon}} \tag{6.1}$$

For most materials $\mu \approx \mu_0$, where μ_0 is the permittivity of free space. Conventional printed-circuit boards (PCBs) have $\epsilon = \epsilon_r \epsilon_0$, where $\epsilon_r \approx 4$ and ϵ_0 is the dielectric constant of free space. We know

$$c = \frac{1}{\sqrt{\mu_0 \epsilon_0}}$$

where c is the speed of light in a vacuum. Standard PCBs, thus, have a propagation rate about half the speed of light.

The characteristic transmission line impedance arises from the distributed inductance and capacitance between the signal conductor and its associated ground path. As long as the geometry of the interconnect is fairly constant, the distributed inductance and capacitance remain constant as well. The distributed inductance and capacitance give rise to a characteristic transmission line impedance, Z_0. While transient voltage waves are propagating down the segment of transmission line, Z_0 defines the transient voltage to transient current ratio. Z_0 is a function of the conductor geometry and the dielectric materials involved. For most conventional PCB technologies and interconnect geometries, $Z_0 \approx 50 \pm 15\Omega$. $Z_0 = 50\Omega$ is a conventional standard for PCB interconnect and high-performance cabling.

As long as the interconnect presents a uniform characteristic impedance, the driven voltage wave propagates along the interconnect at velocity, v. Real interconnect, however, is not infinitely long, so we must consider what happens to the signal when it reaches the end of the transmission line. To understand what happens, let us consider the more general problem of what happens when our propagating wave encounters an impedance discontinuity. When a propagating voltage waves encounters an impedance discontinuity, the discontinuity gives rise to a reflection. At the point of discontinuity, part of the voltage wave may propagate through the discontinuity while part of it may reflect back to the source. In general when we have an incident voltage wave of magnitude V_I from a region of interconnect with impedance Z_0 to a region with impedance Z_1, the reflected and transmitted voltage waves, V_R and V_T, are governed by Equations 6.2 and 6.3.

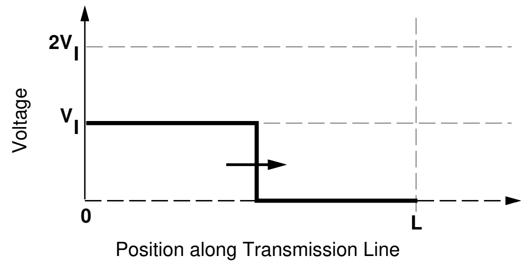
$$V_R = \left(\frac{Z_1 - Z_0}{Z_1 + Z_0}\right) V_I \tag{6.2}$$

$$V_T = V_R + V_I = \left(\frac{2Z_1}{Z_1 + Z_0}\right) V_I \tag{6.3}$$

We can see from these equations that, if we want our signals to be transmitted cleanly between two points, we need to keep the characteristic impedance of the interconnect constant.

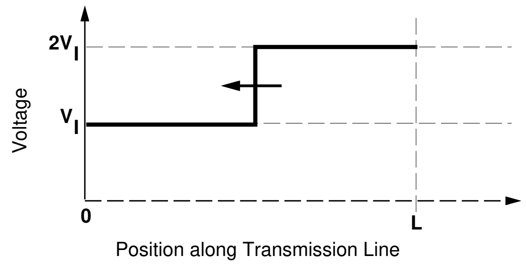
If transmit a signal from a driver at one end of a wire to a receiver at the opposite end, the signalling is called *point-to-point*. Point-to-point signalling can be contrasted to bus oriented signalling where there are several receivers and several potential drivers. The point-to-point signalling case is simpler to understand. Since the signalling between routers is between a single forward and backward port pair, we will focus the remainder of our discussion on point-to-point signalling.

In a point-to-point signalling situation, we generally engineer the wire to have a single characteristic impedance for its entire length. The endpoints of the transmission line, then, become our primary concern. Figures 6.1 through 6.6 show an incident voltage wave and several possible reflection scenarios depending on the ratio of the termination resistance to the characteristic line impedance. If the end of the transmission line is open-circuited (*i.e.* $(Z_{term} >> Z_0)$, Figure 6.2), the reflected voltage wave, V_R, is the same as the incident wave. The receiver at the endpoint thus sees a voltage $2V_I$ following the arrival of the voltage wave. When the transmission line is short-circuited (*i.e.* $(Z_{term} << Z_0)$, Figure 6.6), the reflected voltage wave V_R, is the same magnitude as the incident wave but opposite in polarity. The receiver sees no incident wave. When



At time $t = 0$ there is a voltage transition from 0 to V_I at the source end of the transmission line. Shown above is the voltage profile along a transmission line interconnect during the first transit time (*i.e.* $0 < t < \frac{L}{v}$).

Figure 6.1: Initial Transmission Line Voltage Profile

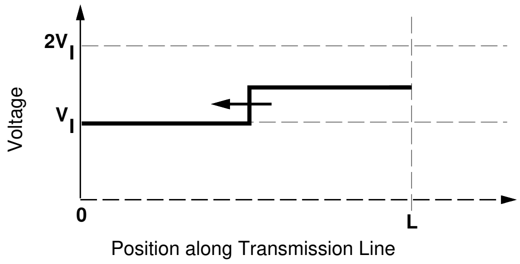


Shown above is the voltage profile along a transmission line interconnect when the incident wave shown in Figure 6.1 encounters an open circuit ($Z_{term} >> Z_0$) at the far end of the transmission line. The voltage profile shown is characteristic of the line during the second transit time across the interconnect (*i.e.* $\frac{L}{v} < t < \frac{2L}{v}$).

Figure 6.2: Transmission Line Voltage: Open Circuit Reflection

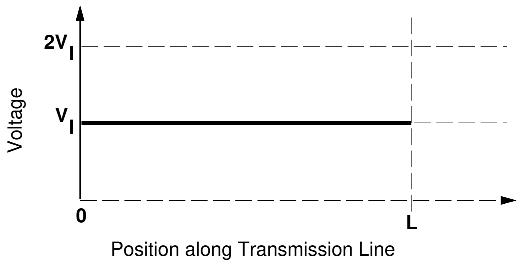
the termination resistance exactly matches the transmission line impedance (*i.e.* $(Z_{term} = Z_0)$, Figure 6.4), there is no reflected wave. If the termination resistance is slightly higher or lower than the transmission line impedance, the reflection will tend between the two extremes (See Figures 6.3 and 6.5). It is important to note that the reflected voltage wave will return to the driving end of the transmission line and encounter the same reflection scenario with the termination impedance defined by the impedance seen at the source end of the transmission line. Reflected waves will continue to propagate along the transmission line until the line reaches the steady-state voltage level defined by the endpoints – *i.e.* in the steady state, the voltage level along the wire will settle to the voltage which the wire would possess if the transmission line were replaced by an ideal wire.

For high-speed signalling, we want to engineer the termination to avoid any unwanted reflections. That is, we want the destination endpoint to settle to the correct voltage as quickly as possible and remain there. Two common methods for achieving this goal for point-to-point signalling are



Shown above is the voltage profile along a transmission line interconnect when the incident wave shown in Figure 6.1 encounters an higher impedance termination ($Z_{term} > Z_0$) at the far end of the transmission line. The voltage profile shown is characteristic of the line during the second transit time across the interconnect (*i.e.* $\frac{L}{v} < t < \frac{2L}{v}$).

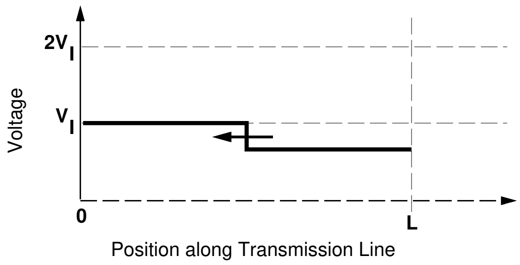
Figure 6.3: Transmission Line Voltage: $Z_{term} > Z_0$ Reflection



Shown above is the voltage profile along a transmission line interconnect when the incident wave shown in Figure 6.1 encounters a matched impedance termination ($Z_{term} = Z_0$) at the far end of the transmission line. The voltage profile shown is characteristic of the line after the first transit time across the interconnect (*i.e.* $t > \frac{L}{v}$).

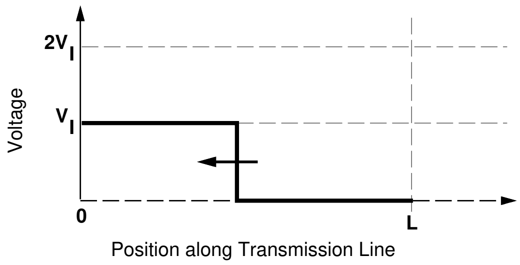
Figure 6.4: Transmission Line Voltage: Matched Termination

series termination and *parallel termination*. Figure 6.7 shows a parallel termination arrangement and voltage profiles at both ends of the transmission line when the driven voltage changes. With parallel termination, we select the driver so that it can drive the transmission line to the desired voltage and select the termination resistance matched to the transmission line characteristic impedance ($Z_{term} = Z_0$). A voltage wave originates at the driver and takes one transmission line transit time to arrive at the receiver. Once the receiver sees the voltage wave, the line remains at the driven voltage until the next transition occurs. Figure 6.8 shows a series termination case. The driver is selected to drive the line voltage to one-half of the desired voltage through a driver impedance equal to the line impedance ($Z_{drive} = Z_0$), and the receiver is left open-circuited ($Z_{term} >> Z_0$). Here the one-half voltage wave arrives at the destination and reflects completely. The receiver thus sees a full-swing transition when the one-half voltage wave arrives and reflects one transit



Shown above is the voltage profile along a transmission line interconnect when the incident wave shown in Figure 6.1 encounters a lower impedance termination ($Z_{term} < Z_0$) at the far end of the transmission line. The voltage profile shown is characteristic of the line during the second transit time across the interconnect (*i.e.* $\frac{L}{v} < t < \frac{2L}{v}$).

Figure 6.5: Transmission Line Voltage: $Z_{term} < Z_0$ Reflection



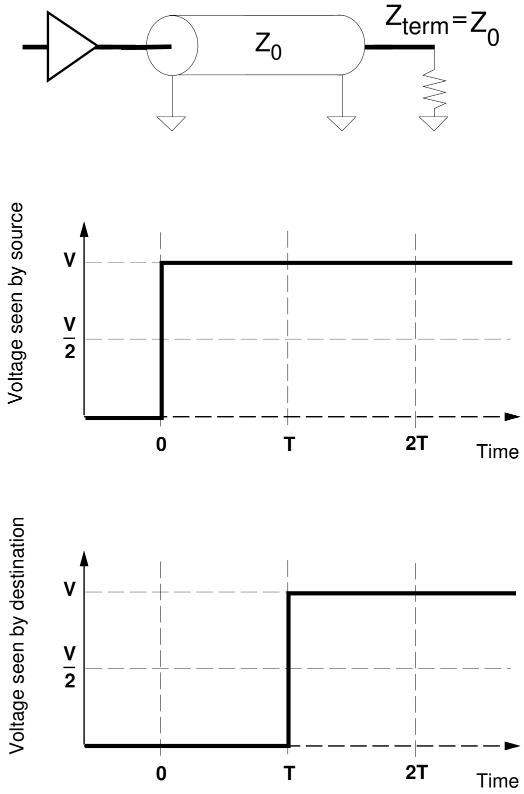
Shown above is the voltage profile along a transmission line interconnect when the incident wave shown in Figure 6.1 encounters a short circuit ($Z_{term} << Z_0$) at the far end of the transmission line. The voltage profile shown is characteristic of the line during the second transit time across the interconnect (*i.e.* $\frac{L}{v} < t < \frac{2L}{v}$).

Figure 6.6: Transmission Line Voltage: Short Circuit Reflection

time after the source drives the transmission line. The reflected wave arrives at the source end of the transmission line one transit time later or one round-trip transit time after the source drove the original one-half voltage wave. When the reflected wave arrives at the source, it is terminated by Z_{drive} and no further reflections result.

6.3 Issues in Transmission Line Signalling

Now that we have reviewed the key features associated with transmission line signalling, we can consider the issues associated with high-speed, point-to-point signalling in terms of transmission line design. By using series or parallel termination, we can control the voltage wave form on the



Shown at top is a parallel-terminated transmission line. Below the transmission line are the voltage profiles seen by the source and destination ends of the transmission line after the driver forces a $0 \rightarrow V$ transition at the source.

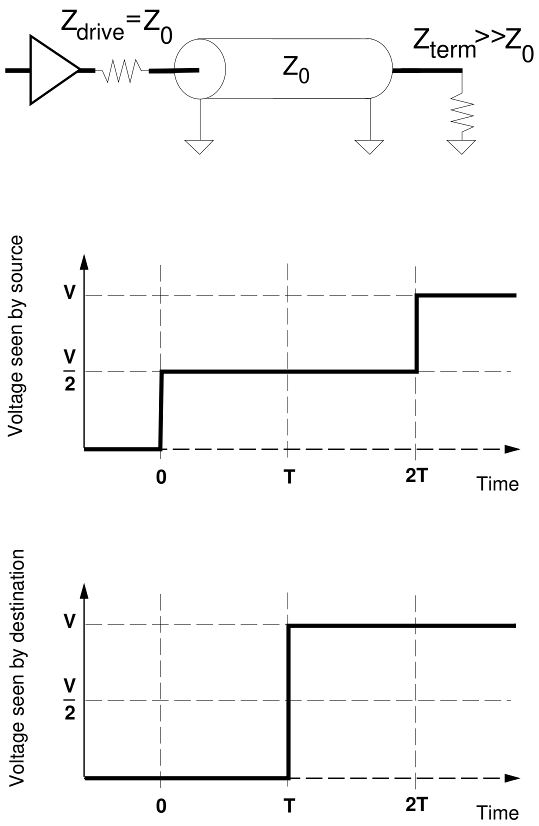
Figure 6.7: Parallel Terminated Transmission Line

transmission line so that the destination settles to the desired voltage in one transit time.

$$T_t = \frac{L}{v} \tag{6.4}$$

As shown in Equation 6.4, the transit time depends on the length of the transmission line, L, and the rate of signal propagation, v. From Equation 6.1 we know that the rate of propagation depended on the properties of the materials. For most conventional, low-cost PCB technologies, $v \approx \frac{c}{2}$. High-performance substrates with slightly higher propagations speeds exist, but their cost and reliability currently limits their use to small, high-end designs.

A key issue to guaranteeing that the destination end of the transmission line does settle to the desired voltage level in a single transit time is proper termination. In both the series and parallel termination cases, we require a termination which is matched to the transmission line characteristic impedance. Process variation in the manufacture of printed-ciruit boards and components complicates the ease with which we can achieve matched termination. A typical PCB manufacturer



Shown at top is a series-terminated transmission line. Below the transmission line are the voltage profiles seen by the source and destination ends of the transmission line after the driver forces a $0 \rightarrow V$ transition.

Figure 6.8: Serial Terminated Transmission Line

will only guarantee the impedance of the PCB signal runs within about $\pm 15\%$. Tighter bounds can be specified but always at higher costs. Additionally, there is the issue of how the matched termination is fabricated. External, surface-mount resistors or resistor packs can be used for termination with moderately high accuracy. However, termination for a large-pin count component, such as a routing component, can require a sizable area on the PCB. This translates into cost for the termination components, for the PCB real-estate, and for the added complication of assembly. The space required for termination also translates into larger distances between components and hence longer transit latency and lower reliability. External, fixed resistors also pose problems if we wish to reverse the direction of signal transmission across our transmission line, as is desired when we reverse an open connection in our network.

Another key concern when driving transmission line interconnects is the power required to drive the transmission line. The power supplied by the driver is determined by the driven voltage and resistance seen by the driver (Equation 6.5).

$$P_{drive} = \frac{(\Delta V_{line})^2}{R} \tag{6.5}$$

A parallel terminated transmission line will dissipate power as shown in Equation 6.6 when driving the transmission line to V.

$$P_{parallel\_drive} = \frac{V^2}{Z_0} \tag{6.6}$$

A series terminated transmission line will dissipate power given by Equation 6.7 for one round-trip transit time following any transition. Once the reflection returns to the source, no power is dissipated in the steady-state condition.

$$P_{serial\_drive} = \frac{V^2}{2Z_0} \tag{6.7}$$

Additionally, power is required to charge the capacitance associated with the driver. This power is given by Equation 6.8, where f is the frequency at which the driver switches, ΔV_{driver} is the voltage swing driving into the driver, and C_{driver} is the capacitance which must be charged or discharged to change the voltage on the driver.

$$P_{charge} = \frac{1}{2} C_{driver} (\Delta V_{driver})^2 f \tag{6.8}$$

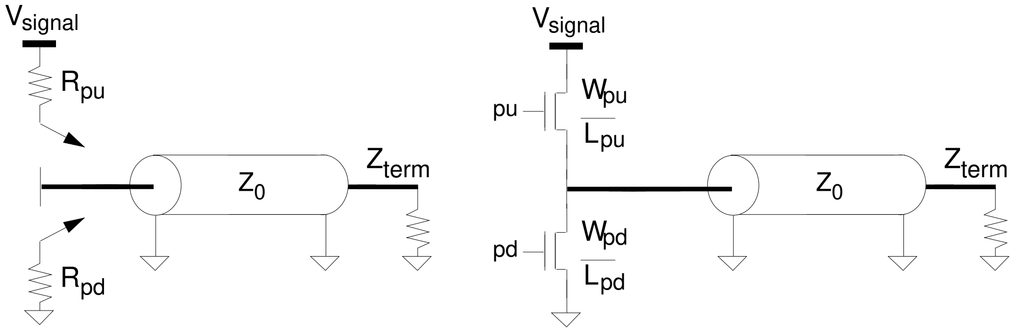
6.4 Basic Signalling Strategy

To meet the needs of point-to-point signalling with high speed and acceptable power, we utilize a series-terminated, low-voltage swing signalling scheme which uses on-chip termination and feedback to match termination and transmission line impedances. For the purposes of the following discussion, we focus on a CMOS integrated circuit technology.

Low-voltage swing signalling is dictated by the need to drive the resistive transmission line load with acceptable power dissipation. We see in Equation 6.5 that going to a lower voltage swing saves power quadratically. In the designs which follow, we specifically consider signalling between zero and one-volt. Limiting the voltage swings to one-volt saves a factor of 25 in power over traditional five-volt signalling (*i.e.* $P_{serial_drive} = 250$mW with five-volt signal swings and $P_{serial_drive} = 10$mW with one-volt swings).

To achieve one-volt signalling, we provide components with a one-volt power supply for the purpose of signalling. This frees the individual components from needing to convert between the logic supply voltage and the signalling voltage level. Any power consumed generating the one-volt supply is dissipated in the power supply, and not in the individual ICs.

Series termination offers several advantages over parallel termination for point-to-point signalling. We can integrate the termination impedance into the driver. In the parallel terminated case, we needed to drive the transmission line voltage close to the signalling supply rail. The effective resistance across the driver between the supply rails and the driven transmission line must be small compared to the transmission line impedance, Z_0, in order to drive the transmission line voltage close to the signalling supply (See Figure 6.9). In a CMOS implementation, this means that the size

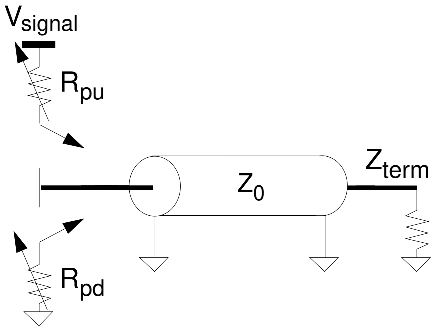


Shown here is the basic CMOS transmission line driver. Shown at right is the basic driver. Shown on the left is a simplified model of the driver making explicit the fact that each transistor, when enabled, can be modeled as a resistor of some resistance determined by the transistor's W/L ratio and process parameters.

Figure 6.9: CMOS Transmission Line Driver

of the transistors implementing the final driver must have a large W/L ratio to make the resistance small. As a consequence, the final driver is large and, therefore, has considerable associated capacitance, C_{driver}. This means it will take additional time to scale the drive capacity of an input signal up large enough to drive the final driver. It also means that the charging power, P_{charge} (Equation 6.8), will be large. In contrast, the series terminated driver can use a higher-impedance driver. The higher impedance of the series terminated driver allows the series driver to have a smaller W/L ratio and hence smaller C_{driver} and less latency driving the output.

The series terminated configuration gives us the opportunity to use feedback to adjust the on-chip, series termination to match the transmission line impedance. We expect both the transmission line impedance and the conductance of the drive transistors to vary due to process variations. By monitoring the stable line voltage during the round-trip transit time between the initial transition at the source end of the transmission line and the arrival of the reflection, the component can identify whether the driver termination is high, low, or matched to the transmission line impedance. With a properly terminated series transmission line, we expect the voltage to settle half-way between ground and the signalling supply during the first round-trip transit time. If the voltage settles much above the half way point, the drive impedance is too low. If the voltage settles much below the half-way point, the driver impedance is too high. By monitoring the voltage on the line and adjusting the drive impedance appropriately, the integrated circuit can compensate for process variation in both the silicon processing and PCB manufacture to match the termination impedance to the line impedance. Most CMOS circuit designers are familiar with the practice of designing circuitry to compensate for the wide variations associated with silicon processing. This adjustable termination technique takes the strategy one step further to compensate for variations in the component's external environment.



Functionally, we want an adjustable resistance for the path to both the high and low signalling rails. To drive the output to a particular signalling rail, we connect the appropriate rail to the output pad via the tuned impedance.

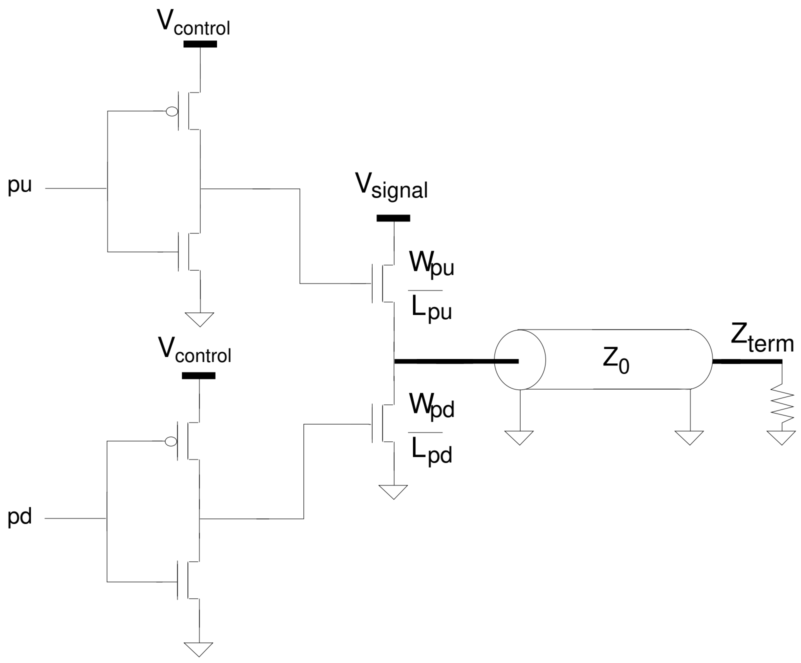
Figure 6.10: Functional View of Controlled Output Impedance Driver

6.5 Driver

To allow adjustable driver impedance, the output driver is designed to connect the transmission line on the a chip's output pad to the signalling supply through a controllable impedance gate. Logically, this configuration is shown in Figure 6.10. Several options exist for controlling the variable output impedance. Knight and Krymm suggest controlling the output impedance by controlling the gate voltage on the final stage output drivers [KK88] (See Figure 6.11). Branson suggests using exponentially sized pass gates between the signalling supplies and the output pad [Bra90]. The impedance is controlled by only allowing the appropriate subset of pass gates to turn on to achieve the desired impedance. Gabara and Knauer suggest a scheme which is virtually equivalent using a set of exponentially sized transistors in parallel in each of the pull-up and pull-down networks to allow digital control of the output impedance [GK92] (See Figure 6.12).

DeHon, Knight, and Simon consider a variant that places the impedance control transistors and the gating transistor in series between the signal supply and the output pad [DKS93] (See Figure 6.13). This configuration achieves lower latency by moving the impedance control logic out of the critical signal path through the output pad. Unlike the other two digitally-controlled impedance schemes, the impedance setting is controlled separately from the drive enables and held static during operation. The generation of the pull-down and pull-up enables is the only logic that must be performed in the signal path to the final stage of the pad driver. Since the enables to the impedance control devices do not change with each data cycle, less power is consumed charging the final stage drivers, P_{charge}.

In all of these driver schemes, when the high signalling supply is a device threshold drop or

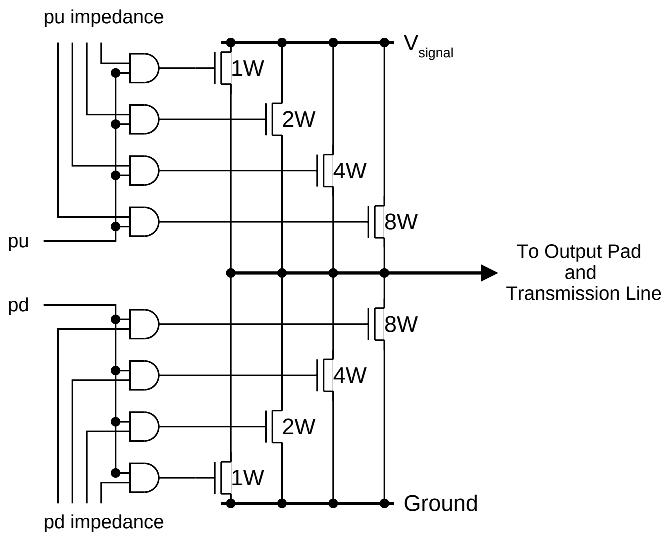


Shown above is a voltage-controlled controlled-impedance CMOS driver from [KK88]. By varying $V_{control}$ below the logic supply voltage, one varies the gate voltage applied to the final driver when it is enabled. Modulating the gate voltage in this manner controls the total conductance of the final driver and hence the impedance seen by the transmission line.

Figure 6.11: CMOS Driver with Voltage Controlled Output Impedance

more below the high logic supply, N-devices can be used to form the pull-up network, as well as the pull-down network. That is, when the internal logic can drive the control voltage onto the final driver more than a threshold above the desired high signalling supply, it becomes unnecessary to use a P-device pull-up to allow the output to swing all the way up to the high signalling rail. NMOS devices have size, speed, and power advantages. Since the mobility of electrons is about two and a half times the mobility of holes, an N-device with a given transconductance, and hence impedance, can be roughly two and a half times smaller than a corresponding P-device with the same transconductance. The smaller devices present less capacitance which must be driven by the internal logic and hence operate faster while dissipating less power when switching. The output driver layout becomes smaller and simpler since the final driver is built entirely out of N-devices. Output drivers that rely on both P-devices and N-devices require guard-rings between the PMOS and NMOS devices to protect against latch-up.

Figure 6.14 shows a sized version of the output driver shown in Figure 6.13 which was implemented in CMOS26, Hewlett Packard's 0.8μ effective gate-length process [DKS93]. This output driver exhibits a 2 ns output latency and 1 ns rise/fall times. The fabricated driver was capable of matching external impedances between 30Ω and 100Ω. In total, the driver consumes



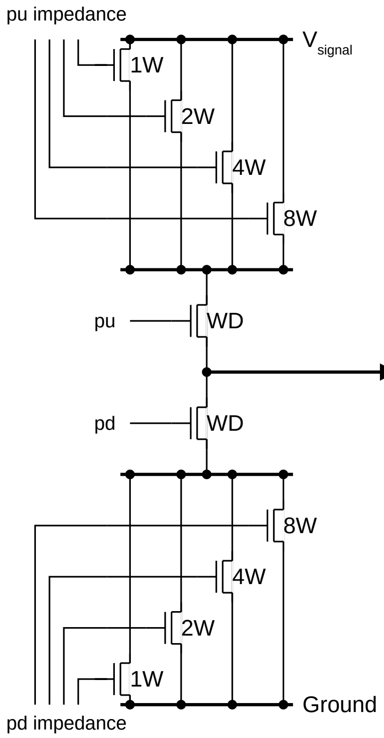
Shown above is a digitally controlled variable resistance driver from [GK92] ([GK92] actually uses PMOS devices in both resistance networks since it focuses on operation in a different voltage region). The digital values on $pu\_impedance$ and $pd\_impedance$ determine which transistors are enabled whenever the driver drives a high or low signal, respectively. The transconductances of the enabled parallel transistors combine to determine the transconductance between the signalling rail and the output pad.

Figure 6.12: CMOS Driver with Digitally Controlled Output Impedance

approximately 10 mW + 2mW/100MHz of power.

6.6 Receiver

The receiver must convert the low-voltage swing input signal to a full-swing logic signal for use inside the component. In the interest of high-speed switching, we want a receiver which has high gain for small signal deviations around the mid-point between the signalling supplies. [CCS+88] and [KK88] introduce suitable differential receivers. Figure 6.15 shows one such receiver. The rightmost inverter pair (I1 and I2) in Figure 6.15 forms a differential receiver biased to trip when the input voltage seen by the input pad exceeds half the low-voltage supply. I1 and I2 are identical inverters. The inputs to each are taken through resistors to what would normally be the ground connections of the inverters. The resistor between the pad and I2 is the diffused input protection resistor which must exist on CMOS input pads. The resistor between the half signalling voltage level and I1 is an identical resistor for reference. The normal input and output of the I1 inverter structure are shorted together so that I1 serves as a bias generator placing I2 in its high-gain region of operation. If the input pad voltage connected to I2 were also at half the signalling voltage level, the two devices would be in identical voltage states and the output of the I2 inverter structure would also be mid-range. As the pad input voltage seen by I2 varies away from the half voltage level,

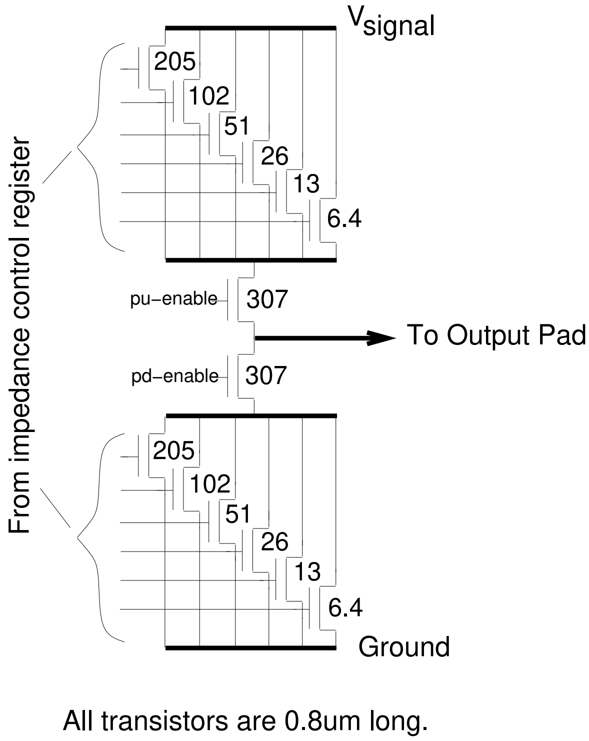


Shown above is a digital controlled-impedance driver after [DKS93]. The digital values on $pu\_impedance$ and $pd\_impedance$ enable the parallel impedance control transistors. Driver transistors are placed in series between the impedance control networks and the output pad. The desired signalling voltage is connected to the pad by enabling the appropriate drive transistor. The digital impedance controls remain static during normal operation.

Figure 6.13: CMOS Driver with Separate Impedance and Logic Controls

I1 and I2 rapidly become unbalanced leading to a high-gain output from I2. If the pad input to I2 is slightly above the half voltage level, the switching threshold of I2 becomes higher than the bias supplied by I1. The bias on the gates of I2 devices appears like a low input to I2 inverter, and I2 drives a high output. Similarly, if the pad input to I2 is slightly below the half voltage level, the switching threshold of I2 becomes lower than the I1 bias causing the I2 gate input to appear high. In response, I2 drives a low output. Finally, I3 serves to restore the recovered input signal from I2 to a full rail-to-rail voltage swing for driving internal logic. In order for I3 to rectify properly, it should be sized so that its midpoint voltage tracks the midpoint voltage of I1 and I2 with process variation.

Figure 6.16 shows a version of the receiver shown in Figure 6.15 which was implemented in CMOS26, Hewlett Packard's 0.8μ effective gate-length process [DKS93]. The input latency through this receiver is approximately 1 ns. The total i/o latency, t_{io}, for this receiver and the driver described in the previous section is 3 ns.

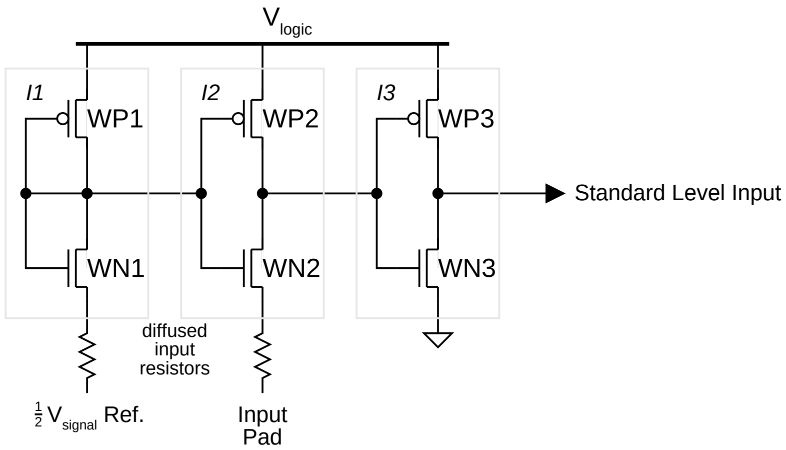


Shown here is a sized CMOS driver circuit from [DKS93]. All widths are shown in microns. This driver was designed for CMOS26, Hewlett Packard's 0.8μ effective gate-length process.

Figure 6.14: Controlled Impedance Driver Implementation

6.7 Bidirectional Operation

The drivers and receiver shown in the previous sections can be combined to allow half-duplex, bidirectional signalling, as needed for the routing components described in Chapter 4. A single i/o pad would contain both a driver and a receiver. At any point in time one side of the transmission line would be configured to drive the line and the other to receive. The receiver would have both its pull-down and pull-up enables turned off. In this mode, both the output driver and the input receiver look like high-impedance connections to the transmission line. The receiver thus behaves as the high-impedance connection we expect on the destination end of a series-terminated transmission line. The driving end of the transmission line drives either the pull-up or pull-down enable connecting a signalling supply to the transmission line through the adjustable impedance network. When it is necessary to turn the connection around to reverse the flow of data in the network, the i/o pads can swap roles as driver and receiver.



Shown above is a receiver after [KK88]. Inverters $I1$ and $I2$ are identical devices ($WP1 = WP2$, $WN1 = WN2$). $I1$ biases $I2$ into its high-gain region. When the voltage on the input pad is slightly higher or lower than the half signalling voltage reference, $I2$ amplifies the voltage. $I3$ standardizes the output of $I2$ for use inside the component. It should be sized to have the same voltage midpoint as $I1$ and $I2$.

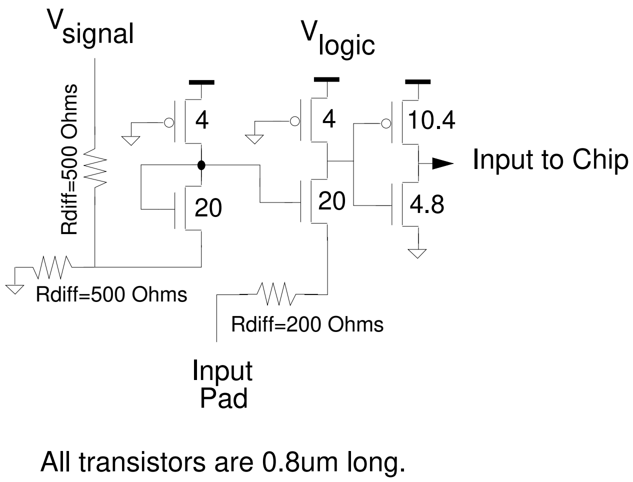
Figure 6.15: CMOS Low-voltage Differential Receiver Circuitry

6.8 Automatic Impedance Control

The drivers described in Section 6.5 all allowed the output impedance to be varied. In this section we turn our attention to the task of automatically matching the controlled impedance to the attached transmission line impedance. In any suitable scheme, we need both a sensor to indicate whether the termination impedance is high or low and a mechanism for feeding the matching information back to update the impedance setting. Starting with the drivers and receiver described in this chapter, we can obtain the information necessary and close the feedback loop by adding a discrete-time sample register and allowing access to the impedance setting and sample values through the test-access port (TAP) (Section 2.2 and Chapter 5).

6.8.1 Circuitry

Figure 6.17 shows the scan architecture for a bidirectional signal pad. In addition to the standard, boundary-scan registers, each pad has an impedance control register and a sample register. The impedance register holds the digital impedance setting for the pull-up and pull-down transistors in impedance controls schemes such as the ones shown in Figures 6.12 and 6.13. The impedance register can be written under scan control through the TAP to configure the pull-up and pull-down network impedances. The sample register is shown in Figure 6.18. When a transition occurs on the logic value to be driven out of the pad, an enable pulse is fed into the sample register. This pulse ripples through the inverter chain enabling each sample register to store the digital input value two inverter delays apart. The digital input value to the sample register comes from the pad's receiver.



Shown here is the differential receiver from [DKS93]. All widths are shown in microns. This driver was designed for CMOS26, Hewlett Packard's 0.8μ effective gate-length process. Grounded-P inverters are used in the differential receiver rather than complementary CMOS inverters as in Figure 6.15. Experimental evaluation suggests that the geometry of the transistors used for the differential receiver should be larger in order to provide good stability to processing variation.

Figure 6.16: CMOS Low-voltage, Differential Receiver Implementation

This receiver may be the same one used for receiving signals when the i/o pad is serving as an input. The key requirement is that the input receiver generates one logic value when the pad is slightly above the signalling supply mid-point voltage and the opposite logic value when the pad is slightly below. Following a transition of the output logic value, the sample register will contain a sequence of closely spaced time samples of the digital value seen by the receiver. The sample register can be read under scan control through the TAP to provide a digital, discrete-time profile of the voltage on the output pad. Figure 6.19 shows the combined i/o pad circuitry from [DKS93]. Since the TAP provides access to read the sample register and write the impedance register, we can place the task of analyzing the data recorded by the sample register and selecting the impedance control on an off-chip controller.

6.8.2 Impedance Selection Problem

The goal is to set the output impedance to achieve matched series termination. If the transmission line were ideal, the signals had no appreciable rise-time, and the round-trip delay on the transmission line was definitely longer than our sample register, the impedance selection task would be easy. The voltage at the pad of a matched transmission line would look like Figure 6.20 following a low to high transition. If the series resistance was a little lower than optimal, the transmission line voltage

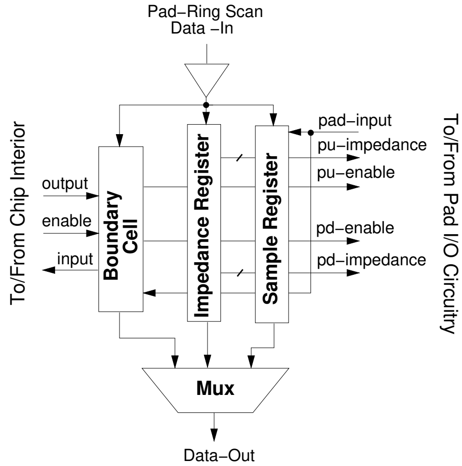
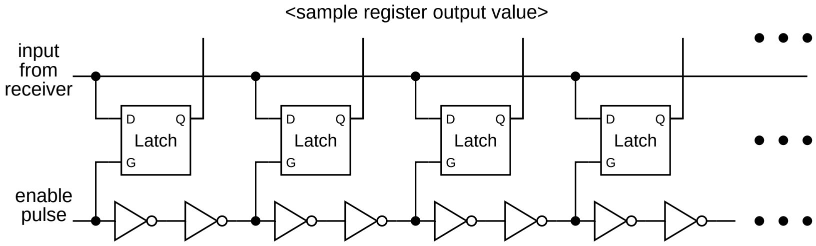


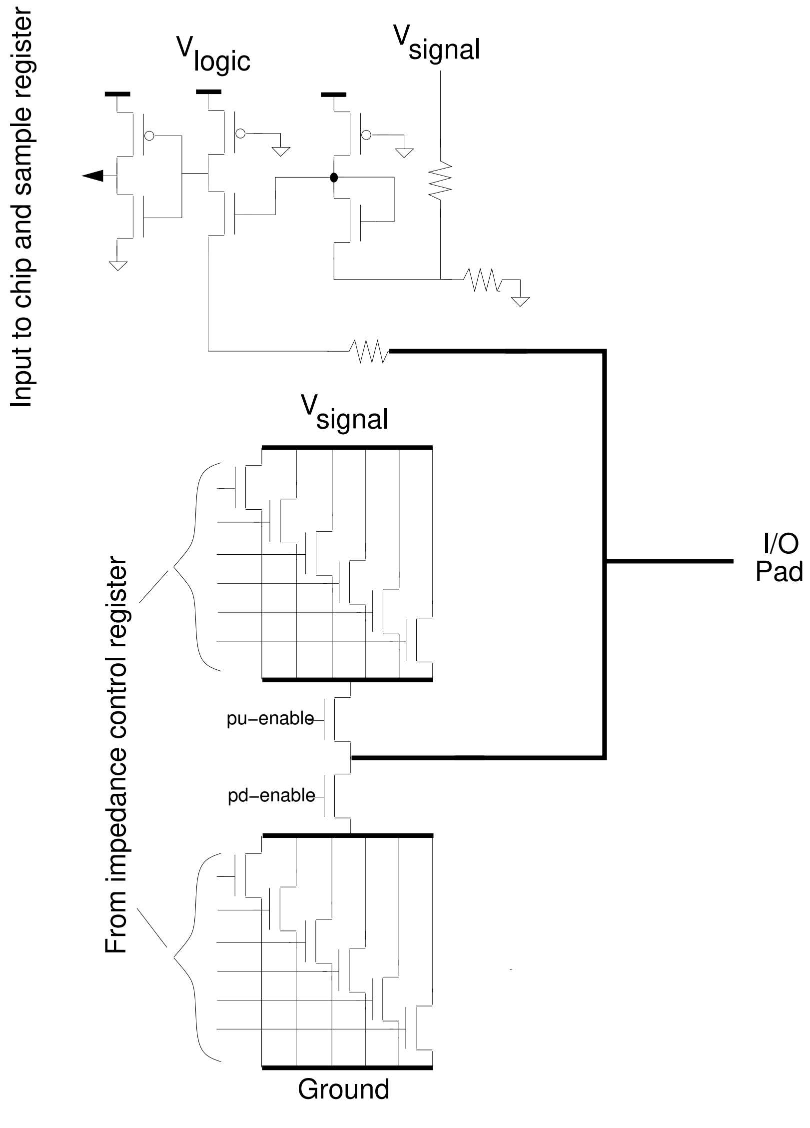
Figure 6.17: Bidirectional Pad Scan Architecture



A simple sample register is composed of a sequence of latches each enabled two inverter delays apart. When a transition occurs on the output pad, a short enable pulse is driven into the sample register. Each sample latch records the value seen by the receiver when it was last enabled. After the enable pulse propagates through the sample register, the sample register holds a discrete-time sample of the value seen by the receiver.

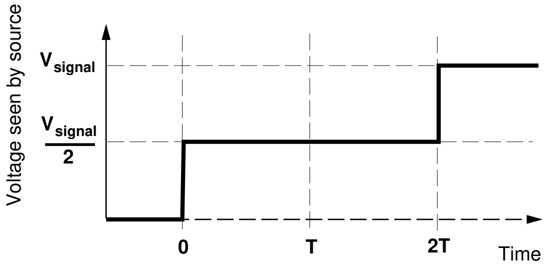
Figure 6.18: Sample Register

would settle a little above the mid-point voltage and trip the input causing the sample register to read all ones. Similarly, if the series resistance is a little higher, the line will settle a little below the trip point and the sample register would read all zeros. To set the pull-up impedance value, we scan through various impedance settings. At each setting, we configure the output impedance and then force a transition of the output using the scan capabilities. Following the transition, we recover the value of the sample register, again using the scan TAP. When we find the impedance settings where the sample register changes from reading all zeros to reading all ones, we have identified the appropriate pull-up impedance setting. The same basic operation can be applied to high to low transitions to configure the pull-down impedance.



A bidirectional pad will contain both driver and receiver circuitry. Shown here is a typical bidirectional pad configuration integrating the driver detailed in Figure 6.14 and the receiver detailed in Figure 6.16.

Figure 6.19: Driver and Receiver Configuration for Bidirectional Pad



Shown above is the voltage at the source-end of an ideal, matched series-terminated transmission line following a low to high transition from the driver.

Figure 6.20: Ideal Source Transition

Unfortunately, there are many non-ideal effects which cannot be ignored. Real signal transitions look more like the ones shown in Figure 6.21. Since there is a finite rise-time associated with the real signal, the driver will always require some time to drive the transmission line up to the input trip point. The sample register will not contain all ones or all zeros. Due to typical process variation in a CMOS process, the time between subsequent samples in the sample register, and hence the timing of a sample bit relative to the output transition may vary widely from component to component. This variation can easily cause the inter-sample time to vary by as much as a factor of two. Additionally, it takes finite time for the signal to get from the input pad to the sample register. Even if the first sample were taken when the output started changing, several sample may elapse times before the input to the sample register reflects the voltage on the output pad. Processing variation will cause this skew between the pad voltage transition and the received input to vary from component to component.

As a result, the sample values returned from an impedance scan look like those shown in Table 6.1. As the impedance decreases, the line does trip from low to high. The point in time where the low to high trip occurs becomes earlier as the source impedance is lowered. This is commensurate with our expectations of a finite rise-time. Eventually, the trip point stops moving. This is an indication of the number of sample times which elapse between the firing of the first sample register and the arrival of the fastest pad transition through the input. The exact number of bit times this requires will vary from component to component due to processing. Properly setting the control impedance requires that we take this data (*e.g.* Table 6.1) and identify the best impedance setting for proper series termination.

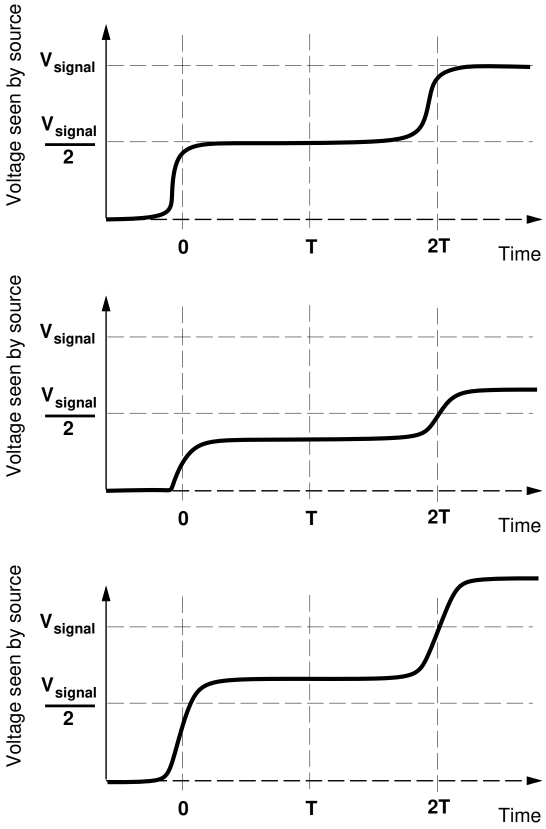
6.8.3 Impedance Selection Algorithm

A heuristic strategy which works well in practice for selecting a matched impedance is to look at the derivative of the sample profile (*e.g.* Table 6.1) and center in on the center of the largest derivative region. That is, we look at where the transition occurs in each sample and take the deltas between impedance pairs. The search focuses on finding the largest regional deltas, not just the largest deltas between an adjacent pair of impedance values. For a truly ideal case, below the trip-point, the sample register would never trip and above, it would always trip. The point where

Impedance Setting	**Sampled Data**
0	0000000000000000
1	0000000000000000
2	0000000000000000
3	0000000000000000
4	0000000000000000
5	0000000000000000
6	0000000000011111
7	0000000011111111
8	0000001111111111
9	0000001111111111
a	0000011111111111
b	0000011111111111
c	0000011111111111
d	0000011111111111
e	0000011111111111
f	0000011111111111
10	0000011111111111
11	0000011111111111
12	0000011111111111
13	0000011111111111
14	0000011111111111
15	0000011111111111
16	0000011111111111
17	0000111111111111
18	0000111111111111
19	0000111111111111
1a	0000111111111111
1b	0000111111111111
1c	0000111111111111
1d	0000111111111111
1e	0000111111111111
1f	0000111111111111

Shown above is sample data for a 16-bit sample register. The impedance setting corresponds to the binary encoding of the enables for 5 exponentially sized impedance transistors. 0 implies all the transistors are disabled, while 1f indicates that all the transistors are on (*i.e.* the lowest impedance setting).

Table 6.1: Representative Sample Register Data



Shown above are a series of more realistic depictions of the voltage waveform seen at the source end of a series terminated transmission line. At the top, we have a matched impedance situation. The middle diagram shows a case where the series terminating impedance is too large, and the bottom diagrams shows a case where the series impedance is too small.

Figure 6.21: More Realistic Source Transitions

the change occurs would clearly be the point where the largest delta occurs and be the correct impedance setting.

Naively, we could scan through looking only at adjacent impedance pairs. We could identify the pair of impedance values between which the largest difference occurred and select one of those impedance settings to configure the impedance network. However, such a purely local selection strategy can be misled. It is often the case that the difference between any two impedances is small, perhaps one or two bit positions. That makes it difficult to decide which pair of impedances is best – *e.g.* consider the case in which there is a run of five impedance settings all differing by one bit position, followed by five impedance settings all identical, then a difference of two bit positions, and no subsequent differences. A pair oriented algorithm would select the two, whereas the largest change is really in the middle of the five impedances which differ by only one bit position.

Instead, we use an algorithm which looks at successively smaller impedance intervals in an

Set Impedance:

```
old_himped ← middle impedance value
old_limped ← 0
limped ← middle impedance value
himped ← 0
while the old and current impedances differ
    old_himped ← himped
    himped ← find_high_impedance(limped)
    old_limped ← limped
    limped ← find_low_impedance(himped)
set impedance to limped, himped
```

Figure 6.22: Impedance Selection Algorithm (Outer Loop)

attempt to zero in on the largest delta. To avoid missing large gaps which may straddle a midpoint, the recursive search divides the region into pieces and recurses on the portion of the remaining space with the largest gap. Further, since the value of the opposite impedance setting does have a second-order effect on the optimal impedance setting, we iterate through searching for the high and low-impedance settings until the solutions converge. Figure 6.22 details the basic algorithm for converging on a pair of impedance values. Figure 6.23 describes the algorithm to actually zero in on an impedance value using the heuristic strategy just described.

6.8.4 Register Sizes

When adapting the impedance control strategy described here to a particular application and process, it is important to consider the amount of granularity available in the impedance control and the time window covered by the sample register. The number of different drive transistors and hence the number of bits used by the impedance control register will depend on the range of impedances to which the pad needs to match, the potential process variation, and the amount of mismatch which is considered negligible. The number of bits in the sample register will depend on the size of the window required to guarantee that the transition is recorded at all process corners. If one could determine, *a prior*, exactly when to sample the output, only a single bit sample register would be necessary. However, since processing and operating temperature variation affect the timing of the input and output circuitry, the number of bits in the sample register should be chosen such that the window spans all potential timing variations.

6.8.5 Sample Results

The test component described in [DKS93] was configured with a 16-bit sample register and six bits of both pull-up and pull-down impedance control. Figure 6.24 shows the voltage profile at both ends of a 50Ω transmission line for an impedance selection determined automatically by

Find Impedance:

$lp \leftarrow$ highest impedance value with no transition
$hp \leftarrow$ lowest impedance setting with same value
  in the sample register as the highest impedance setting
while $((hp - lp) > 2)$
  $hmid \leftarrow hp - \left(\frac{hp-lp}{3}\right)$
  $lmid \leftarrow lp + \left(\frac{hp-lp}{3}\right)$
  if transition difference between $hmid$ and lp is greater than
    that between $lmid$ and hp
      $hp \leftarrow hmid$
  else
      $lp \leftarrow lmid$
return $\left(\frac{hp+lp}{2}\right)$

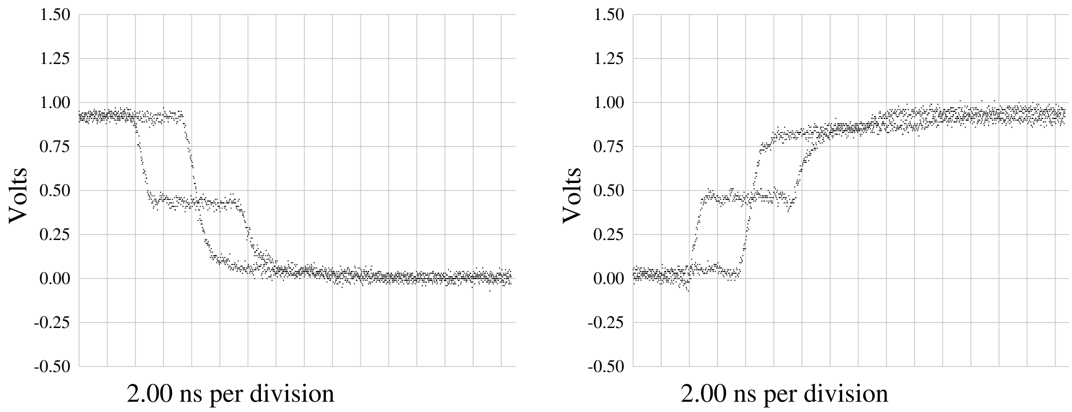
This version of the algorithm divides the remaining distance into thirds and compares the overlapping two-thirds regions. A larger fraction could be used for more overlap and hence greater selection accuracy, at the expense of slower convergence.

Figure 6.23: Impedance Selection Algorithm (Inner Loop)

the algorithm described above. At the process corner represented by the fabricated components, six bits of impedance control were more than sufficient to cleanly match a 50Ω transmission line. Figure 6.25 shows the matching achieved for this same component and automatic impedance selection algorithm when fewer control bits are used. Of course, a different point on the processing curve would provide a different impedance resolution and range. Figures 6.26 show the comparable diagrams when the same pad is automatically matched to a 100Ω transmission line.

6.8.6 Sharing

One option which may make sense in many situations is to share the sample registers, and perhaps impedance control, between several pads. If a group of pads all connect through the same physical media (*e.g.* same PCB), the external transmission lines they drive will have essentially the same characteristic impedance. If the pads are physically close on the same die, there will be little process variation from pad to pad. In such cases, it makes sense to share the sample register and impedance control within the group of pads. In cases where we cannot make the same assumptions about the external impedance, it would still be possible to share a single sample register among a group of physically local pads. Such a shared sample register only needs an additional multiplexor to select among the possible input sources.



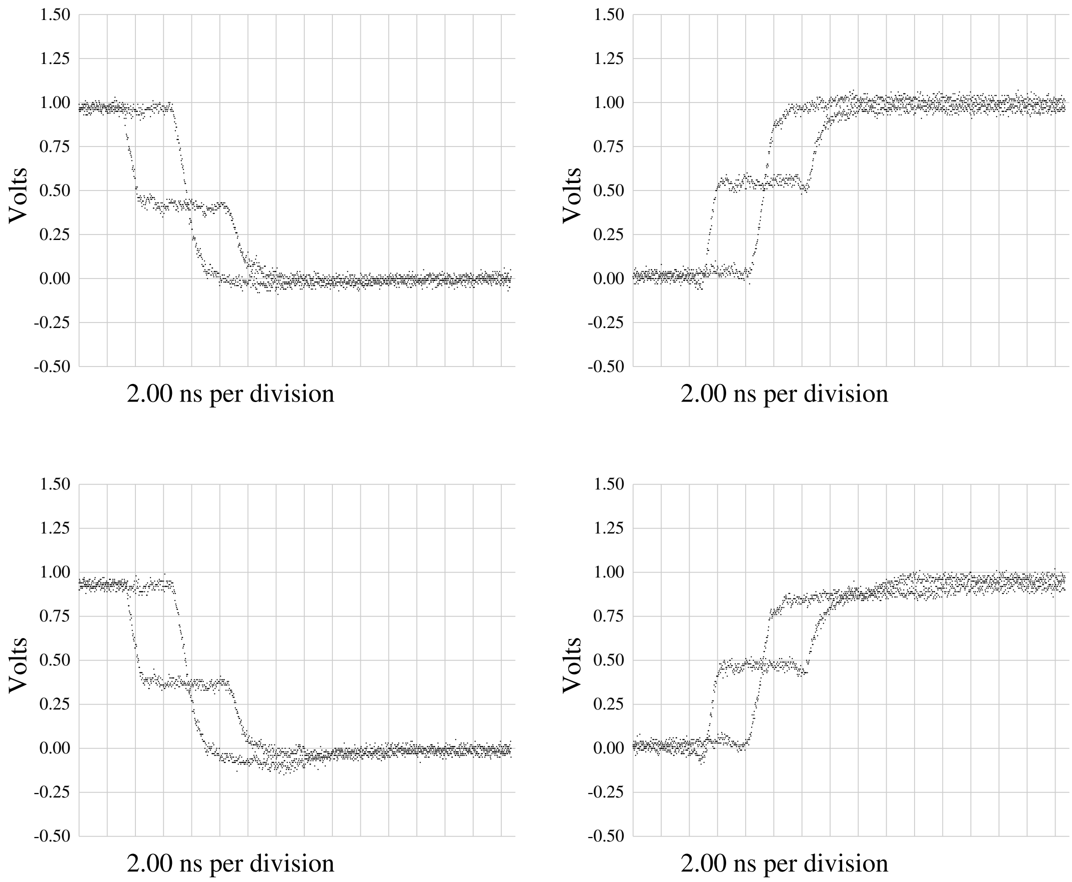
Shown above is the voltage profile seen at both the driver and receiver ends of a nominal 50Ω transmission line. The impedance matching shown here was determined automatically using the algorithm detailed in Figures 6.22 and 6.23.

Figure 6.24: Impedance Matching: 6 Control Bits

6.8.7 Temperature Variation

Temperature is an important environmental factor which affects the behavior of an integrated circuit. Temperature affects the transconductance of devices in a CMOS integrated circuit and hence the termination impedance. The automatic impedance matching described in this section allows us to adjust the termination impedance to be matched at the temperature of operation.

The process described above would normally be performed as part of the configuration sequence for each component. In some environments, it is possible for the component temperature to vary widely during operation. As the temperature varies from the point where the last calibration took place, the termination impedance will deviate from the transmission line's impedance. If this effect is significant enough to affect transmission reliability, the routing protocol will notice a higher than normal rate of corrupted messages through the component. When the scan controller system attempts to localize errors (See Chapters 5), it can rerun the matching algorithm to re-adjust the impedance for the current operating temperature. A more proactive approach can be taken by integrating a temperature sensor onto the integrated circuit. With a scan-accessible, on-chip temperature sensor, the scan controller could periodically check the temperature of each component. When a component's temperature indication differs significantly from the temperature indication when the component was last calibrated, the scan controller can recalibrate the impedance setting. Using the port-deselection and port-by-port scan facilities introduced in Chapter 5, the scan controller can isolate individual port pairs and recalibrate their drivers during operation without having a significant performance impact.

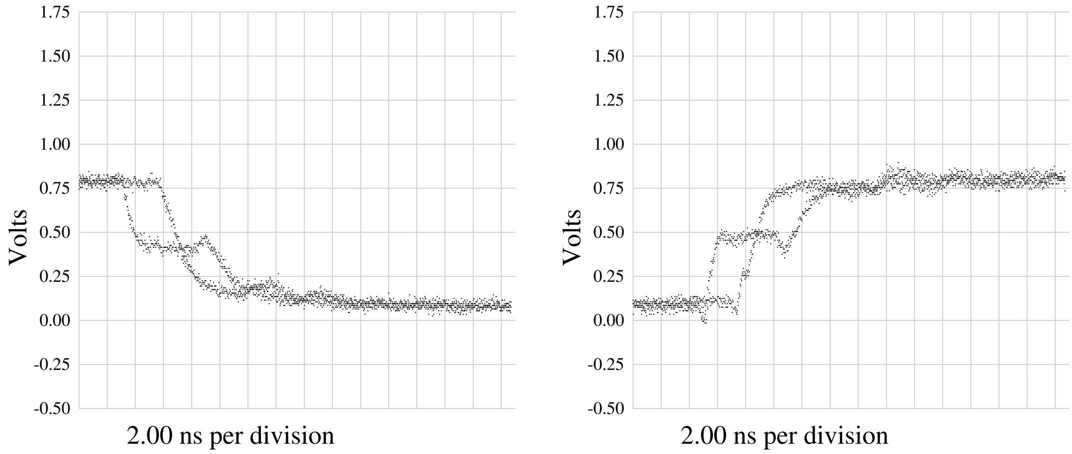


Shown above are the voltage profiles seen close to the driver and receiver ends of a nominal 50Ω transmission line following both high and low output transitions. The impedance was selected automatically. Since only 3 bits of control were used, the highest unused bit was used to simulate parameter variation – in the top pair of traces the bit was turned off, but in the bottom it was enabled.

Figure 6.25: Impedance Matching: 3 Control Bits

6.9 Matched Delay

In Section 3.2 we pointed out that pipelining bit transmissions over long wires is important to prevent the transit times across wires in the system from having a negative impact on communication bandwidth and latency. With the circuitry developed in the previous sections, we can now consider how to reliably pipeline multiple bits on the wires between routing components.



Shown above is the voltage profile seen at both the driver and receiver ends of a nominal 100Ω transmission line. The impedance was matched automatically. The reduced high-voltage level is the result of the finite impedance of the measurment apparatus.

Figure 6.26: 100Ω Impedance Matching: 6 Control Bits

6.9.1 Problem

The wires interconnecting components in the network vary in length. Due to processing and temperature variations, the exact delay through an input or output pad is variable and unknown. With arbitrary length wires and uncertain i/o delays, there is no guarantee about when a signal arrives at the destination component relative to the system clock. If the signal arrives during the setup time just before the clock rises or during the hold time just after the clock has risen, the receiver can clock in indeterminate data. To avoid this potential problem, we seek to adjust the delay through each output pad to guarantee that the signal transition arrives at the destination at a reasonable time with respect to the clock.

6.9.2 Adjustable Delay Pads

To control the arrival time of signals at the destination, a variable delay buffer is inserted between the internal logic and the final output driver. This buffer is designed to have sufficient delay variation such that it can always move the arrival time of the signal out of the danger zone regardless of processing and temperature. For the granularity of control necessary for this application, the variable delay buffer could simply be a multiplexor providing access to a sequence of taps off of a chain of inverters (See Figure 6.27). For finer control, of course, a voltage controlled delay could be used instead of, or in addition to, a variable delay multiplexor (See Figure 6.28). Figure 6.29

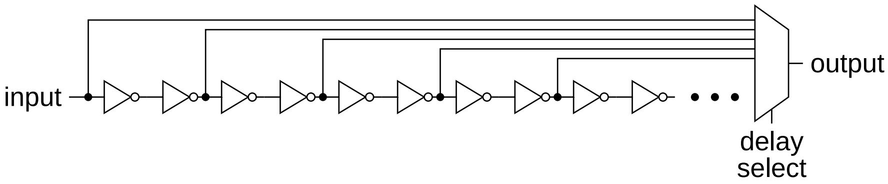
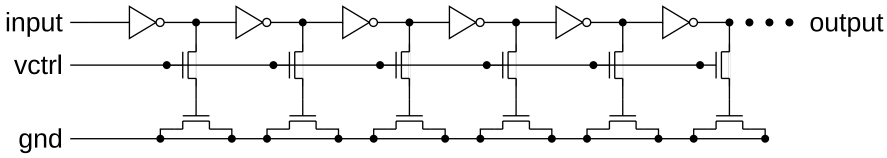


Figure 6.27: Multiplexor Based Variable Delay Buffer



Shown here is a voltage controlled delay line (VCDL) after [Baz85] and [Joh88]. Varying VCTRL effectively controls the amount of capacitive load seen by the output of each inverter stage and hence the delay through each inverter stage. The number of stages one uses in the VCDL will depend on the range of delays required from the buffer.

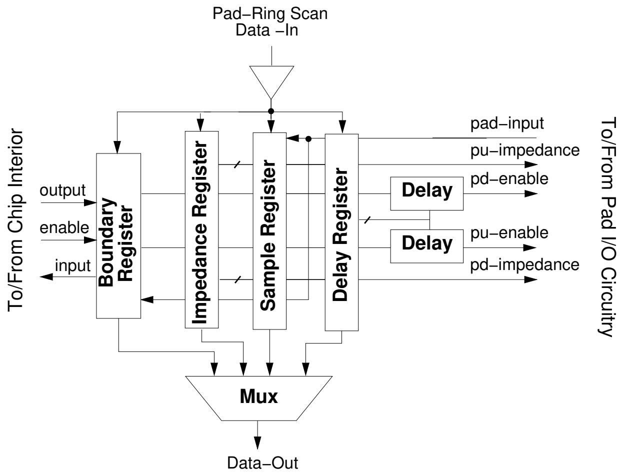
Figure 6.28: Voltage Controlled Variable Delay Buffer

shows a revised pad architecture which incorporates the variable delay and a control register for configuring the delay through the component's TAP. The pad driver and receiver circuitry remain the same as in the matched impedance pads described above.

6.9.3 Delay Adjustment

We can use the same basic strategy used for matching impedance to match the delay. That is, by watching the voltage level at the source end of the transmission line, we can determine the round-trip transit time across the transmission line. Since we can assume that the signal takes the same time to propagate from the source to the destination as it does to propagate back from the destination to the source, we know that the signal arrived at the destination in the center of the round-trip transit time. All we need to do is determine when the source transitions from the half-way point to the full signal voltage rail as well as when it transitioned to the midpoint.

The information which we record in the sample register when scanning through the possible impedance values, in effect, already provides us with this information up to the length of the sample register. The input receiver is set to trip whenever the voltage on the source end of the transmission line exceeds the half-way point between the signalling rail. In normal operation, we select the driver impedance to match the transmission line so that the source end settles right around this half-way point during the round trip-time. If, for example, we were to set the driver impedance at



Shown above is the revised bidirectional pad architecture incorporating variable delay buffers in the output path as well as a scannable register to control the delay buffers.

Figure 6.29: Adjustable Delay Bidirectional Pad Scan Architecture

three times the characteristic impedance of the transmission line, the voltage level would cross the midpoint and trip the receiver not when the line was driven, but when the reflection returned from the far end of the transmission line. That is:

$$\begin{aligned} V_I &= (\frac{Z_0}{4Z_0})V_{signal} \\ V_{R_{dst}} &= V_I \\ V_{R_{src}} &= \left(\frac{1}{2}\right) V_{R_{dst}} \end{aligned}$$

For a transition at $t = 0$, the pad voltage becomes

$$V = \frac{V_{signal}}{4}$$

for the period $0 < t < \frac{2L}{v}$. After the reflection returns and reflects against the unmatched source termination,

$$V = V_{R_{src}} + V_{R_{dst}} + V_I = \left(\frac{5}{8}\right) V_{signal}$$

for the period $\frac{2L}{v} < t < \frac{4L}{v}$. Qualitatively, the situation resembles the case where the series termination is too large as shown in Figure 6.21. In fact, it is not necessary for the drive impedance to be $3Z_0$, as assumed, for this behavior to hold. As long as the driver impedance is in the range

$$Z_0 < Z_{drive} < 4Z_0$$

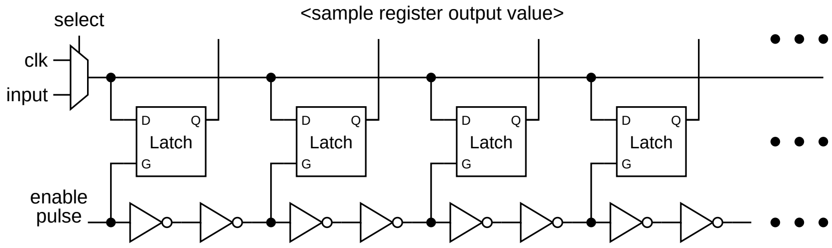


Figure 6.30: Sample Register with Selectable Clock Input

the sample register will trip after the arrival of the return reflection.[1] As long as the sample register is sufficiently long, we can determine not only the impedance setting, but also "when" the outgoing and reflected waves occur.

Notice that the delay through the input receiver is the same when searching for the transition to the midpoint as when searching for the midpoint to full-rail transition. Thus, the delay through the receiver is canceled out when looking at the delta times to determine when the signal arrived at the destination. Also note that this is a discretized time sample limited to a two inverter-delay granularity.

We still have the problems that the delay between sample bits is process dependent and timing of the samples relative to actual signal transitions is uncertain. These problems can be addressed by allowing a version of the clock to be switched into the sample register instead of the value from the input receiver (See Figure 6.30). In this way, the inter-sample bit times can be calibrated in terms of fractions of the component's clock period. If both the logic transition for the output pad and the enable pulse on the sample registers are synchronized to the clock, we can determine the alignment of a signal transition at the far end of the transmission line to within the delay variation through the input receiver. Adding a delay margin for variation in the receiver circuitry and the margin necessary for clean signal reception, the variable delay can be adjusted such that transitions always arrive at the far end of the transmission line at a well defined time relative to the clock. One other piece of information which we get from this configuration is the number of clock cycles it requires for a bit to travel across a piece of interconnect. This information can be used to configure the routing component to account for the pipeline delays associated with transmitting bits across the associated interconnect (See Section 4.11.3).

6.9.4 Simulating Long Sample Registers

The discussion in the previous section assumed the sample register was sufficiently long to actually record samples until sometime after the reflection returned. For CMOS26, Hewlett Packard's 0.8μ process, inverter delays run about 100 ps to 200 ps. Thus each sample bit is enabled roughly 200 ps to 400 ps apart. Each nanosecond of wire, or about 15 cm of wire, would require 4 to 5 sample bits in the sample register. Actually building a long sample register in this manner to match

[1] Actually, in an ideal setting the impedance can be as high as $(2+\sqrt{5})Z_0 \approx 4.24Z_0$.

the range of wire lengths of interest would be impractical. However, we can simulate a long sample register by delaying the sample pulse into a short sample register by a controlled amount. That is, if we can delay the pulse into the sample register by multiples of the sample register length, we can slide the sample register forward in the time sequence. By performing the experiment repeatedly with varying offsets for the sample register pulse and recording the sample register value after each transition, we can virtually reconstruct the waveform which a long sample register would see.

Figure 6.31 shows a simple sample register architecture for simulating long sample registers in this manner. The enable pulse ripples through the inverter chain as before. However, when the pulse reaches the end of the inverter chain it is optionally recycled through the chain. After the pulse finishes cycling through the inverters the configured number of times, the sample register will contain the values recorded during the last cycle. Care, of course, must be taken in timing the recycle path and in reconstructing the waveform. If the recycle path is not carefully tuned, the delay between the last sample bit in one cycle and the first sample bit in the subsequent cycle may not be identical to the inter-sample bit delay for bits enabled during the same cycle. If the variation is small, it may only make the sample granularity slightly coarser. Figure 6.32 shows a variation that recycles the sample bit before completing a cycle. As a result of the overlap between samples, all transition can be pin-pointed to inter-sample bit times. The waveform can be reconstructed from the overlapping samples to more accurately mimic a single long sample register.

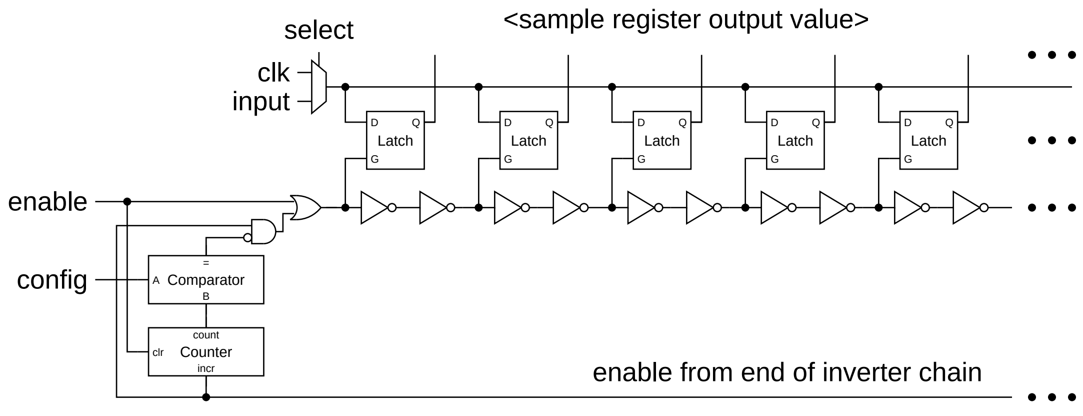
The maximum operating frequency of the counter and comparator will set a limit on how small of a sample register we can use in this scheme. The sample register must be chosen sufficiently long to allow the comparator and counter logic to settle and be prepared for the next enable pulse. If we assume a delay of at least 100 ps through each inverter and we assume a 200 MHz counter, we need a sample register which is at least 25 inverter-pairs long for proper operation.

6.10 Summary

In this chapter we addressed the issue of high-speed signalling between components. We identified the problem of transmitting bits between routing components as a point-to-point transmission line signalling problem. We saw that a low-voltage swing, series-terminated transmission line signalling scheme provided the low-latency signalling we desired while keeping power consumption low. In order to address the issue of termination matching, we introduced feedback at the source end of the series terminated transmission line. This allowed us to match the series termination impedance to the characteristic line impedance, compensating for variations in processing and environment. Finally, we showed that the basic matching mechanisms could be extended to provide the delay alignment necessary to reliably pipeline data across wires of arbitrary length.

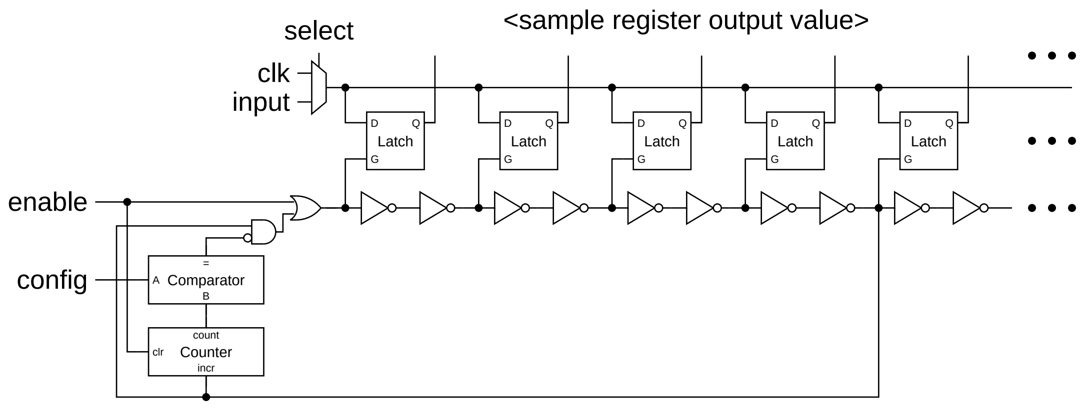
6.11 Areas to Explore

In this chapter, we have detailed a matching scheme that uses digital time samples to identify a matched impedance. It would also be possible to use multiple, biased receivers to detect whether the impedance selection was high or low. Such a scheme may require less post-processing and be more amenable to automatic, on-chip calibration.



To simulate a larger sample register, we allow the enable pulse to be recycled through the sample register inverter chain. Varying the number of cycles which the enable pulse makes through the inverter chain, allows us to move the window of time recorded by the sample register. We can combine the samples recorded at each recycle configuration to reconstruct the waveform seen by the input pad over a large period of time.

Figure 6.31: Sample Register with Recycle Option



To gain higher accuracy when reconstructing a waveform from many sample register windows, we can recycle the enable before the end of sample register so that the sample register windows will overlap each other.

Figure 6.32: Sample Register with Overlapped Recycle

The extensions necessary to allow delay matching have not, as yet, been implemented and tested. No doubt, we stand to learn more about this problem from such an implementation.

7. Packaging Technology

When we actually build any system, we must physically package its primitive components. We must provide a physical medium for the wires interconnecting components, and we must provide a mechanical support substrate to organize and house the components. The technology used in packaging a network will directly affect the size of the packaged network and hence the interconnection distances and transit latency between components. In this chapter we review present packaging technologies and develop hierarchical schemes for packaging the networks addressed in Chapter 3. The packaging scheme developed here makes use of all three spatial dimensions to minimize interconnection distances.

7.1 Packaging Requirements

When packaging a network we have many, often conflicting, goals. We want to:

- Minimize the interconnect distances (and hence T_t)
- Provide controlled-impedance signal paths (Chapter 6)
- Supply adequate power to all components
- Facilitate synchronous clock distribution to all components
- Cool components by removing the heat generated by ICs during operation
- Facilitate physical repair
- Minimize packaging cost

To keep the interconnect distances short, we seek dense packaging strategies that place components as close as possible. Excessive density, however, makes supplying power, removing heat, and physically repairing faults difficult. High-performance packaging and interconnect can often be the most expensive part of a system to manufacture and assemble. While devising a packaging strategy, we must keep in mind the economics of the available technologies.

7.2 Packing and Interconnect Technology Review

7.2.1 Integrated Circuit Packaging

Conventional packaging technology starts with packaged silicon integrated circuits as the base level building block. Silicon ICs are diced from the fabrication wafer and placed in IC packages. Fine pitch wires are bonded from pads around the periphery of the diced integrated circuit to bonding shelves along the perimeter of the die cavity in the IC package. The package seals the component and bonding wires from the environment. Dicing and packaging allows the manufacturer to sort components by functionality and speed. The packaged IC can be more easily handled for testing and assembly than the bare die. Packaged ICs can be replaced as defects are discovered.

Today, most IC packages are plastic encapsulants, ceramic, or fine-line printed-circuit boards. Plastic packages are inexpensive, but only allow connections around their perimeter and have limited capacity for heat removal. Ceramic packages are much more expensive, but provide hermetic sealing for the die cavity and allow greater heat transfer from the die to the outside package. High power IC packages provide paths of high thermal conductivity to the exterior of the package where a heat sink may be mounted to disperse the heat removed from the die. Printed-circuit board IC packages utilize the same technologies in use in PCB production and can leverage experience, tools, and manufacturing developed for fine-line PCBs. Ceramic and printed-circuit IC packages can support i/o connections on most of the package surface. This gives rise to the popular pin-grid array (PGA) arrangement.

Whether the pins are arranged around the periphery of the IC for a plastic package or arranged in a gridded fashion, the package size is generally determined by the quantity of i/o pins and the size and achievable density of external i/o connections. As a result, a packaged IC is much larger than the housed die. The size of an IC package may only be correlated to the size of the housed die because both the package size and the die size are often directly determined by the number of i/o pins on the component.

7.2.2 Printed-Circuit Boards

Packaged ICs are assembled on printed-circuit boards (PCBs). These boards provide mechanical support for the ICs and provide the first level of interconnection among packaged components. Conventional printed-circuit board technology allows the manufacture of multilayer PCBs. Multiple layers of etched copper provide interconnect in two-dimensional planes. The etched copper layers are separated by layers of insulating materials. Drilled and plated holes allow vertical interconnect among the two-dimensional etched copper layers within a single multilayer PCB. Packaged ICs may be located on one or both sides of a composite PCB.

Some packaged ICs have pins which can be inserted in mating holes in the PCB, sometimes via a socket. During assembly, solder is used to connect the component or socket pins to the PCB. Component packages with pins require holes drilled all the way through the PCB and are hence termed *through-hole* components. Another common form of packaged ICs have peripheral pins which can be soldered to exposed metal lands on a surface of the PCB. Packages such as these, which sit on the surface of the PCB and solder to PCB lands without protruding into the PCB, are called *surface-mount* components. Surface-mount components only consume space on the surface of the PCB connected to the IC, whereas through-hole components consume space on all layers of the PCB include the opposing PCB surface.

A common discipline when designing printed-circuit boards for digital electronic systems is to dedicate a power plane for each power level required in the system (*e.g.* Ground, V_{logic}, V_{signal}). This discipline allows power distribution with minimal resistive losses between the power supplies and the components. Dedicated power planes also provide low inductance paths between the power supplies and the packaged ICs' power leads. Solid conductor planes also serve to eliminate or reduce the cross-talk between signal traces on the same PCB. To guarantee consistent controlled-impedance interconnect, signal traces are generally run over a conductor plane or between a pair of solid conducting planes (See [RWD84]).

As a practical matter, there is a limit to the system size we can place on a single printed-circuit board. Despite the fact that PCBs are generally composed of multiple conductor layers, PCB technology is essentially two-dimensional. The size of a single printed-circuit board is limited by mechanical stability, yield, and manufacturing constraints. Today, 24 to 30 inches is the maximum viable side length for PCB technology. In fact, for manufacturing reasons, most PCB manufacturers limit one PCB side dimension to less than 14 inches. For fine-line PCB technologies, yield considerations will generally provide more severe limits on the size of any PCB.

Today, PCB features down to 8 mils (1000 mils = 1 inch) are considered standard. Many manufacturers can produce features down to 3 or 4 mils, but the overall board size is limited. PCB manufacturing costs are roughly proportional to the number of layers and the area of the PCB. Below the feature sizes used in volume production, the cost increases with decreasing feature size. When dealing with multilayer PCB technologies, cost is also dependent on the variety of inter-layer holes required.

7.2.3 Multiple PCB Systems

When a system design exceeds the size which can be efficiently placed on a single printed-circuit board, the system must be built from multiple PCBs interconnected via connectors and cables. Boards interconnected via a backplane PCB is, by far, the dominant paradigm for multiboard interconnect, today. In this case, one PCB is used to interconnect many other boards. Connectors on the "backplane" board allow other boards to mate orthogonally to the backplane board. This produces a structure which takes some advantage of three-dimensional space.

When a system exceeds the size practical to build in backplane fashion, or when location, physical, or mechanical requirements limit backplane use, portions of the system can be connected via cabling. Again, connectors on each printed-circuit board provide an interface to the interconnect. Rather than directly attaching two or more boards, a cable of insulated wires is used to electrically connect the boards.

Cables for controlled-impedance interconnects come in three primary forms:

1. Ribbon cables
2. Coaxial cables
3. Flexible printed-circuit cables

Ribbon cables are composed of a sequence of conductors each separated by an insulating material. Flat-ribbon cables can be used in a manner which generally provide acceptable controlled-impedance interconnect. Coaxial cables place a conductor inside a cylindrical ground. Coaxial cables have more stable impedance characteristics, but are often bulkier and more expensive than the alternatives. Flexible printed-circuits use the well established PCB technologies on flexible laminates. Typically, flexible printed-circuit cables achieve controlled impedance using the signal over ground plane topology familiar for PCBs.

7.2.4 Connectors

To date, most board-to-board, board-to-cable, and board-to-packaged-IC connections are made using pin-and-socket connectors. One board, cable, or IC has a connector which houses one or more rows of pins. The mating unit has a connector which houses a set of sockets in a complementary geometry. The two pieces are connected by mating the pin and socket connections.

More recently, a number of compression connectors have become available. One compression connectors can mate directly with lands on two PCBs or packages. When compressed, the connector provides electrical interconnect between the lands. Compressional connectors have several characteristics which make them preferable to pin-and-socket connectors.

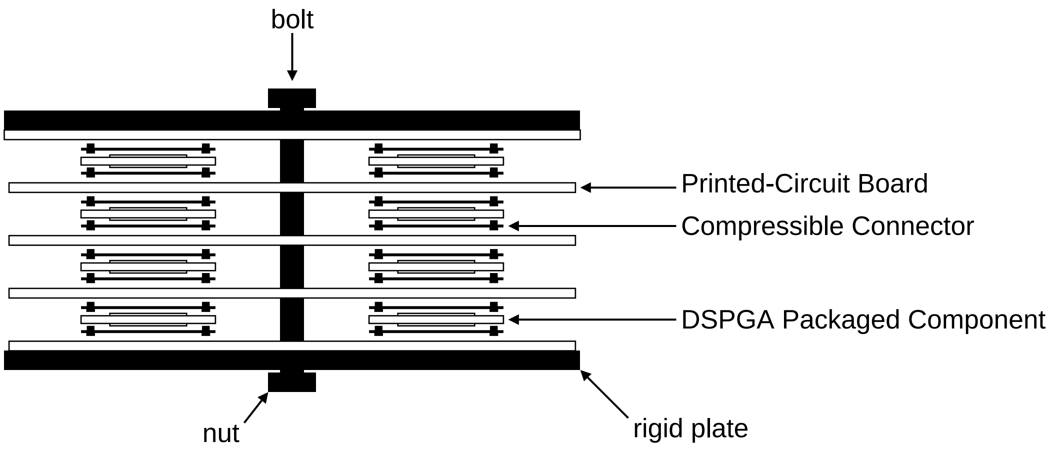
1. Higher density
2. Superior signal integrity
3. Lower insertion force
4. Function without solder

Pin-and-socket connectors are limited by the achievable density for drilled-holes and pins, whereas compressional connectors are limited by the area required to carry sufficient current and the spacing of separate conductors. Removing the need for solder makes assembly and repair easier. The insertion force required for inserting pin-and-socket connectors is proportional to the number of pins on the component. As the number of i/o pins increase, so does the insertion force. Today's 200+ pin PGAs are already experiencing excessive insertion force – forcing connector manufacturers to move to more complicated socketting schemes which mate pins at different heights to smooth out the required insertion force. Traditional pin-and-socket connectors do not provide well defined controlled-impedance paths, while many of the emerging compressional connectors offer cleaner signal paths.

Several technologies currently available for compressional interconnect are:

- Anisotropic conductive elastomer
- "Button balls"
- Springs

Several manufacturers now produce strips or sheets of elastomer with embedded conductors. The conductors are arranged to conduct only along one axis. In this way, they provide interconnect between conductors placed on opposite sides of the connector and lined up along the conduction axis. The elastomer will compress under pressure allowing the conductors to make positive electrical contact (*e.g.* [Inc90] [Pol90] [Tec88] [ND90]). "Button balls" are composed of 25μ spun gold wire compressed into small diameter cylindrical holes (*e.g.* 20 mil diameter by 40 mil high) in a plastic carrier. They provide multiple points of contact between the PCB lands on each side of the ball when compressed (*e.g.* [Div89] [Smo85]). Spring style connectors house shaped pieces of metal which behave as springs in a flexible carrier. When compressed between boards, the spring nature of the metal forces positive contact with the PCBs on both sides of the connector (*e.g.* [GPM92] [Cor90]).



Shown here is a cross-sectional view of a packaging stack. Components and PCBs are sandwiched in alternating layers to form a three-dimensional stack structure of components. This stack structure serves as the next level of the packaging hierarchy above PCBs and forms the basic building block out of which larger systems can be built.

Figure 7.1: Stack Structure for Three-dimensional Packaging

7.3 Stack Packaging Strategy

Leveraging mostly conventional packaging technologies, we can package networks that densely utilize all three spatial dimension. We continue to use fairly conventional IC packaging and PCB technology but stack components and PCBs in the dimension orthogonal to the PCB planes to form a *stack* structure sandwiching layers of packaged ICs between PCB layers (See Figure 7.1). Compressional board-to-package connectors provide signal continuity between vertically stacked printed-circuit boards and integrated-circuit components. This stack structure forms a densely packed three-dimensional cube of components and interconnect which can serve as the basic building block for even larger networks and systems.

7.3.1 Dual-Sided Pad-Grid Arrays

While there is novelty in our design and use of the IC package, the basic packaging technology we employ is conventional. The integrated-circuit is housed in a package with a gridded array of contacts. Rather than being pins, the contacts are land grids similar to the PCB lands used for attaching surface-mount components. These land grids are connected through compressional connectors to similar land grids on the PCBs. Due to the low-insertion force and high-density available from these land-grid arrays (LGAs), they have recently become an attractive option for packaging high pin-count ICs (*e.g.* [Buc89]). Intel, for example, has adopted the LGA package for its 80386SL microprocessor [Mal91].

We make one final addition to the LGA structure. Rather than placing the land grid only on the bottom side of the package, we place the land-grid array on both sides of the IC package and, optionally, provide continuity through the package between the top and bottom pads. We call the resulting structure a *dual-sided pad-grid array* (DSPGA) to emphasize the fact that pads are placed on both sides of the package. Each vertical pad pair can be configured in one of three ways:

1. The corresponding pads on the top and the bottom of the package can be connected straight through without connecting to an IC pin to support vertical interconnect.

2. The corresponding pads can be connected together and to an IC pin allowing the IC pin to make contact to traces on either or both of the boards above and below the package.

3. The corresponding pads can be connected to different IC pins to support higher pin density.

Not connecting the corresponding top and bottom pads as in (3) requires more complicated manufacture and will make the package more expensive than if only configurations (1) and (2) are used.

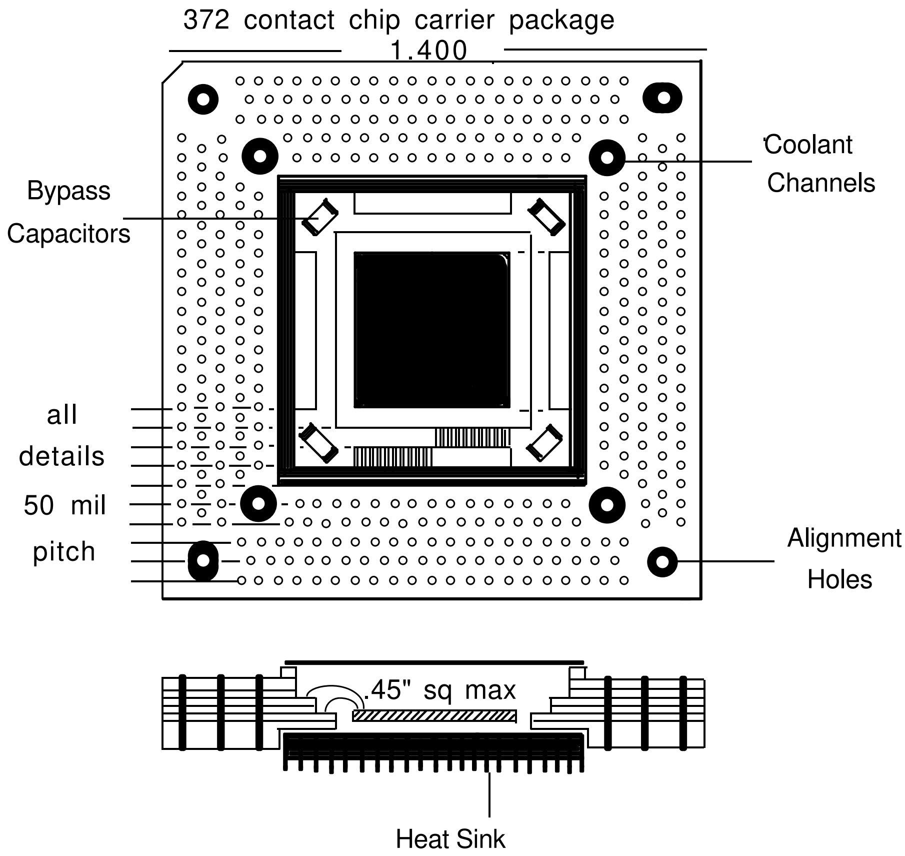
Figure 7.2 shows DSPGA372, a 372 pad DSPGA we have developed. DSPGA372 supports 160 IC signal connections, 76 through signals which do not connect to the IC, and three power supplies supported by 72, 40, and 24 pads, respectively. All lands are 30 mil diameter pads centered around 10 mil plated holes. Contacts are gold plated for high nobility contact. DSPGA372 has three internal power planes for providing a low-resistance and low-inductance path between the IC and the external power supplies. The nominal ground plane is supported by 72 pads. The other two planes supply the logic power supply, V_{logic}, and the signalling power supply, V_{signal}. Additionally, space is provided in the package for surface-mount bypass capacitors across the power supplies. The 76 through pins allow the package to supply through interconnect between adjacent PCBs for signals which do not connect to the IC. The remaining 160 pads support IC signal connections. Each of these 160 signals is available on both the top and the bottom of the DSPGA372 package. Table 7.1 summarizes the physical dimensions of our DSPGA372 package. Figure 7.3 shows pictures of the package. DSPGA372 was fabricated by Ibiden using BT (Bismaleimide Triazine) as a seven-layer printed-circuit board.

The DSPGA372 package has dedicated cooling and alignment holes. The outer set of holes can be used to align the package to the compressional connector and adjacent printed-circuit boards. The inner holes open into the heat sink cavity underneath the die allowing air or liquid coolant to flow across the heat-sink for heat removal.

7.3.2 Compressional Board-to-Package Connectors

Packaged ICs are mated with adjacent PCBs, both above and below, through compressional connectors. These connectors provide through contact between the DSPGA package and the PCB. Using self-aligning compressional connectors, no solder is needed to make reliable contacts. Properly selected compressional connectors will provide consistent, controlled-impedance contact as required for high-speed signalling.

Figure 7.4 shows a picture of BB372, a compressional, button-board connector designed to mate with DSPGA372. BB372 houses 372 buttons aligned with the land grids on DSPGA372. The buttons used by BB372 are housed in 20×40 mil cylinders. Figure 7.5 shows a closeup



DSPGA372 is a 372 pad dual-sided pad-grid array. All pads are connected straight through between the top and bottom of the package. 76 pads do not connect to the internal IC and exist simply to provide through interconnect. (Artwork by Fred Drenckhahn)

Figure 7.2: DSPGA372

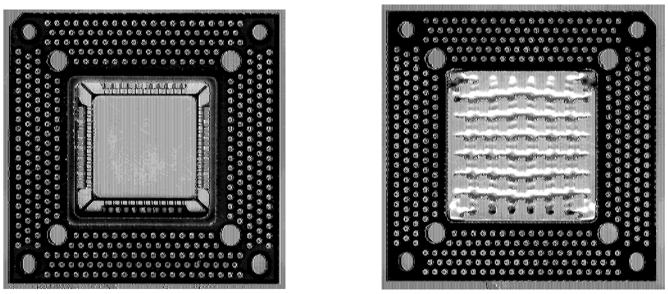
picture of a button in the BB372 connector. The buttons provide low-resistance, controlled-impedance interconnect. The center of the BB372 connector is open to accommodate the heat-sink or lid associated with the mating DSPGA372. BB372 is 30 mils thick allowing the heat sink or lid attached to DSPGA372 to extend at most 30 mils vertically above or below the land grid. Complementary holes are provided for coolant flow to match those on DSPGA372. Each BB372 has two stubs at opposite corners which mate with the DSPGA372 alignment holes. The stubs protrude on both sides of the carrier, allowing the carrier to make positive alignment with both the attached PCB and DSPGA package. The BB372 carrier is made from Vectra Liquid Crystal Polymer [Cor89] and was fabricated by Cinch. Table 7.2 summarizes the physical dimensions.

Our main disappointment with BB372 has been the handling care required. The fine wire

Feature	Size
Package Outline	1.4″ × 1.4″
Package Height	
including heat sink and lid	< 140 mil
excluding heat sink and lid	80 mil
Die Cavity	540 mil × 540 mil
Die Side Length (maximum)	500 mil
Pad Diameter	30 mil
Pad Spacing	50 mil
Cooling Hole (diameter)	100 mil
Mounting Hole	
(diameter)	78 mil
(slot length)	100 mil

Table 7.1: DSPGA372 Physical Dimensions

Top (die cavity) Bottom (heat sink)

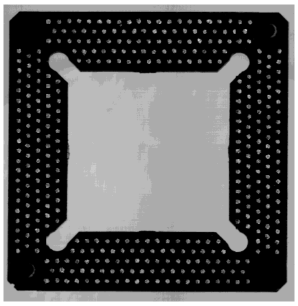


Pictures of DSPGA372 shown actual size

Figure 7.3: DSPGA372 Photos

composing the buttons can easily be pulled out of the cylindrical hole. If a person actually touches the buttons, the wires often attach to the ridges on the person's fingers and begin to unravel when the fingers move away from the connector. As a result, the connector will deteriorate when handled improperly. With proper equipment, the buttons can be restuffed, so repair is possible. Once inserted into a system and compressed, the buttons remain situated during normal use.

Initial experiments with a conductive elastomer from Fujipoly [Fuj92] suggest that elastomeric technology is a viable alternative for this application. Elastomeric connectors are more robust to human handling. The elastomer provides a sheet of anisotropic contacts such that no specialization is required to match the pad geometry of the package or PCB. Size and shape selection is the only customization required for each application. As a result NRE costs are minimal. Elastomer gives



Picture of BB372 shown actual size

Figure 7.4: BB372

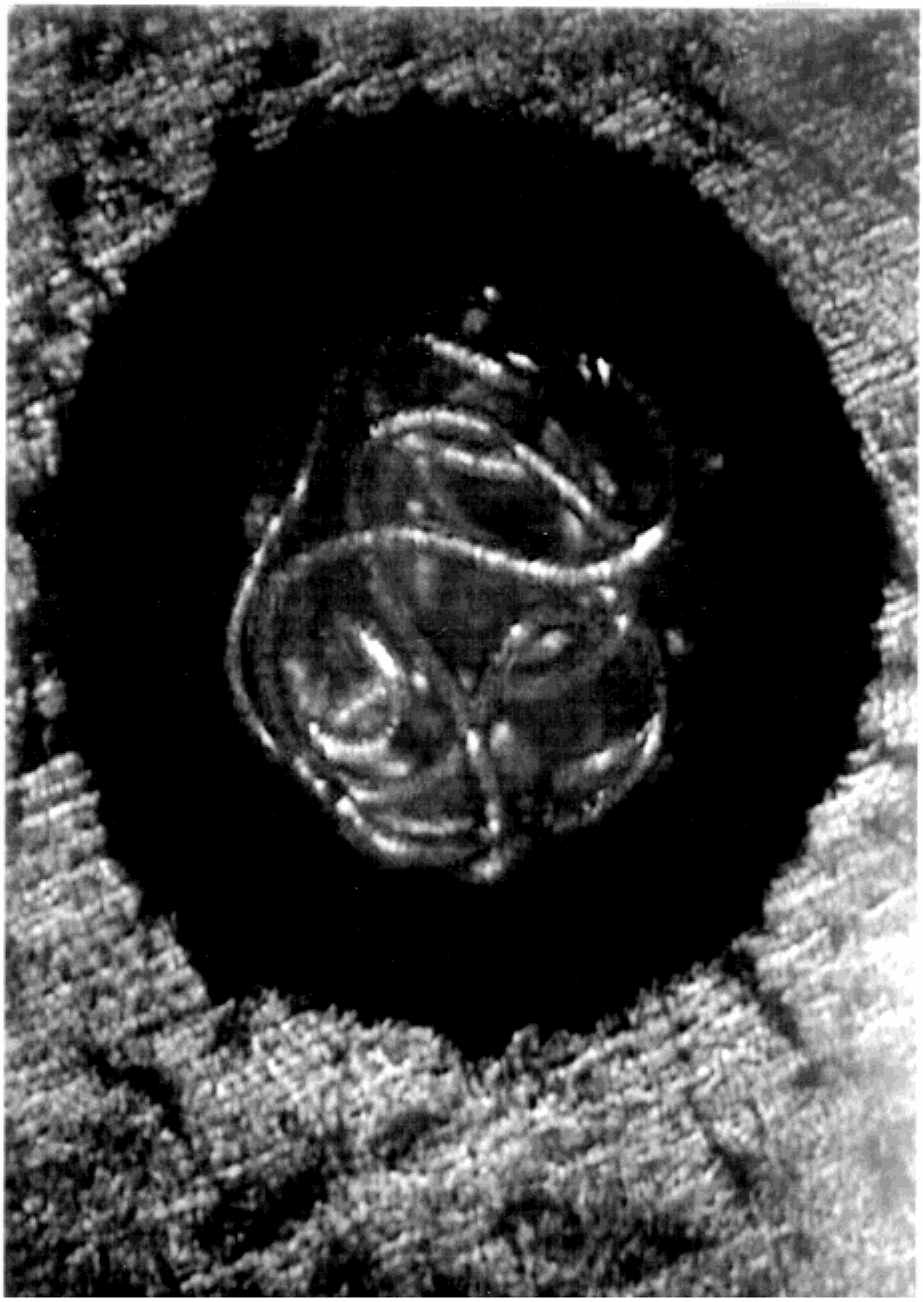
Feature	Size
Connector Outline	$1.4'' \times 1.4''$
Connector Height	30 mil
Center Opening	800 mil $\times$ 800 mil
Button Diameter	20 mil
Button Spacing	50 mil
Cooling Hole	
(diameter)	100 mil
Alignment Stub	
(top height)	28 mil
(bottom height)	48 mil
(diameter)	78 mil

Table 7.2: BB372 Physical Dimensions

us slightly more freedom to choose the connector height. As a side benefit, the elastomer serves to gasket the coolant forced through the cooling holes. On the negative side, the elastomer has a lower current capacity and higher resistance than the button-board connectors.

7.3.3 Printed Circuit Boards

Printed-circuit boards are sandwiched between component layers to provide planar interconnect. These PCBs are fairly conventional multilayer, controlled-impedance PCBs. The only special accommodations required are the land grids and alignment and cooling holes required to mate with the DSPGA packages. The PCB lands mimic the geometry of the DSPGA lands. For higher nobility contact, the PCB surfaces should be gold plated. Alignment holes allow connectors, such as the BB372, to align with the PCB land pattern. Cooling holes are required to facilitate coolant flow through the coolant holes provided in the connector and IC packages.



Closeup of button on BB372

Figure 7.5: Button

Inside the stack, each printed-circuit board mates with two components at each component site, one above the PCB and one below it. Most of the pins on the components which connect to a PCB should not be connected to the corresponding pins on the adjacent component on the opposite side of the PCB. The PCB must not provide continuity between the pads on each of its surfaces which occupy the same planar location. This requirement can be satisfied with conventional PCB manufacturing technology by either using vias which only connect into a portion of the internal routing layers on the PCB, or by offsetting the vias associated with each land so they do not intersect. In some case, continuity between corresponding pins on the components on opposite side of a board is desirable. Power signals, busses, and global signal lines are common examples of such cases.

7.3.4 Assembly

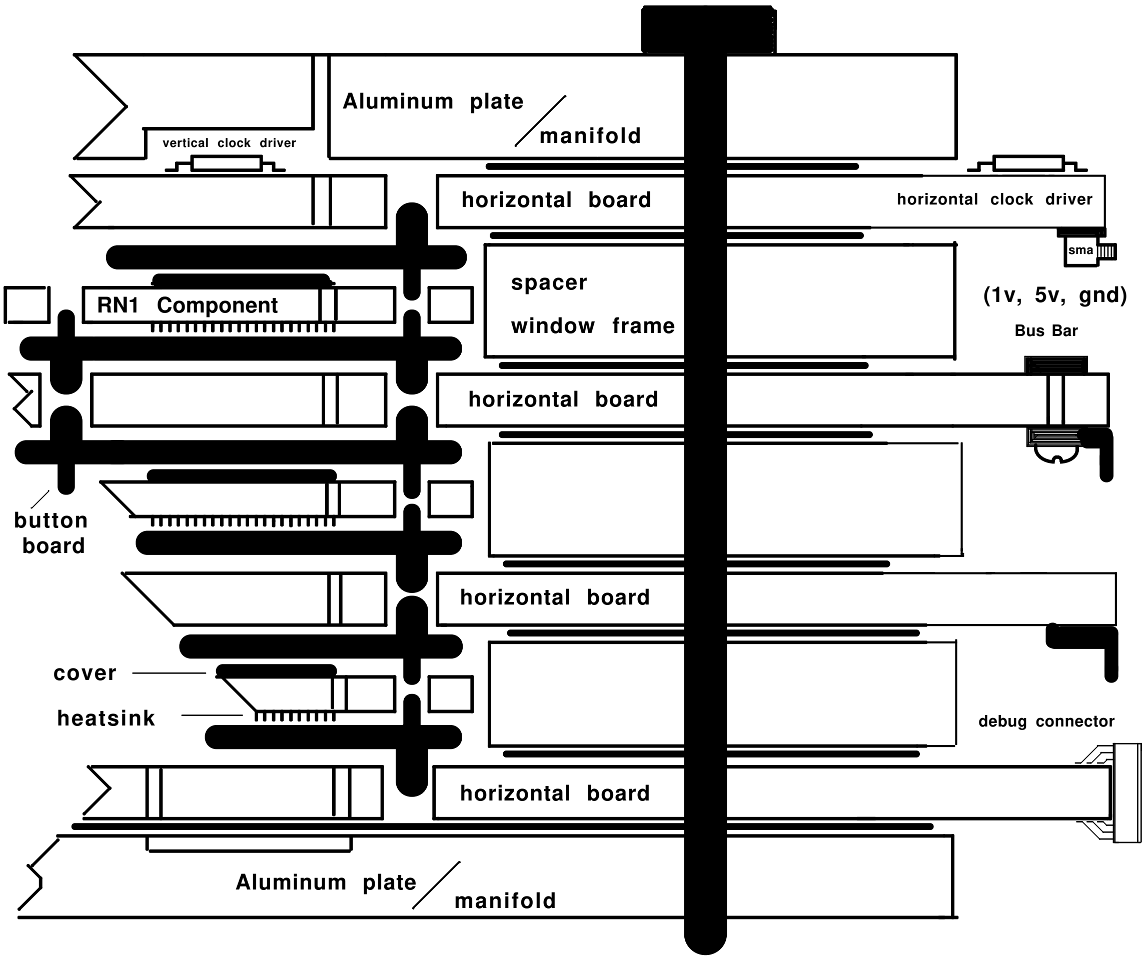
A stack is assembled by building up layers of PCBs, connectors, and packaged ICs. Figure 7.1 depicts the composition of a typical stack. Figure 7.6 shows a more detailed cross-section of a component stack. Figure 7.7 shows a close-up cross-section of an assembled stack. Strategically placed bolts throughout the stack provide vertical compressive force and provide crude board-to-board alignment. A rigid metal plate at the top and the bottom of the stack provides uniform compressive force across the stack. The alignment pins and holes provide fine alignment of PCBs, carriers, and packaged components. The alignment pins provided in BB372 allows each button board to align to the adjacent DSPGA and PCB independently. As a result, components realign at every stage. Alignment tolerances from layer to layer are not additive. Since each BB372 is 30 mils thick and the mating portion of each DSPGA372 is 80 mils thick, the space between PCBs in an assembled stack using these components is 140 mils.

7.3.5 Cooling

Once stacked, the coolant holes in the DSPGA packages, carriers, and PCBs will align defining vertical cooling channels through the stack structure. The heat sink attached to each DSPGA component is adjacent to the four cooling channels associated with its component. The channels allow forced air or liquid coolant to be circulated through the stack and across each heat sink. With proper heat sink design, the coolant flowing across a component between coolant channels will experience turbulent fluid flow to effect efficient heat transfer from the component to the coolant. All coolant channels can be left open allowing parallel flow across the heat sinks in a vertical column. Alternately, every other coolant hole can be plugged forcing serial flow across the heat sinks in a coolant column.

7.3.6 Repair

Component replacement is simplified by the solderless connections. To replace a faulty component, PCB, or connector, we simply need to disassemble the stack, substitute a known good replacement for the faulty unit, and re-assemble the stack. The solderless connections obviate the need to desolder components and rework fine-line PCBs. Of course, power must be disconnected from the stack and coolant drained before the stack can be disassembled.

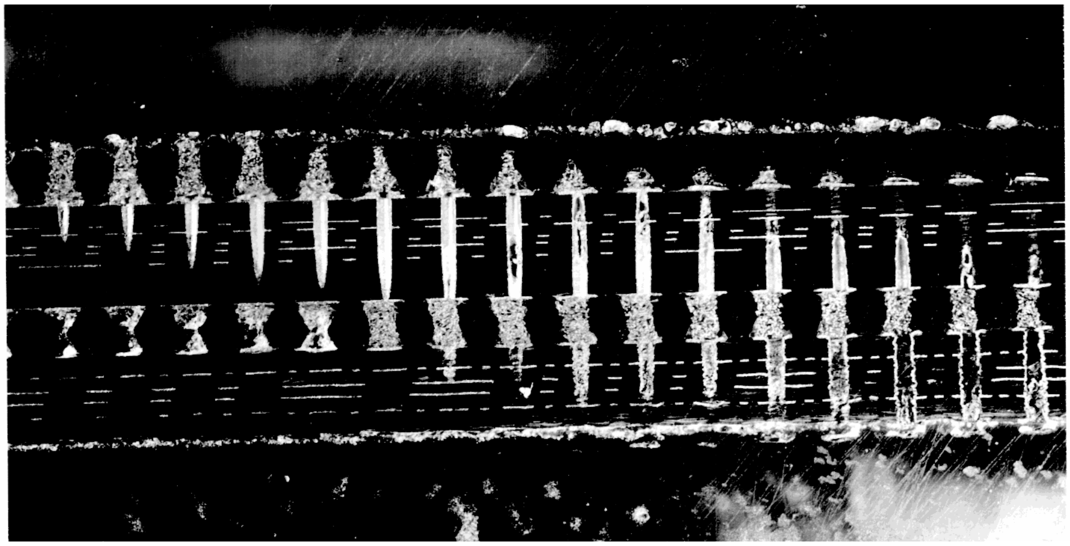


Shown above is an enlarged cross-section of a network component stack.
(Diagram courtesy of Fred Drenckhahn)

Figure 7.6: Cross-section of Routing Stack

7.3.7 Clocking

Any synchronous system requires that clocks be distributed to all clocked components so that the components see the clock edges at approximately the same time. The variation in clock edge arrival times is known as the *clock skew* and, generally, acts to limit the clock rate by increasing setup and hold times. The clock distribution problem in the stack is not much different from the problem of clock distribution on any large PCB or multi-PCB system. Carefully routed clock distribution trees and low-skew clock buffers can help minimize the skew.



Shown here is a close-up picture of a mated set of BB372 and DSPGA372 components which has been cut at an oblique angle to expose the topology of the mated components. The stack shown above is composed of a BB372, DSPGA372, BB372, and DSPGA372 sandwiched inside a plastic encapsulant. The encapsulant serves to hold the assembly together after the cross-sectional cut was made.

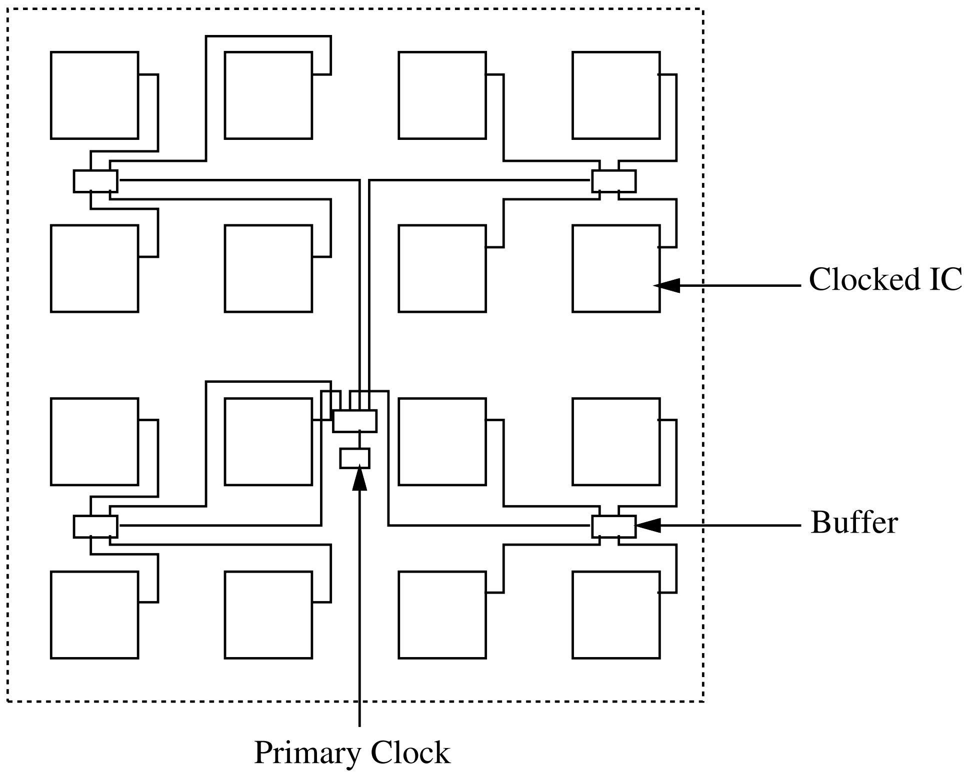
Figure 7.7: Close-up Cross-section of Mated BB372 and DSPGA372 Components

For short stacks in which the propagation delay vertically through a column of stacked components is small, it may be sufficient to carefully distribute the clock to each column on one layer of PCB (See Figure 7.8) then connect the clock signals vertically through each column. For tall stacks, the propagation delay through the column may be intolerable. Additionally, the impedance through the vertical interconnects may not be sufficiently controlled to rely on for clock distribution. Alternately, we can use a two-tier fanout scheme. Each PCB layer supports indepedently and identical clock fanout, similar to the single layer fanout. The input to the PCB fanout stage comes from another fanout tree through carefully tuned lengths of controlled-impedance cabling, such as flexible printed-circuit cables. The additional stage of fanout adds somewhat to the total skew.

Another option is to provide a direct connection to each clocked IC from a central driver where the edge arrival time is carefully tuned [Sim92]. Qualitatively, this kind of clock distribution is similar to the matched delay drivers described in Section 6.9. However, the requirements for low clock skew make it a much more difficult problem.

7.3.8 Stack Packaging of Non-DSPGA Components

As described so far, the packaging scheme requires all ICs be packaged in DSPGA packages. The networks described in this document are built out of homogeneous routing components. As



Shown above is a representative clock fanout scheme. The trace lengths in all clock runs should be balanced so as to guarantee as little skew between clock edges as possible.

Figure 7.8: Sample Clock Fanout on Horizontal PCB

long as we can package our routing component in DSPGA packages, the entire network can be easily constructed as described. It is, nonetheless, worthwhile to consider how to accommodate other components in the stack. The network endpoints, for example, constitute components other than routers, and we may not have the freedom to package all such components in DSPGA packages.

The stack structure will readily accommodate low-profile components in the spaces between DSPGA components. As noted, using DSPGA372 and BB372 components there is 140 mils of clearance between PCB layers. ICs which can fit comfortably within this height can be accommodated in the stack. The height requirement precludes almost all through-hole components including PGAs. Through-hole components further complicate the matter since their pins generally extend through the PCB and into the space below the attached PCB. Most leadless chip carriers (LCC) and gull-wing surface-mount components are around 100 mils thick and can easily be accommodated. J-lead surface-mount components are generally thicker and leave insufficient clearance. Smaller surface mount components such as TSOPs are easily accommodated and may be thin enough to allow components to be mounted on both sides of adjacent boards. Of course, the non-DSPGA components only have direct access to signals on the PCB to which they are mounted. Such components must make use of the spare, through connections provided by DSPGA packages when they require vertical interconnect. The non-DSPGA packages are not part of the assembled stack cooling channels. Cooling for these components is limited to horizontal forced-air between PCB layers.

7.4 Network Packaging Example

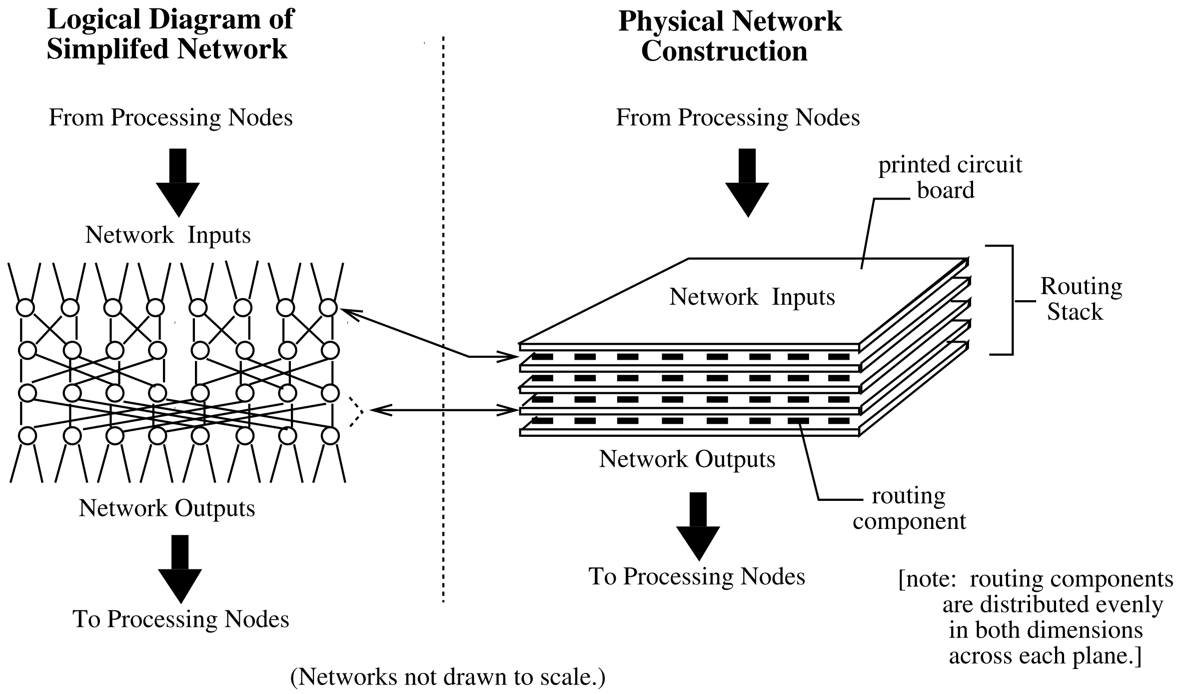
For the sake of concreteness, let us consider how we package a small multistage, multipath network. Figure 7.9 depicts a mapping of a multistage network into a stack package. Each stage of routers is assigned its own plane in the stack. The routers are distributed evenly in both dimensions within the plane. The PCB between planes of routing components implements the interconnect between adjacent stages of routing components. Since there are $\Theta(N)$ routers in each stage, distributed in two dimensions, each side is $\Theta(\sqrt{N})$ long, making the wire lengths between stages $\Theta(\sqrt{N})$ long. The transit latency growth for this structure will thus match our expectations from Section 3.1.5. If the inputs and outputs are not all segregated to opposite sides of the network as shown in the figure, it will be necessary to run the input and output connections which originate on the wrong side of the packaged network vertically through the network layers to connect the inputs or outputs into the network. These loop-through connections are one class of signals which use the straight-through interconnect provided by the DSPGA packages.

7.5 Packaging Large Systems

7.5.1 Single Stack Limitations

Unfortunately, there is a limit to the size of our stacks and hence the size of network which we can build in a single stack package. Recall from Section 7.2.2, our PCB size is limited somewhere under 30 inches. fine-line technology we use. Vertical layers are relatively thin. Consequently, if we package the layers as suggested in the previous section, we normally do not run into any physical constraints in the vertical packaging dimensions. For example, a typical PCB thickness for the horizontal PCB would be 100 mils. Section 7.3.4 noted that using DPGA372 and BB372 components, the space between PCBs is 140 mils. In this scenario, each additional network layer will increase the stack height by just under 0.25 inches. Since the PCB side size is increasing as $\Theta(\sqrt{N})$ and the number of stages, and hence height, is increasing as $\Theta(\log(N))$, we encounter the PCB size limitations before any vertical constraints.

Nonetheless, the vertical constraints that may arise are mostly dominated by cooling and signal integrity constraints. As the number of components in a vertical column increases, in a parallel cooling scheme we will require greater pressure and flow-rates to cool the components. Similarly, in a serial cooling scheme, the temperature gradient between inlet and outlet will increase. As noted in Section 7.3.7, for sufficiently tall stacks we cannot rely on vertical column interconnect for high-speed, low-skew, global signal distribution.



The diagram above depicts how a logical stack is mapped into the stack structure. The interconnect between each pair of routing stages is implemented as a PCB in the stack. Each stage of routing components becomes a layer of routing components packaged in DSPGA packages.

Figure 7.9: Mapping of Network Logical Structure onto Physical Stack Packaging

7.5.2 Large-Scale Packaging Goals

To build large networks, we seek two things:

1. A network stack primitive which represents a logical portion of the network and can be replicated to realize the connectivity associated with the target network

2. A topology for packaging and interconnecting these primitives

As developed in Chapter 3, for large machines we focus on fat-tree networks. Our problem is finding a decomposition of the fat tree into representative sub-networks which can be implemented in a single stack structure. We desire homogeneity in stack primitives for the same reasons we do in integrated-circuit components (See Section 2.7.2). When selecting a packaging infrastructure for assembling the primitive stacks, we must address the same general packaging issues raised in Section 7.1.

7.5.3 Fat-tree Building Blocks

Recall from Section 3.5.6 that we can think of a fat-tree, multistage network as composed of three parts:

1. a *down* network which recursively sorts connections as they head from the root toward the leaves

2. an *up* network to route connections upward toward the root

3. lateral crossovers which allow a connection to change from the up network to the down network when it has reached the least common ancestor of the source and destination nodes

Using radix-r, dilation-d crossbar routers, we build an r-ary down tree sorting network much like a flat, multistage network. Routers in the upward path allow a connection to connect into one of the next $(r-1)$ downward routing stages or continue routing upward. The upward routers compose both the up network and the crossover connections. For every $(r-1)$ logical tree levels, we have one upward routing stage.

We can collect the $(r-1)$ downward routing stage, the associated upward routing stage, and the crossover connections into a physical tree level. Each such physical tree level encompasses $(r-1)$ levels of the original tree. Taking our $(r-1)$ down tree stages, the total sorting performed by a physical tree level is r_p as given in Equation 7.1.

$$r_p = r^{(r-1)} \tag{7.1}$$

The size of the logical node at each physical tree level will increase as we head towards the root since the bandwidth at each tree stage increases toward the root. As a result, we need to further decompose each physical tree level into primitive units which can be assembled to service the varying bandwidth requirements at each tree stage.

We use the term *unit tree* to refer to any primitive stack structure which implements a fixed bandwidth slice of each physical tree level. There is a large class of unit trees based on the parameters of the router and packaging technology. Table 7.3 summarizes the parameters associated with a unit tree stack. The router parameters, r, d, and w have been discussed in detail in Chapters 3 and 4. At the "bottom" of the unit tree, the number of channels headed to and from physical tree levels closer to the leaves is denoted c_l. c_l is a multiple of the router dilation, d, which determines the size of the bandwidth slice handled by the unit tree. Once r is determined, c_l can be chosen such that the size of the unit tree is accommodated in a single packaging stack. One generally wants c_l large for increased fault tolerance and resource sharing. The available packaging technology will limit the size of any stacks and, hence, limit c_l. In these respects, c_l is much like the router dilation, d; we generally select as large a value as we can afford given our physical and packaging limits. At the leaves of a tree, we need to connect the processors into the tree, and hence we need a unit tree with channel capacities matched to the input and output channel capacity of each node (*i.e.* $c_l = ni = no$). The channel capacity in and out of the top of a unit tree, c_r, is fully determined once c_l and r are chosen and is given by Equation 7.2.

$$c_r = c_l \cdot r^{(r-2)} \tag{7.2}$$

r	router radix
d	router dilation
w	router width
c_l	channels per logical direction toward leaves (depends on packaging technology and system requirements)
r_p	logical tree levels per physical tree level (determined by router radix, r)
c_r	channels out of unit tree toward root (determined by c_l and r)

Table 7.3: Unit Tree Parameters

	Routing Components	
	$UT_{64\times2}$	$UT_{64\times8}$
Up Routing Stage	16	64
Final Down Routing Stage	16	64
Middle Down Routing Stage	12	48
Initial Down Routing Stage	8	32

Table 7.4: Unit Tree Component Summary

7.5.4 Unit Tree Examples

For the sake of illustration, let us consider two specific unit tree configurations introduced in [DeH91] and [DeH90]. Here, we denote each unit tree as $UT_{r_p \times c_l}$. Both of these unit trees use the RN1 routing component, a radix-4, dilation-2 routing component (See Chapter 8). $UT_{64\times2}$ has $c_l = 2$ and, consequently, $c_r = 32$. $UT_{64\times8}$ has $c_l = 8$ and $c_r = 128$. Both have $r_p = 64$. Table 7.4 summarizes the number of components composing each stage of the network in each unit tree. If each routing component is housed in a DSPGA372 package measuring 1.4 inches along each side, and we leave as much space between routing components, the $UT_{64\times2}$ stack measures just under 1 foot along each side, and the $UT_{64\times8}$ measures just under 2 feet. Assuming BB372 connectors, the four layers of routing components in both unit trees are 1 inch tall. With compressional plates, each stack is under 2 inch tall.

At the leaves, a single unit tree with $c_l = ni = no$ is connected to each cluster of r_p processors. To build the next size larger machine, we replicate the lower physical subtree r_p times and build a new tree level out of enough unit trees to support the channels entering or leaving the roots of all of the lower physical subtrees. To quantify, if c_n channels come out of each subtree with n levels, the total number of unit trees required to form the root of the physical subtree at physical tree level $n + 1$ is given by;

$$N_{n+1} = \frac{c_n}{c_l} \qquad (7.3)$$

In turn, the physical subtree rooted at physical tree level $n + 1$ will have a total channel capacity out of its root given by;

$$c_{n+1} = c_r \cdot N_{n+1} \tag{7.4}$$

With the particular unit trees just introduced, we form the leaves of the tree by connecting one $UT_{64\times2}$ to each cluster of 64 processors. If we use $UT_{64\times2}$ unit trees in the second level as well as the first, we need $\frac{32}{2} = 16$ $UT_{64\times2}$ unit trees to form the root of each physical subtree rooted in the second physical tree level. The subtree rooted in the second physical tree level support $64^2 = 4096$ nodes and includes 64 $UT_{64\times2}$ unit trees in the first physical tree level. Alternately, we can use $\frac{32}{8} = 4$ $UT_{64\times8}$ unit trees to compose the root of each subtree rooted in the second physical tree level to support the same number of nodes and first level unit trees. To build an even larger machine, we can make 64 copies of a 4096-node tree and interconnect them using a third physical tree level composed of $\frac{32\cdot16}{2} = 256$ $UT_{64\times2}$ unit trees or $\frac{128\cdot4}{8} = 64$ $UT_{64\times8}$ unit trees. The resulting subtree rooted in the third physical tree level supports $64^3 = 262,144$ nodes and has a total of $64 \cdot 16 = 1024$ $UT_{64\times2}$ or $64 \cdot 4 = 256$ $UT_{64\times8}$ unit trees composing the internal tree nodes in the second physical tree level and 4096 $UT_{64\times2}$ unit trees composing the nodes in the first physical tree level.

7.5.5 Hollow Cube

To physically organize the unit trees which make up a large fat tree in an efficient manner, we must consider the interconnect topology. Each group of r_p subtrees at physical tree level n will be connected to the subset of unit trees at physical tree level $n + 1$ which compose the root of the subtree. If each of the subtrees at tree level n is composed of U unit trees and the parent tree level is constructed from the same size unit trees, we know the parent set of unit trees will be composed of

$$U_{parent} = \frac{c_r}{c_l} \cdot U$$

unit trees (See Equations 7.3 and 7.4). This gives us

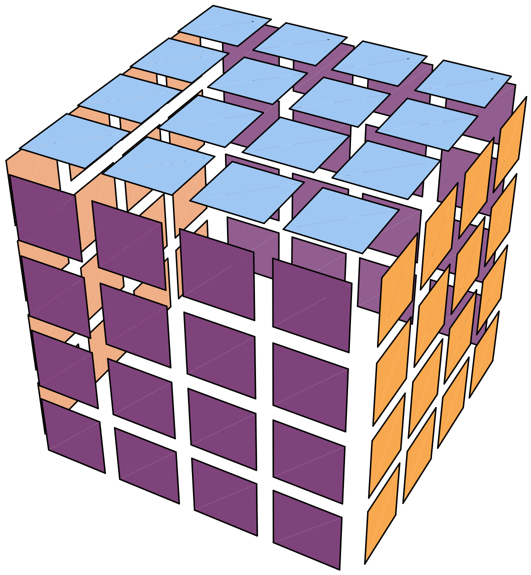
$$U_{children} = r_p \cdot U = r^{(r-1)} \cdot U \tag{7.5}$$

unit trees at tree level n connecting to

$$U_{parent} = \frac{c_r}{c_l} \cdot U = r^{(r-2)} \cdot U \tag{7.6}$$

unit trees at tree level $n + 1$. From these relations we see that the group of unit trees composing the root of a physical subtree will generally be connected to r times as many similarly sized unit trees in the immediately lower physical tree level.

When $r = 4$, as in our examples from the previous section, a natural approach to accommodating this r:1 convergence ratio in our three-dimensional world is to build *hollow cubes*. We select one cube face as the "top" of the cube and tile the unit trees composing the root of a physical subtree in the plane across the top face of the cube. Together the four adjacent faces in the cube have four times the surface area of the top and hence can be tiled with four times as many unit tree stacks. The sides can house all of the unit trees composing the roots of the $4^{(4-1)} = 64$ immediate children



Shown here is an example where the unit trees in both physical tree levels shown are the same size. This is comparable to the case described in Section 7.5.4 where a 4096 processor machine was built from two physical tree levels composed entirely of $UT_{64\times 2}$ unit trees. Based on the stack sizes assumed in Section 7.5.4, this hollow cube would measure about 4 feet on each side.

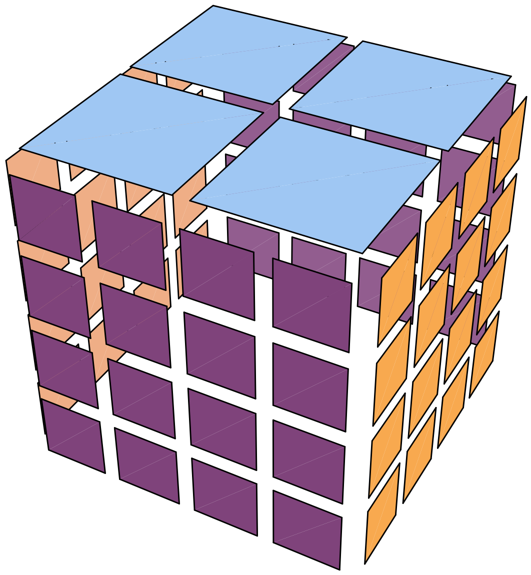
Figure 7.10: Two Level Hollow-Cube Geometry

of the cube top. If the top is part of physical tree level n, the sides contain unit trees which are part of physical tree level $n-1$. We leave the "bottom" of the cube open to increase accessibility to the cube's interior. Figures 7.10 and 7.11 show two hollow-cube arrangements for machines composed of two physical tree levels. Figure 7.12 depicts the hollow-cube arrangement for a machine with three physical tree levels.

The hollow-cube topology is optimized to expose the surfaces of stacks which interconnect to each other. All of the interconnect between unit trees within a hollow-cube fat tree will occur between the sides and face of some cube. The hollow-cube topology is a three-dimensional fractal-like geometry which attempts to maximize the surface area exposed for interconnect within a given volume.

7.5.6 Wiring Hollow Cubes

Each of the unit tree stacks in the sides of a cube feeds connections to and from unit trees composing the parent subtree in the top of the cube. All of the connection in and out of the top of a unit tree stack are logically equivalent. These logically equivalent channels should be distributed among the unit trees composing the parent subtree for fault tolerance. This fanout from a unit tree to multiple unit trees in the parent subtree is desirable for the same reasons fanout from the dilated



Shown here is an example where the unit trees in the higher physical tree level are four times as large as the ones in the lower physical tree level. This is comparable to the case described in Section 7.5.4 where the lowest tree level was composed of $UT_{64\times 2}$ unit trees while the higher level was composed of $UT_{64\times 8}$ unit trees. Like Figure 7.10, this hollow cube measures about 4 feet on each side.

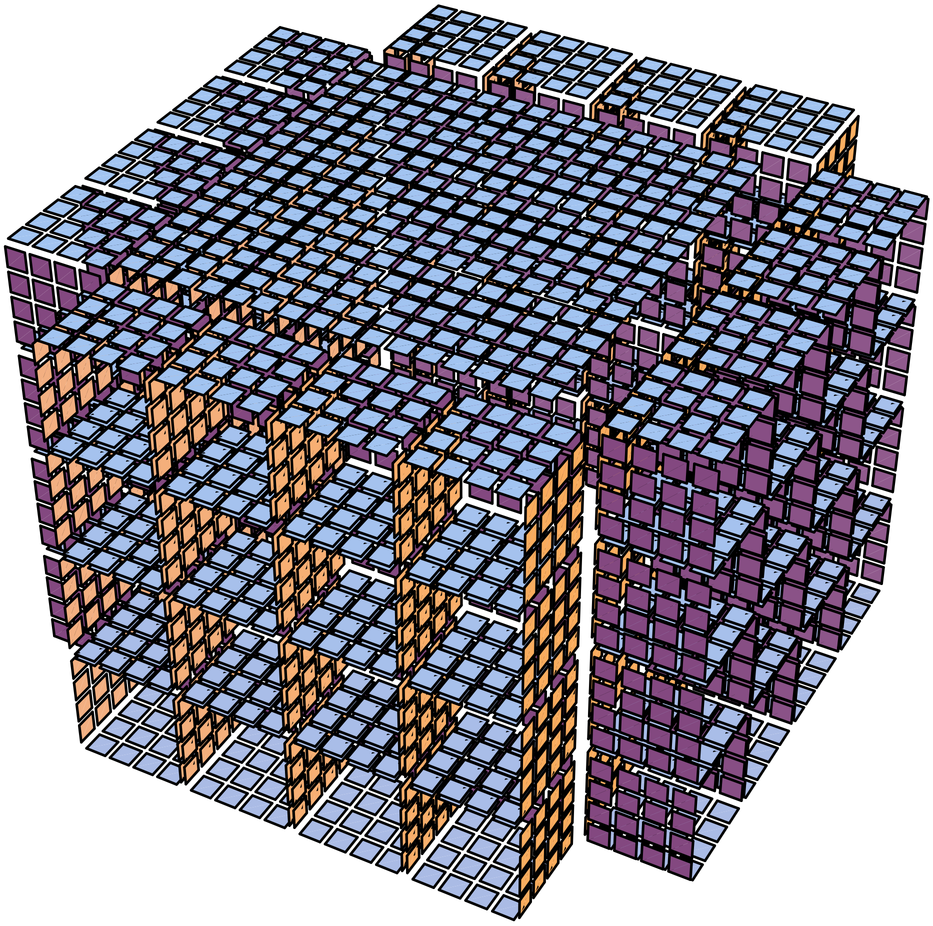
Figure 7.11: Two Level Hollow Cube with Top and Side Stacks of Different Sizes

connections of a single router is desirable (See Section 3.5.3). With proper fanout entire unit tree stacks can be removed from the non-leaf, physical tree levels, and the network still retain sufficient connectivity to route all connections.

Wire connections are made through the center of each hollow cube using controlled-impedance cables. The worst-case wire length between two physical tree levels is proportional to the length of the side of the cube which the wire traverses. Any route through the network traverses a cube of a given size at most twice, once on the path to the root and once on the path from the root to the destination node.

7.5.7 Hollow Cube Support

To support the unit trees making up a hollow cube, we build a gridded support substrate much like the raised floors used in traditional computer rooms. Due to the physical size of the hollow cubes, they occupy room-sized or building-sized structures. The structure to house the hollow-cube network is built with these gridded walls and ceilings to accept the unit trees which are used as building blocks. Conduits for power and coolant are accommodated along grid lines in the gridded substrate. The sixth face of each hollow cube, vacant of unit trees, supplies access to the interior and provides a location for cooling pumps and power supplies.



Shown above is a hollow cube containing three physical tree levels. If all the unit trees were $UT_{64\times 2}$ unit trees, this structure would house 262,144 endpoints. Making the same assumptions as in Figure 7.10, the central cube in this structure measures about 16 feet along each side. The whole unit, as shown, would measure 24 feet along each side and be 16 feet tall.

Figure 7.12: Three Level Hollow Cube

Rather than directly connecting the wires into a unit tree to the unit tree itself, we can build a wiring harness which mates with the unit tree. This harness collects all the wires connecting to a single unit tree. The harness makes compressional contact with either the top or bottom of a unit tree stack to connect the wires to the unit tree. This wiring harness simplifies the task of replacing a unit tree stack. Without the harness it would be necessary to unplug all of the connections into the outgoing unit tree and then reconnect them to the replacement. Since each unit tree generally supports hundreds of connections, this operation would be involved and highly error prone.

7.5.8 Hollow Cube Limitations

As introduced, hollow cubes are only well matched to radix four fat-trees and only retain many of their nice properties up to three physical tree levels. For the first three tree levels, the cube side lengths increase by a factor of four between tree levels. Since each successive tree level accomodates 64 times as many processors, side length, and hence worst-case wire lengths, grows as $\sqrt[3]{N}$. Starting at the fourth physical tree level, the need to accommodate space occupied by lower tree levels increases the growth factor to six. As a result worst-case wiring lengths grow faster beyond this point. Also starting at the fourth physical tree level, the "bottoms" of many of the hollow cubes become blocked by other hollow cubes limiting maintenance access.

7.6 Multi-Chip Modules Prospects

The Multi-Chip Module (MCM) is an emerging packaging technology that further improves component packaging density by dispensing with the IC package. Bare die are bonded directly to a high-performance substrate which serves to interconnect the die. The removal of the IC package allows components to be situated more closely lowering interconnect latency. Recall from Section 7.2.1 that package size is proportional to the spacing of external i/o pins not the die size. Avoiding the package allows the component to only take up space relative to the die size.

Unfortunately, MCM technology has a number of drawbacks which relegate its use to small, high-end systems today. Few IC manufacturers are in the practice of supplying bare, tested die. Final, full-speed IC testing is generally done after the die is packaged. The facilities available for full-scale testing of unpackaged die are limited. As a result, it is generally not possible to know whether all of the die will work before assembling an MCM. Since component speed grading is also generally only performed on packaged ICs, one has little knowledge of the yielded operational speed for each IC. These drawbacks are compounded by the fact that repair and rework technology for MCMs is in its infancy. The MCM technologies available today generally are not amenable to die replacement. Consequently, stocked MCM yield is low. Additionally, NRE costs on MCM substrates are comparable to silicon IC NRE costs rather than PCB NRE costs. The combination of the fact that MCMs have yet to become a high-volume technology, the lower yield due to lack of repairability, and high NRE costs make MCM technology uneconomical for most designs at present.

When MCMs become an economically viable technology, they may be able to replace packaged ICs and PCBs. Stacks composed from layers of MCMs could be a factor of 3 to 4 smaller in each of the planar dimensions than the stacks described with DSPGA372 style components. To build MCM stacks, we would need the ability to connect signals into both sides of an MCM substrate.

If the MCM is limited to single-sided or peripheral i/o, the size of the MCM required to satisfy i/o requirements alone may negate much of the density benefits. Unless significant advances are made in MCM repair, MCM stacks would not have the repair advantages of the current stack packaging scheme. The hollow-cube topology can be used to interconnect MCM stacks with most of the same benefits and limitations.

7.7 Summary

In this chapter we developed a three-dimensional stack packaging technology. Using dual-sided pad-grid array IC packages, compressional board-to-package connectors, and conventional PCBs, we developed a stack structure with alternating layers of components and PCBs. The combination of DSPGA packages and compressional connectors served to both connect packaged ICs to horizontal PCBs and to provide vertical board-to-board interconnect within the stack. We demonstrated how to map a multistage networks into this three-dimensional stack structure. We then looked at the limitations on the size of stacks we can build. To accommodate larger systems, we developed a network decomposition for fat-tree, multistage networks which allows us to build large fat-tree networks from one or two primitive unit tree stack designs. We also showed how these unit tree stacks can be arranged in a hollow-cube geometry to construct large fat-tree machines and commented on the limitations of the hollow-cube structure.

7.8 Areas To Explore

Many areas of packaging are quite fertile for exploration.

- We have pointed out the limitations with current MCM technology and suggested some requirements necessary for the technology to provide real benefits.
- The hollow-cube topology has many nice properties up to its limitations. It would be useful to find alternative topologies with a wider range of application.
- If free-space optical interconnect becomes a viable technology on this scale, the hollow cubes can become truly hollow. Using free-space optical transmission across the long distances through the cube, we could exploit the propagation rate of light to keep transit latencies low. The distance across the larger hollow-cube stages is sufficiently large that the savings due to higher propagation rate may make up for the latency associated with converting electrical signals to light and back again. Recent work in optics promises to integrate electrical and optical processing so that we will be able to build optical conversions into our primitive routing elements [Mil91]. Since photons do not interfere with each other, free-space optics makes the task of wiring the interconnections through the center of the hollow cube trivial. [BJN+86] and [WBJ+87] discuss early work on large-scale, free-space optical interconnect for VLSI systems. They use holographic optical elements to direct optical beams for interconnections. The flexibility of the holographic media holds out promise for adaptive and dynamic connection alignment and reconfiguration. At present, much work is still needed on efficient conversion between electrical and optical signals and emitter-detector alignment.

Part III

Case Studies

RN1 is a circuit-switched, crossbar routing element developed in the MIT Transit Project [MDK91]. RN1 may be configured either as an 8-bit wide, radix-4, dilation-2 crossbar router (*i.e.* $i = 8$, $r = 4$, $d = 2$, $w = 8$) or as a pair of independent, radix-4, dilation-1 routers (*i.e.* two routers with $i = 4$, $r = 4$, $d = 1$, $w = 8$) (See Figure 8.1). In both configurations, RN1 supports the basic routing protocol detailed in Section 4.5. RN1 has no internal pipelining. Each RN1 router establishes connections and passes data with a single clock cycle of latency.

Figure 8.2 shows the micro-architecture for RN1. Each forward and backward port contains a simple finite-state machine for maintaining connection state and processing protocol signalling. The line control units keep track of available backward ports and handle random port selection. Backward port arbitration occurs in a distributed fashion along each logical output column. When several forward ports attempt to open a connection to the same logical backward port during the same cycle, an 8-way arbitration for the available backward ports takes place. [Min91] contains a detailed description of the design and implementation of RN1.

RN1 was implemented as a full-custom, CMOS integrated circuit using a combination of standard-cell and full-custom layout. Standard, five-volt, CMOS i/o pads were used with this first-generation routing component. RN1 was fabricated in Hewlett Packard's 1.2μm CMOS process (CMOS34) through the MOSIS service. The RN1 die measures 1.2 cm on each side. The die size

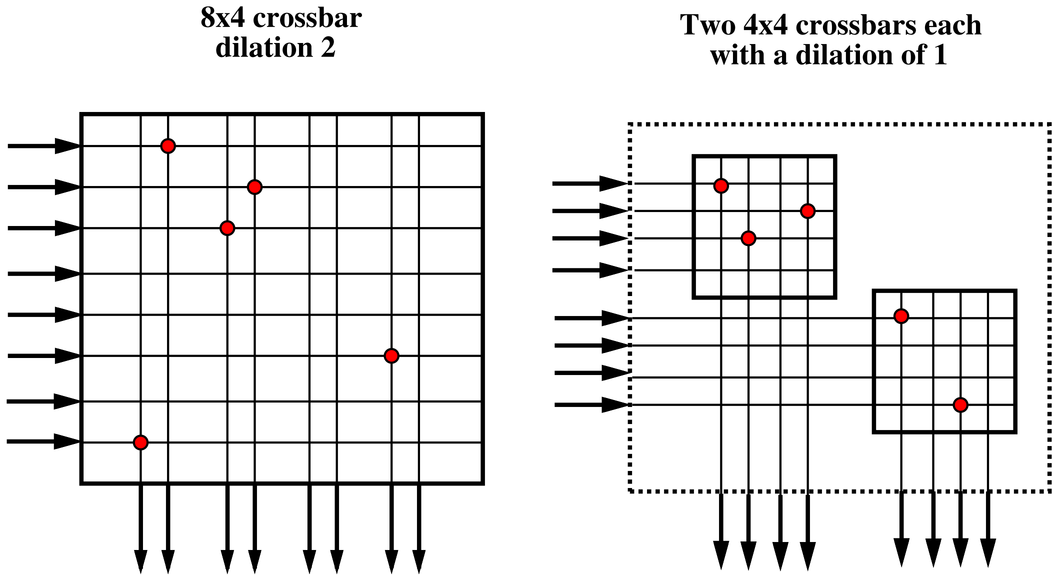


Figure 8.1: RN1 Logical Configurations

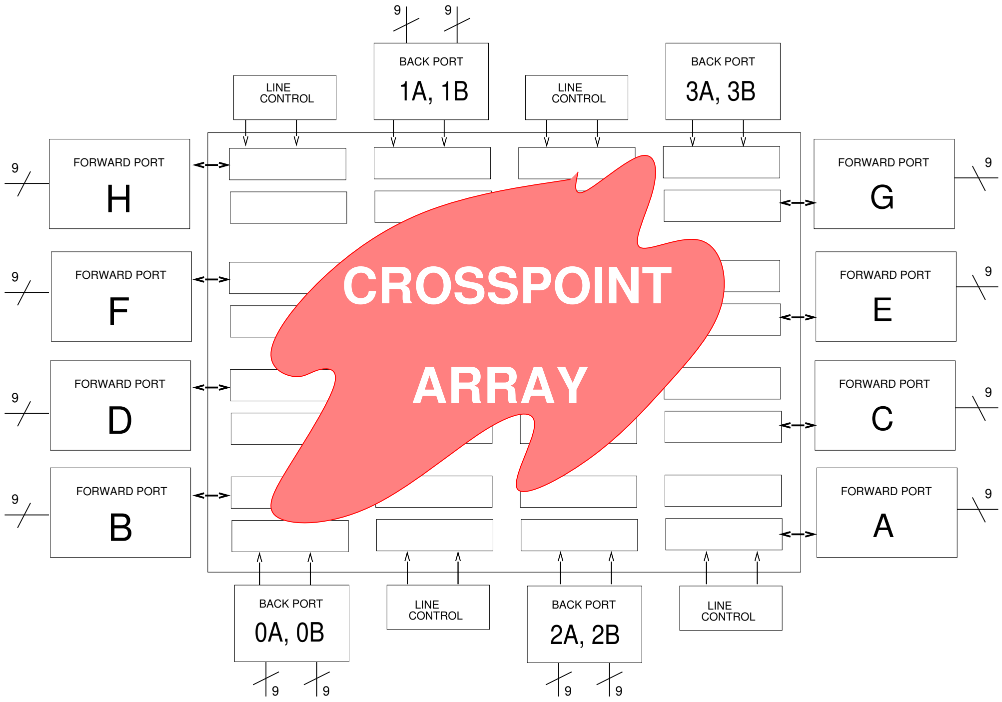
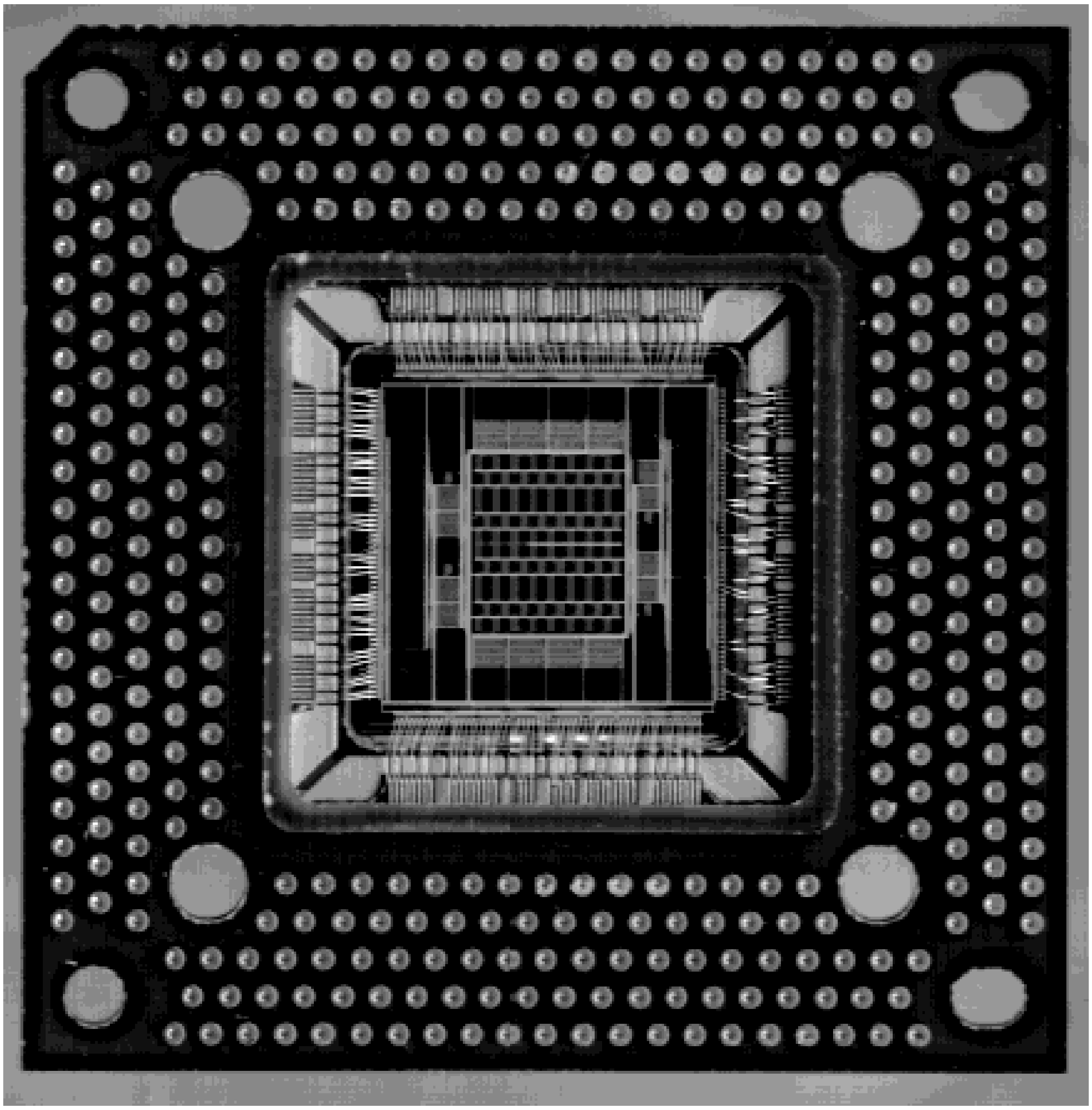


Figure 8.2: RN1 Micro-architecture

was fully determined by the perimeter required to house 160 signal pads plus power and ground connections in a single row of peripheral bonding pads. RN1 is packaged in a DSPGA372 package as shown in Figure 8.3.

RN1 can support clock rates up to 50 MHz. Analysis of the critical-path timing indicates that the standard, five-volt i/o pads and the standard clock buffer are key contributors limiting clock frequency. The input and output latencies for the RN1 i/o pads are each roughly 10 ns (*i.e.* $t_{io} = 20$ ns). Inside the i/o pads, the latency through the IC logic is around 14 ns (*i.e.* $t_{switch} = 14$ ns).



RN1 is housed in the DSPGA372 package introduced in Section 7.3.1.

Figure 8.3: Packaged RN1 IC

9. Metro

The Multipath Enhanced Transit Routing Organization (METRO) is an architecture for second generation Transit routing components. The METRO architecture encompasses the MRP-ROUTER protocol described in Chapter 4 including the enhancements described in Section 4.9. In addition to the basic router protocol implemented by RN1, METRO includes multi-TAP scan, port-by-port deselection, partial-external scan, width cascading, fast path reclamation, and pipelining provisions.

9.1 METRO Architectural Options

The METRO architecture encompasses a large space of routing component configurations and router behavior. Some architectural parameters must be fixed when constructing a particular routing component. Any particular router will have a fixed data width (w), a fixed number of inputs and outputs (i, o), a fixed number of pipeline delays routing data through the router (dps), a fixed number of header words swallowed during connection establishment (hw), and a fixed number of scan paths (sp). Each particular router will have configuration options, accessible via the TAP, which allow one to choose among a set of possible router behaviors. One can configure any METRO routing component to act as a radix-r, dilation-d router ($o = r \times d$) by setting the effective dilation. Each particular router will have a maximum limit on the dilation setting (max_d). Each forward and backward port on any METRO router can be enabled or disabled (See Section 4.9.1). It is also possible to configure each forward and backward port on any METRO routing component to accommodate the pipeline delay cycles associated with pipelining data on the wires between routers (See Section 4.11.3). The configured value, vtd, defines the number of cycles which will transpire between the time when a port sends a TURN and the time when the first piece of return data arrives. Each particular routing component will allow vtd to take on any value up to some component specific maximum, max_vtd. Each forward and backward port can also be configured to either use fast path reclamation or detailed connection shutdown (See Section 4.9.2). Table 9.1 summarizes the architectural variables which must be selected during the construction of a routing component. Table 9.2 summarizes the configuration options which are available on a METRO routing component.

9.2 METRO Technology Projections

Based on our experience with RN1 and the signalling technology described in Chapter 6, we believe we can build a comparably sized METRO router in Hewlett Packard's 0.8μm effective gate-length CMOS process (CMOS26) which operates at clock frequencies up to 200 MHz. As noted in the previous section, the critical path for connection establishment in RN1 was under 14 ns. Through a combination of technology scaling and clever circuit techniques, we can reduce this allocation latency to 5 to 10 ns. The flow-through latency on RN1 was much less than the allocation latency. Routing data between a forward port and backward port of an open connection with less than 5 ns of latency should be quite manageable. Consequently, we expect a part operating at 200 MHz can

Variable	Function	Range
sp	Number of Scan Paths	$sp \geq 1$
w	Bit Width of Data Channel	$w \geq max\_d$
max_d	Maximum Dilation	$max\_d = 2^n$ for some $n > 0$ $max\_d \leq o$
i	Number of Forward Ports	$i = 2^n$ for some $n > 0$
o	Number of Backward Ports	$o = 2^n$ for some $n > 0$ $o \geq max\_d$
ri	Number of Random Inputs Bits	$ri \geq 1$
hw	Number of Header Words Consumed Per Router	$hw \geq 0$
dps	Number of Data Pipestages Through Router	$dps \geq 2$
max_vtd	Maximum Number of Delay Slots Available for Variable Turn Delay	$max\_vtd \geq 0$

This table summarizes the architectural variables which distinguish any particular METRO routing component.

Table 9.1: METRO Architectural Variables

Option	One for Each	Number of Instances	Bits Each
Dilation (d)	component	1	$\lfloor \log_2(log_2(max\_d)) \rfloor + 1$
Port (De)select	port	$i + o$	1
Deselected Port Drives Output	port	$i + o$	1
Fast Reclamation	port	$i + o$	1
Turn Delay (vtd)	port	$i + o$	$log_2(max\_vtd)$

Each METRO router has several configuration options which control its behavior. This table summarizes the options common to all METRO routing components.

Table 9.2: METRO Router Configuration Options

be built with one cycle of data latency ($dps = 1$) and one or two cycles of connection establishment latency ($hw = 0$ or $hw = 1$). Using the i/o pads detailed in Chapter 6, the delay through a pair of i/o pads is 3 ns. As long as the propagation delay between a wire's endpoints is under 2 ns, the i/o pads and interconnect serve as a single pipeline stage. With conventional PCB ($\epsilon_r = 4$), wire runs up to 30 cm in length can be traversed in a single 200 MHz clock cycle.

10. Modular Bootstrapping Transit Architecture (MBTA)

The Modular Bootstrapping Transit Architecture (MBTA) is a series of small multiprocessors based around multistage routing networks composed of RN1 or METRO routing components. MBTA integrates a number of minimal processing nodes with a multistage network organized as described in Section 3.5.

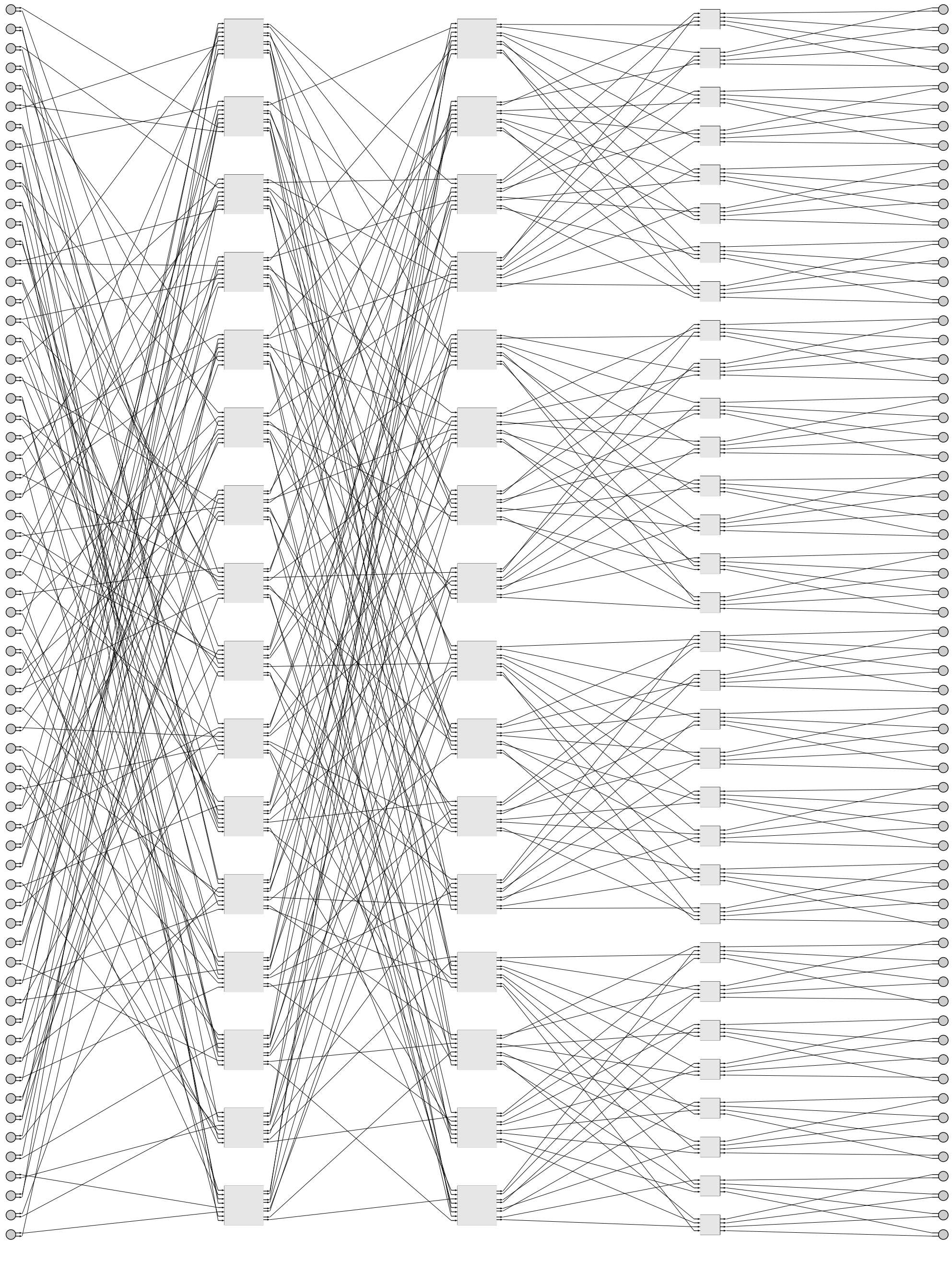
10.1 Architecture

Figure 10.1 shows the network used for a 64-processor MBTA machine. Each processing node has two network inputs and two network outputs ($ni = no = 2$) for fault tolerance. The network shown is composed of RN1-style routing component and uses the dilation-1 router configuration in the final stage so that two different routing components may provide network outputs from the network to each processing node. Since RN1 is a radix-4 routing component, the network is comprised of $\log_4(64) = 3$ routing stages.

Figure 10.2 shows the architecture of the MBTA processing nodes. Each node is composed of a RISC microprocessor (*e.g.* Intel's 80960CA [MB88] [Int89]), fast, static memory, network interfaces, and support logic. Four logical network interfaces service the two connection into and the two connection out of the network. The processor performs computation, initiates network communications, and services non-primitive network operations. The processor is also responsible for the highest levels of MRP-ENDPOINT, which are not handled by the network interface. A single, high-speed memory bank serves to hold instructions and data for the processors, store data coming and going from the network, and store connection status information. The basic node architecture also has provisions to support co-processors and alternate forms of memory. In order to interface MBTA machines with existing computers and data networks, there are provisions for some nodes to accommodate external interfaces.

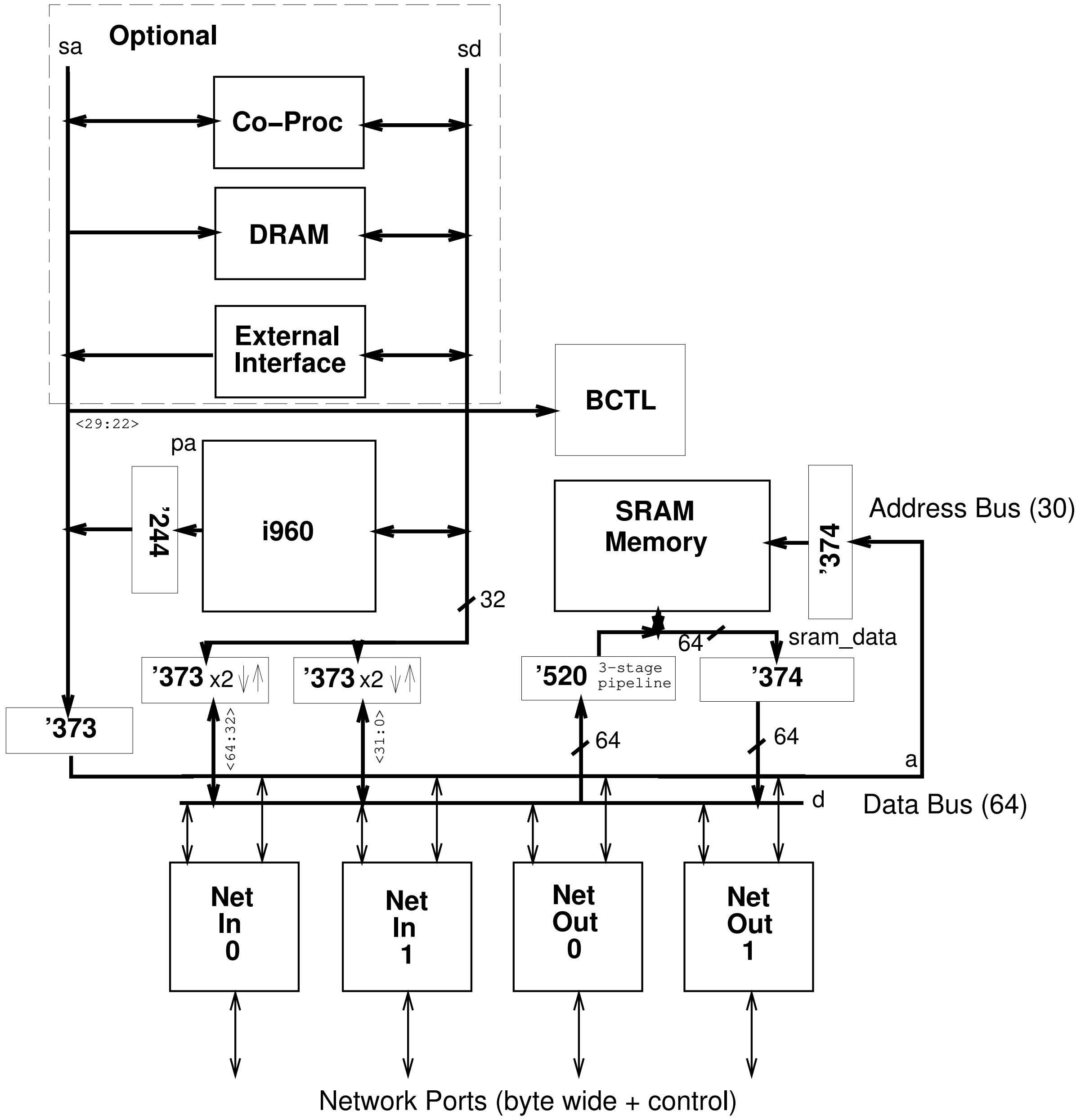
10.2 Performance

The MBTA architecture has been balanced to support byte-wide network connections running at 100 MHz. The network interfaces send data from the fast, static memory and receive data into the memory, as well. Consequently, each network interface requires 100 megabytes/second (100 MB/s) of bandwidth into memory during sustained data transfers. The processor is running at 25 MHz and may read up to one word, or four bytes, per cycle during burst memory operations. To prevent the processor from stalling, it, too, needs 100 MB/s of bandwidth into memory. To run all network interfaces and the processor simultaneously at full-speed, we would need 500 MB/s of bandwidth into memory. To simplify the problem, we restrict operation so that only one network input may be feeding data into the network at a time. This restriction limits the contention in the network while giving us the fault tolerance benefits of having two connections into the network. To provide the 400 MB/s of bandwidth required, we use 64-bit wide, 20 ns, synchronous SRAM



Shown here is the network for a 64-processor MBTA machine composed of RN1 routing components.

Figure 10.1: MBTA Routing Network



Shown above is the architecture for each MBTA node. The units outside of the dotted box are common to all MBTA nodes. Within an MBTA machine, a few nodes would support external interfaces.

Figure 10.2: MBTA Node Architecture

for the high-speed memory on a pipelined bus. Each of the four units using the memory gets the opportunity to read or write one, 8-byte value to or from memory every 80 ns. This allows each unit to sustain 100 MB/s data transfers without internal buffering as long as data can be transferred as contiguous double-words.

If all nodes are busy sending data at 100MB/s, a 64-processor network, like the one shown in Figure 10.1, can support a peak bandwidth of 6,400MB/s = 6.4GB/s. With one network input and both network outputs in operation, a single node can simultaneously transfer up to 300MB/s. Running the METRO component described in Section 9.2 at 100 MHz, it takes one cycle to traverse each router and one cycle to traverse each wire in the network. The unloaded latency through the network, $T_{unloaded}$, is 70 ns arising from 10 ns of latency through each of the three routing components in any path through the network and 10 ns of latency through each of the four chip crossings between network endpoints. If our technology projections for METRO hold, we could implement a version with RN1-style pipelining and cut this latency in half. Alternately, if we could cycle the pipelined memory bus twice as fast or increase the memory width to 128-bits and require 16-byte data transfers, we could support 200 MB/s network connections using the METRO router at full speed. This would cut the unloaded network latency in half to 35 ns. This change would also double the bandwidth figures above and cut the transmission time, $T_{transmit}$, in half. For this size of a network, the total time to communicate a message from one node to another, $T_{message}$, will be dominated by the network input and output latency, T_p and T_w and the transmission latency, $T_{transmit}$.

11. Metro Link

METRO Link (MLINK) is a network interface designed to connect the processor and memory on an MBTA node to a METRO network. MLINK handles the core portions of MRP-ENDPOINT (See Section 4.7) and provides support so the node processor can handle the remainder.

11.1 MLINK Function

MLINK performs all of the low-level operations necessary for an endpoint to send and receive data over a METRO network. MLINK handles control and signalling which must operate at the network speed. It also handles those operations which must be implemented in hardware to exploit the full bandwidth of the network ports and keep end-to-end network latency low. MLINK leaves infrequent diagnostic operations, certain kinds of message formatting, and policy decisions to the node processor.

MLINK's primary function is to convey data between the network and a node's memory. MLINK moves data between the double-word wide memory bus, on which it gets one cycle once every 80 ns, and the byte-wide network port operating at 100 MHz. MLINK adds control bytes to the data stream (*e.g.* ROUTE, TURN, DROP) to open, reverse, and close network connections. MLINK also generates and verifies the end-to-end message checksums used to guard message transmission. MLINK will retry failed connections without processor intervention up to some processor specified number of trials. A pair of MLINK network inputs will arbitrate using randomization to determine which input is used for each connection trial. MLINK network outputs can handle the reception of a small set of primitive messages (See Section 11.3) without processor intervention. For all other messages, MLINK queues the incoming data to be handled by the processor

Operations which are more complicated and infrequent are left to the node processor. The processor is responsible for packet launch and for source-queuing of messages while the network input is busy. The processor determines how to proceed when MLINK fails to deliver a message in the specified number of trials. The processor is also responsible for allocating space for incoming remote function invocation messages and for processing and dequeuing the messages as they arrive.

When configured to do so, MLINK will store connection status information for successful and failed connections. This information includes the status and checksum words returned from each router in an allocated path. MLINK leaves the task of interpreting this information to the processor.

A few tasks are also left to the processor in order to limit the details MLINK needs to know about the message protocol or attached network. The remote function invocation provides an efficient and flexible opportunity to customize low-overhead messages for a particular application. MLINK only provides basic transport and queuing of these message types, leaving formatting and interpretation to the processor. MLINK also leaves the selection of routing words to the processor. This allows the processor to select a particular path in extra-stage, multipath networks (See Section 4.7.1) and prevents MLINK from needing to know the details of formatting routing words for any particular network (See Section 4.6).

11.2 Interfaces

On the node side, MLINK connects to the 64-bit, pipelined bus. This bus serves two purposes for each MLINK interface. Recall from Chapter 10 that each network interface on an MBTA node has a designated cycle on the pipelined bus once every 80 ns. MLINK uses this slot to read or write data from the fast memory at 100MB/s. The processor also has a designated slot on the pipelined bus. During the processor's slot, it may read or write 32-bit values at memory-mapped MLINK addresses. These memory-mapped addresses allow the processor to:

1. configure each MLINK
2. launch or abort network operations
3. check on the status of each MLINK's ongoing or recently completed operations

On the network side, MLINK has a byte-wide network port which behaves like a METRO forward or backward port. The network port has the same set of configuration options as each METRO forward and backward port (See Table 9.2).

11.3 Primitive Network Operations

MLINK distinguishes five kinds of primitive network operations:

1. READ
2. WRITE
3. RESET
4. NOOP
5. ROP (remote function invocation)

The RESET, NOOP, READ, and WRITE operations are handled entirely by the receiving MLINK without involving the processor, whereas the remote function invocation is only queued by MLINK to be handled by the node processor. The READ operation performs a multi-word, memory read operation on the remote node, returning the data at the specified address to the source. The WRITE operation performs the complementary function, allowing data to be written into a remote node's memory. These operations are direct, hardware reads and writes and are associated with no guards for coherence. The RESET operation signals MLINK to release the associated processor and allow it to boot. Combined with the WRITE operation, this primitive allows each node to be remotely configured and booted across the network. The NOOP operation performs no function on the destination node but does return connection status information which is useful during testing.

Remote function invocation is a generic primitive which allows software configuration of arbitrary message types and remote network functions. MLINK simply conveys the specified data and a distinguished address from the source endpoint to the destination via the network. The destination MLINK queues the arriving data and address on the incoming message queue for the

MLINK **Message Formats:**

(ROUTE)* $\circ$ RESET $\circ$ $(\text{DATA}_{cksum})^2$ $\circ$ TURN

(ROUTE)* $\circ$ NOOP $\circ$ $(\text{DATA}_{cksum})^2$ $\circ$ TURN

(ROUTE)* $\circ$ READ $\circ$ *len* $\circ$ $(\text{DATA}_{addr})^3$ $\circ$ $(\text{DATA}_{cksum})^2$ $\circ$ TURN

(ROUTE)* $\circ$ WRITE $\circ$ *len* $\circ$ $(\text{DATA}_{addr})^3$ $\circ$ $(\text{DATA}_{cksum})^2$ $\circ$ $(\text{DATA}_{write})^{(8 \cdot len)}$ $\circ$ $(\text{DATA}_{cksum})^2$ $\circ$ TURN

(ROUTE)* $\circ$ ROP $\circ$ *len* $\circ$ $(\text{DATA}_{addr})^3$ $\circ$ $(\text{DATA}_{cksum})^2$ $\circ$ $(\text{DATA})^{(8 \cdot len)}$ $\circ$ $(\text{DATA}_{cksum})^2$ $\circ$ TURN

MLINK **Reply Formats:**

$(\text{DATA}_{mlink\_status})^2$ $\circ$ ACK/NACK $\circ$ DROP

$(\text{DATA}_{mlink\_status})^2$ $\circ$ (DATA_IDLE)* $\circ$ $(\text{DATA}_{read})^{(8 \cdot len)}$ $\circ$ $(\text{DATA}_{cksum})^2$ $\circ$ DROP

Each primitive message type has its own initial message format. Where determined by MLINK, superscripts indicate the number of bytes composing each portion of the message data. The read operation is the only primitive message which receives data along with its reply message.

Figure 11.1: MLINK Message Formats

destination processor to service. The destination processor dequeues each message and invokes the function at the specified address with the associated data. [E+92] calls this kind of low-overhead, remote code invocation an *Active Message*. This primitive exports the basic functionality of the network to the software level where custom message handlers can be crafted in software for each application or run-time system.

Figure 11.1 summarizes the message formats used by MLINK network interfaces to perform the primitive network operations. Where appropriate, the target address and data length are guarded with their own checksum so that data can be written into memory while it is being received (See Section 4.7.4). In reply to a WRITE, RESET, NOOP, or ROP message, MLINK sends status information and an acknowledgement. When replying to an uncorrupted READ operation, MLINK returns the data associated with the read. The DATA_IDLE words preceding the read data in the read reply message are used to fill in any delays before the first byte of read data is available. This delay arises partially from the need to wait for the first read data on the node data bus and partially from the need to fill pipeline stages within MLINK with reply data.

These primitives form a minimal set of network primitives. They provide a high-degree of flexibility for a general-purpose multiprocessor. In situations where the application and programming model are limited or biased to a particular domain, it may make sense to customize a network interface with additional network primitives implemented directly in hardware without the processor's intervention. For instance, when building a dedicated, shared-memory machine with a particular memory-model in mind, it would be beneficial to provide primitive network operations to handle coherent memory transactions.

12. MBTA Packaging

In Section 7.4, we showed how to package a network using the stack packaging technology introduced in Chapter 7. Here, we consider packaging an entire 64-processor MBTA machine.

12.1 Network Packaging

Packaging the network is a simple application of the network to stack packaging mapping introduced in Section 7.4. As shown in Figure 10.1, a 64-processor network built out of RN1 components, or comparably sized METRO components, has 16 routers in each stage. We arrange these routers in a 4×4 grid arrangement as shown in Figure 12.1. With the routers packaged in DSPGA packages and placed on 3 inch centers, each routing board is roughly 12 inches square.

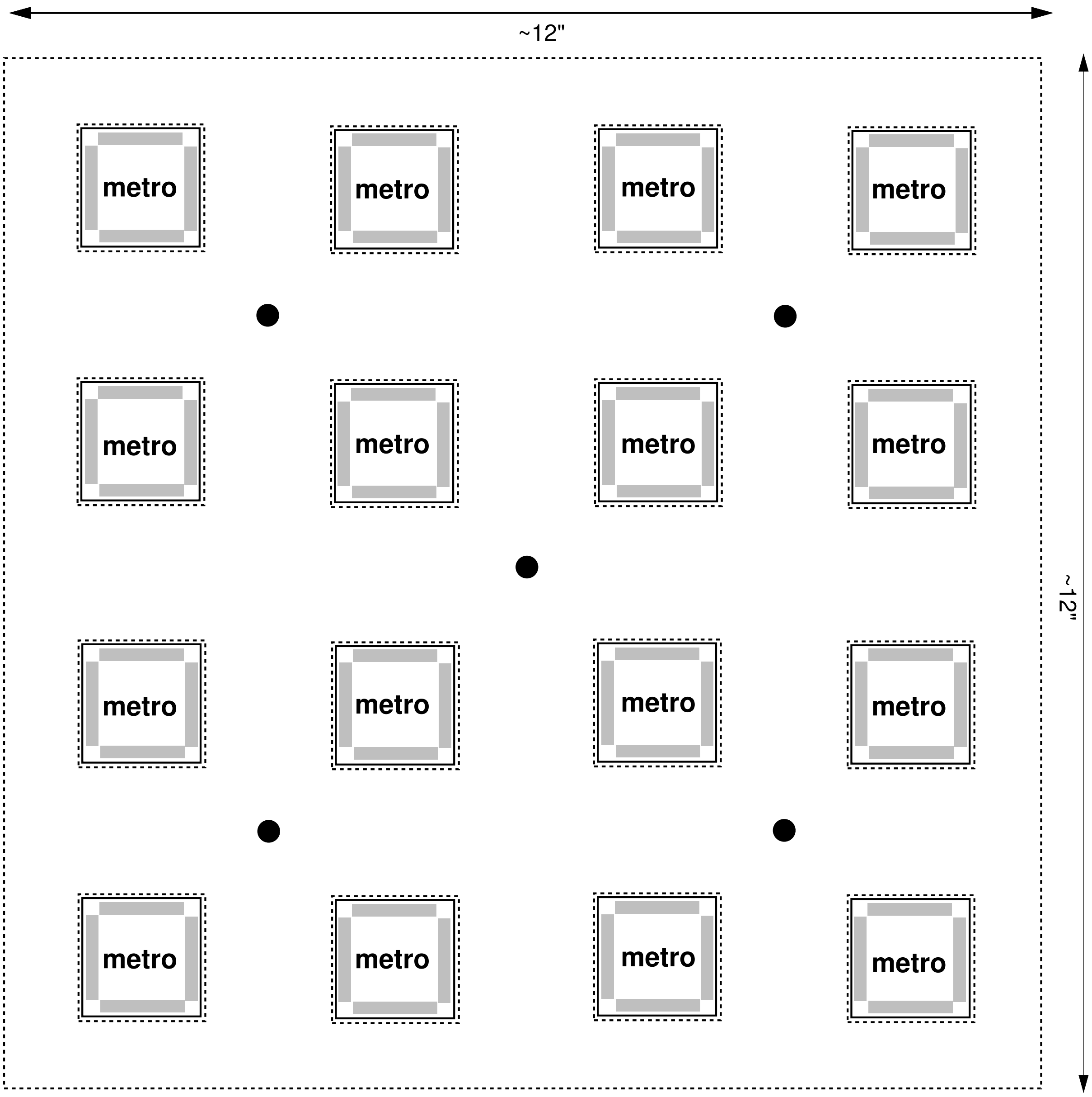
12.2 Node Packaging

We can package the nodes inside the same stack by housing the larger, VLSI components in DSPGA packages and using gull-wing surface-mount components for memory and bus logic. By sharing the i/o pads associated with the 64-bit, node data bus, we can integrate all four logical network interfaces on a single die and place the die in a DSPGA package. The processor and custom bus control logic can each be placed in their own DSPGA package. The memory can be obtained in gull-wing, surface-mount packages. The bus interface logic can be packaged in SSOP packages with a 25 mil pad pitch [Tex91]. By adding a fourth DSPGA package, we can package a node on a 6 inch square PCB with the DSPGA components centered 3 inches apart. The memory and glue logic can be placed on the surface of the node PCB between the DSPGA packages as shown in Figure 12.2. The fourth DSPGA package can be used either to house additional node logic or as a blank for mechanical support and vertical signal continuity. This arrangement allows us to stack four node PCBs on top of the network routing boards and align the DSPGA packages in the network and nodes (See Figures 12.3 and 12.1).

To accommodate additional logic or memory for each node, we can build daughter boards of the same size and use vertical connectivity to interconnect the boards. As long as the signals which connect to the additional logic or memory are available on the pads of one of the four DSPGA packages on the node, an adjacent PCB has access to these signals. A DRAM memory card, for example, could be built by housing the DRAM controller in a DSPGA package, and packaging the DRAM in TSOP packages between the DSPGA component sites. Blank DSPGA packages will be necessary in any unused DSPGA grid sites.

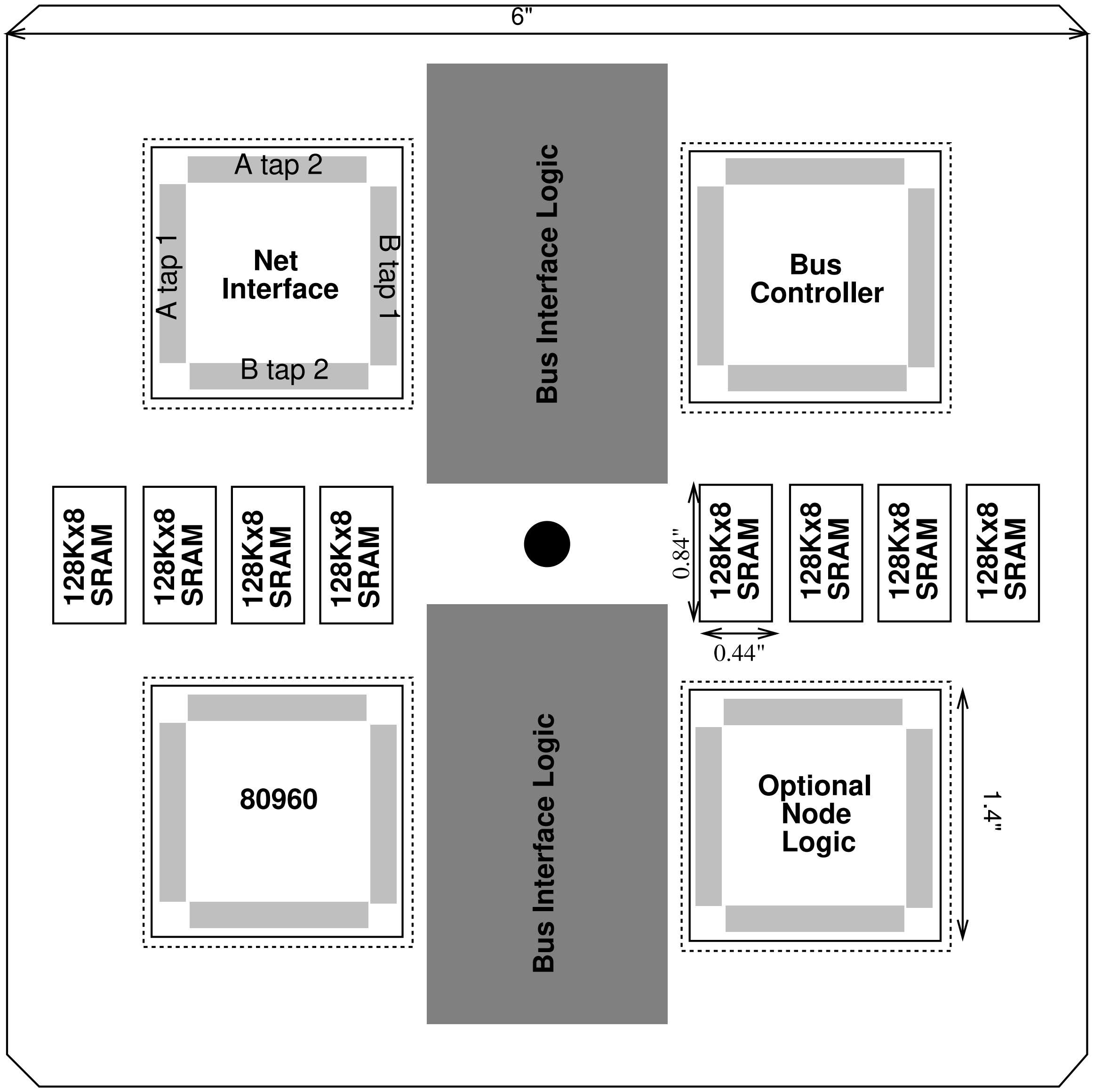
12.3 Signal Connectivity

Each node needs to be connected to two network input channels and two network output channels. We use the through vias on the DSPGA packages to vertically connect each node into



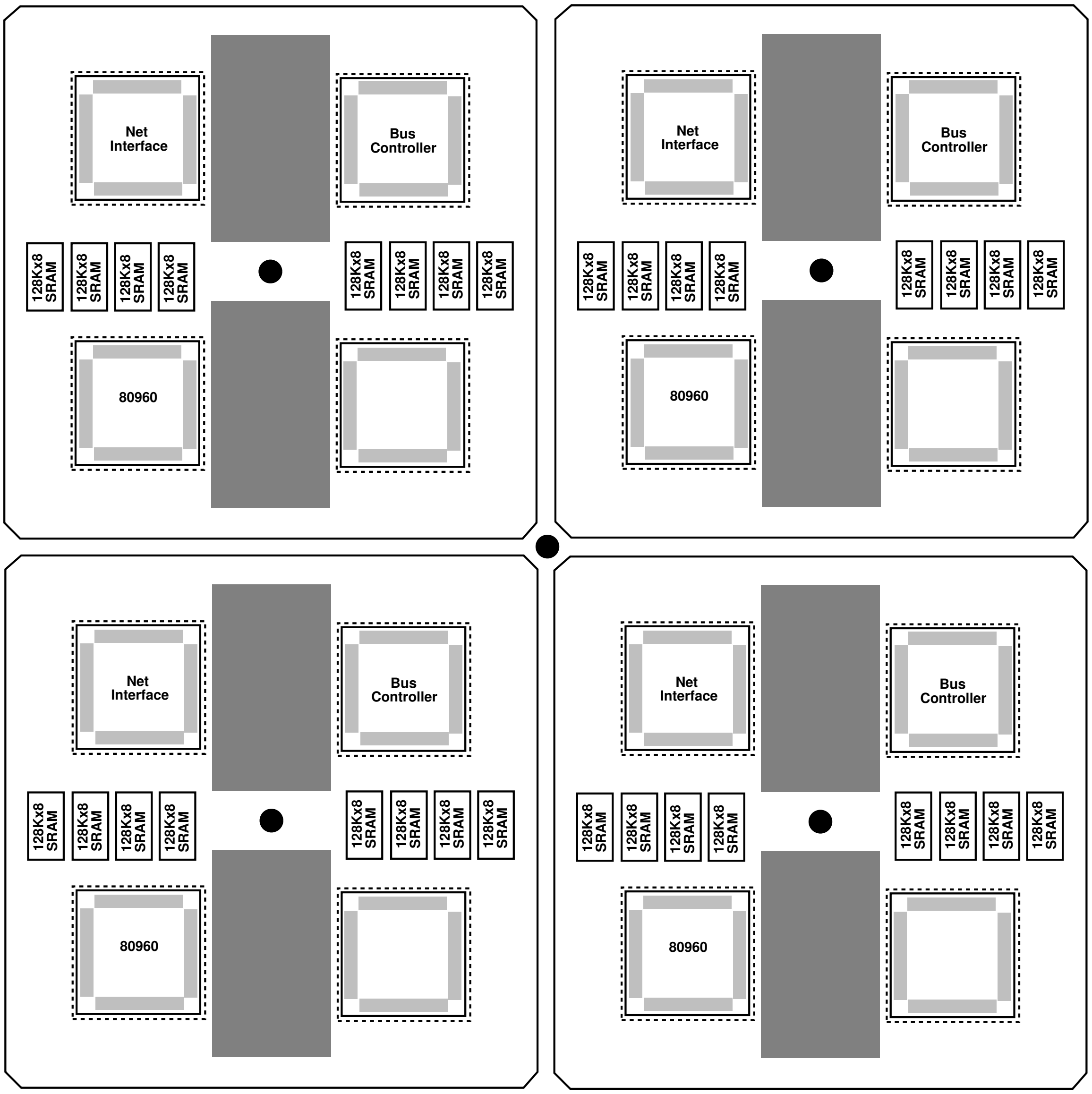
The 16 routers in each stage of the network (See Figure 10.1) are arranged in a 4×4 grid. The routers are housed in DSPGA packages and spaced 3 inches apart.

Figure 12.1: Routing Board Arrangement for 64-processor Machine



By housing the processor, network interface, and bus control logic in DSPGA packages, we can construct a 6 inch square node suitable for stack packaging. The fourth DSPGA can be blank or house additional logic, such as an optional co-processor. Memory and bus interface logic are housed in gull-wing surface-mount packages in the space between the DSPGA packages.

Figure 12.2: Packaged MBTA Node



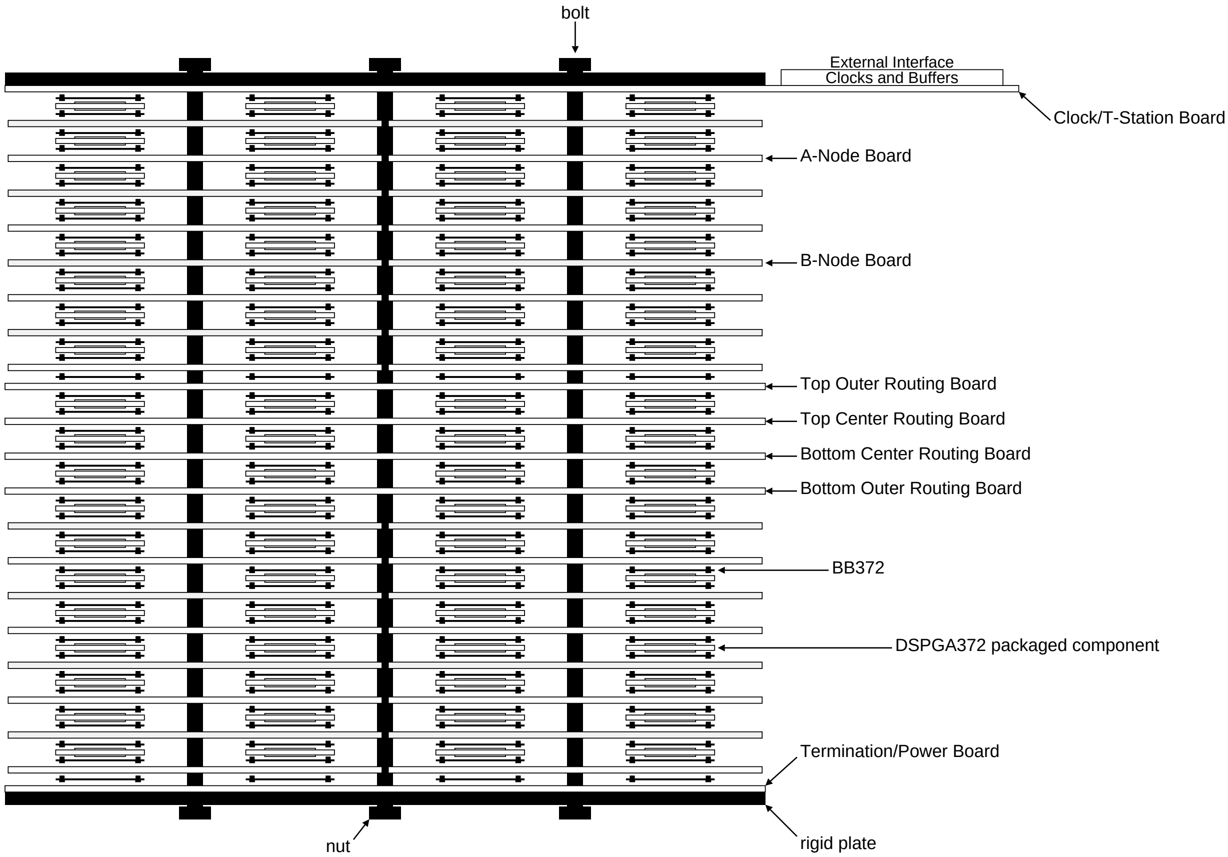
Four nodes can be arranged in one 12 inch square stack layer which mates mechanically and electrically with the routing component layers (Figure 12.1).

Figure 12.3: Layer of Packaged Nodes

the network layers. As shown in Figure 12.3, we can place four nodes in each stack layer above, or below, the group of three network boards. To house 64 nodes, we need $\frac{64}{4} = 16$ such layers of nodes. We can place half of the nodes above and half below the network layer to minimize the distance of any node from the network. This segregation leaves us with eight layers of nodes on each side of the network. The vertical through signals on each node must run network connections for the eight nodes in each node column on each side of the network. Each of the eight nodes taps off the appropriate subset of these signals to connect into the network. Mechanically, the node arrangement described has a rotational symmetry of four. With proper signal arrangement, we can exploit this symmetry to allow a single node PCB design to tap into any of four different vertical signal runs. We can tap into the eight different vertical signal runs using only two different basic node designs. In the network layers, the vertical through interconnect will be used to arrange the network inputs and outputs so that half of the inputs and half of the outputs are available on each side of the network.

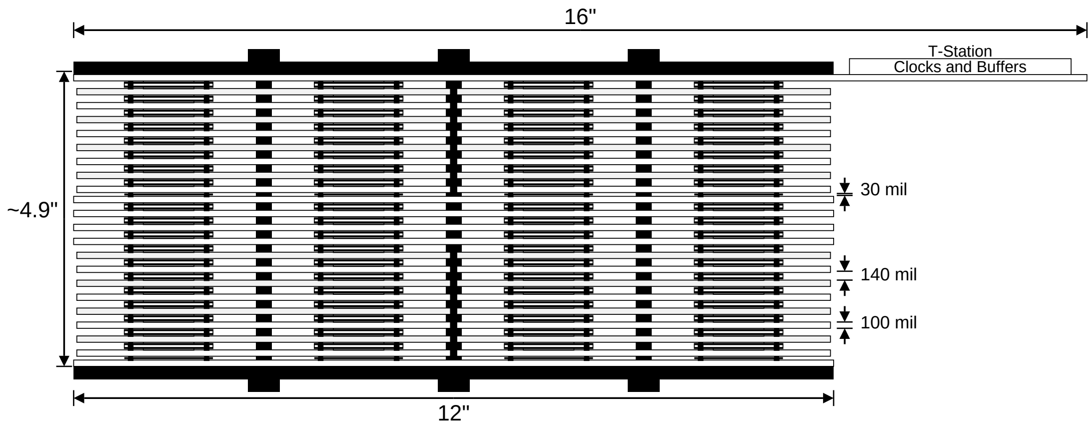
12.4 Assembled Stack

Figure 12.4 shows an exploded view of the packaged 64-processor machine. External interfaces mate with the nodes in the top-most node layer using the same vertical interconnect scheme suggested for node daughter boards. The complete stack houses the network and all 64 nodes in a cubic structure roughly $12'' \times 12'' \times 5''$ (See Figure 12.5).



A complete 64-processor machine stack is composed of 3 network layers (Figure 12.1) and 16 node layers (Figure 12.3). Two different node PCB designs coupled with the rotational symmetry of the node PCBs, allow each of the eight nodes in a vertical column to tap into different network connections.

Figure 12.4: Exploded Side View of 64-processor Machine Stack



Scale Drawing

Figure 12.5: Side View of 64-processor Machine Stack

Part IV

Conclusion

We have examined the latency and fault tolerance associated with large, multiprocessor networks. We developed techniques at many levels for building low-latency networks. We also developed networks capable of sustaining faults and techniques allowing proper operation of these networks in the presence of faults. In the development, we found no inherent incompatibilities between our goals of low latency and fault tolerance. Rather, we found commonality between techniques which decrease latency and those which improve fault tolerance.

Consequently, we were able to identify a rich class of networks with good latency and fault-tolerant characteristics. We parameterized the networks in this class in several ways. We developed an understanding of how the network parameters effect network properties. This understanding allows us to tailor networks to meet the requirements of particular applications.

13.1 Latency Review

Combining the latency contributions from Section 2.4 and collapsing into a single equation, we get:

$$T_{net} = \gamma(\text{application, topology}) \cdot \left(s_n \cdot (t_{io} + t_{switch}) + \Sigma_i \left\lceil \frac{\frac{d_i}{\frac{1}{\sqrt{\mu\epsilon}}}}{t_c} \right\rceil \cdot t_c \right) + \left\lceil \frac{L}{w} \right\rceil \cdot t_c \quad (13.1)$$

We see that there are many aspects which contribute to network latency. To achieve low latency, we must pay attention to all potential latency contributors and work to simultaneously minimize their effects. In Chapters 3 through 7, we addressed all of these latency components and examined ways to minimize their contributions.

We considered how to minimize the transit time between routers (T_t).

$$T_t = \Sigma_i \left\lceil \frac{\frac{d_i}{v}}{t_c} \right\rceil \cdot t_c \quad (13.2)$$

This latency is determined by the speed of propagations, v, and the total distance traversed, $d = \Sigma_i d_i$. We saw that the maximum speed of signal propagation was determined by material properties.

$$v = \frac{1}{\sqrt{\mu\epsilon}}$$

We also saw that this maximum was only achievable with proper signal termination. In Chapter 6, we saw signalling techniques for achieving this maximum rate of propagation with minimal power dissipation. We saw that the traversed interconnect distance, d, depends on the growth characteristics of the network topology and the achievable packaging density. In Chapter 3, we looked at the interconnect distance growth characteristics for a large class of networks and identified those with the most favorable growth characteristics. In Section 2.4.2, we noted that, in some situations, locality can be exploited to minimize, on average, the distances which must be traveled inside the network. Consequently, in Chapter 3 we also identified network topologies with locality. In Chapter 7, we looked at technologies for high-density packaging. We also looked at topologies for mapping networks onto the packaging technology in a way that exploits the density to minimize interconnection distances.

We considered how to minimize the total number of routers which must be traversed in a network, s_n. In Chapter 3, we found that log structured sorting networks gave us the lowest number of switches as long as we restricted ourselves to bounded degree switching nodes (Section 2.7.1). We also noted that the router radix, r, gives us a parameter we can use to control the actual number of switches traversed in an implementation. Again, for some applications locality exploitation may allow us to further reduce the average number of switches traversed when routing through the network.

We noted that the latency contributed by each routing component was composed from the switching time and the i/o latency.

$$t_{nl} = t_{io} + t_{switch} \tag{13.3}$$

In Chapter 6, we identified a signalling discipline which minimized transit and chip i/o latencies. We looked at technologies for implementing CMOS drivers and receivers for this signalling discipline and saw how to design circuitry for realizing low-latency i/o. In Chapter 4, we developed a simple routing protocol that was well matched to the capabilities of our CMOS IC implementation technologies. The routing scheme combines simple, local decision making with a minimum complexity routing protocol to allow the switch to perform all of its functions quickly.

To keep the contention latency, γ, and the transmission time, $T_{transmit}$, low, we looked at how to provide high bandwidth in these networks. We saw that increasing the bandwidth available for of each connection will decrease the transmission latency.

$$T_{transmit} = \left\lceil \frac{L}{w} \right\rceil \cdot t_c \tag{13.4}$$

We can increase this bandwidth either by increasing the signalling rate, $\frac{1}{t_c}$, or by increasing the data channel width, w. We can also note that the lower the transmission latency, $T_{transmit}$, the faster the resources used by a connection are freed. As a result, decreasing transmission latency will also decrease contention latency.

We noticed that we can often reliably send data faster than the data can traverse wires or routing components. As a result, we saw that pipelining the transmission of data often allows us to decrease the signalling clock, t_c, considerably, and hence increase bandwidth, without any negative impact on latency. To this end, we showed how the routing protocol can accommodate pipelining of data across wires and inside routers (Section 4.11). In Chapter 6, we also saw how the i/o circuitry and signalling discipline allow us to reliably pipeline bits across wires of arbitrary length.

Further, we saw that contention latency arises from inadequate or improperly utilized resources inside the network. We saw in Chapter 3 that dilated routers gave connections a choice of resources to utilize throughout the network. This freedom reduced the likelihood that blocking will occur within the network and hence reduced contention latency. In Section 4.9.2, we saw how fast path collapsing reduced contention latency further by quickly reclaiming resources allocated to blocked connections.

In Chapter 3, we also saw that we have several options for reducing contention latency:

1. We can increase the per connection bandwidth by increasing the channel width or signalling frequency, as described above.

2. We can also increase the aggregate bandwidth of the machine by increasing the number of input and output connections between each node and the network, ni and no.

3. We can decrease the likelihood of blocking by increasing the dilation, d, so that each connection has more options at each routing stage.

13.2 Fault Tolerance Review

In Section 2.5, we saw that we cannot depend on the correct operation of every component in the network if we need to achieve reasonable MTTF for large-scale multiprocessor networks. We also saw that the ability to operate in the presence of even a small number of faulty components improves our system reliability, considerably. This observation led us to look for networks in which we could maximize the distinct resources available to make any connections and hence to minimize the likelihood that any set of faults will render the network disfunctional.

In Section 2.1.1, we noted that transient faults were much more likely than permanent faults. This fact, coupled with the single-component fault rate derived in Section 2.5, led us to be concerned with robust operation in the face of dynamically arising faults. We found that we must devise protocols which do not assume the correct operation of any component in the network at any point in time. Rather, we must arrange the protocol to verify the integrity of each network operation.

In Section 3.3 we examined multipath networks and noted their potential for providing fault tolerance. In these networks, the multiple paths between endpoints use different routing resources. These alternate routing resources provide the basis for fault-tolerant operation. When a faulty component renders one path inoperative, another path is available which avoids the faulty component. In Section 3.5 we examined many of the detailed wiring issues associated with multipath, multistage networks. We saw that the number of connections to each endpoint, ni and no, is the weakest link between a node and a multipath network, and we saw how to make the best use of the endpoint connections available in a particular network. We also visited the issue of wiring the multiple paths inside the network to maximize fault tolerance. We saw different evaluation criteria based on whether network connectivity is viewed as a yield problem or as a harvest problem. If we allow node isolation, we saw that randomly-wired networks generally behave most robustly in the face of faults. When node isolation is not permitted, we found that deterministic, maximum-fanout networks generally survive more faults.

We also saw that we can control the amount of redundancy, and hence fault tolerance in these networks, by selecting the router dilation, d, and the number of node input and output connections, ni and no. We can adjust ni and no to control the mean time to node isolation. For the harvest case, where node isolation is not allowed, increasing the number of node inputs and outputs is probably the most effective way of increasing fault tolerance. We can adjust the dilation to control the amount of path fanout within the network and hence the number of paths provided between endpoints. Increasing the dilation is effective for increasing fault tolerance in both the the harvest and yield situations. However, due to the different reliability metrics we use in these two cases, increasing the dilation is much more effective in the yield case than in the harvest case.

Multipath network topology, however, only gave us the potential for fault-tolerant operation. To realize that potential, we noted that the routing scheme must be able to detect when failures occur and be able to exploit the multiple paths to avoid faults. In Chapter 4, we developed such a scheme for routing on the multistage, multipath networks detailed in Chapter 3. End-to-end message checksums guard each data transmission against unnoticed corruption. End-to-end acknowledgments and source-responsible retry work together to guarantee each message is delivered at least once without corruption. Random selection of a particular path through the multipath network coupled with source-responsible retry, guarantees that any non-faulty path between any source-destination pair can eventually be found. Combining these features, the routing protocol achieves correct operation without requiring any knowledge of the faults within the network.

We saw that we could minimize the performance impact of faulty components and interconnect on the network by identifying them and masking them from the network. A known, masked fault is deterministically avoided. This avoidance allows the random path selection to converge more quickly on a good path by removing all known bad paths from the space of potential paths. We also saw that identifying faults allows us to make assessments about the integrity of the network (Section 5.1, Section 5.7).

We developed minimally intrusive mechanisms for locating faults. The routing protocol uses the pipeline delay cycles associated with reversing the direction of data flow across the network to transmit router checksums and detailed connection information back to the source. This information helps narrow down the source of any faults. Port-by-port deselection and partial-external scan (Chapter 5) allow the system to isolate regions of the network and test for faults. Since the network has redundant paths, portions of the network can be isolated and tested in this manner without interfering significantly with normal operation.

Finally, we saw that the mechanisms used for fault isolation and testing, coupled with the multiple paths within each network, provide facilities for in-operation repair. Physically replaceable subunits can be isolated, repaired, and returned to service without taking the entire network out of service (Section 5.6, Section 7.5.6). On-line repair allows us to minimize or eliminate system down-time and hence maximize system availability.

13.3 Integrated Solutions

We have described a set of techniques for building robust, low-latency multiprocessor networks. These solutions span a range of implementation levels from VLSI circuits, packaging, and interconnect up through architectures and organizations. Each technique presented is interesting in its own right for the features and benefits it offers. However, the collection of techniques presented here is most interesting because the techniques integrate smoothly into a complete system. When assembled, we do get a system which reaps the cumulative benefits offered by all of the techniques. The features of many of the techniques compliment each other such that the overall features and benefits of the composite system are greater than the features of the individual pieces.

A. Performance Simulations[1]

In this appendix, we will describe the simulations used to measure network performance. We will begin by describing some features of the basic architecture modeled. We revist some practical issues of network construction involving the inputs and outputs of the network. Finally, we develop methods for exercising networks based on representative network loads taken from shared memory applications.

A.1 The Simulated Architecture

The networks simulated use a circuit-switched routing component based upon RN1 (See Chapter 8) and Metro (See Chapter 9). The particular component used throughout these experiments can act either as a single 8-input, radix-4, dilation-2 router, or as two independent 4-input, radix-4, dilation-1 routers.

To aid the routing of messages, each component will have a pin dedicated to calculating flow control information according to the following blocking criterion taken from Leighton and Maggs in [LM92]. A router is *blocked* if it does not have at least one unused, operational output port in each logical direction which leads to a router which is not blocked. To route a message, a router attempts to choose a single output port by first looking at unused ports, and second eliminating any ports which are blocked. If no unique choice arises — all ports unused, but all unblocked or all blocked — then the router randomly decides between ports.

Each chip also incorporates a serial test-access port (TAP) which accesses. These ports, in turn, are connected together in a diagnostic network which can provide in-operation diagnostics and chip reconfiguration as detailed in Chapter 5.

A.2 Coping with Network I/O

Much of the fault tolerance and routing behavior of our networks is dominated by the first and last stages. Although multipath networks provide multiple paths between any two nodes, these paths can only use a large number of physically distinct routers towards the middle of the network. Near the nodes, these paths must concentrate towards their specific sources and destinations. This concentration is most severe in the first and last stages, where each node only has a small number of connections to the network (See Section 3.5.2). For the sake of these simulations we assume dilation one routing components are used in the final stage of the networks (*e.g.* Figure 3.11).

[1] This information is reprinted with slight modification from [Cho92]

A.3 Network Loading

In this section, we derive network loads for use with our performance simulations. We need to run a large number of simulations to obtain average performance of each network at various fault levels. Consequently, we use simple synthetic loadings to keep simulation time manageable. We start by using uniformly distributed random destinations for our messages. We refine our model by looking at shared-memory applications studied by the MIT Alewife Project [CFKA90].

A.3.1 Modeling Shared-Memory Applications

Our goal is to provide a realistic model of network utilization that can be used to compare many different networks and parameters. To keep simulation time tractable, we use a variant of uniform traffic. Our simulation sends messages to random destinations in a uniform distribution. However, message lengths are randomly generated according to distributions derived from specific parallel applications. These application were taken from caching studies done by the Alewife Project [CFKA90]. These studies simulate a shared memory architecture with coherent caches at each processing node. Data taken from this study corresponds to the following system parameters:

- Shared memory, coherent caches
- Full-map directories
- 16-byte cache lines
- 64 nodes, corresponding to 3-stage, radix-4 networks.
- Single thread
- CISC instructions
- 1 memory reference per instruction
- Processors stall after 1 outstanding memory reference
- Barrier synchronization

A.3.2 Application Descriptions

[CFKA90] studied four applications: SIMPLE, SPEECH, FFT, and WEATHER. SIMPLE models the hydrodynamic behavior of fluids using finite difference methods to solve the equations in two dimensions. SPEECH is the lexical decoding stage of a phonetically-based spoken language understanding system. It uses a variant of the Viterbi search algorithm. FFT is a radix-2 Fast Fourier Transform. WEATHER uses finite-difference methods to solve partial differential equations which model the atmosphere around the globe.

These applications attempt to represent three major classes of problems: graph problems, continuum problems, and particle problems. Graph problems involve searching and irregular communication. Continuum problems generally have localized communication in regular patterns. Particle problems often involve communication over long distances to simulate interactions such as those due to gravitational forces.

A.3.3 Application Data

Cache-coherent shared memory systems utilize a small set of message types, each of a fixed length. The messages sent through the network read cache lines, write cache lines, and maintain cache coherency. A cache-line read, for example, requires an 8-byte read request followed by a reply containing the data. The reply consists of an 8-byte header followed by 16 bytes representing the desired cache line.

Table A.1 lists the frequency of all transactions for our four applications. The messages sent for each transaction are also listed. In contrast to the Alewife study, it is important for us to distinguish which transactions are split phase. Our networks are circuit-switched and can save routing time if a reply to a request is immediately available. However, if the transaction must be split into two phases, two messages will have to be routed separately. Table A.1 distinguishes those messages which are single phase, always split phase, and sometimes split phase. Table A.2 gives the percentage split phase for those which are sometimes split phase. Assuming a two router cycles per processor cycle, our data gives us a 3, 6, 7, and 9 percent approximate message generation rate per router cycle for WEATHER, SIMPLE, SPEECH, and FFT, respectively.

Table A.2 also gives the approximate grain sizes of each application. These figures will be used to determine frequency of barrier synchronization, to be discussed in Section A.3.4. Finally, Table A.3 gives the relative frequency of each length of message for each application. This is summarized by the average length of messages given for each application.

A.3.4 Synchronization

Our performance simulation includes barrier synchronization to eliminate *skew*. Our simulation models applications which assume some degree of synchronization between the multiple processor nodes of the system. When a simulation violates this assumption, results are skewed. We prevent this skew by performing periodic barrier synchronization according to grain sizes estimated for each application.

It is important to eliminate simulation skew because it can mask the effects of localized network degradation. Analytic models suggest that faults and congestion may severely affect the performance observed by specific destination nodes while leaving others largely unaffected [KR89]. Without synchronization, such localized degradation would be lost in the *average* I/O bandwidth utilization. Modeling synchronization, however, forces all processors to wait for those falling behind, resulting in a more realistic decrease in I/O bandwidth utilization.

A.3.5 The FLAT24 Load

We also simulate a uniform message distribution, FLAT24. FLAT24 uses 24-byte messages and other simulation parameters which are similar to the messages and parameters of SIMPLE and SPEECH. FLAT24 serves as a basis for network comparison.

For our later studies, we shall also use FLAT24, but we shall use parameters which differ slightly from those in the Alewife study and correspond more closely to our target architectures.

To derive the frequency of message loading, several "reasonable" parameter values were chosen. Note that our results are not overly sensitive to loading, so rough figures are adequate. The program code for each processor is assumed to be resident in local memory. Consequently, only data

Message Type	WEATHER	SIMPLE	SPEECH	FFT
read miss, write mode [hdr,hdr+data] (hdr,hdr+data)	0.4400	0.4500	1.8700	0.9600
read miss, not write mode (hdr,hdr+data)	0.8400	4.2400	1.3500	1.7500
read miss, not in any cache (hdr,hdr+data)	0.6700	0.7700	0.0800	0.1300
write miss, write mode [hdr,hdr+data] (hdr,hdr+data)	0.6400	0.6300	0.0100	2.5100
write miss, not write mode ?hdr,hdr+data?	0.0000	0.1900	0.0000	0.1000
write hit, not write mode ?hdr,hdr?	0.5500	0.4500	1.8500	0.9900
write miss, not in any cache (hdr,hdr+data)	0.3300	0.3000	0.1500	0.0000
instruction miss (hdr,hdr+data)	0.0700	0.1900	0.0000	0.2700
private misses (hdr,hdr+data)	0.1159	0.1038	0.0013	0.1821
invalidations (hdr,hdr)	0.4318	1.2562	2.7455	2.8004
evictions (hdr,hdr)	0.0000	0.0000	0.0000	0.0000
replacements of dirty data (hdr+data)	0.0407	0.4512	0.0090	0.0196
synchronizations not cached (hdr,hdr+data)	0.0000	0.0000	0.0000	0.0000

hdr = packet header, data = cache line

[...] = split phase message combination

(...) = single phase

?...? = sometimes split phase

Relative transaction frequencies for each of our four applications, in transactions per processor cycle, are given above. Messages sent for each kind of transaction are also given. (Data Courtesy of David Chaiken)

Table A.1: Relative Transaction Frequencies for Shared-Memory Applications

	WEATHER	SIMPLE	SPEECH	FFT
percent split phase	85.5560	91.7210	100.0000	88.8396
grain size	59769($\frac{1}{2}$) 187714($\frac{1}{2}$)	7271	1000 to 10000	28289

For each application, the percent split phase is given for those listed as sometimes split phase (?. . . ?) in Table A.1. Approximate grain sizes are also given for each application. There are two grain sizes for WEATHER, one for each of two phases in the application.

Table A.2: Split Phase Transactions and Grain Sizes for Shared-Memory Applications

	WEATHER	SIMPLE	SPEECH	FFT
8-byte	2.9211	2.0798	5.5800	5.3179
16-byte	0.5112	1.2936	2.7455	2.9109
24-byte	1.1207	1.7055	1.8890	3.5784
32-byte	2.4359	5.9295	3.3813	5.6832
Average Length	21.2178	24.3463	17.8074	20.4033

Relative frequencies of each length of message are given. These are summarized by the average length of messages for each application.

Table A.3: Message Lengths for Shared-Memory Applications

references will result in non-local memory references. We assumed a data cache miss rate of 15 percent. For each data read or write, we get 0.15 misses per processor cycle. We assumed that 50 percent of the references are to local memory, which gives us 0.075 references to non-local memory per processor cycle. With a 50 MHz processor and the RN1 part running at better than 100 MHz, we have two router cycles per processor cycle. This gives us 0.0375 non-local memory references per router cycle. Adding an additional 10 percent to account for cache coherency messages, we end up with approximately 0.04 messages per router cycle.

We also examine time between synchronizations, or, equivalently, application grain size. A grain size representative of applications studied is 10,000 cycles, or $10,000 \times 0.04 = 400$ messages. Altogether, our performance metric is the time to route the following task: all processors in the system must each send 400 24-byte messages at a rate of 0.08 messages per active processor cycle. We assume that each processor can have up to 4 threads, or tasks, each with an outstanding message, before stalling.

Note that our task explicitly models barrier synchronization. Other styles of synchronization may involve smaller processor groups, but may also tend to synchronize these groups more often. In any case, it is important to model the synchronization requirements of an application. For our purposes, barrier synchronization incorporates an appropriate component of these requirements into our performance metric. We see that synchronization plays a major role in performance degradation. This degradation occurs when network failure results in a small number of nodes with particularly poor communication bandwidth.

Let us take one more look at our numbers. Since messages are 24 bytes long, we are basically running our network at 1 byte per router cycle, or 100 percent. If the message rate were any higher, the processors would just be stalled more often, and the network loading would not really change. Note that our analysis assumes low-latency message handling, a concept demonstrated in the J-Machine [D^+92]. If, as with many commercial and research machines, there exists a high latency for message handling, the latency induces a feedback effect which prevents full utilization of the network [Joh92]. Although cache miss and message locality numbers are open to debate, we observe that the technological trends of multiple-issue processors and wider cache lines will only increase demands on the network. However, performance results presented in this paper were also verified to be qualitatively unchanged under network loading half of that used here.

A.4 Performance Results for Applications

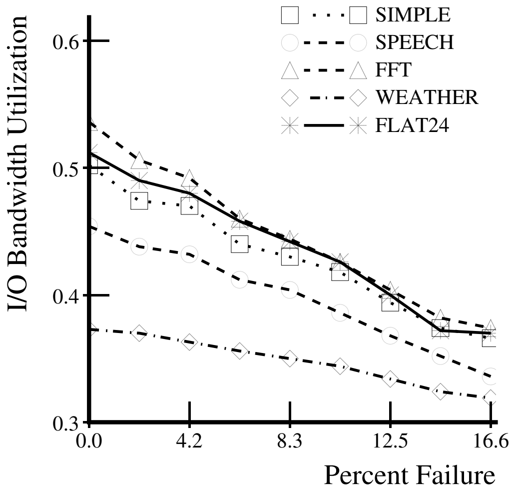
In this section we summarize our performance results for each network in the presence of faults. Performance was measured for complete networks only. For each fault level shown, multiple trials were run in which faults were randomly chosen. After fault insertion, applications were simulated on those networks which remained complete. Data shown represent the average I/O bandwidth utilization and latencies over those trials involving complete networks. These results do not represent the actual performance of these applications on hardware. Rather, our data provides a basis for network comparison by illustrating performance trends in the presence of faults.

To isolate the effects of interwiring, the deterministic and random networks presented use dilation-2 components in the last stage. We studied 3-stage non-interwired, randomly-interwired, and deterministically-interwired networks. I/O bandwidth utilization varied by less than 2 percent, a variation not significant for our simulation accuracy.

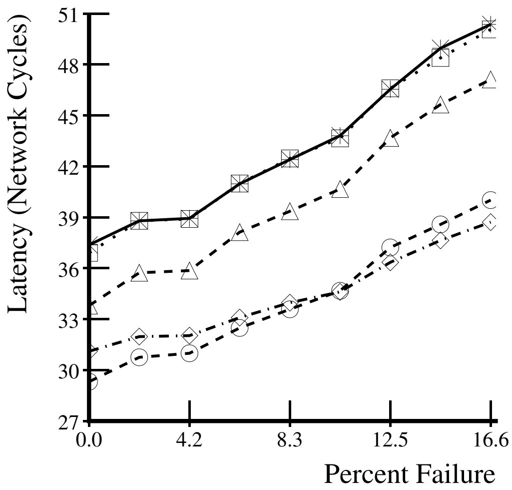
However, to avoid single-point disconnections in the last stage, the interwired networks we shall analyze for fault performance are constructed from dilation-1 components in the last stage. Although dilated components provide better performance, the use of these dilation-1 components substantially increases fault tolerance (See Section 3.5.2). For applications studied on 3-stage networks, the decrease in I/O bandwidth utilization was less than 6 percent.

Figures A.1 and A.2 details the fault performance of our applications on 3-stage, radix-4, networks which can withstand faults. The application FLAT24, shown as a solid line, is representative of graph trends and will be used in network comparisons.

Figures A.3 and A.4 compares the performance of those networks which can tolerate faults. The performance of the random network is slightly better than that of the deterministic network. However, recall that our figures are for complete networks only. For each fault level shown, the random networks have a lower probability of remaining complete than the deterministic networks.



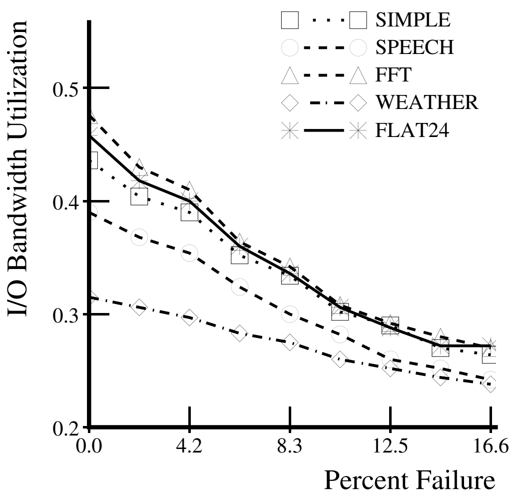
Random Network I/O



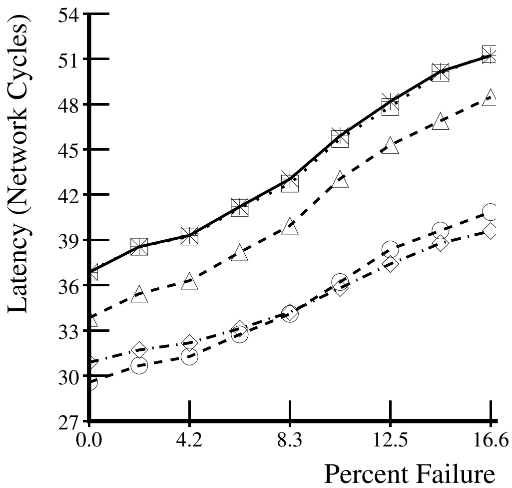
Random Network Latencies

I/O bandwidth utilization and latencies for applications on 3-stage random networks. The application FLAT24, shown as a solid line, is representative of graph trends and will be used for network comparison.

Figure A.1: Applications on 3-stage Random Networks



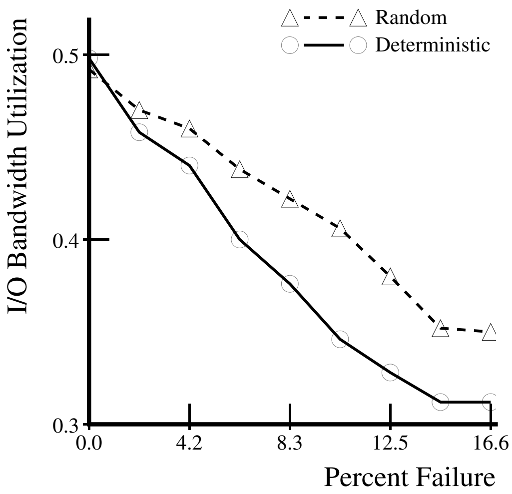
Deterministic Network I/O



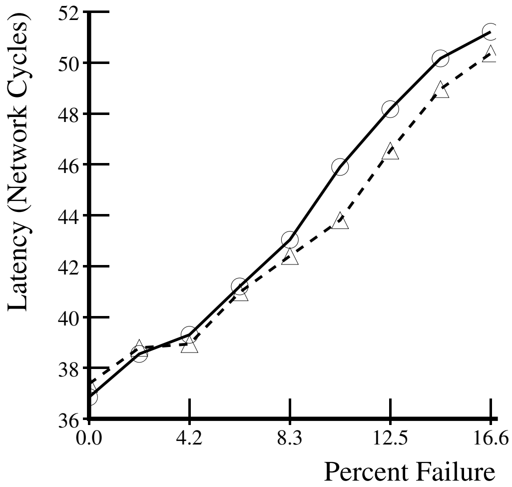
Deterministic Network Latencies

I/O bandwidth utilization and latencies for applications on 3-stage deterministic networks. The application FLAT24, shown as a solid line, is representative of graph trends and will be used for network comparison.

Figure A.2: Applications on the 3-stage Deterministic Network



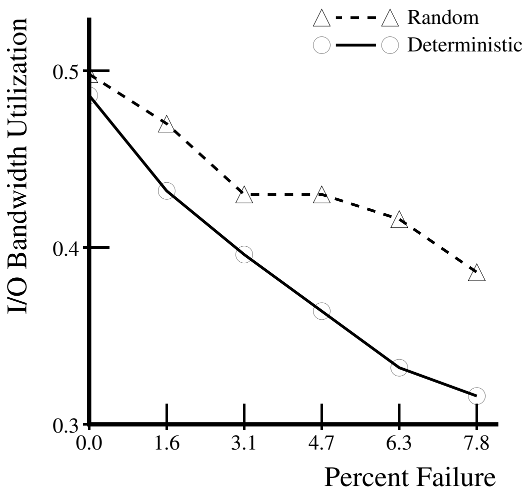
3-Stage Network I/O



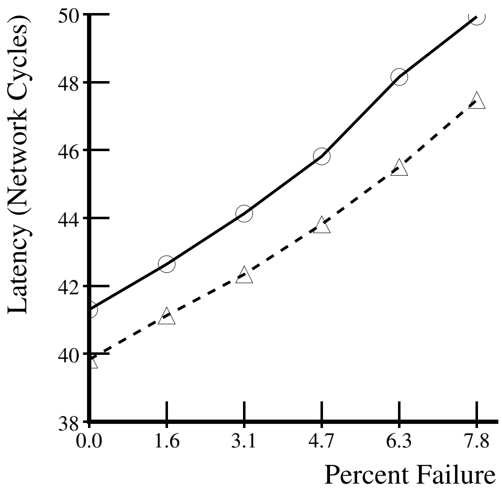
3-Stage Network Latencies

Comparative I/O bandwidth utilization and latencies for 3-stage deterministic and random networks on FLAT24. Recall from Table 3.4 that expected percentages of failure tolerated by random and deterministic networks are, respectively: 10% and 16%. Note that the performance degradation appears to level off because only complete networks are measured. Although the surviving networks suffer less degradation as percentage of failure increases, the number of surviving networks is becoming substantially smaller.

Figure A.3: Comparative Performance of 3-Stage Networks



4-Stage Network I/O



4-Stage Network Latencies

Comparative I/O bandwidth utilization and latencies for 4-stage random and deterministic networks on FLAT24. Recall from Table 3.4 that expected percentages of failure tolerated by random and deterministic networks are, respectively: 4.6%, and 8.8%. Note that the performance degradation appears to level off because only complete networks are measured. Although the surviving networks suffer less degradation as percentage of failure increases, the number of surviving networks is becoming substantially smaller.

Figure A.4: Comparative Performance of 4-Stage Networks

Bibliography

[ACD+91] Anant Agarwal, David Chaiken, Godfrey D'Souza, Kirk Johnson, David Kranz, John Kubiatowicz, Kiyoshi Kurihara, Beng-Hong Lim, Gino Maa, Dan Nussbaum, Mike Parkin, and Donald Yeung. The MIT Alewife Machine: A Large-Scale Distributed-Memory Multiprocessor. MIT/LCS/TM 454, MIT, 545 Technology Sq., Cambridge, MA 02139, November 1991.

[ACM88] Arvind, D. E. Culler, and G. K Mass. Assessing the Benefits of Fine-Grain Parallelism in Dataflow Programs. *The International Journal of Soupercomputer Applications*, 2(3), November 1988.

[AI87] Arvind and R. A. Ianucci. Two Fundamental Issues in Multiprocessing. In *Proceedings of DFVLR Conference on Parallel Processing in Science and Engineering*, pages 61–88, West Germany, June 1987.

[ALKK90] Anant Agarwal, Beng-Hong Lim, David Kranz, and John Kubiatowicz. APRIL: A Processor Architecture for Multiprocessing. In *Proceedings of the 17th International Symposium on Computer Architecture*, pages 104–114. IEEE, May 1990.

[And85] T. Anderson, editor. *Resilient Computing Systems*, volume I, chapter 10, pages 178–196. Wiley-Interscience, 1985. Chapter by: C. I. Dimmer.

[Baz85] M. Bazes. A Novel Percision MOS Synchronous Delay Line. *IEEE Journal of Solid-State Circuits*, 20(6):1265–1271, December 1985.

[BJN+86] L. A. Bergman, A. R. Johnston, R. Nixon, S. C. Esener, C. C. Guest, P. K. Yu, T. J. Drabik, and M. R. Feldman S. H. Lee. Holographic Optical Interconnects for VLSI. *Optical Engineering*, 25(10):1009–1118, October 1986.

[Bra90] Christopher W. Branson. Integrating Tester Pin Electronics. *IEEE Design and Test of Computers*, pages 4–14, April 1990.

[Buc89] Leonard Buchoff. Elastomeric Connectors for Land Grid Array Packages. *Connection Technology*, April 1989.

[CCS+88] Barbara A. Chappell, Terry I. Chappell, Stanley E. Shuster, Hermann M. Segmuller, James W. Allan, Robert L. Franch, and Phillip J. Restle. Fast CMOS ECL Receivers With 100-mV Worst-Case Sensitivity. *IEEE Journal of Solid-State Circuits*, 23(1):59–67, February 1988.

[CED92] Frederic Chong, Eran Egozy, and André DeHon. Fault Tolerance and Performance of Multipath Multistage Interconnection Networks. In Thomas F. Knight Jr. and John Savage, editors, *Advanced Research in VLSI and Parallel Systems 1992*, pages 227–242. MIT Press, March 1992.

[CFKA90] David Chaiken, Craig Fields, Kiyoshi Kurihara, and Anant Agarwal. Directory-Based Cache-Coherence in Large-Scale Multiprocessors. *IEEE Computer*, 23(6):41–58, June 1990.

[Cho92] Frederic T. Chong. Performance Issues of Fault Tolerant Multistage Interconnection Networks. Master's thesis, MIT, 545 Technology Sq., Cambridge, MA 02139, May 1992.

[CK92] Frederic T. Chong and Thomas F. Knight, Jr. Design and Performance of Multipath MIN Architectures. In *Symposium on Parallel Architectures and Algorithms*, pages 286–295, San Diego, California, June 1992. ACM.

[Com90] IEEE Standards Committee. *IEEE Standard Test Access Port and Boundary-Scan Architecture*. IEEE, 345 East 47th Street, New York, NY 10017-2394, July 1990. IEEE Std 1149.1-1990.

[Cor89] Engineering Plastics Division Hoechst Celanese Corporation. Vectra Liquid Crystal Polymer Molding Guidelines and Specifications (VC-6), 1989.

[Cor90] Rogers Corporation. Isocon Interconnection Features and Applications, 1990.

[Cor91] Intel Corporation. Paragon XP/S. Product Overview, 1991.

[CSS+91] David E. Culler, Anurag Sah, Klaus Erik Schauser, Thorsten von Eicken, and John Wawrzynek. Fine-grain Parallelism with Minimal Hardware Support: A Compiler-Controlled Threaded Abstract Machine. In *Proceedings of the Fourth International Conference on the Architectural Support for Programming Languages and Operating Systems*, April 1991.

[CYH84] Chin Chi-Yuan and Kai Hwang. Connection Principles for Multipath Packet Switching Networks. In *10th Annual Symposium on Computer Architecture*, pages 99–108, 1984.

[D+92] William J. Dally et al. The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms. *IEEE Micro*, pages 23–39, April 1992.

[Dal87] William J. Dally. *A VLSI Architecture for Concurrent Data Structures*. Kluwer Academic Publishers, 1987.

[Dal91] William J. Dally. Express Cubes: Improving the performance of *k*-ary *n*-cube interconnection networks. *IEEE Transactions on Computers*, 40(9):1016–1023, September 1991.

[DeH90] André DeHon. Fat-Tree Routing For Transit. AI Technical Report 1224, MIT Artificial Intelligence Laboratory, April 1990.

[DeH91] André DeHon. Practical Schemes for Fat-Tree Network Construction. In Carlo H. Séquin, editor, *Advanced Research in VLSI: International Conference 1991*. MIT Press, March 1991.

[DeH92] André DeHon. Scan-Based Testability for Fault-tolerant Architectures. In Duncan M. Walker and Fabrizio Lombardi, editors, *Proceedings of the IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, pages 90–99. IEEE, IEEE Computer Society Press, 1992.

[Div89] Cinch Connector Division. CIN::APSE Product Brochure, 1989.

[DKS93] André DeHon, Thomas F. Knight Jr., and Thomas Simon. Automatic Impedance Control. In *ISSCC Digest of Technical Papers*, pages 164–165. IEEE, February 1993.

[DS86] William J. Dally and Charles L. Sietz. *The Torus Routing Chip*, volume 1 of *Distributed Computing*, pages 187–196. Springer-Verlag, 1986.

[DZ83] J. D. Day and J. Zimmermann. The OSI Reference Model. *IEEE Transactions on Communications*, 71:1334–1340, December 1983.

[E$^+$92] Thorsten von Eicken et al. Active Messages: a Mechanism for Integrated Communication and Computation. In *Proceedings of the 19th Annual Symposium on Computer Architecture*, Queensland, Australia, May 1992.

[Fuj92] Fujipoly. Connector W Series, 1992.

[GC82] C. Clark Jr. George and J. B. Cain. *Error-Correction Coding for Digital Communications*. Plenum Press, New York, 1982.

[GK92] Thaddeus J. Gabara and Scott C. Knauer. Digitally Adjustable Resistors in CMOS for High-Performance Applications. *IEEE Journal of Solid-State Circuits*, 27(8):1176–1185, August 1992.

[GL85] Ronald I. Greenberg and Charles E. Leiserson. Randomized Routing on Fat-Trees. In *IEEE 26th Annual Symposium on the Foundations of Computer Science*. IEEE, November 1985.

[GM82] P. Goel and M. T. McMahon. Electronic Chip-In-Place Test. In *Proceedings 1982 International Test Conference*, pages 83–90. IEEE, 1982.

[GPM92] Dimitry Grabbe, Ryszard Prypotniewicz, and Henri Merkelo. AMPSTAR Connector Family, 1992.

[HLwn] Johan Hastad and Tom Leighton. Analysis of Backoff Protocols for Multiple Access Channels. In *Unknown*. Unknown, Unknown.

[Hui90] Joseph Y. Hui. *Switching and Traffic Theory for Integrated Broadband Networks*. Kluwer Academic Publishers, Boston, 1990.

[Inc90] AMP Incorporated. Conductive Carbon Fiber Connectors Product Announcement, July 1990. Catalog 90-749.

[Int89] Intel Corporation, Literature Sales, P.O. Box 58130, Santa Clara, CA 95052-8130. *80960CA User's Manual*, 1989.

[Joh88] Mark G. Johnson. A Variable Delay Line PLL for CPU-Coprocessor Synchronization. *IEEE Journal of Solid-State Circuits*, 23(5):1218–1223, October 1988.

[Joh92] Kirk L. Johnson. The Impact of Communication Locality on Large-Scale Multiprocessor Performance. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, Queensland, Australia, May 1992.

[Jor83] J. F. Jordan. Performance Measurement on HEP – A pipelined MIMD Computer. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*. IEEE, June 1983.

[Kah91] Nabil Kahale. Better Expansion for Ramanujan Graphs. In *32nd Annual Symposium on Foundations of Computer Science*, pages 398–404. IEEE, October 1991.

[KK88] Thomas F. Knight Jr. and Alexander Krymm. A Self-Terminating Low-Voltage Swing CMOS Output Driver. *IEEE Journal of Solid-State Circuits*, 23(2), April 1988.

[KLS90] Kathleen Knobe, Joan D. Lukas, and Guy L. Steele, Jr. Data Optimization: Allocation of Arrays to Reduce Communication on SIMD Machines. *Journal of Parallel and Distributed Computing*, 8:102–118, 1990.

[KMZ79] Bernd Könemann, Joachim Mucha, and Günther Zwiehoff. Built-In Logic Block Observation Techniques. In *Proceedings 1979 International Test Conference*, pages 37–41. IEEE, 1979.

[KR89] Vijay P. Kumar and Andrew L. Reibman. Failure Dependent Performance Analysis of a Fault-Tolerant Multistage Interconnection Network. *IEEE Transactions on Computers*, 38(12):1703–1713, December 1989.

[KS86] Clyde P. Kruskal and Marc Snir. A Unified Theory of Interconnection Network Structure. In *Theoretical Computer Science*, pages 75–94, 1986.

[Lak86] Ron Lake. A Fast 20K Gate Array with On-Chip Test System. *VLSI Systems Design*, 7(6):46–55, June 1986.

[LeB84] Johnny J. LeBlanc. LOCST: A Built-In Self-Test Technique. *IEEE Design and Test*, pages 45–52, November 1984.

[Lei85] Charles E. Leiserson. Fat-Trees: Universal Networks for Hardware Efficient Supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.

[Lei89] Charles E. Leiserson. VLSI Theory and Parallel Supercomputing. MIT/LCS/TM 402, MIT, 545 Technology Sq., Cambridge, MA 02139, May 1989. Also appears as an invited presentation at the 1989 Caltech Decennial VLSI Conference.

[LHM$^+$89] Anthony L. Lentine, H. Scott Hinton, David A. B. Miller, Jill E. Henry, J. E. Cunningham, and Leo M. F. Chirovsky. Symmertric Self-Electrooptic Effect Device: Optical Set-Reset Latch, Differential Logic Gate, and Differential Modulator/Detector. *IEEE Journal of Quantum Electronics*, 25(8):1928–1936, August 1989.

[LLG$^+$91] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-Dietrich Weber, Anoop Gupta, and John Hennessy. Overview and Status of the Stanford DASH Multiprocessor. In Norihisa Suzuki, editor, *Proceedings of the International Symposium on Shared Memory Multiprocessing*, pages 102–108. Information Processing Society of Japan, April 1991.

[LM89] Tom Leighton and Bruce Maggs. Expanders Might Be Practical: Fast Algorithms for Routing Around Faults on Multibutterflies. In *IEEE 30th Annual Symposium on Foundations of Computer Science*, 1989.

[LM92] Tom Leighton and Bruce Maggs. Fast Algorithms for Routing Around Faults in Multibutterflies and Randomly-Wired Splitter Networks. *IEEE Transactions on Computers*, 41(5):1–10, May 1992.

[LP83] Duncan Lawrie and Krishnan Padmanabhan. A Class of Redundant Path Multistage Interconnection Networks. *IEEE Tr. on Computers*, C-32(12):1099–1108, December 1983.

[LR86] Frank T. Leighton and Arnold L. Rosenberg. Three-Dimensional Circuit Layouts. *SIAM Journal on Computing*, 15(3):793–813, August 1986.

[Mal91] David Maliniak. Pinless Land-Grid Array Socket Houses Intel's 80386SL Microprocessor. *Electronic Design*, March 28 1991.

[MB88] Glenford J. Myers and David L. Budde. *The 80960 Microprocessor Architecture*. Wiley-Interscience, 1988.

[MDK91] Henry Minsky, André DeHon, and Thomas F. Knight Jr. RN1: Low-Latency, Dilated, Crossbar Router. In *Hot Chips Symposium III*, 1991.

[Mil91] David Miller. *International Trends in Optics*, chapter 2, pages 13–23. Academic Press, Inc., 1991.

[Min91] Henry Q. Minsky. A Parallel Crossbar Routing Chip for a Shared Memory Multiprocessor. AI memo 1284, MIT Artificial Intelligence Laboratory, 545 Technology Sq., Cambridge, MA 02139, 1991.

[MR91] Silvio Micali and Phillip Rogaway. Secure Computation. MIT/LCS/TR 513, MIT, 545 Technology Sq., Cambridge, MA 02139, August 1991.

[MT90] Colin M. Maunder and Rodham E. Tulloss, editors. *The Test Access Port and Boundary-Scan Architecture*, chapter 20, pages 205–213. IEEE, 1990. Chapter by: Patrick F. McHugh and Lee Whetsel.

[ND90] Michael Noakes and William J. Dally. System Design of the J-Machine. In William J. Dally, editor, *Advanced Research in VLSI: Proceedings of the Sixth MIT Conference*, pages 179–194, Cambridge, Massachusetts, 1990. MIT Press.

[NPA91] R. S. Nikhil, G. M. Papadopoulos, and Arvind. *T: A Killer Micro for A Brave New World. CSG Memo 325, MIT, 545 Technology Sq., Cambridge, MA 02139, 1991.

[NPA92] Rishiyur S. Nikhil, Gregory M. Papadopoulous, and Arvind. *T: A Multithreaded Massively Parallel Architecture. In *Proceedings of the 19th International Symposium on Computer ARchitecture*. ACM, May 1992.

[oD86] U.S. Department of Defense. *Military Standardization Handbook: Reliability Prediction of Electronic Equipment*, mil-hdbk-217e edition, 1986. This handbook is updated periodically to track the evolution of electronic equipment technology. MIL-HDBK-217 was first issued in 1965.

[PC90] G. M. Papadopoulos and D. E. Culler. Monsson: an Explicit Token-Store Architecture. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*. IEEE, 1990.

[Pol90] Shin Etsu Poly. Shin-Flex GD Product Summary, 1990.

[Pos81] (Ed.) Jon Postel. Transmission Control Protocol – DARPA Internet Program Protocol Specification. RFC 793, USC/ISI, Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Rey, California, 90291, September 1981.

[PW72] W. Wesley Peterson and E.J. Weldon Jr. *Error-Correcting Codes*. MIT Press, Cambridge, MA, 1972.

[RWD84] Simon Ramo, John R. Whinnery, and Theodore Van Duzer. *Fields and Waves in Communication Electronics*. John Wiley and Sons, 2nd edition, 1984.

[SBCvE90] R. Saavedra-Barrerra, D. E. Culler, and T. von Eicken. Analysis of Multithreaded Architectures for Parallel Computing. In *Proceedings of the 2nd Annual Symposium on Parallel Algorithms and Architectures*, July 1990.

[Sim92] Thomas Simon. Matched Delay Clock Distribution. Personal Communications, 1992.

[Smi78] B. J. Smith. A Pipelined, Shared-Resource MIMD Computer. In *Proceedings of the 1978 International Conference on Parallel Processing*, pages 6–8, 1978.

[Smo85] R. Smolley. Button Board, A New Technology Interconnect for 2 and 3 Dimensional Packaging. In *International Society for Hybrid Microelectronics Conference*, November 1985.

[SS92] Daniel P. Siewiorek and Robert S. Swarz. *Reliable Computer Systems: Design and Evaluation*. Digital Press, Burlington, MA, 2nd edition, 1992.

[Tec88] Elastomeric Technologies. Elastomeric Connector Application Guide, 1988.

[Tex91] Texas Instruments. *Shrink Small Outline Package Surface Mounting Packages*, 1991.

[Thi91] Thinking Machines Corporation, Cambridge, MA. *CM5 Technical Summary*, October 1991.

[Tho80] C. D. Thompson. A Complexity Theory of VLSI. Technical Report CMU-CS-80-140, Department of Computer Science, Carnegie-Mellon University, 1980.

[Upf89] E. Upfal. An $O(\log N)$ deterministic packet routing scheme. In *21st Annual ACM Symposium on Theory of Computing*, pages 241–250. ACM, May 1989.

[Wag87] Paul T. Wagner. Interconnect Testing with Boundary Scan. In *Proceedings 1987 International Test Conference*, pages 52–57. IEEE, 1987.

[Wal92] Deborah Wallach. PHD: A Hierarchical Cache Coherent Protocol. Master's thesis, MIT, 545 Technology Sq., Cambridge, MA 02139, September 1992.

[WBJ+87] W. H. Wu, L. A. Bergman, A. R. Johnston, C. C. Guest, S. C. Esener, P. K. Yo, M. R. Feldman, and S. H. Lee. Implementation of Optical Interconnections for VLSI. *IEEE Transactions on Electron Devices*, 34(3):706–714, March 1987.

[Web90] Steve Webber. The Stratus Architecture. Stratus Technical Report TR-1, Status Computer, Inc., 55 Fairbanks Blvd., Marlboro, Massachusetts 01752, 1990.