Fault Tolerant Adaptive Routing in Multicomputer Networks

by

Thucydides Xanthopoulos

B.S.E.E., Massachusetts Institute of Technology

(1992)

Submitted to the Department of Electrical Engineering and Computer Science
In Partial Fulfillment of the Requirements for the Degree of
Master of Science
in Electrical Engineering and Computer Science
at the
Massachusetts Institute of Technology
February 1995

Signature of Author ______________________________

Department of Electrical Engineering and Computer Science
January 20, 1995

Certified by ______________________________

William J. Dally
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by ______________________________

Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Fault Tolerant Adaptive Routing in Multicomputer Networks

by

Thucydides Xanthopoulos

Submitted to the
Department of Electrical Engineering and Computer Science
on January 20, 1995, in partial fulfillment of
the requirements for the Degree of Master of Science in
Electrical Engineering and Computer Science

Abstract

Interconnection networks play a major role in the performance and reliability of massively parallel processors (MPPs). This work is concerned with the design and implementation of a wormhole fault-tolerant adaptive routing algorithm for k-ary n-meshes called Reliable Adaptive Routing (RAR).

RAR when coupled with a fault-detection mechanism and a retransmission protocol is capable of handling a single link or node failure anywhere in the network without interruption of service. Furthermore, RAR can assign multiple message paths between each source and destination pair. RAR uses virtual channels to prevent deadlocks. A total of three virtual channels per physical channel are necessary for deadlock prevention. The routing algorithm is formally defined and a formal proof of deadlock freedom is presented.

This work also presents the Virtual Channel Dependency Analyzer (VCDA) a software tool with a graphical user interface that helps in the visualization and study of the channel dependency graph produced by Reliable Adaptive Routing.

A sample circuit implementation of RAR is also presented. The circuit uses $236\mu m \times 904\mu m$ of silicon area and has a worst case delay of 6 ns. This work concludes by describing higher level design issues such as the integration of the above circuit in a complete routing system, and also the coupling with a retransmission protocol to produce a reliable network layer.

Thesis Supervisor: William J. Dally
Title: Associate Professor of Electrical Engineering and Computer Science

Keywords: Adaptive routing, virtual channels, channel dependency graph, reliable routing, fault tolerance.

Dedication

Για το Θοδωρη (1963 – 1992)

Με περνουσες εφτα χρονια. Τωρα με περνας μονο τεσσερα.

Στη φωτογραφια πανω πρεπει να ειμαστε στην ιδια ηλικια.

Εσυ στην Καλυμνο, ωραιος, δυνατος, μαυρισμενος, χαβαλες, με γυαλια ηλιου.

Εγω στη Βοστωνη, χοντρος, δυσκινητος, μπροστα απο μια οθονη με γυαλια μυωπιας.

Εσυ στον ουρανο κι εγω στη γη.

Φαινεσαι ετοιμος να διηγηθεις παλι καμια ιστορια.

Τις εχω πιστεψει ολες εκτος απο μια : *Πως θα μεινεις για παντα εικοσιεννια.*

You were seven years older than me. Now, you are just four years older. In the picture above, we probably have the same age.

You are in Kalymnos, handsome, strong, tanned, having fun, wearing sunglasses. I am in Boston, fat, pathetic, in front of a computer screen wearing prescription glasses.

You are in heaven, I am still in earth.

You look ready to tell a story again. I have believed them all except for one: That you will always be twenty-nine.

Acknowledgments

I would like to thank a number of people that have contributed to one way or another to the completion of this project:

- Bill Dally for his energy, enthusiasm and most constructive supervision.
- Larry Dennison for his valuable experience and enormous patience.
- All the past and present members of the Reliable Router team including Larry Dennison, Kin Hong Kan, David Harris, Ivan Oei, Dan Hartman and Jeff Bowers for their valuable contributions to our project.
- All the members of the CVA group for providing a most pleasant and stimulating work environment. I would like to thank especially Rich Lethin for volunteering to proofread the most basic chapters of this thesis.
- All my friends here in Boston, especially Sylvi Dogramatzian for helping me get a life!
- Finally, I would like to thank my parents Maria and Nicholas Xanthopoulos as well as my uncle John Xanthopoulos for their invaluable love and support throughout all these years that I have been away.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Massively parallel processors (MPPs) are considered today a very promising approach of achieving teraflops of computational power. Multicomputers of this kind are usually organized as a number of nodes where each node has its own local processor and memory. The nodes are connected to each other by means of an interconnection network. MPPs are capable of exhibiting such desirable properties as scalability, reconfigurability and fault tolerance. Moreover, MPPs as opposed to other architectural approaches such as networks of workstations exhibit cost balance [Dal93].

The interconnection network is the key component of a parallel computer. Such networks may accept a message from any processing node, and deliver it to any other processing node. Interconnection network design greatly affects both system cost and performance [Dal93].

This thesis is concerned with one important aspect of interconnection network design: routing. Routing [Dal90] is the method used for a message to choose a path from source to destination over the network channels. Routing decisions can be classified as oblivious or adaptive. An oblivious routing method does not use information about the network state in choosing a path. An example of an oblivious routing method is Dimension-Ordered routing [DS87]. Adaptive routing on the other hand, may use information concerning the state of the network. This thesis is concerned with the design and implementation of an adaptive routing algorithm for k-ary n-meshes.

1.1 A Case for Adaptive Routing

Most existing multicomputer routing networks use oblivious deterministic routing [DFK$^+$92] [SS93]. Oblivious deterministic routing assigns a single path between each source and

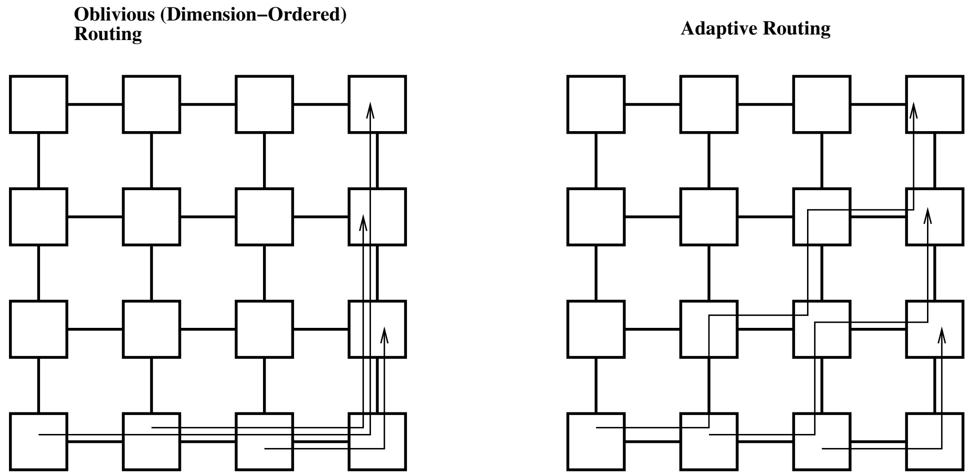

Figure 1.1: Adaptive routing achieves better physical link utilization from oblivious routing.

destination pair without taking into account the state of the network. The problem is illustrated in Figure 1.1.

As can be seen from the figure, oblivious routing cannot achieve good physical link utilization because of strong restrictions in the use of physical channels. In Figure 1.1, three nodes try to send messages to three different destinations. Dimension-Ordered routing forces the messages to use the same physical channels, thus overloading the physical links and decreasing performance. Messages first have to reduce their offset along x and then along y. Adaptive routing, on the other hand, is capable of assigning different paths between each source and destination based on channel availability. If a desired physical channel is busy, another channel leading towards the destination may be chosen. Adaptive routing algorithms can achieve a much better utilization of physical links and thus increase performance.

A number of studies regarding the performance of adaptive routing have been published. Among those, we pick the one by Duato and Lopez [DL94] because the routing algorithm evaluated has been derived using the same methodology that will be used in chapter 4 to construct Adaptive Routing. This study examines a variety of traffic patterns and also takes into account the fact that adaptive routers need more complicated routing logic than oblivious routers. This assumption is incorporated in the study in the form of increased cycle time for the adaptive router. Chien's parametric speed model [Chi93] was used to derive specific cycle times. The results indicate that adaptive routing consistently produces lower average message latency than Dimension-Ordered routing for all traffic patterns.

Adaptive routing has critics too. Recently, Pertel [Per92] has published a study in which he presents simulation results that allegedly refute the claims that adaptive routing is superior to Dimension-Ordered routing in terms of performance (average message latency). The main conclusion of the study is that under heavy traffic, Dimension-Ordered routing tends to queue more packets at their source and fewer packets in the network compared to adaptive routing. Queueing in the network is the major factor that contributes to the increased message latency exhibited by adaptive routing.

There are two main issues that undermine the significance of Pertel's results:

1. All results assume random uniform traffic patterns. There is no reason that adaptive routing should perform better in such cases.

2. Pertel's simulator assumes infinite buffering to resolve deadlocks in adaptive routing. This is not a realistic assumption. Infinite buffering practically eliminates backpressure at the source and can increase unrealistically network queue sizes. Given this assumption, Pertel's conclusion regarding network queue sizes is self-induced.

Adaptive routing is a very promising approach to take advantage of the rich and regular connectivity of multicomputer networks. Increasing the utilization of the network bisection bandwidth will increase performance in most traffic cases. The only serious disadvantage of adaptive routing is the design complexity introduced to the switching element.

1.2 Contributions of this work

A number of wormhole adaptive routing algorithms have appeared in the literature [NM93]. Yet, there are no hardware implementations of such algorithms in part because of the difficulty and design complexity involved.

This thesis presents a complete solution to the problem of adaptive routing ranging from the formal definition of a simple but powerful routing algorithm to the circuit implementation and layout. The algorithm presented is fully adaptive, can handle a single non-transient fault at a time, and has advanced features such as neighbor routing if the original message recipient is not available due to a fault. The circuit presented has a worst case delay of 6 ns (slowest process corner) when implemented in the hp26 process, and has a total area of $236\mu m \times 904\mu m$.

1.3 Thesis Overview

Chapter 2 gives some background information on adaptive routing. It presents adaptive routing algorithms proposed in the current literature and briefly analyzes their strong and weak points.

Chapter 3 presents a number of formal definitions regarding interconnection networks, routing functions and dependencies between virtual channels. It also presents two important theorems, the first by Dally [DS87] and the second by Duato [Dua93] regarding deadlock-freedom in wormhole networks.

Chapter 4 formally defines Adaptive Routing, an adaptive routing algorithm for k-ary n-meshes. It also proves that Adaptive Routing is deadlock-free.

Chapter 5 augments the Adaptive Routing algorithm presented in the previous chapter with the capability to handle a single non-transient fault at a time anywhere in the network without interruption of service. The resulting algorithm is called Reliable Adaptive Routing. The chapter also includes a proof of deadlock freedom.

Chapter 6 describes the Virtual Channel Dependency Analyzer (VCDA). The VCDA is a software tool that helps in the visualization and analysis of Reliable Adaptive Routing. It can also prove by exhaustion that Reliable Adaptive Routing is deadlock-free.

Chapter 7 is concerned with implementation issues. It describes the physical design of a two-dimensional version of Reliable Adaptive Routing. The resulting circuit is actually used in the MIT Reliable Router [DDH+94b], a high-performance network switching element.

Chapter 8 describes how the circuit presented in the previous chapter fits into the Reliable Router design. Moreover, it presents a message retransmission mechanism which must be used in conjuction with adaptive routing for a reliable network layer.

Finally, Chapter 9 summarizes and conludes the document.

Chapter 2

Background

This chapter constitutes a survey of wormhole adaptive routing algorithms for low dimensional bidirectional meshes that have appeared in the literature. A brief overview of each algorithm is presented along with a discussion of strong and weak points. The first section of the chapter introduces a common study framework in terms of a number of evaluation criteria. These criteria are mainly concerned with the degree that a routing algorithm can be easily and efficiently implemented in a VLSI system.

2.1 Evaluation Criteria

Reasonable Virtual Channel Requirements Virtual channels [DS87] can be very expensive to implement. Not only do they require additional on-chip memory, but also they require allocation mechanisms and arbitration layers for other resources that are shared among them. Chien [Chi93] presents a parametric model that quantifies the cost of virtual channels in a generic router architecture in terms of number of gates and added latency. Virtual channels in adaptive routing algorithms are used to guarantee deadlock-freedom. We are interested in adaptive routing algorithms that require a small number of virtual channels in order to be deadlock-free.

Minimality A routing algorithm is minimal if it does not allow misrouting steps – steps that will move a message away from its destination. Minimality is important for two reasons: First, it guarantees freedom from livelock. Second, it prevents wasting network resources in terms of channels and buffers used to take a message further from its destination [Dal90]. Minimality is a desirable property in a routing algorithm. Departures from minimality should only take place when there is absolutely no alternative due to a faulty link.

Full Adaptivity Full adaptivity means that a routing algorithm gives maximum routing freedom (within the limits placed by minimality) to a message trying to reach its destination. A fully adaptive algorithm can assign a next step channel of any possible direction that will move the message one step closer to its destination. Full adaptivity is a desirable property because more choices result in increased performance for non-uniform traffic patterns.

Fault Tolerance Fault-tolerance is the ability of a routing algorithm to bypass faulty links in the network. We are interested in routing algorithms that can handle gracefuly one non-transient fault at a time anywhere in the network.

Dependence on Local Information This last requirement means that the routing decision depends only on the current position of a message and the position of the message destination. It does not depend on any past routing decisions. This requirement simplifies the physical design of the routing subsystem because it does not involve updating the header of a message with certain state information. Moreover, it makes possible to employ a higher level end-to-end error detecting (or correcting) protocol that includes the message header as well.

2.2 The Linder-Harden Algorithm

Linder and Harden [LH91] have proposed a fully adaptive routing algorithm for bidirectional k-ary n-cubes. The main idea behind the Linder-Harden approach is the partitioning of the bidirectional physical network into several *Virtual Networks*. These virtual networks are unidirectional in all dimensions except for dimension 0. Messages are routed within a single virtual network throughout their course. The selection of the virtual network to be used by each message is based on the relative address of the message source with respect to the message destination. Routing within each virtual network is fully adaptive (minimal). Disadvantages of this algorithm are:

1. The Linder-Harden approach requires 2^{n-1} virtual networks where n is the dimension of the mesh, and therefore requires 2^{n-1} virtual channels per physical channel. The exponential dependence of the number of virtual channels on the mesh dimension makes the Linder-Harden algorithm impractical for network dimensions greater than two.

2. The Linder-Harden scheme is not designed for fault-tolerance. Routing is minimal and does not allow side steps. Fault tolerant capabilities can be added in an ad hoc fashion on top of the existing algorithm at the expense of more virtual channels.

2.3 Planar Adaptive Routing

Chien and Kim have proposed Planar Adaptive Routing [CK92]. Planar Adaptive Routing provides adaptivity in only two dimensions at a time. Figure 2.1 contrasts Planar Adaptive Routing with a fully adaptive routing algorithm in three dimensions. Within each adaptive plane, messages are routed in a fully adaptive fashion in a way very similar to the Linder-Harden algorithm. More specifically, in every adaptive plane A_i, the physical network is partitioned into two virtual networks: The increasing network which is used for messages that need to travel in the positive direction along dimension i, and the decreasing network, used by messages that need to travel in the negative direction along dimension i. When the distance in dimension i is reduced to zero, the message proceeds to adaptive plane A_{i+1}. In the two-dimensional mesh case, Planar Adaptive Routing is equivalent to the Linder-Harden approach.

By providing adaptivity to only two dimensions at a time, Chien and Kim have managed to reduce the virtual channel requirements of the Linder-Harden algorithm. Planar Adaptive Routing needs 3 virtual channels per physical channel for an n-dimensional mesh except for $n = 2$. In this case only two virtual channels are necessary.

The Planar Adaptive Algorithm can be extended to handle faults that form convex regions within the network. The fault-tolerant version still requires 3 virtual channels per physical channel. The two-dimensional case requires 3 virtual channels as well. There are three major drawbacks in the way that the Planar Adaptive algorithm handles faults:

1. Misrouting steps result in tagging the message header. Therefore, the routing decision does not depend on local information only.

2. The algorithm requires an intermediate deactivation phase each time a fault occurs to ensure that the faulty regions are convex. In the case where there is a fault along the edge of the network, the deactivation algorithm will deactivate all k nodes along that particular edge.

In conclusion, Planar Adaptive Routing is not an attractive routing algorithm because of limited adaptivity and complications and inefficiencies in the fault-tolerant version.

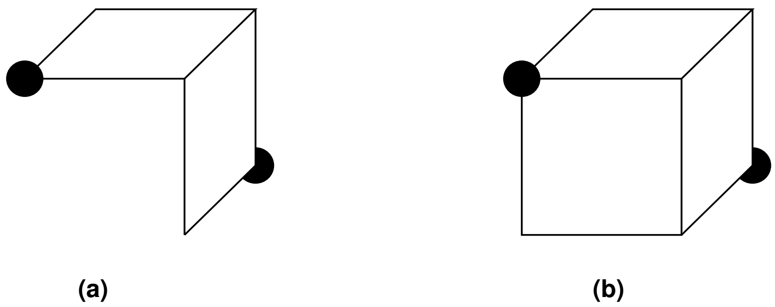

Figure 2.1: An example of Planar Adaptive Routing (a) vs. fully adaptive routing (b).

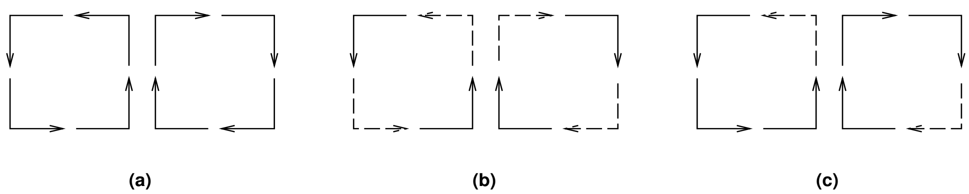

Figure 2.2: An illustration of the Turn Model in a 2D mesh: (a) abstract cycles in a 2D mesh; (b) four turns (solid arrows) allowed in Dimension-Ordered Routing; (c) six turns (solid arrows) allowed in West-First routing. This figure has been reproduced from Ni and McKinley [NM93]

2.4 The Turn Model For Adaptive Routing

Glass and Ni have proposed the Turn Model for Adaptive Routing [GN92]. The Turn Model is an elegant framework that can produce a family of partially adaptive routing algorithms which have the highest possible degree of adaptivity that can be achieved using a single virtual channel.

Deadlocks in wormhole routing are caused by messages waiting on each other in a cycle. The Turn Model prevents deadlocks by placing restrictions on the turns that a message can take throughout its course. The fundamental concept in this algorithm is the prohibition of the smallest number of turns so that cycles are prevented.

Figure 2.2 illustrates the concept behind the Turn Model in a two-dimensional mesh. Figure 2.2 (a) shows the two abstract cycles that can cause deadlocks in a 2D routing algorithm. Figure 2.2 (b) shows the four turns that are prohibited in Dimension-Ordered routing. Finally 2.2 (c) shows that Dimension-Ordered routing places more restrictions than necessary for deadlock-freedom. The formation of cycles that lead to deadlock can be pre-

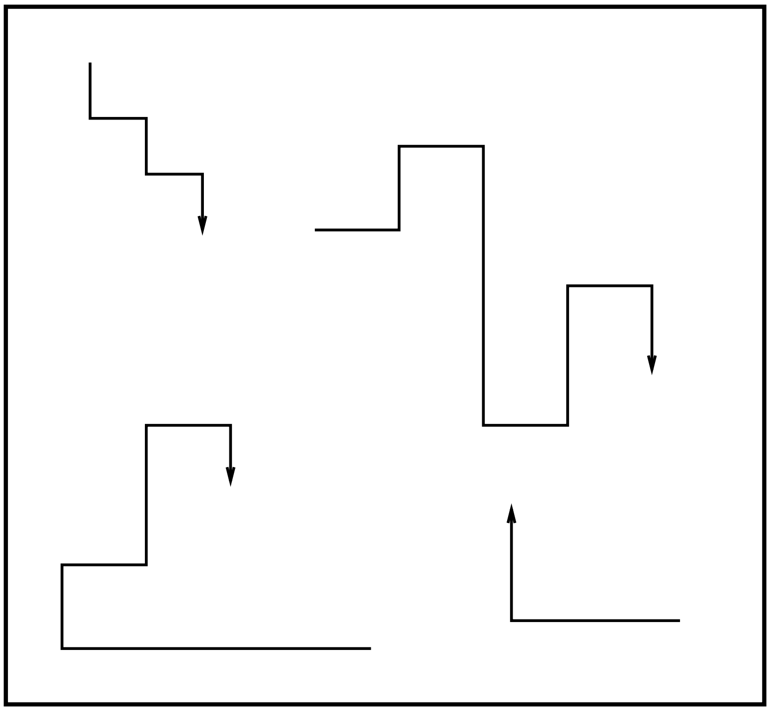

Figure 2.3: Example message paths under the West-First routing algorithm

vented by prohibiting only two turns instead of four. The resulting routing algorithm that can be derived from the prohibition of the two turns in Figure 2.2 (c) is called *West-First*: It routes a message first west if necessary and then it routes it nonminimally adaptively south, east and north. Example message paths under the West-First algorithm are shown in Figure 2.3. Other algorithms can be constructed by prohibiting a different pair of turns.

Although the Turn Model is a simple and elegant framework for adaptive routing with minimal resource requirements, it has a number of weak points:

1. It does not scale easily to more than two dimensions. The framework with the abstract cycles does not extend in an obvious fashion to more than two dimensions because of the possibility of cycle formation that spans three or more dimensions.

2. The Turn Model produces algorithms that are not symmetric and place an uneven load on the physical links of the network. Independent studies by Glass and Ni [GN92] and Boppana and Chalasani [BC93] have concluded that the performance of Turn Model routing is inferior to other routing schemes especially for uniform traffic.

3. The Turn Model does not produce fault-tolerant algorithms.

2.5 Dimension Reversals

Dally and Aoki [DA93] have proposed an adaptive routing algorithm which is based on the notion of "dimension reversals". Dimension reversals is the count of the number of times a packet has been routed from a channel in one dimension to another in violation of the dimension order imposed by the underlying Dimension-Ordered routing algorithm. The static version of the algorithm divides the virtual channels of each physical link into r classes where r is the maximum number of permitted dimension reversals. Messages with a dimension reversal count less than r can be routed adaptively but only using channels of class equal to their dimension reversal count. Once a message has a dimension reversal count equal to r it can only be routed using Dimension-Ordered routing on virtual channels of class r.

The dynamic version of the algorithm does not place an upper bound on the dimension reversal count. It divides the virtual channels of each physical channel into two classes: adaptive and oblivious. Messages originate in adaptive channels. While in those channels messages can be routed along any direction. The dimension reversal count is maintained in the same way as before. Each time a virtual channel is allocated to a message, it is marked with the dimension reversal count of the message. To avoid deadlock, a message may not use a channel labeled with a count less than or equal to its own dimension reversal count. If an appropriate adaptive channel is not found, the message switches to the oblivious channels under Dimension-Ordered routing. It may not reenter the adaptive channels.

This routing scheme is not attractive for a number of reasons:

1. Updating dimension reversal counts and marking virtual channels increases routing complexity significantly and does not meet the requirement that the routing decision should depend on local information only.

2. The algorithm is not fault-tolerant.

2.6 Compressionless Routing

Kim, Liu and Chien have recently proposed Compressionless Routing [KLC94]. The main concepts behind this scheme are: Compressionless Routing (CR) supports deadlock recovery instead of prevention because according to the authors deadlocks are infrequent. Second, CR uses the feedback that wormhole routing provides in the form of flow control.

A message routed under Compressionless Routing does not release intermediate virtual channel buffer storage until the head flit reaches its destination. In this way, the tail flit of the message does not leave the source node before the head flit reaches the destination. If the message length is shorter than the distance between the message source and destination, then the message is padded with idle flits to ensure that the message header reaches the destination before the last flit has been injected by the source. The sender keeps track of the number of flits it has injected, and increments a timeout counter each time that it cannot inject a flit. The timeout counter is reset each time a flit is injected. After a number of flits equal to the distance between the source and the destination have been injected, the sender releases the path because the header has succesfully reached the destination and there is no possibility for deadlock. If on the other hand, the timeout counter reaches a threshold value before the header reaches the destination, then this indicates a possible deadlock. In such a case, the sender tears down the connection and will attempt retransmission after a certain time period.

Since this scheme supports deadlock recovery instead of prevention, routing is fully adaptive with no restrictions and no virtual channels. Compressionless Routing can be thought of as a recast of circuit switching adapted to wormhole networks.

Compressionless Routing can have serious performance problems:

1. CR is an unstable routing scheme because it resolves conflicts by dropping [Dal90]. This means that for traffic over a certain threshold, network throughput will drop dramatically.

2. CR is geared to small flit size, small physical channel bandwidth and very narrow flit buffers in each network node. Unless the above are true, sending medium and short messages under Compressionless Routing will result in sacrificing substantial network bandwidth due to message padding.

2.7 Pipelined Circuit Switching

Another similar approach to Compressionless Routing is Pipelined Circuit Switching (PCS) proposed by Allen, Gaughan, Schimmel and Yalamanchili [AGSY94]. PCS has been implemented in hardware in the Ariadne Router.

PCS is different from wormhole routing in the following respect: In wormhole routing, data flits immediately follow the head flit in a pipeline fashion. In PCS, the data flits do not immediately follow the head flit into the network. Instead, the head flit acts as a probe that sets up a circuit. When the head reaches the destination, an acknowledgment returns to the source. Only then can the data flits be pipelined through the circuit that has been set up.

The fact that the data flits do not immediately follow the head flit results in increased flexibility in the routing of the head. The head flit can backtrack and release previously reserved virtual channels instead of blocking and waiting on busy channels. PCS can use the family of Misrouting-Backtracking (MB-x) protocols [GY93] for head flit routing. The MB-x protocols search the network for a path in a depth-first manner. Routing is minimal with the addition of x misrouting steps. The misrouting count is decremented each time a misrouting step is reversed. Neither deadlock nor livelock can occur because MB-x protocols do not block holding resources and they can use history information to avoid searching the same path repeatedly.

PCS uses three virtual channels per physical channel. Data and control information (header) use separate virtual channels. There is an extra virtual channel that has the opposite direction from the previous two and is reserved for backwards-travelling flits: acknowledgments and backtracking flits.

There are three main concerns with PCS:

1. The added round trip probe latency will have a big impact on network performance.
2. Maintaining misrouting history for a number of misrouting steps greater than one can sunstantially increase system complexity.
3. There is no support for fault tolerance once the circuit has been set up.

2.8 *f*-cube3 Algorithm

An interesting fault-tolerant extension to e-cube (Dimension-Ordered) routing was recently proposed by Boppana and Chalasani [BC94]. Faults are grouped into fault sets that occupy rectangular regions such that the boundary of the rectangle has only fault-free nodes and channels and the interior of the rectangle contains all the faulty channels and nodes that correspond to that particular fault set. This fault model is superior to the convex fault regions proposed by Chien and Kim [CK92] because it deals more effectively with faults along the network boundary. Boppana and Chalasani develop the concept of fault rings

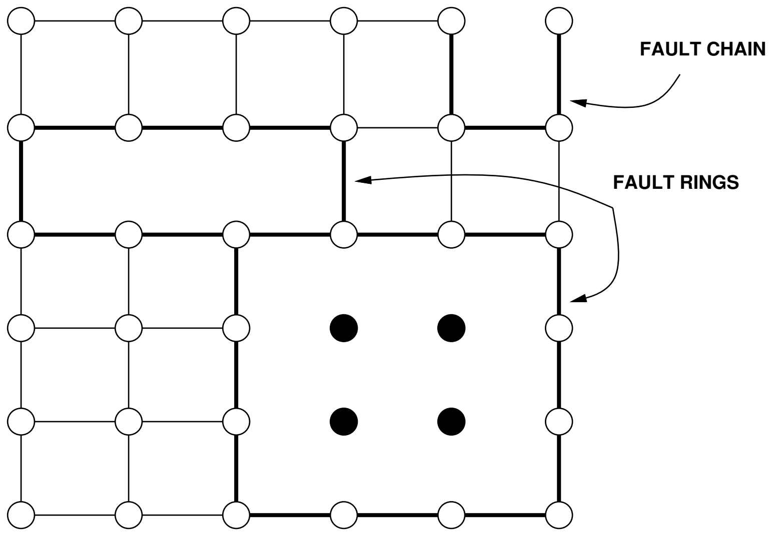

Figure 2.4: Fault chains and fault rings in *f*-cube3 routing. This figure is reproduced from Boppana and Chalasani [BC94].

and fault chains, rectangular paths around the fault regions that consist of fault-free nodes and links. Figure 2.4 shows examples of such structures.

Messages are routed under regular Dimension-Ordered routing until they reach a fault ring or a fault chain. Then, depending on the relative position of the destination, messages are routed clockwise or counterclockwise around the fault ring until all misrouting steps are reversed. Three virtual channels are necessary to ensure deadlock-freedom.

The disadvantages of this algorithm are:

1. *f*-cube3 is not an adaptive routing algorithm. Although it requires a minimum of three virtual channels for deadlock freedom, it is a fully oblivious scheme. Adaptivity can be added on top of the underlying fault-tolerant algorithm at the expense of more virtual channels.

2. *f*-cube3 requires a cumbersome setup phase each time a fault occurs during which nodes around the fault exchange messages to determine their position on the fault ring. Such a phase can add considerable complexity to the system. Moreover, *f*-cube3 requires updating the message header to indicate misrouting and direction around the fault ring.

3. Overlapping fault rings cannot be handled in *f*-cube3. Extra virtual channels are necessary for deadlock-freedom.

Algorithm	VC's (min)	Adaptivity	Minimality	Local Info
Linder-Harden	2^{n-1}	Full	Yes	Yes
Planar Adaptive	3	Partial	Yes	Yes
Turn Model	1	Partial	No	Yes
Dimension Reversals (s)	r	Full	Yes	No
Dimension Reversals (d)	2	Full	Yes	No
Compressionless	1	Full	Yes	Yes
PCS	3	Full	No	No
f-cube3	3	None	Yes	No
Duato's New Theory	2	Full	Yes	Yes

Table 2.1: A brief comparison of existing adaptive routing algorithms (a)

2.9 Duato's New Theory on Adaptive Routing

Duato [Dua91][Dua93] has set up a theoretical framework and provided a methodology for constructing fully adaptive routing algorithms with minimal virtual channel requirements.

Duato divides the virtual channels of each physical channel in two classes: oblivious and adaptive. Messages can use both channel classes and switch freely between the two. Duato studies the dependencies among the oblivious channels. Part of these dependencies are *direct* dependencies and are introduced by messages using two consecutive oblivious channels, and the rest of the dependencies are *indirect* and are introduced by messages using oblivious channels, then switching to adaptive and then switching back to oblivious. Duato proves that such an adaptive routing algorithm is deadlock-free if there are no cyclic direct nor indirect dependencies. The minimum number of virtual channels necessary for algorithms derived using Duato's methodology is two.

Duato's methodology is valuable because it can be used to add full adaptivity on top of an existing deadlock-free oblivious routing algorithm at the expense of one more virtual channel.

2.10 Summary and Conclusion

This chapter has presented a number of wormhole adaptive routing algorithms that have appeared in the literature. Tables 2.1 and 2.2 summarize the algorithms presented in terms of the evaluation criteria set in section 2.1. As one can see from the tables, none of the proposed algorithms is particularly attractive either because it does not meet our criteria or because it does not have adequate performance. Of all the algorithms presented, Duato's

Algorithm	Support for Fault Tolerance
Linder-Harden	Ad hoc and not well integrated
Planar Adaptive	Inefficient and complicated because of deactivation algorithm.
Turn Model	Incomplete
Dimension Reversals (s)	Incomplete
Dimension Reversals (d)	Incomplete
Compressionless	Complete but decreases performance
PCS	Incomplete
f-cube3	Complete but complicated
Duato's New Theory	Left to designer

Table 2.2: A brief comparison of existing adaptive routing algorithms (b)

theory and methodology is the most attractive candidate. In the next chapters we will use Duato's theory to construct a new fault-tolerant adaptive routing algorithm which meets all the requirements set in section 2.1. We call this algorithm Reliable Adaptive Routing.

Chapter 3

Definitions, Theorems and Notation

This chapter presents a number of definitions necessary for clarity and coherence throughout this document. Two very important theorems are also presented which will serve as the basis for the proof that the routing algoritm presented in Chapter 5 is deadlock-free. But first the notation that will be used in subsequent chapters is introduced and explained.

3.1 Notation

The following notation will be used throughout the rest of this document.

c_a	A virtual channel.
src(c_a)	The source node of virtual channel c_a.
dst(c_a)	The destination node of virtual channel c_a.
x(c_a)	The x address of src(c_a).
y(c_a)	The y address of src(c_a).

Figure 3.1 can be a helpful reference for the notation used.

3.2 Definitions

The definitions that appear in this section are based on the definitions given by Dally [DS87] and Duato [Dua93].

Definition 1 An *interconnection network* I is a strongly connected directed graph, $I = G(N, C)$. The vertices of the graph N represent the set of processing nodes. The arcs of

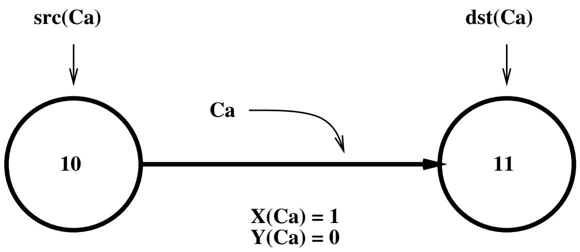

Figure 3.1: Notation used.

the graph C represent the set of communication channels. More than one arcs may connect two nodes.

Definition 2 A *routing function* $R \subseteq C \times N \times C^p$ where p is an integer, maps the current channel c_c and the destination node n_j to a set of channels where the packet can be routed next.

Definition 3 A *routing subfunction* R_1 for a given routing function R and channel subset $C_1 \subseteq C$, is a routing function such that:

$$R_1 : C \times N \times C_1^p \mid R_1(c, n) = R(c, n) \cap C_1^p \quad \forall c, n \in (C, N) \tag{3.1}$$

where p again is a positive integer.

Definition 4 A *channel dependency graph* D for a given interconnection network I and a routing function R, is a directed graph, $D = G(C, E)$. The vertices of D are the channels of I. The edges of D are the pairs of channels connected by R:

$$E = (c_i, c_j) \mid c_j \in R(c_i, n) \text{ for some n} \in \text{N}. \tag{3.2}$$

Definition 5 Given an interconnection network I, a routing function R, a channel subset $C_1 \subseteq C$ which is the range of a routing subfunction R_1, a channel c_s and a pair of channels $c_i, c_j \in C_1$, there is a *direct dependency* from c_i to c_j if

$$c_i \in R(c_s, n) \tag{3.3}$$

$$c_j \in R(c_i, n) \tag{3.4}$$

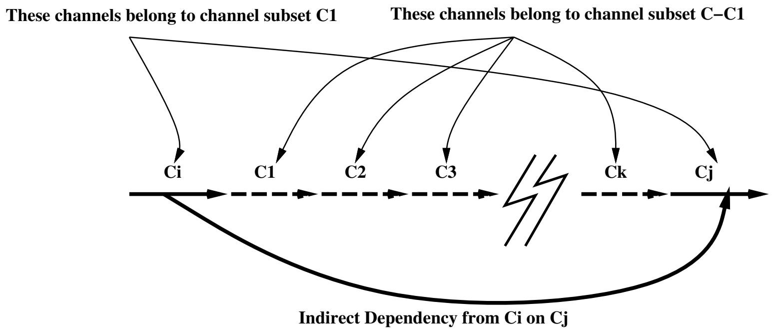

Figure 3.2: Visualizing Indirect Dependencies.

In other words, c_j can be used immediately after c_i by messages travelling from channel c_s to a destination node n.

Definition 6 Given an interconnection network I, a routing function R, a channel subset $C_1 \subseteq C$ which is the range of a routing subfunction R_1 and a pair of channels $c_i, c_j \in C_1$, there is an *indirect dependency* from c_i to c_j iff

$$\exists c_1, c_2, \ldots, c_k \in C - C_1 \mid \begin{cases} c_i \in R_1(c_{i-1}, n) & \\ c_1 \in R(c_i, n) & \\ c_{m+1} \in R(c_m, n) & \text{m = 1, \ldots, k - 1} \\ c_j \in R_1(c_k, n) & \text{for some n} \in \text{N} \end{cases} \tag{3.5}$$

Figure 3.2 can be helpful in understanding better the definition of an extended dependency.

The existence of an indirect dependency between channels c_i and c_j means that it is possible to establish a path from c_i to dst(c_j) where c_i and c_j are the first and last channels of the path and belong to channel subset C_1 and all the intermediate channels belong to channel subset $C - C_1$.

Definition 7 An *extended channel dependency graph* D_E for a given interconnection network I and routing subfunction R_1 of a routing function R, is a directed graph, $D_E = G(C_1, E_E)$. The vertices of D_E are the channels which are the range of the routing subfunction R_1. The arcs of D_E are the pairs of channels (c_i, c_j) such that there is either a direct or indirect dependency from c_i to c_j.

3.3 Theorems

Theorem 3.1 (Dally) *A routing function R for an interconnection network I is deadlock-free if there are no cycles in the channel dependency graph D.*

Dally [DS87] presents a formal proof of this theorem. Duato [Dua93] presents a slightly different version of the proof.

Theorem 3.2 (Duato) *A routing function R for an interconnection network I is deadlock-free if there exists a subset of channels $C_1 \subseteq C$ which forms the range of a routing subfunction R_1 which is connected and has no cycles in its extended channel dependency graph D_E.*

Duato [Dua93] has formally proved the above theorem. Theorem 3.2 has shown that Theorem 3.1 imposes too strong a constraint on a routing function R in order to be deadlock-free: The acyclic channel dependency graph is a sufficient but not a necessary condition. Theorem 3.2 poses a less strong constraint on R. The result is that "cheaper" adaptive routing algorithms can be made now possible (in terms of the number of virtual channels necessary to ensure freedom from deadlock.) Actually, both Theorems 3.1 and 3.2 only give a necessary but not sufficient condition for a routing function R to be deadlock-free. Duato [Dua94a] has showed that the inverse of Theorem 3.2 is not true and the necessary condition is not as strong as the one suggested in the above theorem. Yet, Theorem 3.2 is sufficient for the development of a simple and symmetric fault-tolerant adaptive routing algorithm which can be easily implemented in silicon. This theorem will be used to show that the algorithm presented in the following chapters is deadlock-free.

There is a difference in the definitions given by Duato for a routing function and a routing subfunction and the definitions presented in the previous section. Duato [Dua93] defines a routing function as follows:

$$R : N \times N \mapsto C^p \tag{3.6}$$

We have decided to define a routing function as Dally [DS87] does.

$$R : C \times N \mapsto C^p \tag{3.7}$$

Duato does not define a routing function as in Equation 3.7 because in such a case Theorem 3.2 would not be valid. Consider for example two subsets of C, C_1 and $C - C_1$.

Let R be a routing function defined in such a way that all messages arriving in a node through a channel that belongs to the $C - C_1$ subset are routed to a channel that belongs to the *same* subset. Suppose that are cyclic dependencies among the channels that belong to $C - C_1$. Moreover, suppose that there exists a routing subfunction R_1 whose range is C_1. R_1 is connected and has no cycles in its extended channel dependency graph. In such a case, R is not guaranteed to be deadlock-free.

We believe it is important make the domain of a wormhole routing function $C \times N$ instead of $N \times N$ for the following reasons:

1. As Dally [DS87] suggests, the formulation of routing as a function from $C \times N$ to C^p has more memory than the conventional definition of routing as a function from $N \times N$ to C^p. This definition has actually helped to develop the notion of channel dependence.

2. The formulation from $C \times N$ to C^p provides information that is actually *used* in certain cases by the algorithm to be described in the following chapters. As an example, our algorithm does not allow the use of $C - C_1$ channels when some conditions hold, one of which is that the message arrives at the node through a C_1 channel. A definition such as 3.7 is most appropriate in such cases.

In order to make Theorem 3.2 valid and still maintain $C \times N$ as the domain of the routing function we need to impose one constraint on R. Given a routing function $R : C \times N \mapsto C^p$, a channel subset C_1 and a routing subfunction $R_1 : C \times N \mapsto C_1^p$, Theorem 2 is valid if

$$R(c, n) \cap C_1^p \neq \emptyset \qquad \forall (c, n) \in C \times N \tag{3.8}$$

In other words, R is defined in such a way that it is never possible for a message to be forced to use only $C - C_1$ channels after a certain point in its course. If this provision is met, the pathological case described above will be avoided.

Chapter 4

Adaptive Routing

This chapter presents an adaptive routing algorithm with a minimum number of virtual channels. This algorithm does not have fault-handling capabilities. We name this algorithm Adaptive Routing (AR) as opposed to Reliable Adaptive Routing (RAR) which will be presented in the next chapter. This particular choice of presentation was made because an incremental approach is easier to organize and easier to be understood and appreciated by the reader.

4.1 Informal Description

AR needs two virtual channels per physical channel to ensure that there will be no deadlocks. AR is minimally adaptive: A message can only make steps that will bring it closer to its destination (productive). Unproductive steps (misrouting) are not allowed.

One of the two virtual channels is the Adaptive channel and the other is the Dimension-Ordered channel. When a message gets injected into the network, it starts to route on the Adaptive channels. AR can assign any free productive Adaptive channel as a possible next step. A virtual channel is considered free if it is currently not allocated to a message. If there are no free Adaptive channels AR assigns the unique Dimension-Ordered virtual channel that corresponds to the current position of the message and its destination. This unique Dimension-Ordered virtual channel is the same one as the one that would be assigned by a pure Dimension-Ordered routing function given the current message position and the message destination. If after the next hop there are free Adaptive channels, the message switches back to them.

A Dimension-Ordered routing function routes messages in strict dimension order: In every dimension d a message is routed along that dimension until it reaches a node whose address

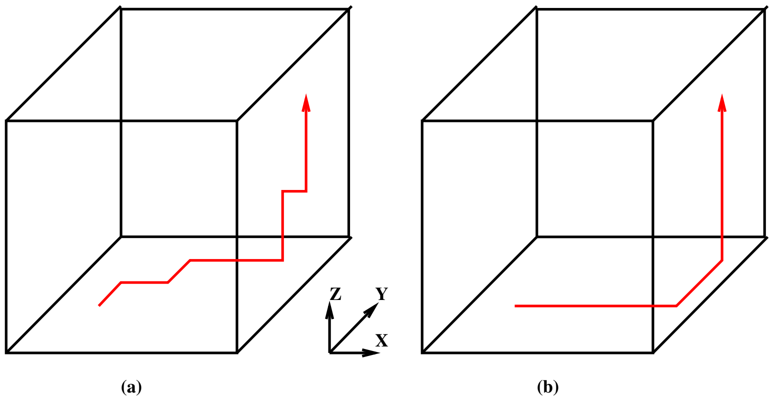

Figure 4.1: Minimally Adaptive Vs. Dimension-Ordered Path.

in dimension d matches the address of the message destination in the same dimension. If the addresses match, then the packet continues to route in the next lower dimension $d-t$, $t > 0$ where the current channel address and the destination address differ.

Figure 4.1 contrasts a minimally adaptive vs. a dimension-ordered message path in a 3-dimensional mesh. The order of dimensions in this case is {x,y,z}.

Dimension-Ordered Routing even with a single virtual channel per physical channel (assuming that the network is not toroidal and there are no wraparound paths) has been proven to be deadlock-free [DS87].

Figure 4.2 shows an example of a message path under AR. The message starts to route on Adaptive channels when it enters the network. It follows an adaptive minimal path. At some point in the path, no free Adaptive channels were available and AR started assigning Dimension-Ordered channels. While routing on Dimension-Ordered channels, the message had to match the x address of its destination before routing on the y dimension. Later in the course, Adaptive channels became available again, and the message reached its destination on Adaptive channels. Figure 4.3 shows a flowchart of the routing algorithm.

AR is deadlock-free. There are cyclic dependencies among the Adaptive channels. Yet, AR does not let a message *block* while waiting for an Adaptive channel to become free. The routing function assigns a Dimension-Ordered channel instead. There are no cyclic dependencies among Dimension-Ordered channels. As a result, messages can block on such channels without the risk of forming a deadlocked configuration.

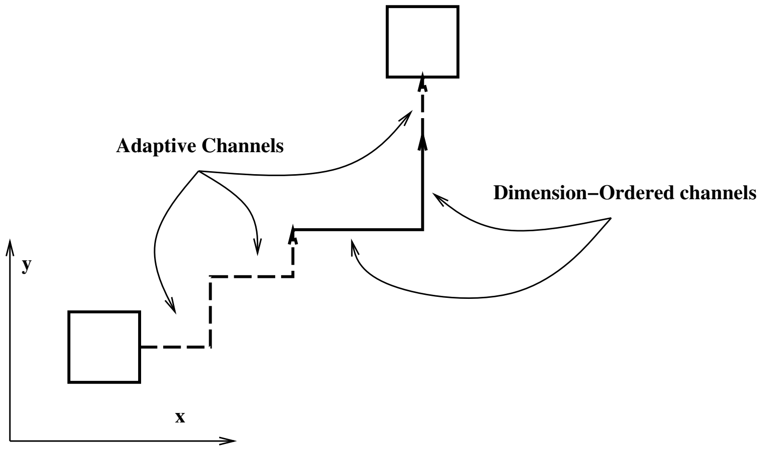

Figure 4.2: Example of a Path Under Adaptive Routing.

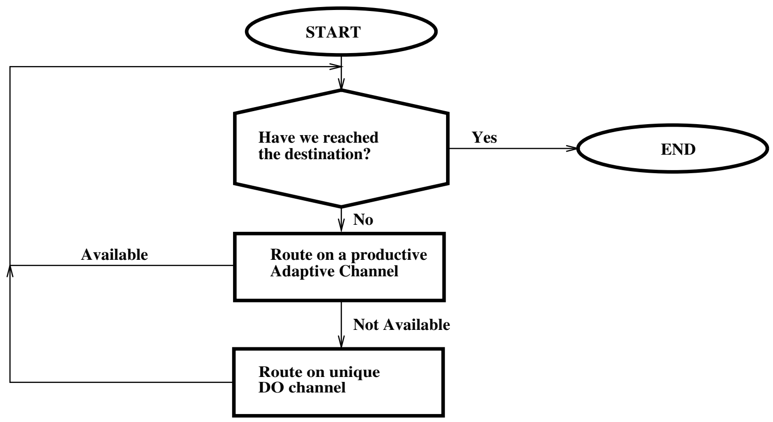

Figure 4.3: Flowchart of Adaptive Routing

4.2 Formal Description

This section will apply the definitions and the theorems of the previous chapter on AR and give a formal proof that the algorithm is deadlock-free.

4.2.1 Notation

Let x be an n-digit radix-k number. x will be represented as $x_{n-1}x_{n-2}\dots x_{i+1}x_i\dots x_1x_0$ where $x_i < k, \forall i \in [0, n-1]$. x_i will denote the ith digit of x. Numbers in such a format will be used to indicate addresses in a k-ary n-cube interconnection network.

The following notation will be used:

n_x	Any node in a k-ary n-cube, where x is the node adress (n-digit radix-k number.)
$p(n_x)$	The consuming (processor) channel associated with node n_x.
c_i i integer	Any virtual channel.
ca_i i integer	An Adaptive virtual channel.
cd_i i integer	A Dimension-Ordered virtual channel.
free(ca_i)	1 if channel ca_i is not allocated to a message, 0 otherwise.

The subscript i of Adaptive and Dimension-Ordered channels $c\{a,d\}_i$ can be partitioned in a number of the form $drfx$ where:

- d the dimension of the channel.
- r the direction of the channel (1 increasing, 0 decreasing.)
- f productive routing indicator ($k-1-x_d$ for increasing channels, and x_d for decreasing channels.)
- x the network address of the channel source node (n-digit radix-k number.)

Figure 4.4 shows the channel subscripts for all channels c_{drfx} in a *4*-ary *2*-cube.

4.2.2 Defining R and R_1

Channel sets C_1 and C are defined as follows:

$$C_1 = \{cd_i, \ \forall i\} \tag{4.1}$$

$$C = \{c_i, \ \forall i\} = \{cd_i, \ \forall i\} \cup \{ca_i, \ \forall i\} \tag{4.2}$$

$$C - C_1 = \{ca_i, \ \forall i\} \tag{4.3}$$

C is the set of all virtual channels in the network. C_1 contains only the Dimension-Ordered channels and $C - C_1$ contains only the Adaptive channels.

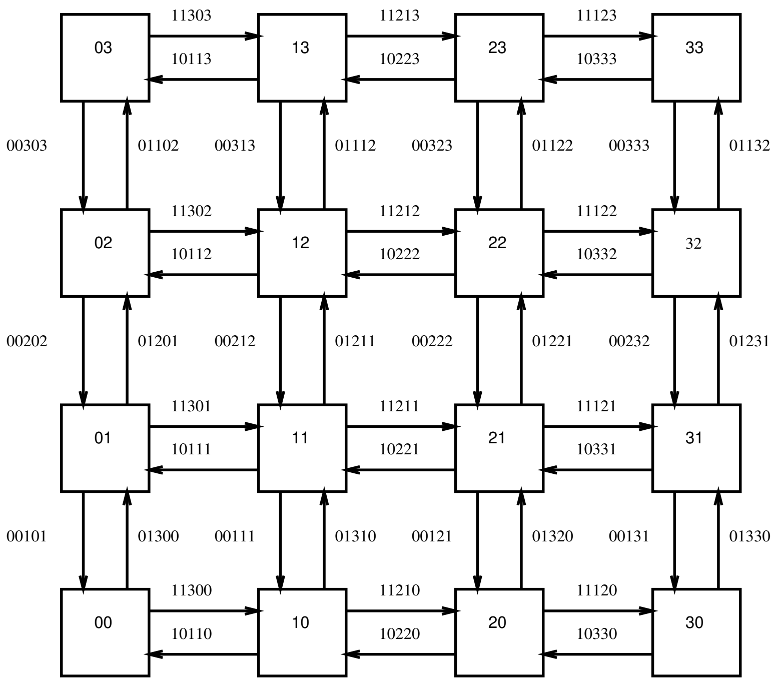

Figure 4.4: Assignment of subscripts to channels in a *4*-ary *2*-cube.

Let $R_1 : C \times N \mapsto C_1$ be the function given by the following definition:

$$R_1(cd_{drfx}, n_j) =$$

$$\begin{cases} p(n_j) & \text{if } \forall m, [x-(-1)^r k^d]_m = j_m \\ cd_{dr[f-1][x-(-1)^r k^d]} & \text{if } [x-(-1)^r k^d]_d \neq j_d \\ cd_{d'1[k-1-x_{d'}][x-(-1)^r k^d]} & \text{if } (\forall m > d', [x-(-1)^r k^d]_m = j_m) \ \wedge x_{d'} < j_{d'} \\ cd_{d'0[x_{d'}][x-(-1)^r k^d]} & \text{if } (\forall m > d', [x-(-1)^r k^d]_m = j_m) \ \wedge x_{d'} > j_{d'} \end{cases}$$

$$R_1(ca_{drfx}, n_j) =$$

$$\begin{cases} p(n_j) & \text{if } \forall m, [x-(-1)^r k^d]_m = j_m \\ cd_{d'1[k-1-x_{d'}][x-(-1)^r k^d]} & \text{if } (\forall m > d', [x-(-1)^r k^d]_m = j_m) \ \wedge [x-(-1)^r k^d]_{d'} < j_{d'} \\ cd_{d'0[x_{d'}][x-(-1)^r k^d]} & \text{if } (\forall m > d', [x-(-1)^r k^d]_m = j_m) \ \wedge [x-(-1)^r k^d]_{d'} > j_{d'} \end{cases} \tag{4.4}$$

For clarity purposes, the definition of R_1 has been broken in two parts: The first part defines R_1 on domain $C_1 \times N$ while the second defines it on $(C - C_1) \times N$. The union of the two domains is $C \times N$. Function R_1 is nothing but the widely used dimension-ordered routing function. This particular routing function routes in decreasing dimension order. This definition is very similar to the one which Dally [DS87] defines, but here it is augmented to accomodate bidirectional channels.

Assertion 4.1 *The Channel Dependency Graph of routing function R_1 is acyclic.*

Proof It is evident from Equation 4.4 that R_1 always assigns a virtual channel with a decreasing subscript. More formally:

$$\forall c_i, c_j \in C_1, \forall n_c \in N : \exists R_1(c_i, n_c) = \ c_j \Rightarrow i > j. \tag{4.5}$$

Therefore, the Channel Dependency Graph is acyclic. □

Let us consider a k-ary n-cube and a routing function $R_a : C \times N \mapsto (C - C_1)^p$ given by the following definition:

$$R_a(c_{drfx}, n_j) = \{ca_{d'r'f'x'} : free(ca_{d'r'f'x'}) = 1\} \tag{4.6}$$

The subscript $d'r'f'x'$ is given by the following equations:

$$d' \ \in \ \{0, 1, \ldots, n-1\} \ \wedge \ [x - (-1)^r k^d]_{d'} \neq j_{d'} \tag{4.7}$$

$$r' \ = \ \begin{cases} r & d' = d \\ 0 & d' \neq d \ \wedge \ [x - (-1)^r k^d]_{d'} > j_{d'} \\ 1 & d' \neq d \ \wedge \ [x - (-1)^r k^d]_{d'} < j_{d'} \end{cases} \tag{4.8}$$

$$f' \ = \ \begin{cases} f - 1 & d' = d \\ x_{d'} & d' \neq d \ \wedge \ [x - (-1)^r k^d]_{d'} > j_{d'} \\ k - 1 - x_{d'} & d' \neq d \ \wedge \ [x - (-1)^r k^d]_{d'} < j_{d'} \end{cases} \tag{4.9}$$

$$x' \ = \ [x - (-1)^r k^d] \tag{4.10}$$

It is important to note that R_a can assign a *set* of possible channels for the next step in the message route. A selection function $S : (C - C_1)^p \mapsto C - C_1$ will have to be applied to pick the single virtual channel on which the message will be routed next. S can be deterministic or stochastic and is of limited importance. In certain cases it is possible to have $R_a = \emptyset$. This means that all the adaptive channels ca_i for some i are allocated to other messages.

Although the definition looks complicated, R_a is nothing but a minimal adaptive routing function. Equation 4.7 simply says that any channel can be assigned as a possible next step as long as it moves the message closer to its destination.

We are now ready to define routing function $R : C \times N \mapsto C^p$.

$$\forall (c,n) \in C \times N \quad R(c,n) = R_1(c,n) \cup R_a(c,n) \tag{4.11}$$

Note that R_1 is a subfunction of R by the definition of Chapter 3.

4.3 Facts about R and R_1

This section states and proves certain facts about functions R and R_1 that will be used to prove that routing function R is deadlock-free.

Assertion 4.2 *Let cd_{d1fx} and $cd_{d0f'x'}$ be two Dimension-Ordered channels. There can be no indirect dependency from cd_{d1fx} on $cd_{d0f'x'}$.*

This assertion merely claims that there can be no indirect dependencies between two channels of the same dimension and opposite directions. If such a dependency exists, then this means that at least one of the intermediate adaptive $(C - C_1)$ steps overshot the destination along the d dimension. More formally:

Proof Suppose that there exists an indirect dependency from channel cd_{d1fx} on channel $cd_{d0f'x'}$. Let $ca_{m_1}, ca_{m_2}, \ldots, ca_{m_k}$ be k intermediate adaptive channels. Moreover, let n_g be the message destination. We have:

$$\begin{aligned}
ca_{m_1} &= S(R_a(cd_{d1fx}, n_g)) && (4.12)\\
ca_{m_2} &= S(R_a(ca_{m_1}, n_g)) && (4.13)\\
ca_{m_3} &= S(R_a(ca_{m_2}, n_g)) && (4.14)\\
&\vdots \\
ca_{m_k} &= S(R_a(ca_{m_{k-1}}, n_g)) && (4.15)\\
cd_{d0f'x'} &= R_1(ca_{m_k}, n_g) && (4.16)
\end{aligned}$$

Since cd_{d1fx} is a channel of positive direction, we know that:

$$x_d < g_d \tag{4.17}$$

By directly applying definition 4.6 on cd_{d1fx} and the subsequent adaptive channels, we see that the following must be true:

$$dst(ca_{m_k})_d < g_d \tag{4.18}$$

If the intermediate adaptive channels belong to different dimensions than d, then we see by 4.6 through 4.10 that this won't affect the dth digit of the destination node of each channel. Therefore the above equation holds. If on the other hand there are intermediate adaptive steps along the d dimension, then according to 4.6 through 4.10 this will increase the destination address by tk^d where t is the number of steps along dimension d. In other words the dth digit of the destination will be increased by t radix-k units. In such a case the inequality may be converted to at most an equality: $dst(ca_{m_k})_d = g_d$. Yet, this does not happen because if it did, the address of the destination along the d dimension would be matched and 4.4 cannot assign a dimension d channel as a possible next step route. Therefore 4.18 is valid in all the cases.

Since 4.18 is true, definition 4.4 will assign a channel $cd_{drf'x'}$ with r=1. A channel $cd_{d0f'x'}$ cannot be assigned by routing function R_1 and therefore no dependency may exist from cd_{d1fx} and $cd_{d0f'x'}$. □

By symmetry arguments, the following assertion is also true:

Assertion 4.3 *Let cd_{d0fx} and $cd_{d1f'x'}$ be two Dimension-Ordered channels. There can be no indirect dependency from cd_{d0fx} on $cd_{d1f'x'}$.*

We can consolidate Assertion 4.2 and 4.3 into the following statement:

Assertion 4.4 *Let cd_{drfx} and $cd_{d\overline{r}f'x'}$ be two Dimension-Ordered channels. There can be no indirect dependency from cd_{drfx} on $cd_{d\overline{r}f'x'}$.*

Assertion 4.5 *There is an indirect dependency from cd_{d1fx} on $cd_{d1f'x'}$ if and only if $x_d < x'_d$.*

Proof If we proceed exactly the same way as we did for Assertion 4.2, then we have:

$$x + k^d < x' \Rightarrow x_d < x'_d. \tag{4.19}$$

□

We also need to show the inverse. Let cd_{d1fx} and $cd_{d1f'x'}$ be two Dimension-Ordered channels such that $x_d < x'_d$. Let $g = dst(cd_{d1f'x'})$. Without loss of generality let us assume that:

$$\forall i \in [0, n-1] \quad x_i < g_i \tag{4.20}$$

There exist adaptive channels such that:

$$\begin{aligned}
ca_{01[k-1-(x+k^0)_0][x+k^0]} &= S(R(c_{d1fx}, n_g)) \\
ca_{01[k-1-(x+2k^0)_0][x+2k^0]} &= S(R([\cdot], n_g)) \\
&\vdots \\
ca_{01[k-1-(x+(g_0-x_0)k^0)_0][x+(g_0-x_0)k^0]} &= S(R([\cdot], n_g)) \\
ca_{11[k-1-(x+k^1)_1][x+(g_0-x_0)k^0+k^1]} &= S(R([\cdot], n_g)) \\
ca_{11[k-1-(x+2k^1)_1][x+(g_0-x_0)k^0+2k^1]} &= S(R([\cdot], n_g)) \\
&\vdots \\
ca_{11[k-1-(x+(g_1-x_1)k^1)_1][x+(g_0-x_0)k^0+(g_1-x_1)k^1]} &= S(R([\cdot], n_g)) \\
&\vdots \\
ca_{d1[k-1-(x+k^d)_d][x+(g_0-x_0)k^0+(g_1-x_1)k^1+\cdots+(g_{d-1}-x_{d-1})k^{d-1}+k^d]} &= S(R([\cdot], n_g)) \\
ca_{d1[k-1-(x+2k^d)_d][x+(g_0-x_0)k^0+(g_1-x_1)k^1+\cdots+(g_{d-1}-x_{d-1})k^{d-1}+2k^d]} &= S(R([\cdot], n_g)) \\
&\vdots \\
ca_{d1[k-1-(x+(g_d-x_d+1)k^d)_d][x+(g_0-x_0)k^0+\cdots+(g_{d-1}-x_{d-1})k^{d-1}+(g_d-x_d+1)k^d]} &= S(R([\cdot], n_g)) \qquad (4.21) \\
&\vdots \\
ca_{[n-1]1[k-1-(x+k^{n-1})_{n-1}][x+(g_0-x_0)k^0+\cdots+(g_{n-2}-x_{n-2})k^{n-2}+k^{n-1}]} &= S(R([\cdot], n_g)) \\
ca_{[n-1]1[k-1-(x+2k^{n-1})_{n-1}][x+(g_0-x_0)k^0+\cdots+(g_{n-2}-x_{n-2})k^{n-2}+2k^{n-1}]} &= S(R([\cdot], n_g)) \\
&\vdots \\
ca_{[n-1]1[k-1-(x+(g_{n-1}-x_{n-1}+1)k^{n-1})_{n-1}][x+(g_0-x_0)k^0+\cdots+(g_{n-1}-x_{n-1})k^{n-1}]} &= S(R([\cdot], n_g)) \\
cd_{d1f'x'} &= S(R([\cdot], n_g)) \qquad (4.22)
\end{aligned}$$

In the above equations the running symbol $[\cdot]$ which appears in each equation refers to the left hand side of the previous equation. It is important to note that this particular choice of adaptive channels for the intermediate steps does not match the destination address along the d dimension (equation 4.21) so that the indirect dependency between cd_{d1fx} and $cd_{d1'x'}$ may exist as indicated by equation 4.22 given our particular choice for the message destination. Therefore $x_d < x_{d'}$ constitutes both a necessary and sufficient condition for an indirect dependency. □

Using the same method, we can also prove the following statement:

Assertion 4.6 *There is an indirect dependency from cd_{d0fx} on $cd_{d0f'x'}$ if and only if $x_d > x'_d$.*

Assertion 4.7 *If there is an indirect dependency from cd_{drfx} on $cd_{d'rf'x'}$ and $d' \neq d$ then $d' < d$.*

Proof Let cd_{drfx} and $cd_{d'rf'x'}$ be two Dimension-Ordered channels such that there is an indirect dependency from cd_{drfx} on $cd_{d'rf'x'}$. Let n_g be the message destination. We know that cd_{drfx} was the result of applying R_1 on a channel c_i such that $dst(c_i) = x$, and the message destination n_g:

$$R_1(c_i, n_g) = cd_{drfx} \tag{4.23}$$

From definition 4.4 we draw the conclusion that:

$$\forall d'' > d \quad g_{d''} = x_{d''} \tag{4.24}$$

In such a case equations 4.6 through 4.10 won't assign a channel of dimension $d'' > d$. Let ca_j be the last Adaptive channel used before $cd_{d'rf'x'}$. It must be that $src(ca_j)_{d''} = g_{d''} \quad \forall d'' > d$. Application of R_1 on (ca_j, n_g) will therefore yield $cd_{d'rf'x'}$ with $d' < d$ if d' is to be different from d. □

4.4 Proving Freedom from Deadlock

We have a rich set of facts about routing functions R and R_1. These facts can help us prove in a couple of steps that routing function R is deadlock-free.

Assertion 4.8 *Let $cd_{drfx}, cd_{d'r'f'x'} \in C_1$ be two Dimension-Ordered channels. If there exists an indirect dependency from cd_{drfx} to $cd_{d'r'f'x'}$ then it must be that $drfx > d'r'f'x'$.*

Proof There are two distinct cases. If $d \neq d'$ then from Assertion 4.7 we have $d' < d$. Therefore $drfx > d'r'f'x'$.

Now let us assume that $r = 1$. From Assertion 4.4 we know that $r' = r$. If $d = d'$ then from Assertion 4.5 we have the following:

$$\begin{aligned} x_d &< x'_d && (4.25) \\ -x_d &> -x'_d && (4.26) \\ k-1-x_d &> k-1-x'_d && (4.27) \\ f &> f' && (4.28) \\ drfx &> d'r'f'x' && (4.29) \end{aligned}$$

If $r = 0$ on the other hand (using Assertion 4.6) :

$$\begin{aligned} x_d &> x'_d && (4.30) \\ f &> f' && (4.31) \\ drfx &> d'r'f'x' && (4.32) \end{aligned}$$

And the proof is complete. □

Finally we come to our conclusion:

Assertion 4.9 *Routing function R is deadlock-free.*

Proof Let $cd_i, cd_j \in C_1$ be two Dimension-Ordered channels. If there is a direct dependency from cd_i to cd_j then from Assertion 4.1 we know that $i < j$. If on the other hand there is an indirect dependency from cd_i to cd_j then from Assertion 4.8 we know that $i < j$. Therefore if there exists *any* dependency either direct or indirect from cd_i to cd_j it is true that $i < j$. Therefore, there are no cycles in the Extended Channel Dependency Graph of function R and by Theorem 3.2 R is deadlock-free.

Before completing the proof we must make sure that condition 3.8 we have set is true:

$$R \cap C_1^p = (R_1 \cup R_a) \cap C_1^p = C_1 \neq \emptyset \qquad (4.33)$$

This completes the proof. □

4.5 Summary

In this chapter we have formally defined a fully adaptive routing function R. R needs two virtual channels per physical channel, one Adaptive and one Dimension-Ordered. R makes messages follow minimally adaptive paths towards their destination. If Adaptive channels are not available at some step due to network contention, R will assign a Dimension-Ordered channel as the unique next step route. If Adaptive channels become free, then R will resume assigning Adaptive channels. We have stated and proven a number of facts about R which led us to the conclusion that R is deadlock-free by directly applying Duato's Theorem (3.2).

R is not fault-tolerant. In the next chapter we will augment R with the capability to handle a single non-transient fault at a time *anywhere* in the network. A lot of the facts stated in this chapter will be used and exploited in the rest of this document.

Chapter 5

Reliable Adaptive Routing

This chapter augments Adaptive Routing and enables it to handle a single link failure anywhere in the network, even along the edges. To achieve this, we need to introduce one more virtual channel per physical channel: The Fault-Handling channel. We call the resulting routing algorithm Reliable Adaptive Routing. The first section will discuss the particular fault model assumed in the design of the algorithm. Next, an informal description of the algorithm will be given along with routing examples. Then, the algorithm will be formally defined and finally a proof of deadlock-freedom will be presented.

5.1 Fault Model

Our fault model is based on the following assumptions:

- We consider as fault a link failure anywhere in the network. A fault can be detected by using parity or any other error detecting code on the wires of each physical link in the network.
- Faults are non-transient. This means that as soon as a parity error is detected on a link, the link is flagged as faulty(0). If the parity error is transient and goes away, the link status is not reset to 1. A link can only be returned to service through human intervention using a dedicated diagnostic port. Such an action does not disrupt the rest of the system.
- There can be only one fault at a time.

Adaptive Routing as presented in the previous chapter cannot always assign a next step route in the presence of faults that conform to this model. Figure 5.1 shows two examples of such faults.

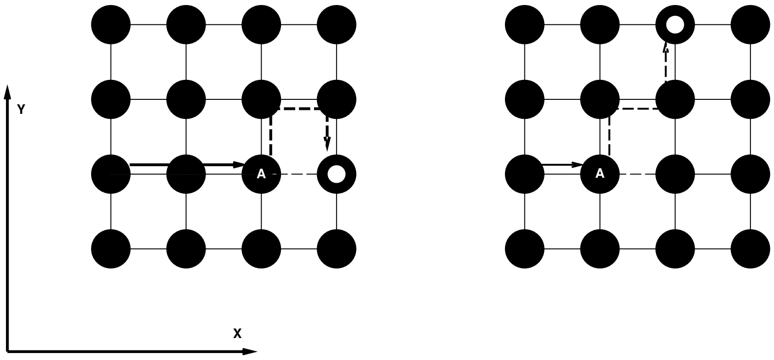

Figure 5.1: Faults that cannot be handled by Adaptive Routing.

In both cases, the destination of the path is indicated by a white dot. In the first case, a message has reached the y address of its destination through Adaptive channels and proceeds to reach the x address too through either Adaptive or Dimension-Ordered channels. There is a broken link in the way. Since R is a minimal adaptive function, the dotted path cannot be allocated and there is no way that this particular message will reach the destination. In such a case if c is the current virtual channel and n is the message destination, we have:

$$R(c, n) = \emptyset \tag{5.1}$$

In the second example equation 5.1 will hold as well if the single eligible Adaptive channel that departs from node A (the $y+$ channel) is busy.

In both cases we see that R clearly needs to be augmented with the ability to assign paths such as the dotted paths shown in figure 5.1.

5.2 Informal Description

This section will give an informal description of Reliable Adaptive Routing. RAR is almost the same with Adaptive Routing. The only difference is that RAR provides an extra class of virtual channels to accomodate paths such as the dotted paths of Figure 5.1.

5.2.1 Examples

First, we will describe RAR through some examples. The following examples assume that the dimension order assumed by routing function R_1 (4.4) is {x,y}. In other words x is dimension 1 and y is dimension 0. Figure 5.2(a) shows an example of routing around a fault that occurs along the x dimension. In such a case, when the message is in node A, RAR will assign the dotted Fault-Handling channel and allow the message to take a non-minimal side step. After the side step has been taken, the message can continue to use Dimension-Ordered or Adaptive channels as it did before. In this case, the side step was taken along the y dimension. Since this constitutes a misrouting step, we know that it has to be reversed in the future. As we can see from Figure 5.2(a), this step is reversed just before the message reaches its destination. Reversing the misrouting step does not introduce any extra dependencies among C_1 channels. Reversing the step involves a step from an x channel to a y channel. Such a dependency already exists in the Extended Channel Dependency Graph of routing function R. This is the rationale behind allowing the message to use Dimension-Ordered channels after it has used the Fault-Handling channel: No additional dependencies are created and therefore no deadlock is introduced.

Case (b) is very similar to case (a) in that the message can continue to use Dimension-Ordered channels after routing on a Fault-Handling channel without introducing extra dependencies.

On the other hand, when the fault occurs along the y dimension as shown in Figure 5.2(c)(d) things are different. In this case the misrouting step has to occur along the x dimension which is a higher dimension than the one we are currently routing on. If this misrouting step is reversed on Dimension-Ordered channels, then additional dependencies must be added to the Extended Channel Dependency Graph from Dimension-Ordered y channels to Dimension-Ordered x channels. Such an action introduces potential deadlocks.

5.2.2 The Algorithm

An informal definition of Reliable Adaptive Routing follows:

1. First find a free and non-faulty Adaptive channel which brings the message closer to its destination.

2. If such a channel is not found, find the unique Dimension-Ordered channel that corresponds to the current channel position of the message and its destination.

3. If the unique Dimension-Ordered channel is not available due to a fault, then find a productive Fault-Handling Channel.

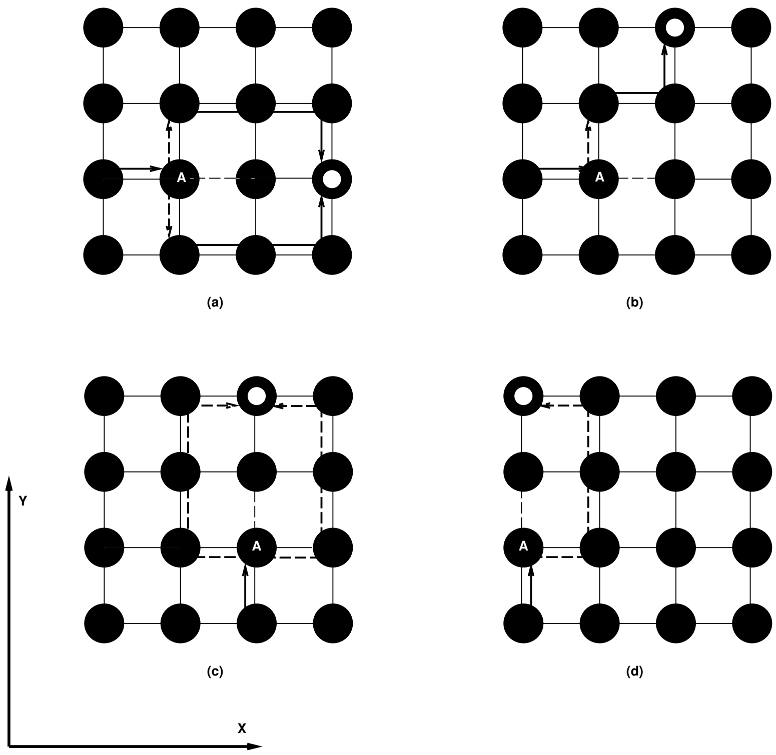

Figure 5.2: Faults handled by Reliable Adaptive Routing.

4. If a productive Fault-Handling channel is not available because the destination address only differs in a single dimension from the current address then if the unmatched dimension is different from dimension 0 do the following steps:

   (a) Route on an unproductive Fault-Handling channel along a dimension lower from the single unmatched dimension at this time.

   (b) Resume routing from step 1.

   If on the other hand, the unmatched dimension is 0 execute the following steps:

   (a) Route on an unproductive Fault-Handling channel along dimension 1.

   (b) Route on productive Fault-Handling channels along dimension 0 until the address of the destination in dimension 0 is reached.

   (c) Route on the Fault-Handling channel along dimension 1 which will bring the message to its final destination.

The algorithm described above is capable of assigning channels that form the message paths shown in Figure 5.2. We see that RAR can bypass a fault even if the fault occurs along the edges of the network. A flowchart of the Reliable Adaptive Routing algorithm is shown in Figure 5.3.

5.3 Formal Definition

This section presents a formal definition of the routing algorithm that has been described in the previous section.

5.3.1 Notation

The notation of section 4.2.1 will be used along with the following additions:

cf_i	A Fault-Handling virtual channel.
valid(c_i)	1 if channel c_i is in service, 0 if it is faulty.

5.3.2 Defining R and R_2

Proof of deadlock freedom will be based again on Theorem 3.2. We need to redefine routing function R and define a routing subfunction R_2 with range C_2 so that the requirements of

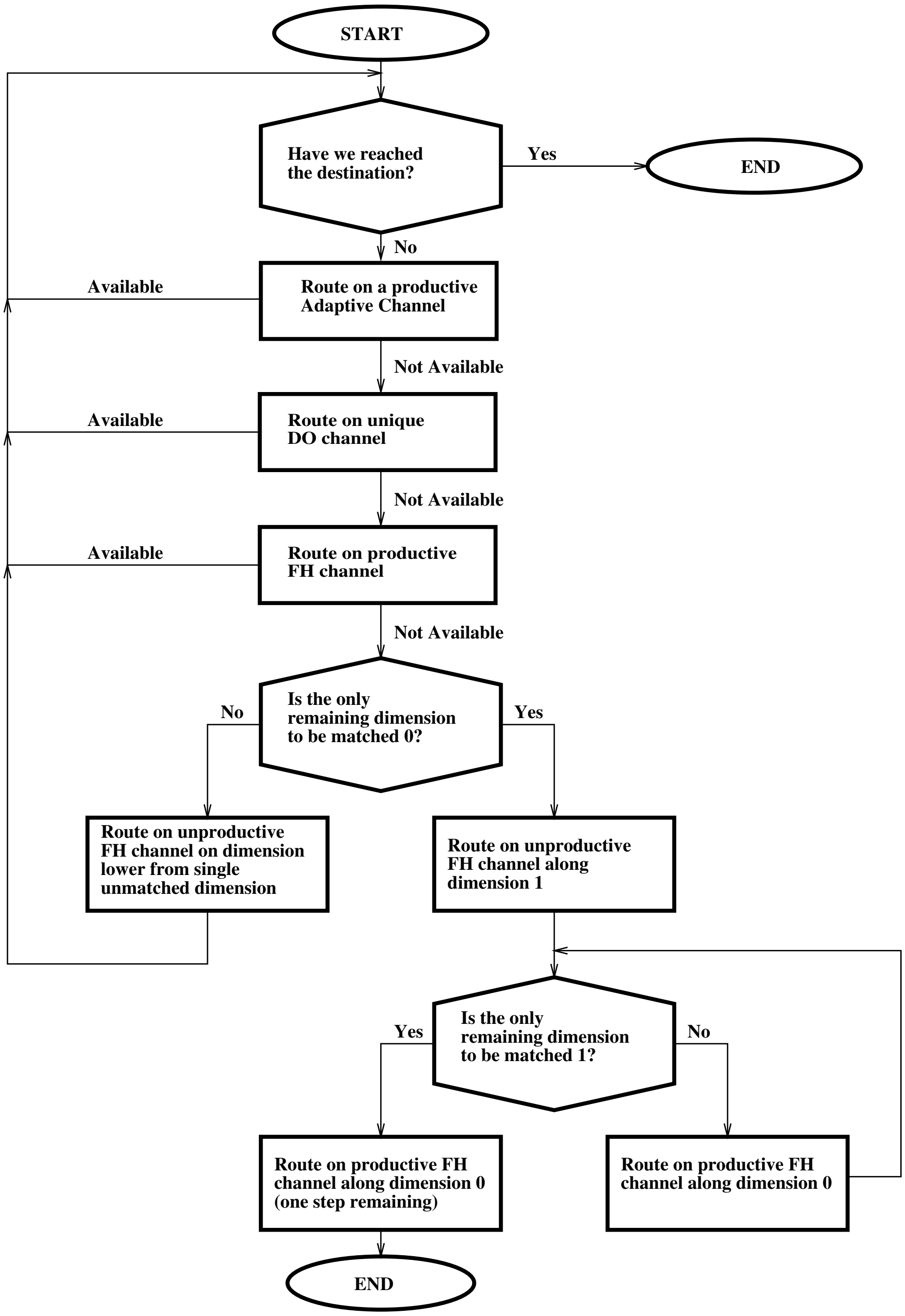

Figure 5.3: Flowchart of Reliable Adaptive Routing

the theorem are satisfied. Channel sets C_2 and C are defined as follows:

$$\begin{aligned} C_2 &= \{cd_i,\ \forall i\} \cup \{cf_i,\ \forall i\} && (5.2)\\ C &= \{c_i,\ \forall i\} = \{cd_i,\ \forall i\} \cup \{ca_i,\ \forall i\} \cup \{cf_i,\ \forall i\} && (5.3)\\ C - C_2 &= \{ca_i,\ \forall i\} && (5.4) \end{aligned}$$

C is again the set of all virtual channels in the network. C_2 contains the Dimension-Ordered channels and the Fault Handling channels. $C - C_2$ contains only the Adaptive channels.

The Fault-Handling channels *must* be included in channel subset C_2 because subset C_2 must be connected for Theorem 3.2 to be valid.

Let R_1 be the function defined in 4.4. Routing function R_2 will be defined on two disjoint domains for reasons of clarity and simplicity. Let $c\{a,d\}_{drfx}$ indicate an Adaptive or Dimension-Ordered channel. R_2 defined on such channels is given by the following definition:

$$R_2(c\{a,d\}_{drfx}, n_j) = \begin{cases} R_1(c\{a,d\}_{drfx}, n_j) & \text{if } \exists c \in R_1 : \text{valid } (c) = 1 \\ cf_{d'r'f'x'} & \text{otherwise} \end{cases} \tag{5.5}$$

Subscripts d', r', f', x' are given by the following equations:

$$\begin{aligned} d' &= \begin{cases} dp & \exists dp, dp' : [x - (-1)^r k^d]_{\{dp,dp'\}} \neq j_{\{dp,dp'\}} \ \wedge \ dp \neq dp' \\ du & \exists d'', d''' : [x - (-1)^r k^d]_{\{d'',d'''\}} \neq j_{\{d'',d'''\}} \ \Rightarrow d'' = d''' \\ & \quad \wedge\ \exists du : du < d'' \\ 1 & \text{otherwise} \end{cases} && (5.6)\\ f' &= \begin{cases} x_{d'} & [x - (-1)^r k^d]_{d'} > j_{d'} \\ k - 1 - x_{d'} & [x - (-1)^r k^d]_{d'} < j_{d'} \end{cases} && (5.7)\\ x' &= x - (-1)^r k^d && (5.8) \end{aligned}$$

When R_2 is defined on Fault-Handling channels, it is given by the following definition:

$$R_2(cf_{drfx}, n_j) = \begin{cases} p(n_j) & \text{if } \forall m, [x - (-1)^r k^d]_m = j_m \\ R_1(cd_{drfx}, n_j) & |[x - (-1)^r k^d]_1 - j_1| \neq 1 \\ cf_{01[k-1-x_0][x-(-1)^r k^d]} & |[x - (-1)^r k^d]_1 - j_1| = 1 \ \wedge \ x_0 < j_0 \\ cf_{00x_0[x-(-1)^r k^d]} & |[x - (-1)^r k^d]_1 - j_1| = 1 \ \wedge \ x_0 > j_0 \\ cf_{11[k-1-x_1][x-(-1)^r k^d]} & |[x - (-1)^r k^d]_1 - j_1| = 1 \ \wedge \ x_0 = j_0 \ \wedge x_1 < j_1 \\ cf_{10x_1[x-(-1)^r k^d]} & |[x - (-1)^r k^d]_1 - j_1| = 1 \ \wedge \ x_0 = j_0 \ \wedge x_1 > j_1 \end{cases} \tag{5.9}$$

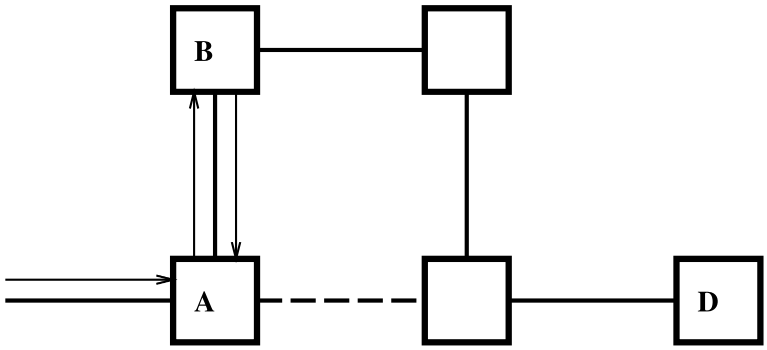

Figure 5.4: Undesirable minimal step

Let R_{a2} be a minimal adaptive routing function much like the one defined in 4.6 through 4.10. One difference between R_a and R_{a2} is shown in the following equation.

$$R_{a2}(c\{a,d,f\}_{drfx}, n_j) = \{ca_i : ca_i \in R_a(c\{a,d\}_{drfx}, n_j), valid(ca_i) = 1\} \tag{5.10}$$

It simply says that the minimal adaptive function R_{a2} can only pick channels that are not faulty. Except for this trivial constraint, there is an additional one which stems from the fact that misrouting steps are now possible: If no further constraints are imposed to the minimal adaptive routing function, then an undesirable situation may occur just after a misrouting step. This is illustrated in Figure 5.4.

Suppose that a message misroutes due to a fault because the address of its current channel differs from the destination address in only one dimension. When the head of the message reaches node B, then the minimal adaptive routing function as defined in Equations 4.6 through 4.10 and Equation 5.10 may immediately reverse the misrouting step and direct the message back to node A (node D in the Figure indicates the message destination.). This is clearly undesirable. We therefore need to place extra restrictions on the minimal adaptive routing function so that misrouting steps are not immediately reversed. We can formulate this as follows: The minimal adaptive routing function R_{a2} may not assign a next step channel of the same dimension but opposite direction when the message reaches the node from Fault-Handling channels. More formally:

$$R_{a2}(cf_{drfx}, n_j) = R_a(c\{a,d\}_{drfx}, n_j) - \{ca_{d\overline{r}f'[x-(-1)^r k^d]}\} \tag{5.11}$$

where f' is given by the following equation:

$$f' = \begin{cases} x_d & \overline{r} = 0 \\ k - 1 - x_d & \overline{r} = 1 \end{cases} \tag{5.12}$$

Now that we have all the routing subfunctions of R we are one step away from the definition of Reliable Adaptive Routing:

$$\forall(c,n) \in C \times N \quad R(c,n) = R_2(c,n) \cup R_{a2}(c,n) \tag{5.13}$$

From the definitions of R_2 and R_{a2} it is evident that defining a routing function on domain $C \times N$ instead of $N \times N$ has been necessary for defining Reliable Adaptive Routing without resorting to other non-elegant schemes that would allow us to provide the routing function with some sort of memory. Reliable Adaptive Routing makes use of the fact that an incoming message comes in from Fault-Handling channels. The next step assignment depends on the channel kind of an incoming message. Yet, RAR does not require modification of the header or any other message state. The routing decision as can be seen from the previous definitions depends only on local information. Never does the routing decision depend on a past routing event. This greatly simplifies the implementation of the algorithm and reduces system complexity.

5.4 Facts About Reliable Adaptive Routing

Before proceeding with the proof of deadlock freedom it is worthwhile to examine the properties of the Fault-Handling channels that are added to the C_1 channel subset. Clearly, the Fault-Handling channels form an independent virtual network on top of the Adaptive and the Dimension-Ordered virtual networks that already exist. Yet, we do not wish to adopt this point of view for two reasons:

1. Not *all* Fault-Handling channels are actually used by the routing algorithm. There is one Fault-Handling channel per physical channel. Yet, the Fault-Handling channels that can actually be assigned by the routing algorithm are very limited, given that there can be only one fault at a time.

2. The Fault-Handling channels must be viewed as an addition to the Dimension-Ordered virtual network for restoring connectivity. This is the only approach that will allow us to use Duato's theorem to justify the claim that Reliable Adaptive Routing is deadlock-free.

The location and number of Fault-Handling channels depends directly on the location of the fault within the network. We are exploring this issue in the following sections.

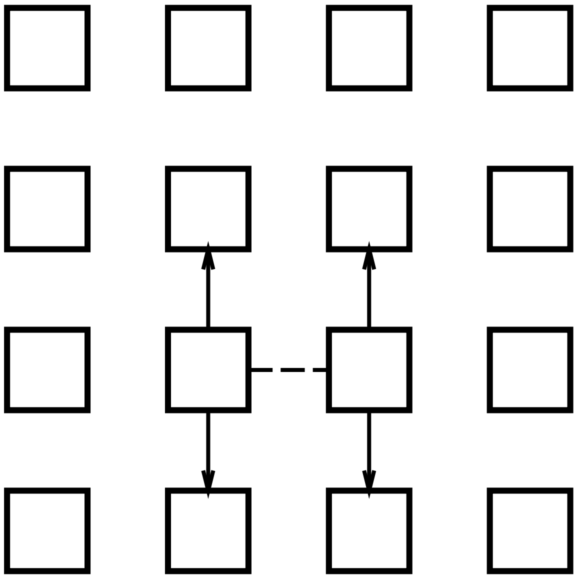

Figure 5.5: Addition of Fault[Handling channels when a fault occurs along a dimension $d \neq 0$. This figure shows the two-dimensional case when the fault occurs along dimension 1.

5.4.1 Fault Along a Dimension $d \neq 0$

Reliable Adaptive Routing will put into service a maximum of $4d$ Fault-Handling channels if we assume that dimension numbering starts from 0. This can be seen in Figure 5.5 for two dimensions and Figure 5.6 for three dimensions. We must note that only Fault-Handling channels of dimension $d' < d$ may be assigned by the routing algorithm and for this reason the number of Fault-Handling channels differs in Figures 5.6 (a) and (b). As was mentioned before, the rationale behind this restriction is that when a misrouting step occurs along a low dimension, this misrouting step can later be reversed without violating the order of dimensions and therefore without adding extra dependencies to the Extended Channel Dependency Graph. The number of added Fault-Handling channels is limited because Reliable Adaptive Routing only assigns Fault-Handling channels on the first step that a message is taking after a fault has affected regular channel assignment.

5.4.2 Fault Along Dimension 0

When a fault occurs along Dimension 0, then according to RAR only dimension 0 and dimension 1 Fault-Handling channels are assigned. Figure 5.7 shows the Fault-Handling

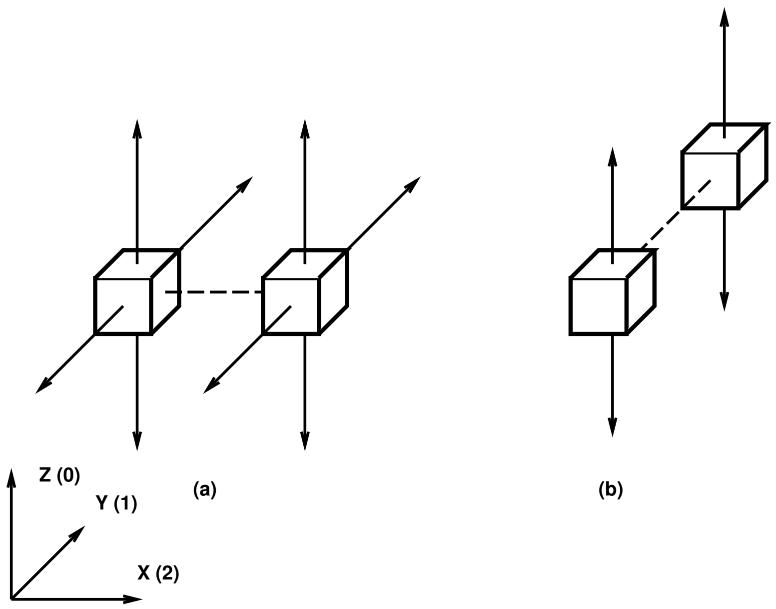

Figure 5.6: Addition of Fault-Handling channels when a fault occurs along a dimension $d \neq 0$. This figure shows the three-dimensional case when the fault occurs along dimensions 2 and 1.

channels that are put into service in such a case. In this case the number of Fault-Handling channels that are added does not depend on the network dimension but only on the network radix k. The total number of added Fault-Handling channels is $12 + 4(k - 2)$. Applying this formula to Figure 5.7 where $k = 6$ we see that we have a total of 28 added Fault-Handling channels.

Understanding the location and number of the Fault-Handling channels that are added to restore connectivity to the C_1 channel subset is essential to to our approach of proving that Reliable Adaptive Routing is deadlock-free.

5.5 Proof of Deadlock Freedom

Without loss of generality and for purposes of simplicity, we will restrict ourselves to two dimensions x (dimension 1) and y (dimension 0). In a subsequent section an argument will be given that extends the proof to an arbitrary number of dimensions. Two different cases may arise when a fault occurs. Reliable Adaptive Routing treats differently the case when a fault occurs on an x channel from the case of a fault along the y dimension. The following two sections will treat each case separately.

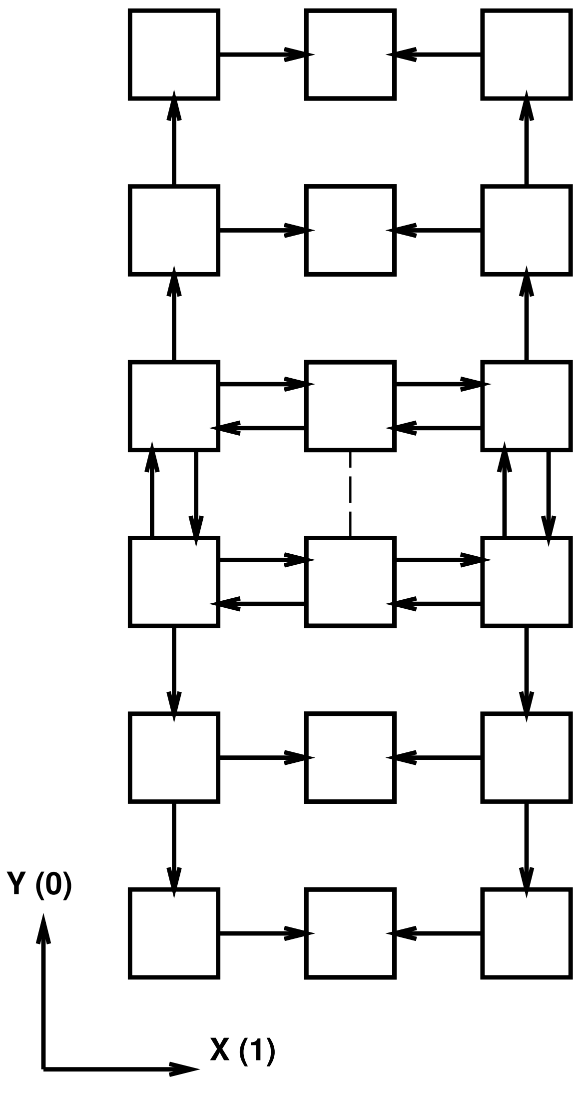

Figure 5.7: Addition of Fault-Handling channels when a fault occurs along dimension 0

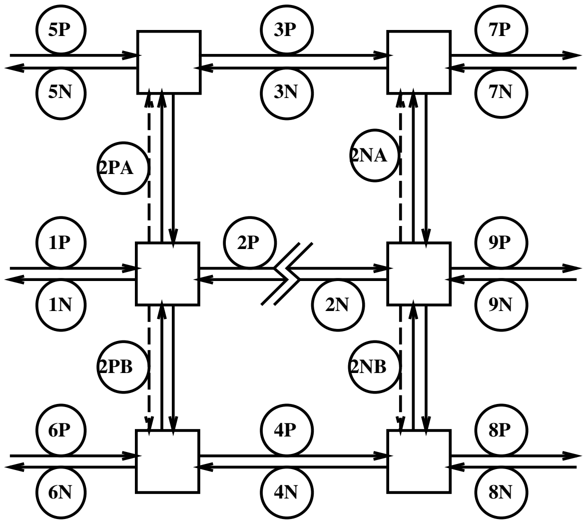

Figure 5.8: Addition of Fault-Handling channels when a fault occurs along the x dimension

5.5.1 Fault along the x dimension

This case is shown in Figure 5.8. The four Fault-Handling channels that have been added to restore connectivity are shown with dotted arrows.

In this case the physical link which carries Dimension-Ordered virtual channels 2P and 2N goes down because a parity error was detected on the lines. Reliable Adaptive Routing restores connectivity in the C_1 channel subset by putting to use Fault-Handling channels 2PA, 2PB, 2NA, 2NB. The proof that the addition of the Fault-Handling channels preserves the fact that the algorithm is deadlock-free is based on a study of the Channel Dependency Graph. The following observation will be helpful in order to simplify our analysis:

Observation 5.1 *If there exist any cycles in the Channel Dependency Graph after the addition of Fault-Handling channels when a fault occurs along the x dimension, these cycles do not contain any nodes that correspond to Dimension-Ordered y channels.*

Discussion There are no direct nor indirect dependencies from any Dimension-Ordered y channel to any Dimension-Ordered x channel before the occurrence of the fault. This can be seen from Equation 4.4 and Assertion 4.7. Moreover, the addition of Fault-Handling channels does not alter the dependency arrows that depart from each Dimension-Ordered y channel in the Channel Dependency Graph. There are three things that we need to consider to see why the above statement is true.

1. When a fault occurs along the x dimension, the routing algorithm continues not to allow a message to use a Dimension-Ordered x channel after it has used a Dimension-Ordered y channel. This means that a fault does not add any new dependencies between Dimension-Ordered y channels and Dimension-Ordered x channels.

2. When a fault occurs along the x dimension, the routing algorithm does not create a dependency between any Dimension-Ordered y channel and a Fault-Handling Channel. A message may not use a Fault-Handling Channel after it has used a Dimension-Ordered y channel. If a message is allowed to use a Dimension-Ordered y channel under Reliable Adaptive Routing, it follows that the message has reached the x dimension of its destination. In such a case, RAR will never assign an x channel as a possible next step, and the message won't perceive that there is fault somewhere on an x channel. It follows that the message will never use a Fault-Handling channel. Therefore, no dependency may exist between a Dimension-Ordered y channel and a Fault-Handling channel when there is a fault along the x dimension.

3. When a fault occurs along the x dimension, the routing algorithm does not create any new dependencies between Dimension-Ordered y channels. The argument is the same as in the previous clause: If a message uses a Dimension-Ordered y channel, it won't ever perceive that there is a fault in the network. Therefore, Reliable Adaptive Routing will create the same dependencies it would create in the fault-free case.

The above three clauses establish the conclusion that a fault along the x dimension and the corresponding addition of Fault-Handling channel nodes and some extra dependencies in the CDG does not cause the addition of extra dependencies that depart from Dimension-Ordered y channels. Therefore, if the addition of new nodes and new dependencies in the Channel Dependency Graph causes the formation of cycles, these cycles will not involve nodes that correspond to Dimension-Ordered y channels, since any path that initiates from a Dimension-Ordered y channel is cycle-free exactly as in the fault-free case.

Observation 5.1 justifies the omission of Dimension-Ordered y channels from the study of the Channel Dependency Graph. Figure 5.9 displays a part of the Channel Dependency Graph that corresponds to Figure 5.8 before the occurrence of the fault. Dimension-Ordered y channels have been omitted.

Remarks on the CDG

The following notation is used in Figure 5.9:

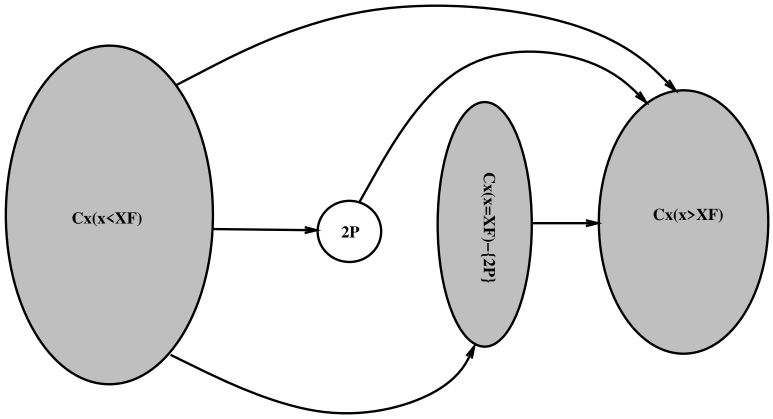

Figure 5.9: Condensed form of the Channel Dependency Graph before the occurrence of a fault. This CDG corresponds to the situation depicted in Figure 5.8 *before* the occurrence of the fault along the x dimension. The gray ovals represent channel sets. The Figure displays only the subgraph that corresponds to positive channels.

XF The x address of the fault. In Figure 5.8, this is the x address of the source node of channel 2P.

$C_x(x =, <, > A)$ The set of all x Dimension-Ordered channels with x-address $x =, <, > A$.

If the Dimension-Ordered y channels are removed from the CDG as we have done in Figure 5.9, the directed graph can be partitioned in two disjoint half-graphs. This can be easily seen be invoking Equation 4.4 and Assertion 4.4: AR does not create dependencies neither direct nor indirect between channels of the same dimension but of different direction. We conclude that the CDG involving only x Dimension-Ordered channels is the union of two completely disjoint directed graphs, the former having as nodes all positive direction channels (from left to right) and the latter having as nodes all negative direction channels (from right to left). Figure 5.9 depicts the subgraph that contains all positive direction channels. We will focus our attention on the positive subgraph since one can apply the exact same analysis on the negative subgraph using symmetry arguments.

The gray ovals represent channel sets and the notation used fully characterizes the contents of each set. The Channel Dependency Graph in its condensed form has only one node that corresponds to individual virtual channels (5P, 6P, 2P). The rest of the channels are collapsed into channel subsets. Each subset is itself an acyclic directed graph. The arrows that depart from each subset represent direct and indirect dependencies of channels in the subset on other channels in other subsets. The arrows that arrive at each subset represent dependencies from channels that belong to other subsets on channels in the particular subset. Each arrow represents more than one dependencies.

Figure 5.9 has been drawn by direct application of Assertion 4.5.

Restructuring the CDG due to a Fault

The key to proving that the algorithm is deadlock-free in this case is understanding exactly the restructuring that takes place in the CDG of Figure 5.9 when the fault occurs and channel 2P goes down. We can identify immediately two things that will change:

1. Node 2P must be deleted from the Channel Dependency Graph.
2. Two nodes that correspond to the Fault-Handling channels 2PA, 2PB must be added to the Channel Dependency Graph.

Two important questions remain to be answered:

1. Which dependency arrows arrive at nodes 2PA, 2PB?
2. Which dependency arrows depart from nodes 2PA, 2PB?

In other words how do we connect these two added nodes to the rest of the graph.

We will address the first question with the following Observation:

Observation 5.2 *The source nodes of the arrows that arrive to each of the nodes 2PA and 2PB in the restructured Channel Dependency Graph form each a subset of the source nodes of the arrows that arrive at 2P in the original Channel Dependency Graph.*

Discussion We can see that easily if we consider a Dimension-Ordered x channel c_x which did not depend on 2P neither through a direct nor through an indirect dependency. By Assertion 4.7 then:

$$x(c_x) \geq x(2P) = XF \tag{5.14}$$

The dependencies that depart from c_x by the same Assertion, arrive at channels c such that $x(c) > XF$. There is no fault associated with that portion of the network, since our fault model allows only a single fault at a time and this fault has occurred already at $x = XF$. Therefore, RAR won't ever assign a Fault-Handling channel as a possible next step route. No dependencies can exist between c_x and any Fault-Handling channel, and therefore no dependencies can exist between c_x and 2PA or c_x and 2PB. The bottom line of this observation is that no new dependencies are introduced from other nodes in the CDG on nodes 2PA and 2PB other than a subset of the original dependencies that arrive at node 2P.

It is worthwhile to note that not *all* the dependencies that arrive at node 2P are transferred to each of the nodes 2PA and 2PB. There can be no dependency between channels 5P and 2PA in Figure 5.8. Such a dependency would imply a non-minimal step in the course of a message. Yet, there is an original dependency between channels 5P and 2P according to Assertion 4.5. No rigorous discussion of this fact will be presented since it is not necessary for the development of the proof.

We will address the second question with another observation:

Observation 5.3 *Let c_x be a Dimension-Ordered x channel. If there exists a dependency from 2PA or 2PB on c_x then $x(c_x) \geq x$(2PA, 2PB).*

Discussion Let us focus on the positive subgraph of Figure 5.9. All messages that perceive the fault at XF have not matched yet the address of their destination along x. More specifically since all the x channels shown are positive we see that the x address of the current channel for such messages is smaller than the x address of their destination. When messages are redirected because of a fault to channels 2PA and 2PB we still have:

$$x(2PA, 2PB) < j_x \tag{5.15}$$

for n_j being a destination node of such a message. After the redirection on the Fault-Handling channels, messages revert to plain Adaptive Routing which is minimal. Successive applications of routing function R on 2PA, 2PB and on the successive outcomes of such computations will only increase the x address of the next step channel or leave it unchanged. Therefore if channel c_x is the result of such a computation then $x(2PA, 2PB) \leq x(c_x) \leq j_x$.

It is worthwhile again to note that the inverse of the previous observation does not hold: Let c_x be a channel such that $x(c_x) \geq x(2PA)$. This does not mean that there is a dependency from 2PA on c_x. As an example, we can consider channels 2PA and 4P in Figure 5.8. If there was such a dependency, that would imply that backtracking would occur on C_2 channels and this is not allowed by the algorithm.

Since we have determined how the nodes 2PA and 2PB are to be connected to the rest of the graph, we are now ready to construct the restructured Channel Dependency Graph after the occurrence of the fault. It is shown in Figure 5.10. It is obvious from the figure that

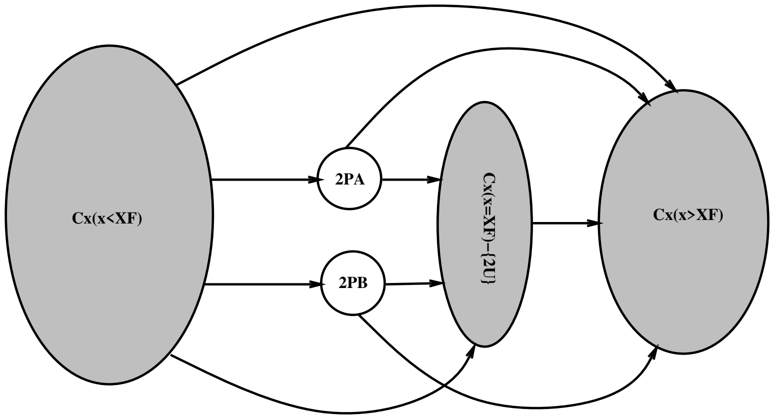

Figure 5.10: The restructured Channel Dependency Graph after the occurrence of a fault along the x dimension.

this restructuring has not introduced any cycles. More specifically: Comparing Figures 5.9 and 5.10 we conclude that Figure 5.10 can be constructed from Figure 5.9 in three simple steps. We are going to examine these steps and show that none of those may produce a cycle in the graph. The steps are illustrated in Figure 5.11.

1. Node 2P is split into two nodes 2PA and 2PB. Everything else stays the same. Each one of the nodes 2PA and 2PB inherits all the arrows that were arriving at node 2P before the occurence of the fault. Moreover, each node inherits all the arrows that were departing from node 2P before the occurence of the fault. This action does not introduce any cycles to the graph. Suppose that a cycle was actually introduced. This cycle could not contain both nodes 2PA and 2PB because there are no dependency arrows between these two nodes. Without loss of generality, let us assume that this cycle contains node 2PA. Such a cycle cannot exist because if it did, it would also exist before splitting node 2P since node 2PA inherits all arriving and departing arrows and everything else stays the same.

2. The next step involves removing some dependencies from the arrow clusters that arrive and depart from nodes 2PA and 2PB. This issue was discussed in Observation 5.2 and 5.3. Obviously, removing dependencies cannot introduce cycles.

3. The third and final step involves adding an arrow cluster from channels 2PA and 2PB to channel subset $C_x(x = XF) - \{2P\}$ according to Observation 5.3. Such an action does not produce a cycle. The case is symmetric and without loss of generality we can

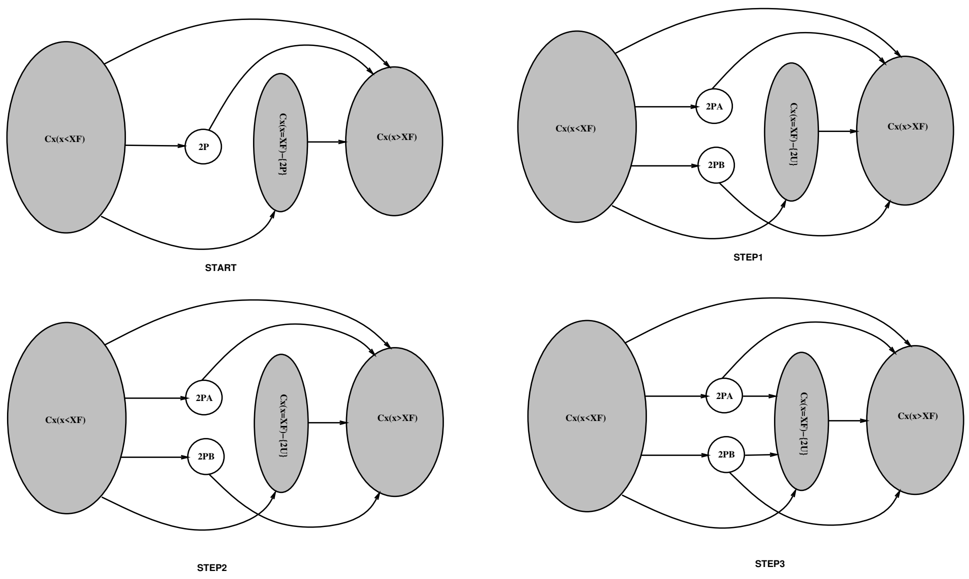

Figure 5.11: Restructuring the Extended Channel Dependency Graph in 3 steps

focus on possible cycles that contain node 2PA. There are three possible ways that a cycle can be formed by adding an arrow cluster from 2PA to $C_x(x = XF) - \{2P\}$:

(a) A cycle that contains 2PA and $C_x(x = XF) - \{2P\}$.

(b) A cycle that contains 2PA, $C_x(x = XF) - \{2P\}$ and $C_x(x < XF)$.

(c) Finally, a cycle that contains 2PA, $C_x(x = XF) - \{2P\}$, $C_x(x < XF)$ and $C_x(x > XF)$

Application of Assertion 4.7 for cases (b) and (c) and application of Assertion 4.7 and Observation 5.2 for case (a) establishes the conclusion that none of the possible cycles may actually exist.

The above discussion has established the following assertion:

Assertion 5.1 *A fault along the x dimension and the resulting addition of Fault-Handling channels by Reliable Adaptive Routing does not introduce any cycles to the Extended Channel Dependency Graph.*

5.5.2 Fault along the y dimension

The Fault-Handling channels that are added by Reliable Adaptive Routing to restore connectivity in this case are shown in Figure 5.7. The proof of deadlock freedom in this case exploits the fact that faults are non-transient and that messages cannot revert back to Dimension-Ordered channels when a Fault-Handling channel has been used.

Assertion 5.2 *A fault along the y dimension and the resulting addition of Fault-Handling channels by Reliable Adaptive Routing does not introduce any cycles to the Extended Channel Dependency Graph.*

Proof Let us add one more digit at the most significant position of the Dimension-Ordered channel subscripts: The full subscript now of a Dimension-Ordered channel is $tdrfx$ where $t = 1$ and the $drfx$ part remains as defined in previous sections. The position of the fault along the y axis divides the xy plane into two parts: The West Half Plane is the part that lies to the west of the fault and the East Half Plane is the part that lies to the east of the fault. For Fault-Handling channels we are employing a subscript of the same format $tdrfx$ where $t = 0$ and where d instead of indicating channel dimension it is used as follows:

$$\begin{array}{ll} d = 3 & \text{if the channel is a West Half Plane } x- \text{ channel} \\ d = 3 & \text{if the channel is an East Half Plane } x+ \text{ channel} \\ d = 2 & \text{if the channel is a } y \text{ channel} \\ d = 1 & \text{if the channel is a West Half Plane } x+ \text{ channel} \\ d = 1 & \text{if the channel is an East Half Plane } x- \text{ channel} \end{array}$$

The remaining digits of the subscript rfx do not change. Digit d of the subscript of the added Fault-Handling channels is shown in Figure 5.12. It is easy to verify from Equation 5.9 that R will assign next step channels that have a decreasing subscript. Therefore the Extended Channel Dependency Graph is acyclic and by Theorem 3.2 R is deadlock-free when a fault occurs along the y dimension. □

5.5.3 Proving Deadlock Freedom In Two Dimensions

Assertion 5.3 *Reliable Adaptive Routing in two dimensions is deadlock-free.*

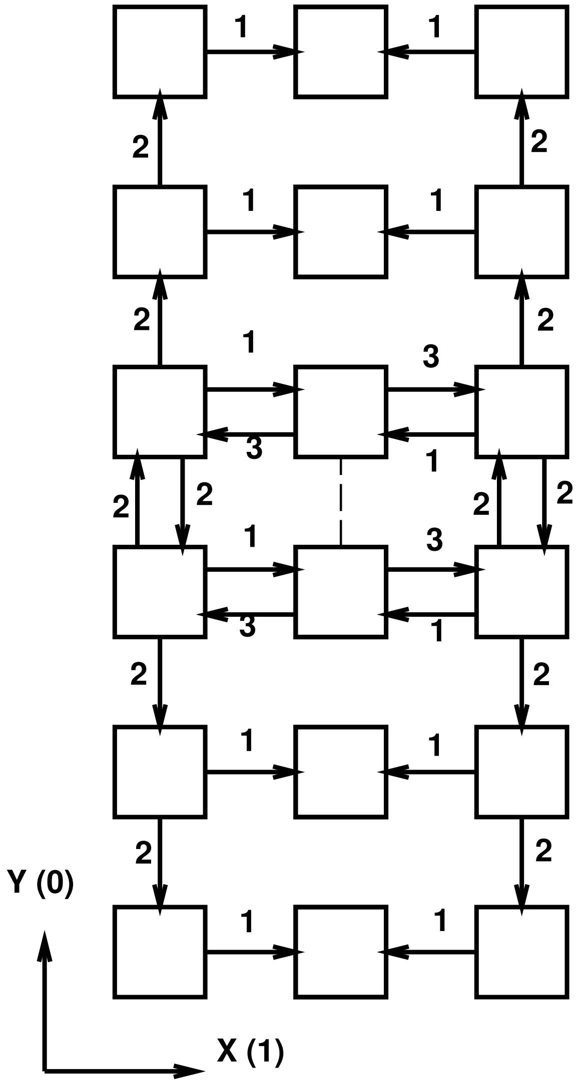

Figure 5.12: dth digit of Fault-Handling channel subscripts

Proof Assertions 5.1 and 5.2 indicate that when a non-transient fault occurs either along the x dimension or along the y dimension, no cycles are introduced to the Extended Channel Dependency Graph. Therefore by Theorem 3.2 Reliable Adaptive Routing is deadlock-free in two dimensions. □

5.6 Extending the Proof to Arbitrary Dimensions

It is very easy to extend the proof of deadlock freedom to a k-ary n-cube with $n > 2$. In this section we will provide a detailed sketch of such a proof.

5.6.1 Fault Along Dimension $n-1$.

In this case, the Channel Dependency Graph before the Fault can be modeled exactly as that of Figure 5.9. The only difference is that all the nodes now are not channels of dimension x but of dimension $n-1$. The CDG after the fault is very similar to the one in Figure 5.10. The only difference is that channel 2P is not split into two separate nodes, but into $2(n-1)$ nodes. The discussion related to the steps necessary to restructure the CDG does not change at all, except for the fact that node 2P has been split into more than two nodes. We conclude that there are no cycles in the Channel Dependency Graph.

5.6.2 Fault Along Dimension d such that $0 < d < n-1$

This is a trivial case. In this case only Fault-Handling channels of dimension $d' < d$ can be added to the C_1 channel subset. The CDG in this case can be derived from the CDG of the previous case by rotating the dimensions, and removing a finite number of nodes and arrows. No cycle can be introduced this way.

5.6.3 Fault Along Dimension 0

This case presents no difference at all from the two-dimensional case. Reliable Adaptive Routing only adds Fault-Handling channels on the plane spanned by dimensions 0 and 1.

5.6.4 Proving Deadock Freedom in n Dimensions

We can formulate the previous discussion in the following Assertion:

Assertion 5.4 *Reliable Adaptive Routing in a k-ary n-cube without wraparound connections is deadlock-free.*

5.7 Summary

This section has presented Reliable Adaptive Routing, an adaptive routing algorithm for k-ary n-cubes without wraparound connections that can tolerate a fault *anywhere* in the network, even along the edges. RAR needs three virtual channels per physical channel: One Adaptive, one Dimension-Ordered and one Fault-Handling. The routing decision under RAR needs only consider local information. It does not depend on any past routing decision except for the fact that a message currently uses Fault-Handling channels. As a result RAR is easy to implement in a real system and does not require message header modification. Reliable Adaptive Routing is provably deadlock-free.

Chapter 6

The Virtual Channel Dependency Analyzer

This chapter describes the Virtual Channel Dependency Analyzer (VCDA), a software tool that helps the designer of a wormhole routing algorithm visualize the dependencies among virtual channels, and also verify that a certain routing algorithm is deadlock-free. VCDA is particularly designed for the Reliable Adaptive Routing algorithm presented in chapter 5. Additional functionality can easily be built into it so that it can work with any wormhole routing algorithm with a small number of virtual channels. The main engine of the Analyzer is written in C. Certain parts (such as the graphical user interface and the graph drawing modules are written in Tcl [Ous94] and use Tk Toolkit widgets.

6.1 Functional Description

The Virtual Channel Dependency Analyzer can perform the following tasks:

1. It creates and displays graphically a structure that represents a 4 by 4 mesh. It displays address and port status information on each network node.

2. It creates and displays graphically a structure that represents the Extended Channel Dependency Graph (chapter 5) that corresponds to Reliable Adaptive Routing. The Extended Dependency Graph includes all the direct and indirect dependencies among the Dimension-Ordered channels of the 4 by 4 mesh. The program can be instructed to display the direct dependencies only, so that the diference between the two types of dependencies can be visualized.

3. The program can be instructed to search for cycles in the Extended Dependency Graph. If it does not find any cycles, it informs the user that the algorithm is deadlock-free. On the other hand, if it succeeds in finding a cycle, it displays the cycle graphically.

4. The user is able to specify a particular fault location by filling out the fault position in a dedicated dialog box. The program marks the link specified as faulty. The user can then instruct the program to display the Fault-Handling channels (chapter 5) that are put into use by RAR. Moreover, the user can obtain a dependency graph that includes only the dependencies that concern Fault-Handling channels. This feature is quite useful for visualization of the alternative paths that are available around a faulty link. The user can then launch again the cycle-search feature and verify that the algorithm is deadlock-free in the presence of faults.

5. Finally, the program can run an exhaustive analysis where it verifies that the algorithm is deadlock-free for all possible fault positions.

6.2 A Sample Session with VCDA

This section essentially constitutes a tutorial on using the Analyzer.

6.2.1 Program Manager

As soon as the program is launched from the command line, it displays the Program Manager window shown in Figure 6.1. All features of the tool are enabled from this window. The commands are grouped according to their functionality.

6.2.2 Building and Displaying the Network

The user must first click on the *Build Network* button. This button enables a C procedure that builds the network data structure. When the structure is built, a Tcl script is invoked which creates a window and graphs the network. This window is shown in Figure 6.2. The white dots on both sides of each link, represent the port status flag associated with each link within each node. White indicates a functional link whereas red indicates a faulty link. In Figure 6.2 all links are functional since we have not specified any faults.

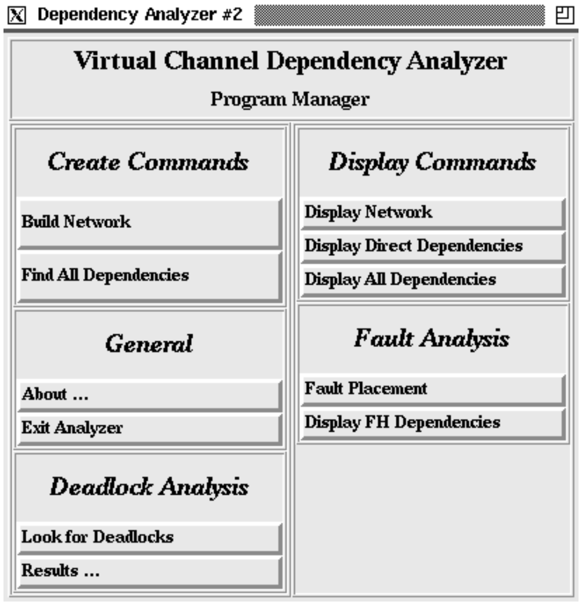

Figure 6.1: The VCDA Program Manager.

The *Display Network* command can be invoked at any time during the session *after* the *Create Network* command has been run. This command simply reproduces the network graph of Figure 6.2 without recreating the underlying network structure.

6.2.3 Creating the Dependency Graph

The next step is to click on the *Create All Dependencies* button. This action invokes a series of routines and scripts that create the Extended Channel Dependency Graph and display it on the screen as shown in Figure 6.3. In the same fashion, the *Display All Dependencies* command can be invoked at any time *after* the *Create All Dependencies* command has been run. It reproduces the graph without rebuilding the underlying structure.

The *Display Direct Dependencies* command may also be invoked after creating all the dependencies. The effect of this command is a graph of the direct dependencies only. This graph is shown in Figure 6.4. Comparing Figures 6.3 and 6.4 can be helpful in visualizing the difference between direct and indirect dependencies.

6.2.4 Looking for Deadlocks

Clicking on the *Look for Deadlocks* button enables the cycle-search algorithm. VCDA almost immediately notifies the user that the analysis is complete and prompts him or her to click

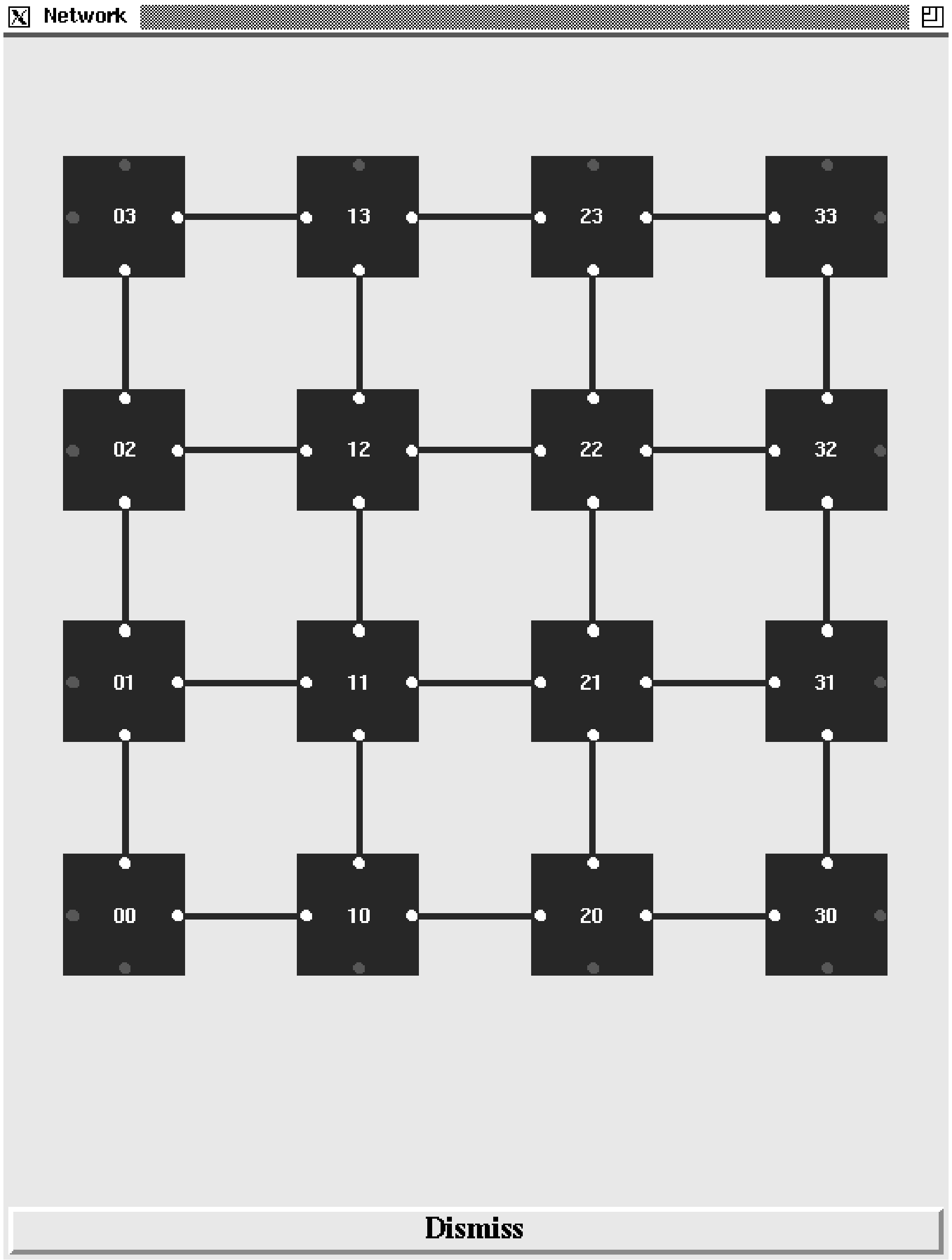

Figure 6.2: The VCDA Network Window.

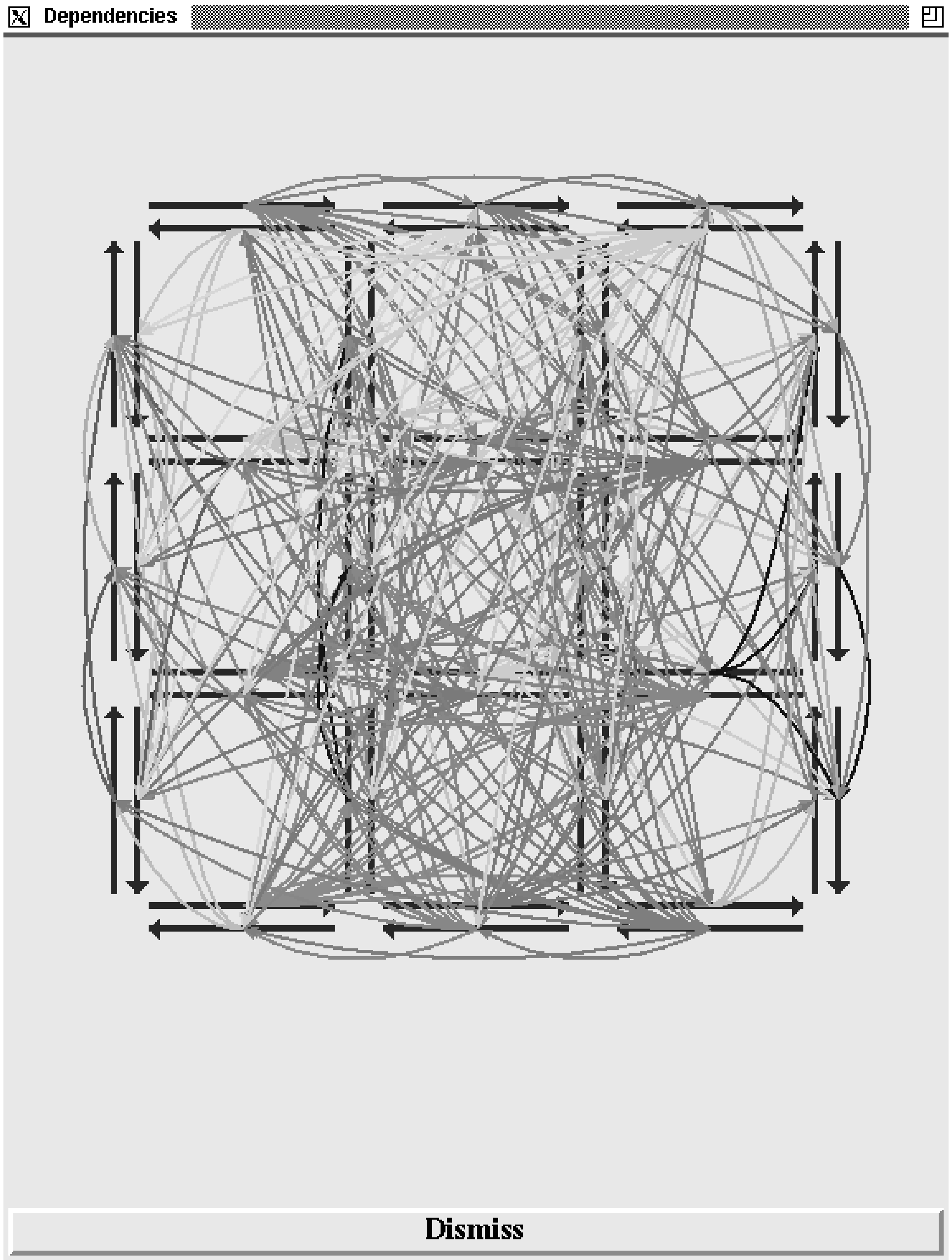

Figure 6.3: The Extended Channel Dependency Graph as displayed by the VCDA.

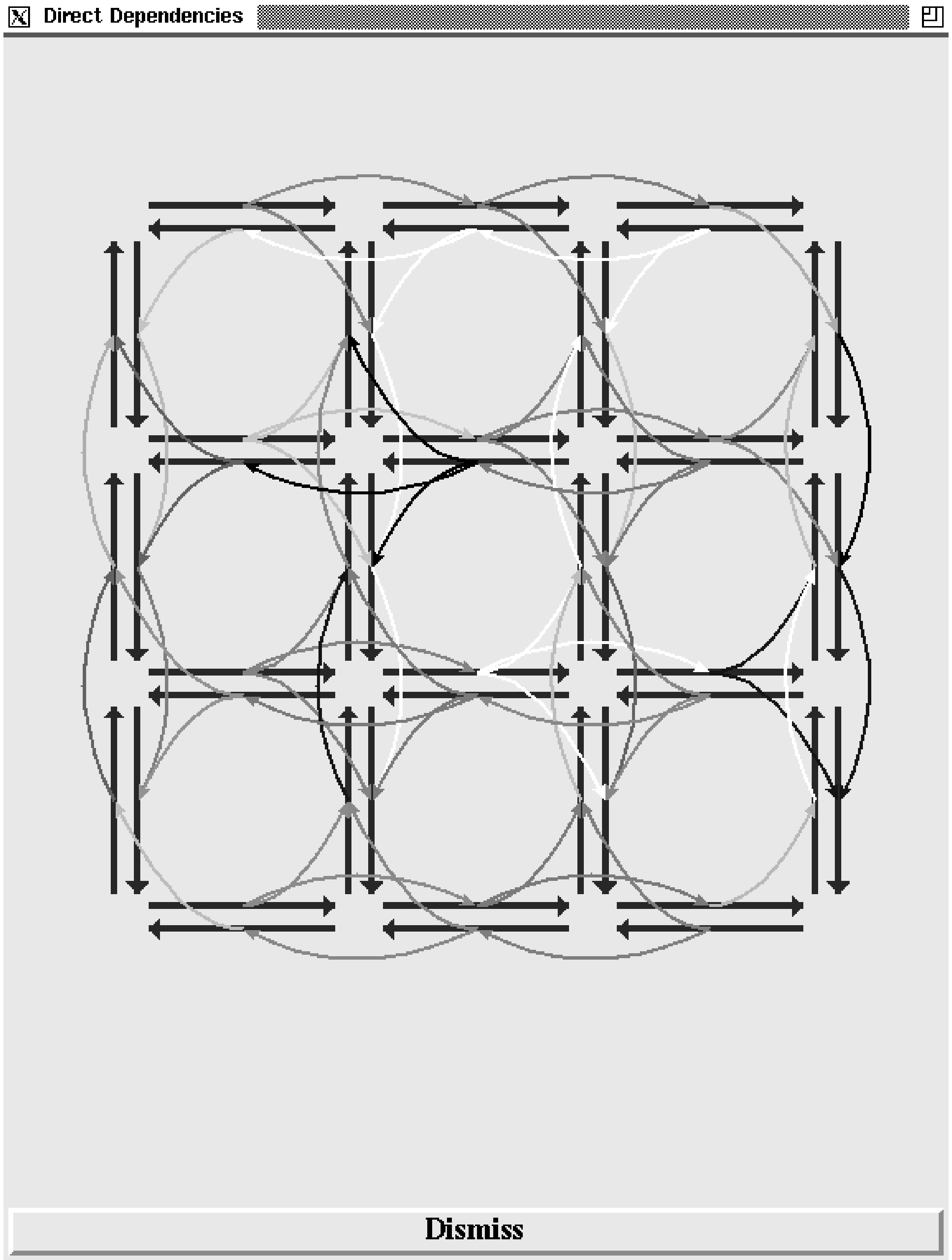

Figure 6.4: The Direct Channel Dependency Graph as displayed by the VCDA.

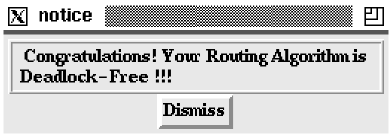

Figure 6.5: Notification that the algorithm is deadlock-free.

Port Number	Link Position
0	$x-$
1	$x+$
2	$y-$
3	$y+$

Table 6.1: Correspondence between port numbers and link positions.

on the *Results* button. If the program did not find any cycles, it will display the notice shown in Figure 6.5. Otherwise, it will display a window such as the one in Figure 6.6 where it will draw the cycle that was found. This information can help the user understand the nature of deadlocks and redesign the routing algorithm.

6.2.5 Placing Faults

The Fault Analysis is actually the most interesting part of the tool. After the user has verified that the fault-free version of the algorithm is deadlock-free, he or she should enter the fault analysis stage. Clicking on the *Fault Placement* button pops the dialog box of Figure 6.7. The three numbers shown in the dialog box uniquely identify a link in the network. The correspondence between port number and link position is shown in Table 6.1.

As soon as a fault location is specified, the user must rerun the *Create Network* and the *Create All Dependencies* commands. The network structure needs to be updated to reflect the faulty link information. Moreover, the dependency graph must also be updated to reflect the addition of Fault-Handling channels and corresponding dependencies. When there is a fault specified, the *Create Network* command also displays the Fault-Handling channels that are put into use by the routing algorithm. Figure 6.8 shows the Fault-Handling channels for fault location (1,1,1) and Figure 6.9 shows the Fault-Handling channels for fault location (1,1,3). It is interesting to observe the similarity between figures 6.8 and 5.5 and figures 6.9 and 5.7.

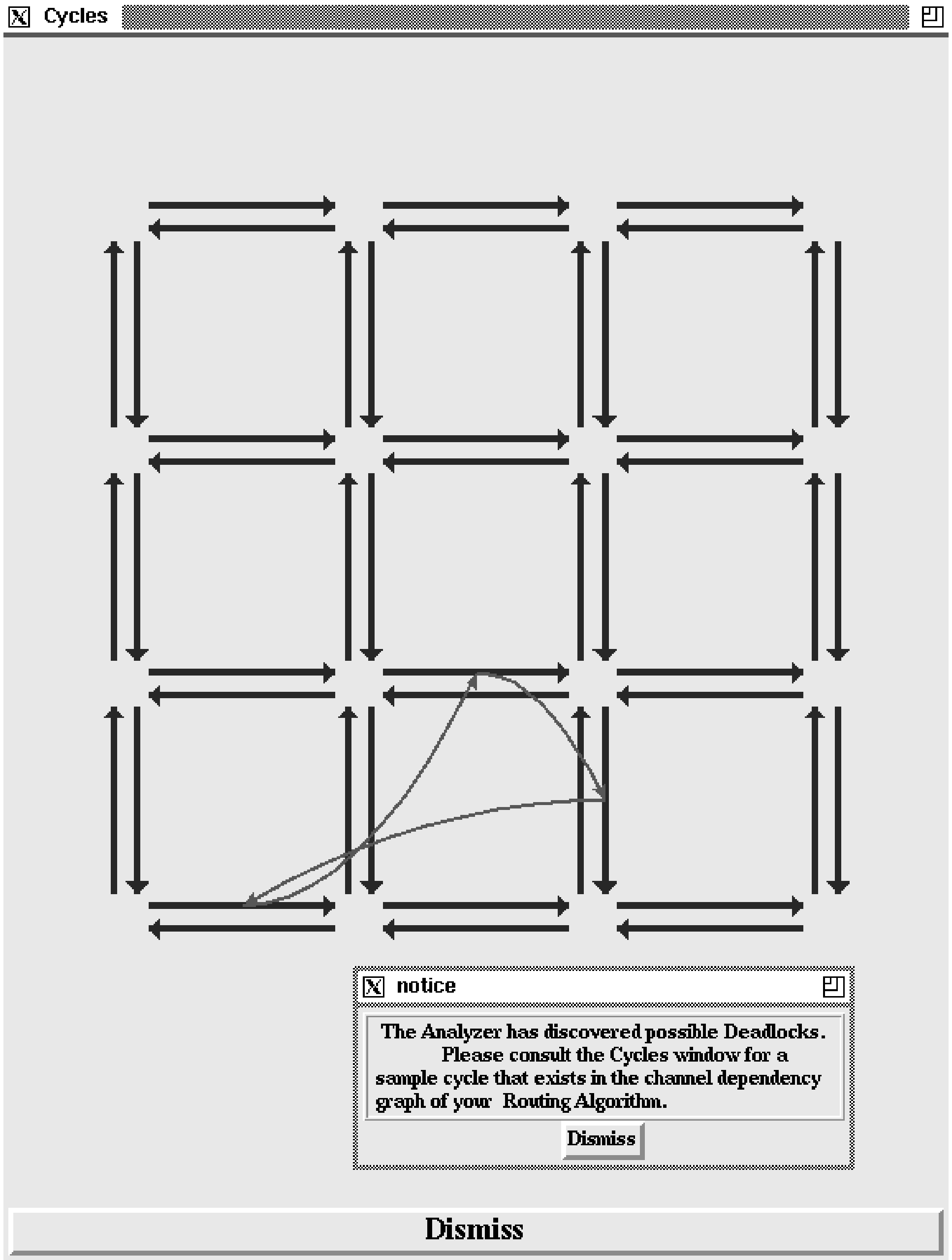

Figure 6.6: The VCDA displays a cycle in the Extended Channel Dependency Graph.

Figure 6.7: The Fault Placement dialog box.

Clicking on the *Display FH Dependencies* button produces the graphs shown in Figures 6.10 and 6.11 respectively. These graphs display the dependencies that involve Fault-Handling channels. Fault-Handling channels are depicted as red arrows placed in parallel with the blue arrows which represent the Dimension-Ordered channels that share the same physical link. These figures have been used by the author to validate the observations and assertions presented in chapter 5.

After going through this step, the user can reinvoke the cycle-search feature, and verify that Reliable Adaptive Routing is deadlock-free in the presence of faults.

6.3 VCDA Internals

This section constitutes a brief overview of the tool implementation.

6.3.1 Object Representation

The most important data structure that is created and maintained by the Analyzer, is the Channel Dependency Graph (CDG). The CDG is an array of structures of the type defined below:

```
typedef struct _vc_node {
  int valid;                     /* Channel used */
  int x_addr;                    /* Source x address */
  int y_addr;                    /* Source y address */
  int port;                      /* port number */
  int type;                      /* 0 do, 1 fh */
  int color;                     /* for cycle search */
  int dep_index;                 /* number of dependencies */
  struct _vc_node *deps[TOTAL_VCS]; /* Dependency array */
```

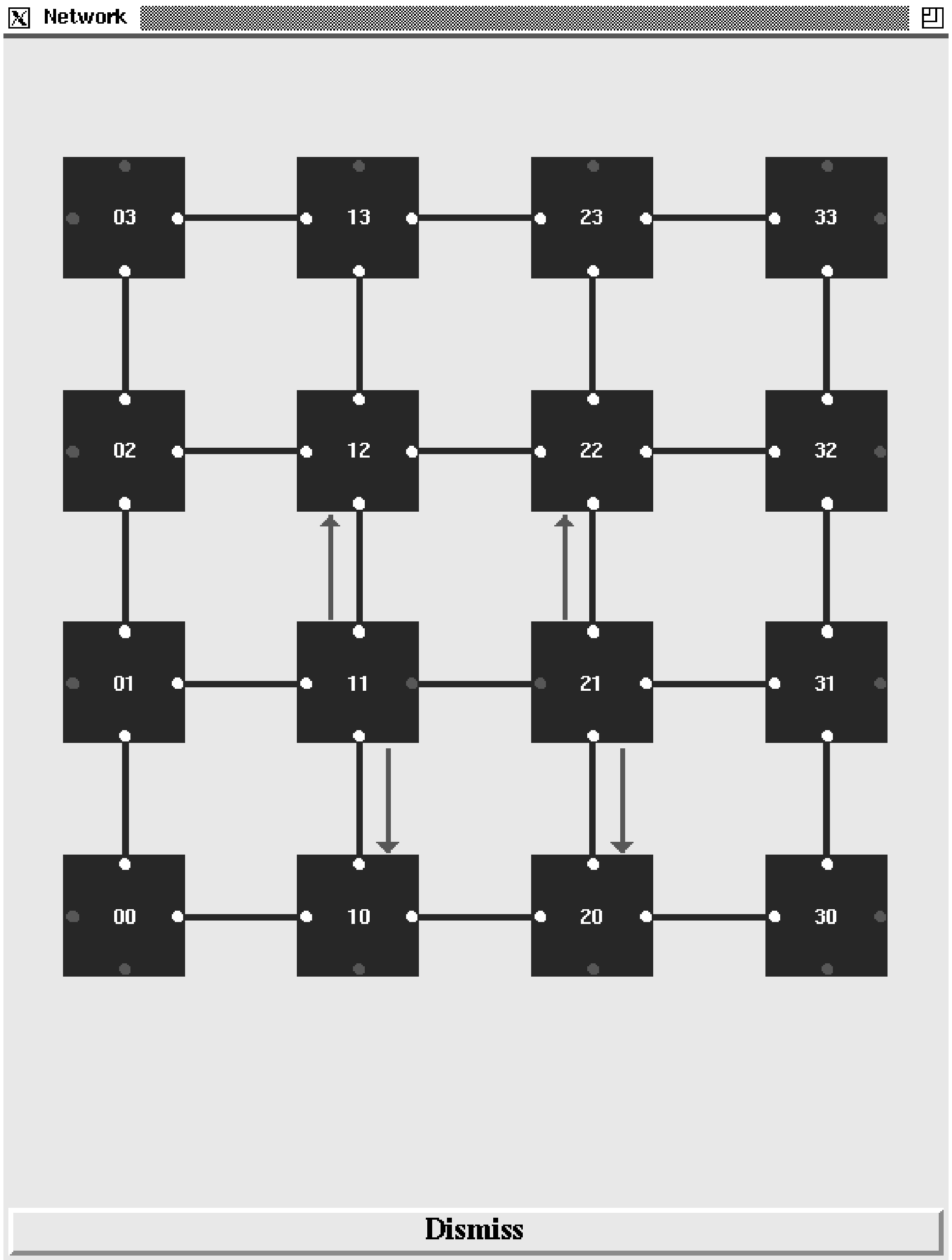

Figure 6.8: Fault-Handling channels added due to a fault along the x dimension. The fault position is (1,1,1).

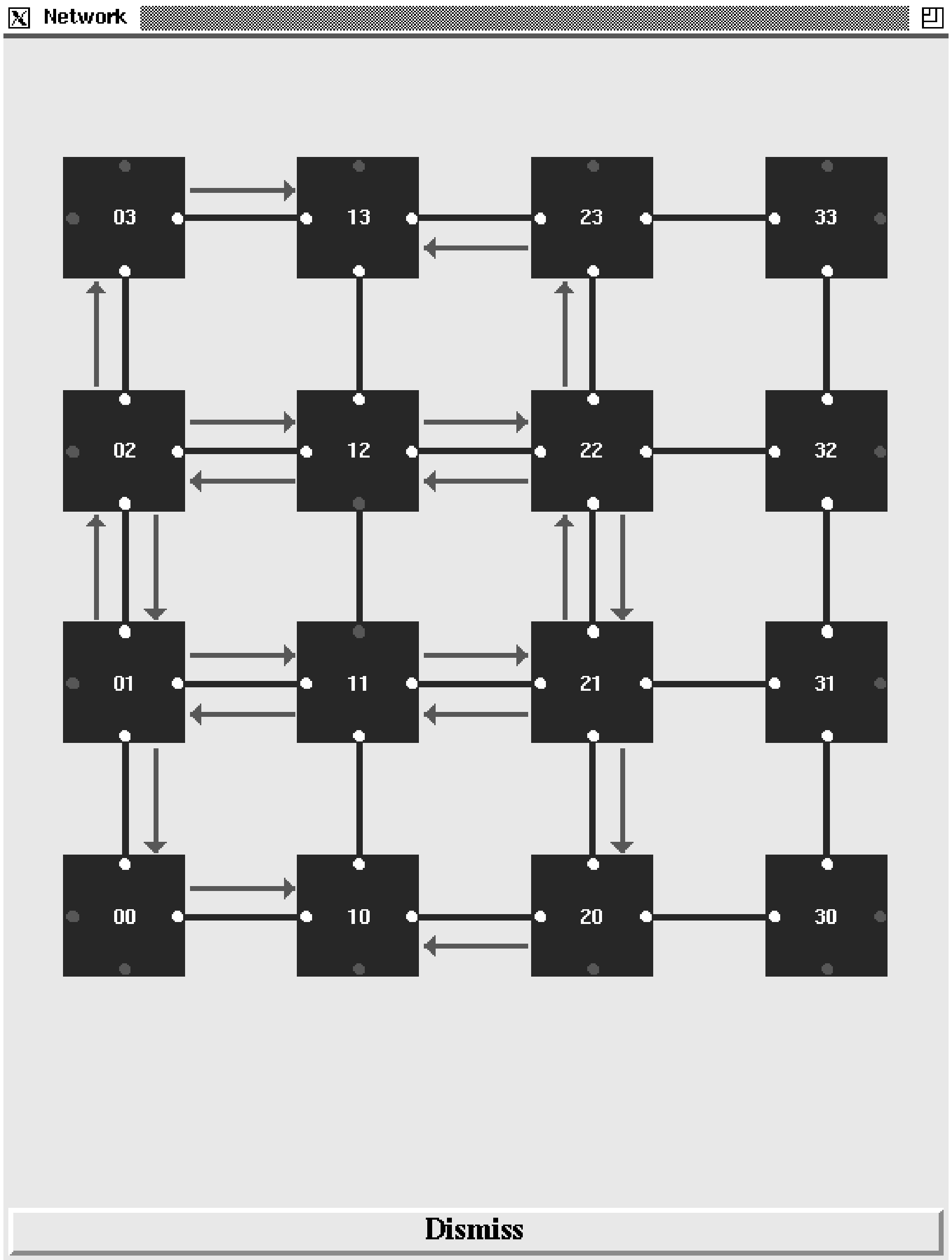

Figure 6.9: Fault-Handling channels added due to a fault along the y dimension. The fault position is (1,1,3).

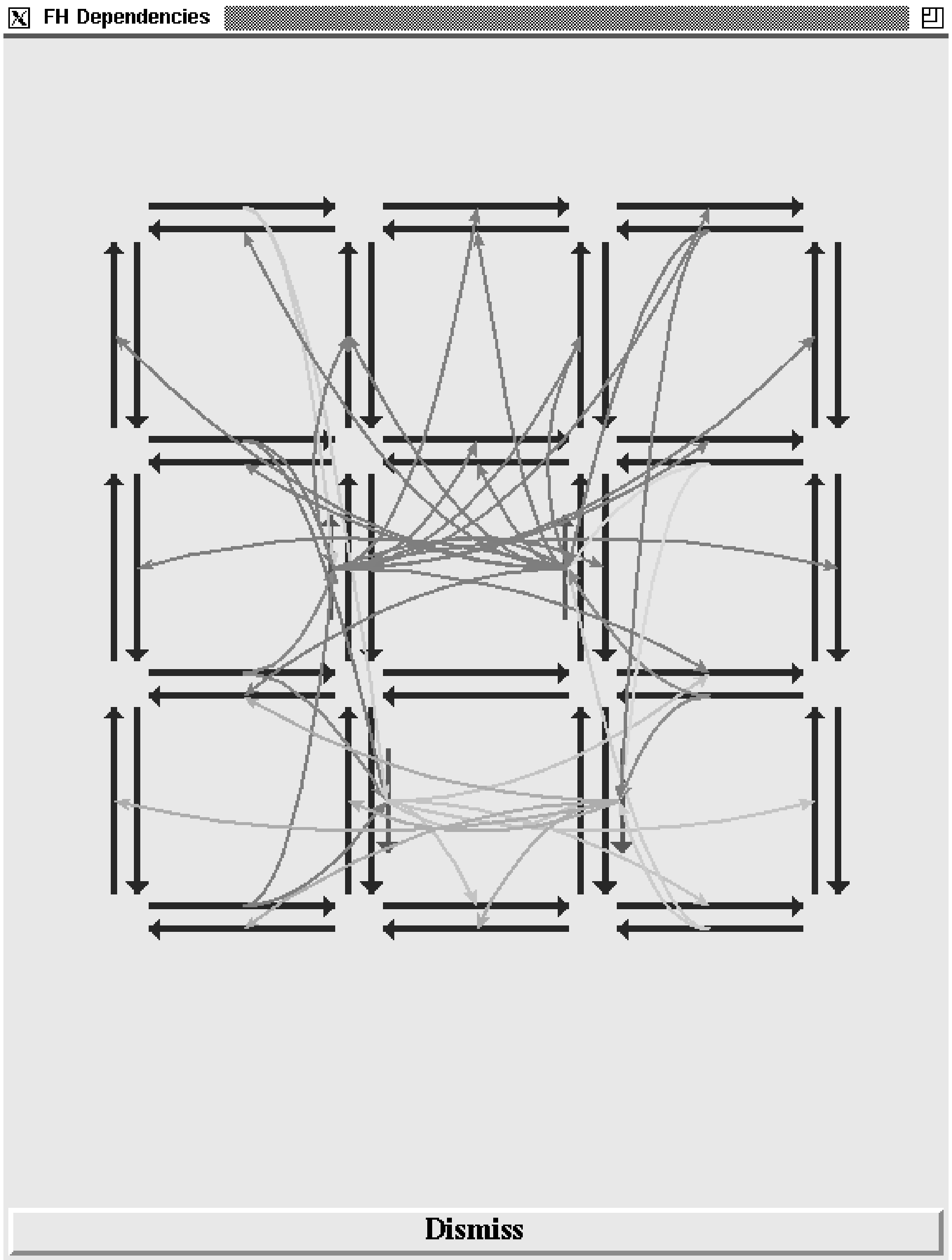

Figure 6.10: Dependencies involving Fault-Handling channels for a fault along the x dimension. The fault position is (1,1,1).

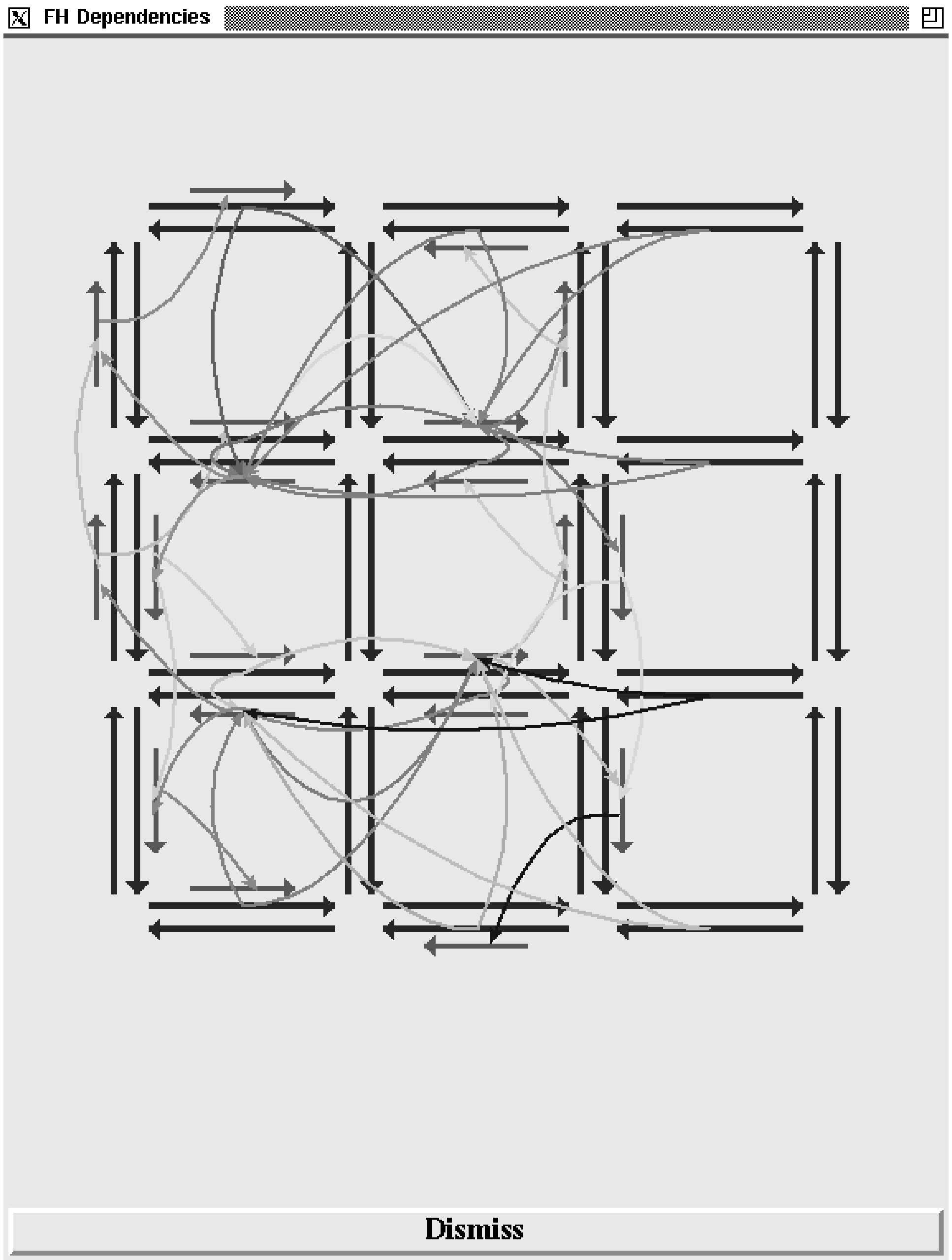

Figure 6.11: Dependencies involving Fault-Handling channels for a fault along the y dimension. The fault position is (1,1,3).

```
} vc_node;
```

This structure represents each virtual channel as a node in the channel dependency graph. The *valid* field is necessary so that Dimension-Ordered virtual channels can be invalidated and Fault-Handling channels be put in use in case of a fault.

The next three fields are self-explanatory. They uniquely identify the position of the virtual channel in the network. The *type* field distinguishes between Dimension-Ordered and Fault-Handling channels. The *color* field is used by the cycle-search algorithm and will be decribed in a following section. The *dep_index* field keeps a count of the number of the dependencies that depart from the particular virtual channel. Finally, the *deps* field is an array of pointers to other virtual channel structures. These pointers represent the direct and indirect dependencies that depart from each virtual channel.

When the program is launched, an array of all the virtual channels is created. When the user presses the *Create All Dependencies* button, the program exhaustively cycles through the virtual channels and creates all possible dependencies using the Reliable Adaptive Routing algorithm described in chapter 5. Dependencies are created by the following function:

```
void create_dependency(vc1, vc2)
     vc_node *vc1;
     vc_node *vc2;
{
  int already_in_dep;
  int i;


  already_in_dep=0;

  if ((vc1!=NULL) && (vc2!=NULL)){
    if ((vc1->valid) && (vc2->valid)){

      for (i=0; i<(vc1->dep_index); i++){
if ((vc1->deps[i]) == vc2)
  already_in_dep=1;
      }

      if (!already_in_dep){
vc1->deps[i]= vc2;
(vc1->dep_index)++;
```

```
      }
    }
  }
}
```

The function avoids duplicate dependecies by first checking whether the dependency that is about to create already exists. Finally upon exiting, it increments the dependency count.

6.3.2 The Cycle-Search Algorithm

The cycle-search algorithm loops through all the valid virtual channels and invokes the following recursive function on each one:

```
int is_there_a_cycle(vc)
     vc_node *vc;
{
  int i;

  if((vc->color)==1)
    return(1);

  (vc->color) = 1;

  for (i=0; i<(vc->dep_index); i++) {
    if (is_there_a_cycle(vc->deps[i]))
      return(1);
  }

  (vc->color) = 0;
  return(0);
}
```

Virtual channels are initialized to have color 0. The above function essentially traverses the dependency tree of each virtual channel and marks with color 1 the nodes it has visited. If it finds a node of color 1, this means that the particular node has been traversed twice, and therefore there is a cycle. If it fails to find a cycle, it changes the color of the nodes back to 0.

6.3.3 Interfacing Tcl and C

Tcl and Tk widgets [Ous94] have been used to construct the graphical user interface of Figure 6.1 as well as all the network and channel dependency graphs. Tcl is invoked within C functions in the following way:

First, a Tcl interpreter is created using the following command:

```
Tcl_Interp *interp;
interp = Tcl_CreateInterp();
```

Then, the Tcl application main window [Ous94] is created in the following way:

```
Tk_Window mainWindow;
mainWindow = Tk_CreateMainWindow(interp, display, "Network", "Tk");
```

A Tcl script can be evaluated using the interpreter that was previously created in the main application window using the following code line:

```
Tcl_EvalFile(interp, "network.tcl");
```

Multiple Tcl interpreters can be instantiated by a single C application. In fact, each button in the Program Manager of Figure 6.1 instantiates a new interpreter and runs a dedicated script.

One-way communication between the C application and Tcl has been implemented using string passing. As a concrete example, the C function that draws the channel dependency graph loops through all the virtual channels and each time it finds two channels linked by a dependency executes the following lines:

```
sprintf (Tcl_command, "make_dep %d %d %d %d %d %d \"%s\"",
         x1,y1,p1,x2,y2,p2, color);
Tcl_Eval(interp, Tcl_command);
```

In this case *make_dep* is a Tcl procedure which was defined in a Tcl script that was evaluated with *interp* sometime in the past. Given two virtual channels in terms of x address, y address, and port number it computes x,y coordinates for a spline curve that connects them. These spline curves are the arrows that represent the channel dependencies in the graphs.

Communication from Tcl to C must also be implemented. As an example consider the dialog box where the user specifies the fault position. The fault coordinates are Tcl variables and C must be enabled to access them. This achieved by the following line:

```
Tcl_LinkVar(interp, "xf", (char *) &xf, TCL_LINK_INT);
```

In this way, variable *xf* is common to both environments.

The final piece of the interface is registering C functions as Tcl procedures. This is extensively used in the implementation of the Program Manager, where clicking on a button invokes an underlying C function registered as a Tcl procedure. This can be done as follows:

```
Tcl_CreateCommand(interp, "CreateNetwork", CreateAndDisplayNetwork,
                  (ClientData) NULL, (Tcl_CmdDeleteProc *) NULL);
```

In this example, *CreateAndDisplayNetwork* is a regular C function, which is registered as Tcl procedure *CreateNetwork*. Additional Tcl code is necessary to bind the Tcl procedure to a specific button.

In general, the interface between the two environments is easy, straighforward and works really well. All the above functions are defined in em tcl.h and *tk.h*. These files must be included in the C code. Finally the code must be compiled with the *-ltcl* and *-ltk* flags.

6.4 Future Work

Currently, the Virtual Channel Dependency Analyzer only implements the Reliable Adaptive Routing algorithm of chapter 5. The routing algorithm portion can be made sufficiently programmable so that it can work with any mesh wormhole routing algorithm. The main difficulty is how to represent more than two virtual channels on the channel dependency graph. Right now, each Fault-Handling channel is drawn as an arrow parallel to the arrow that represents the Dimension-Ordered channel that shares the same physical link. This method helps in the visualization of the algorithm since each virtual channel is drawn at its true location within the network. On the other hand, this scheme does not scale for a large number of virtual channels. It will produce cluttered and indecipherable diagrams.

The ultimate goal is to make the tool available to the parallel computer routing community if and when these problems are solved.

6.5 Summary

This chapter has described the Virtual Channel Dependency Analyzer, a software tool that aids in the visualization of Reliable Adaptive Routing especially in the presence of faults. Moreover, it includes a feature that checks for dependency cycles. This essentially proves deadlock-freedom by directly applying Theorem 3.2. The VCDA has been used to provide independent verification of the Reliable Adaptive Routing algorithm.

Chapter 7

Implementation Issues

This chapter presents an implementation of Reliable Adaptive Routing in two dimensions. This routing logic is used in the MIT Reliable Router [DDH+94b] [DDH+94a], a high performance routing chip for two-dimensional meshes. The routing logic presented in this chapter is a big block of combinational logic that computes the next step route based on current message position, destination address and output virtual channel availability. For reasons that will be explained later, we call this block of combinational logic the Optimistic Router.

The Optimistic Router has been designed to support five virtual channels instead of the three that are required by Reliable Adaptive Routing. We add one more virtual channel to the adaptive pool to be used as an extra lane [Dal92]. Moreover, we add one more Dimension-Ordered virtual channel to implement two separate message priorities. Message priorities are essential in a message-driven computer for software deadlock avoidance.

The Optimistic Router assumes that there is a processor input/ output interface on each network node for message injection and consumption. Moreover, it also assumes that there is a serial diagnostic interface which may also inject and consume messages. This interface can be useful for system testing and diagnostics.

7.1 Input and Output Specification

The Optimistic Router needs the following information to make a routing decision:

Bit	Explanation
0	Message needs to route to x- dimension
1	Message needs to route to x+ dimension
2	Message needs to route to y- dimension
3	Message needs to route to y+ dimension
4	Message needs to route to the processor
5	Message needs to route to the diagnostic interface
6	Message priority
7	Message is one hop from destination
8	Message is one hop from destination in the x dimension
9	Message is on a fault handling channel

Table 7.1: Routing Problem Decomposition

7.1.1 Routing Problem

The Routing Problem (rp[9:0]) is a 10-bit value that indicates all the possible directions that the message can route to move closer to its destination. The Routing Problem also includes bits that indicate whether the message should exit the network and be directed to either the processor or the diagnostic interface, mesage priority and whether message distance from destination along dimension 1 (x in this case) is equal to one. This information is used by the Fault-Handling computation as can be seen from Equation 5.9. The full decomposition of the Routing Problem is shown in Table 7.1.

The need for bits 0 through 5 is obvious and does not require further explanation. Bit 6 (priority) is necessary because the particular implementation of the routing algorithm supports message priorities. Messages that have different priority (0, 1) use a different set of Dimension-Ordered virtual channels but they share the same pool of adaptive channels. Bit 7 is used to direct a message to a neighboring processor instead of its original recipient. When a message is received on a Fault-Handling channel (as indicated by bit 9), it is only one hop away from its destination, and the link that leads it to its original destination is down, then it is consumed by the local processor. This is illustrated in Figure 7.1. Bits 8 and 9 are both used by the routing algorithm to determine whether a message must use Fault-Handling channels throughout the rest of its course.

The Routing Problem is generated by another piece of combinational logic which essentially compares the header of the incoming message with the local address. This scheme implies that the header of each message carries the absolute address of the message destination. An alternative approach could be to use relative addressing: The header of each

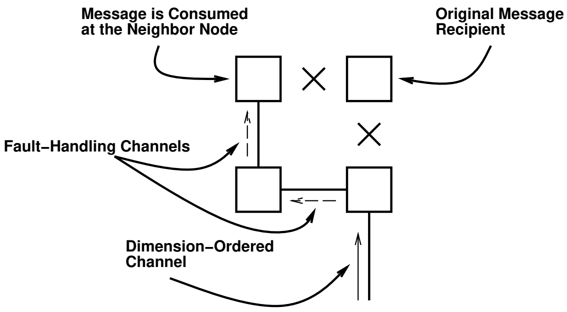

Figure 7.1: Routing a message to the neighboring node instead of the original recipient.

free [4:0]	$x-$ Virtual Channels
free [9:5]	$x+$ Virtual Channels
free [14:10]	$y-$ Virtual Channels
free [19:15]	$y+$ Virtual Channels
free [24:20]	Processor Virtual Channels
free [29:25]	Diagnostic Virtual Channels

Table 7.2: Virtual channel avaliability information.

message carries the current offset of the message from the destination location. When the message is routed along a certain dimension, the corresponding offset is decremented by one. The absolute scheme was prefered over the relative one because it conforms with one important requirement we set back in Chapter 2: Intermediate nodes should not modify the header of a message. The contrary would prohibit an end-to-end error-correcting layer that would also include message headers.

7.1.2 Output Virtual Channel Availability Information

The Optimistic Router also needs as an input a 30-bit quantity (free[29:0]) which represents the state of each output virtual channel (allocated / not allocated). Five bits are necessary to represent the virtual channels of each port including the processor and the diagnostic. A 1 in free[x] means that virtual channel x is not currently allocated to a message. Table 7.2 shows how the bits are divided among the six ports. Table 7.3 shows how the free bits are distributed among the virtual channels within a single port.

The free [29:0] flags are implemented as a bank of RS-latches. Each latch is set when the corresponding virtual channel is allocated to a message head, and is reset when the tail flit

Bit Position	**Virtual Channel**
0	Adaptive Channel 0
1	Adaptive Channel 1
2	Dimension-Ordered Channel, Priority 0
3	Dimension-Ordered Channel, Priority 1
4	Fault-Handling Channel

Table 7.3: Virtual channel bit assignment within a single port.

Port status bit	Link Position
0	$x-$
1	$x+$
2	$y-$
3	$y+$

Table 7.4: Correspondence between port_status bit and link positions.

of the message leaves the node and releases the virtual channel.

7.1.3 Port Status Information

The routing decision also needs information regarding the status (faulty / operational) of each one of the four physical links $(x-, x+, y-, y+)$. The Optimistic Router assumes that the processor and the diagnostic links are always operational. A 4-bit quantity (port_status[3:0]) is used to denote with 1 an operational link, and with 0 a faulty link. Table 7.4 shows the correspondence between port status bits and link positions.

7.1.4 Routing Answer

The output of the Optimistic Router is a 12-bit quantity (routing_answer [11:0]). The first five bits indicate the output virtual channel in fully decoded form, and the next six bits indicate the output port in decoded form as well. The full decomposition of the routing answer is shown in Table 7.5.

Bit Field	**Explanation**
[4:0]	Output virtual channel
5	Output Controller x-
6	Output Controller x+
7	Output Controller y-
8	Output Controller y+
9	Processor Output
10	Diagnostic Output
11	Priority

Table 7.5: Routing Answer Decomposition

7.2 Optimizing Circuit Latency

The high clock frequency of the Reliable Router chip (100 MHz) places tight requirements on the latency through the Optimistic Router. In order for a clock frequency of 100 MHz to be achieved, the total delay for the routing computation must be well below 10 ns. Techniques such as computation parallelization and preprocessing of output virtual channel availability information have helped bring the latency down to the desirable level.

7.2.1 Parallel Computations

Although the flowchart of Figure 5.3 depicts RAR as a serial computation, we have decided to parallelize it as much as possible. Figure 7.2 shows the gross organization of the Optimistic Router. The inputs are broadcast to each of the five computation modules shown. Note that port status information is not broadcast to the processor and diagnostic module. Port status only includes information pertinent to the $x-, x+, y-, y+$ links. Each module independently computes a routing answer. A multiplexer selects the final answer based on the failure or success of each computation to produce an answer. The logic circuit will be described in more detail in section 7.3.

7.2.2 Preprocessing Output Virtual Channel Availability Information

The Reliable Adaptive Routing algorithm as shown in the flowchart of Figure 5.3 imposes a prioritization among the five virtual channels of each port. Let free[4:0] be the avalable/ non-available flags that describe the virtual channels of an output port as shown in Table 7.3. We can make the following observations:

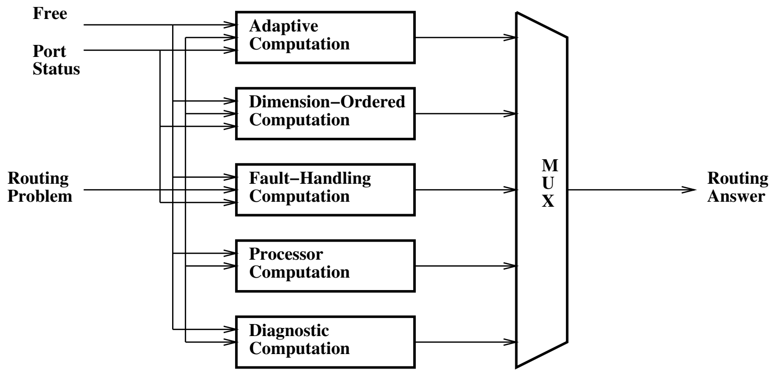

Figure 7.2: Parallel computations in the Optimistic Router.

1. If any of the two available Adaptive virtual channels is free, then the algorithm will never pick a Dimension-Ordered channel. Therefore, we can employ preprocessing logic to set the corresponding free bits (free[3:2]) to zero when either free[1] or free[0] are set. We cannot do the same for the Fault-Handling channel. It is possible for RAR to pick a Fault-Handling channel when there are free Adaptive channels: When there is a fault along the y dimension, all messages that use Fault-Handling channels cannot switch back to the other classes.

2. If both Adaptive channels are free, then we can employ deterministic preprocessing logic that picks one among the two. No such prioritisation logic can be employed for the two Dimension-Ordered channels since they are used by messages of different priority.

The two observations above can lead to an encoding scheme for the free bits (free[4:0]) of each output port as shown in Table 7.6.

The processor and diagnostic output ports also have five virtual channels. These virtual channels do not have the meaning thet they have within each port. They are simply used as five different lanes with equal properties. In this case we can employ prioritisation logic that picks in a deterministic fashion only one out of the free virtual channels. Such an encoding is shown in Table 7.7.

Preprocessing the output virtual channel state has yielded two main benefits:

First, the Optimistic Router circuit has been simplified and made faster. Necessary computation for the routing decision has been pushed back and removed from the critical path. The total silicon area has been reduced. The Reliable Router has a total of 30 copies

free[4:0]	**Encoded free[4:0]**
00000	00000
0xxx1	00001
0xx10	00010
00100	00100
01000	01000
01100	01100
10000	10000
1xxx1	10001
1xx10	10010
10100	10100
11000	11000
11100	11100

Table 7.6: Encoding of free bits within each network port.

free[4:0]	**Encoded free[4:0]**
00000	00000
xxxx1	00001
xxx10	00010
xx100	00100
x1000	01000
10000	10000

Table 7.7: Encoding of free bits within the processor and diagnostic ports.

of the Optimistic Router logic. There is one copy for each input virtual channel so that the routing logic is not a shared resource. In this fashion, an extra level of arbitration and serialization is removed and performance is increased. By factoring out the channel prioritization logic, gains in silicon area are substantial because there is a multiplicative factor of 30 involved.

Second, channel prioritization has made possible the exhaustive validation of the Optimistic Router logic circuit. Channel prioritization has reduced the total input state space of the logic by a huge factor. More specifically, free[29:0] can have a total of $2^{30} = 1.0737 \times 10^9$ different values if it is not encoded. If the encodings of Tables 7.6 and 7.7 are performed, free[29:0] may take a total of $12^4 \times 6^2 = 746496$ different values. This reduces the number of iterations required to check the circuit exhaustively by a factor of about 1438. It currently takes about 18 hours of computation on a relatively unloaded Sparcstation 10 to validate the Optimistic Router. Without channel prioritization, exhaustive validation would take about 2.95 years! Alternative validation methods (i.e. randomized inputs) would have to be sought in such a case.

7.3 Logic Circuit Schematics

The exact schematics of the Optimistic Router have been included in appendix A. Figure 7.3 shows a block diagram of the logic.

Each one of the four blocks contains necessary logic to compute an output port number and an output virtual channel based on the routing problem, the port status and the free virtual channels at that time. We can make the following observations regarding the circuitry external to these blocks:

1. If the processor or the diagnostic port is the output port, the other three computations are invalidated. The reason that this invalidation is necessary is to force the virtual channel output of the three blocks to be zero, so that the final multiplexing can be implemented by a simple 4-wise OR.

2. Success of the Adaptive or Dimension-Ordered computation may invalidate the Fault-Handling computation. This is consistent with the flowchart of Figure 5.3.

3. Success of the Fault-Handling computation may invalidate the Dimension-Ordered and the Adaptive computation. This happens when a fault along the y dimension has forced the use of Fault-Handling channels only throughout the rest of the message course.

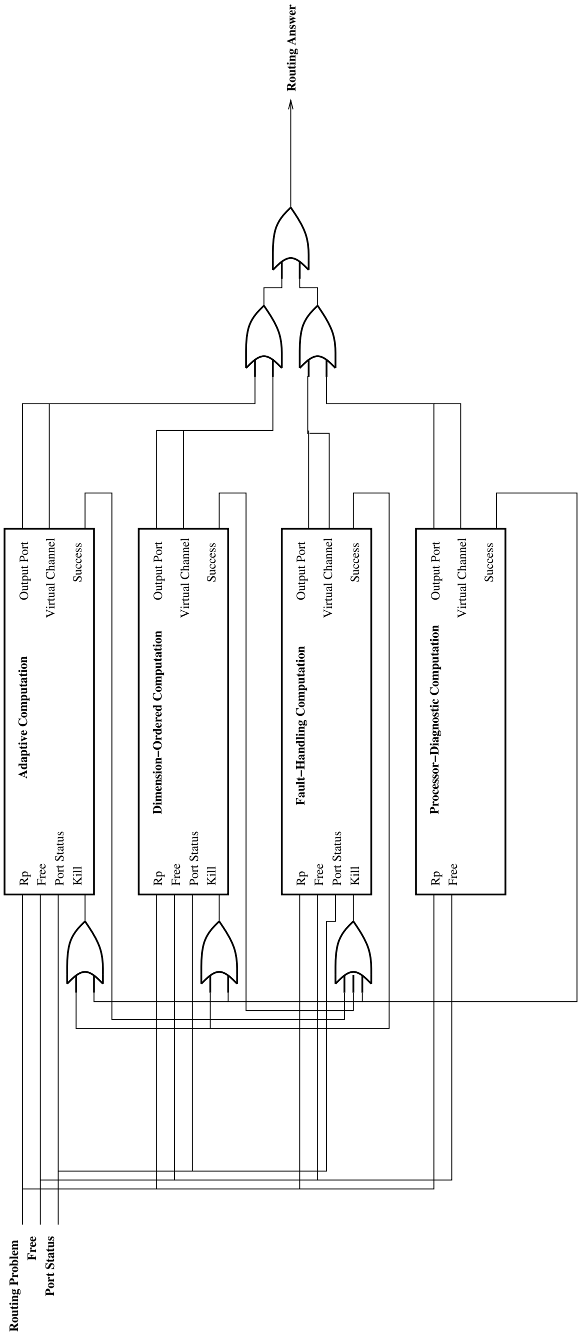

Figure 7.3: Block Diagram of Optimistic Router logic circuit.

4. There is no invalidation of the Dimension-Ordered computation by the Adaptive computation although this seems necessary from the definition of Reliable Adaptive Routing. The Adaptive computation has already priority over the Dimension-Ordered computation through the free channel state encoding of Table 7.6.

For detailed schematics, the reader is encouraged to refer to appendix A.

7.4 Logic Simulation and Validation

The Optimistic Router schematics have been exhaustively validated. Gate-level Verilog code has been extracted from the schematics of appendix A. A wrapper in Verilog has been written which performs the following tasks:

1. It instantiates a Verilog module which contains the extracted schematics.
2. It contains a high-level procedural description of the Reliable Adaptive Routing algorithm.
3. It generates all possible input vectors for rp[9:0], free[29:0], and port_status[3:0].
4. It iterates through all the possible values of the above inputs. There is a total of 6.569164×10^8 such iterations.
5. During each iteration, it compares the routing answer from the extracted schematics with the routing answer from the procedural description. If it finds a mismatch, it stops and prints relevant state.

Figure 7.4 shows the validation process in a flowchart form. The Verilog code which performs the validation has been included in Appendix B.

The Verilog code has been compiled to C using the Chronologic Verilog compiler (VCS). The resulting C code has been compiled using the GNU optimizing compiler (gcc -O3). The complete run requires approximately 18 hours of computation on a Sparcstation 10. Since there is substantial coarse grain parallelism, the validation process has been partitioned in three independent parts and has been run on different machines simultaneously. The schematics have successfully passed the validation stage.

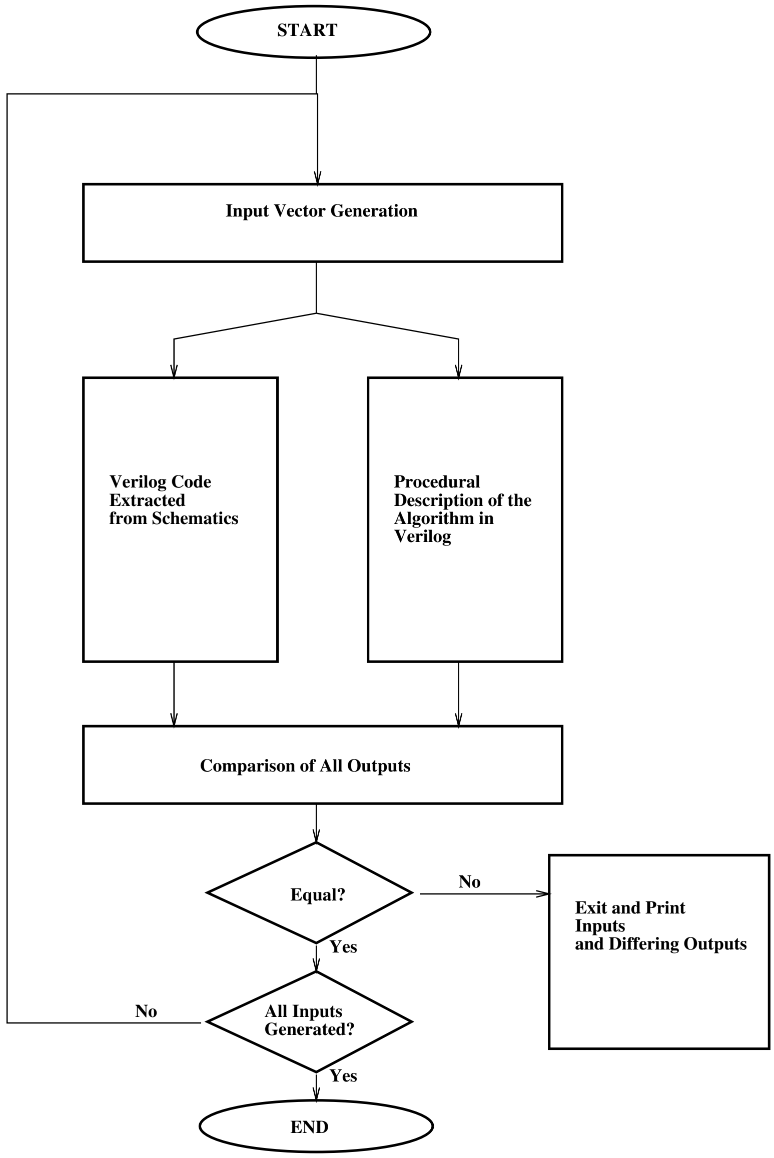

Figure 7.4: Schematic validation process

Process Corner	Rise Delay (ns)	Fall Delay (ns)
Nominal	4.01	1.37
Fast Speed	2.99	1.03
Slow Speed	6.79	2.25
Fast N Slow P	3.92	1.40
Slow N Fast P	3.76	1.34

Table 7.8: Optimistic Router rise and fall delays across all process corners.

7.5 Determining Circuit Delay

The Optimistic Router critical path has been simulated using the Hspice circuit simulator. A load consisting of a single 2x inverter per output was used for the simulation. The device models used are the ones for the hp26 1μ 3-metal layer process. Table 7.8 and Figure 7.5 summarize the results of the simulation across all process corners. The rise and fall delays are measured from when the input achieves 10% of its final value until the output achieves 90% of its final value. An input rise and fall time of 0.5 ns has been assumed. The Hspice driver and the netlist used in these experiments have been included in appendix C.

The critical path simulated has been determined as the path that goes through the largest number of logic gates between input and output. It is necessary to note that inputs free[29:0] and port_status[3:0] do not enter the critical path. The architectural design of the Reliable Router [DDH+94a] pushes the stabilization of these inputs well into the previous cycle. As can be seen from the results, the routing computation needs slightly more than half a clock cycle even in the slowest possible process corner.

7.6 Circuit Layout

The circuit has been laid out using hand placement and routing of standard library cells. Cell instantiation, placement and routing have been done using relevant database calls accessible through the SKILL language and environment. The layout of the circuit is shown in Figure 7.6. It has 10 rows of standard cells, each 472 λ wide. The figure shows the silicon area allocated to each of the separate computations of Figure 7.2. Table 7.9 summarizes the physical characteristics of the circuit.

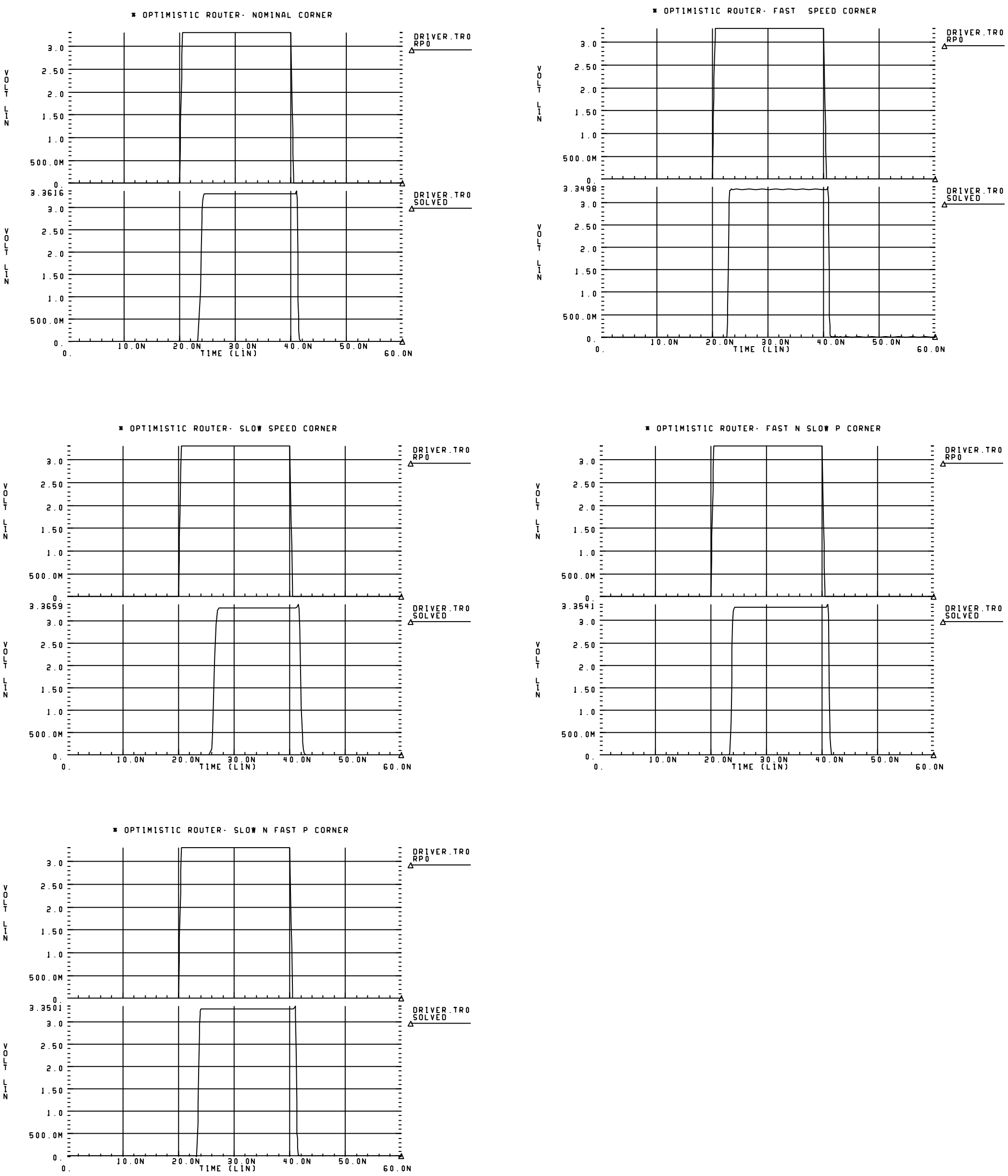

Figure 7.5: Hspice transient analyses of the Optimistic Router critical path across all process corners. The first plot of each of the five diagrams shows a square wave input with rise and fall times equal to 5ns. The second plot of each diagram is the response of the Optimistic Router. Each diagram corresponds to the process corner indicated by its title.

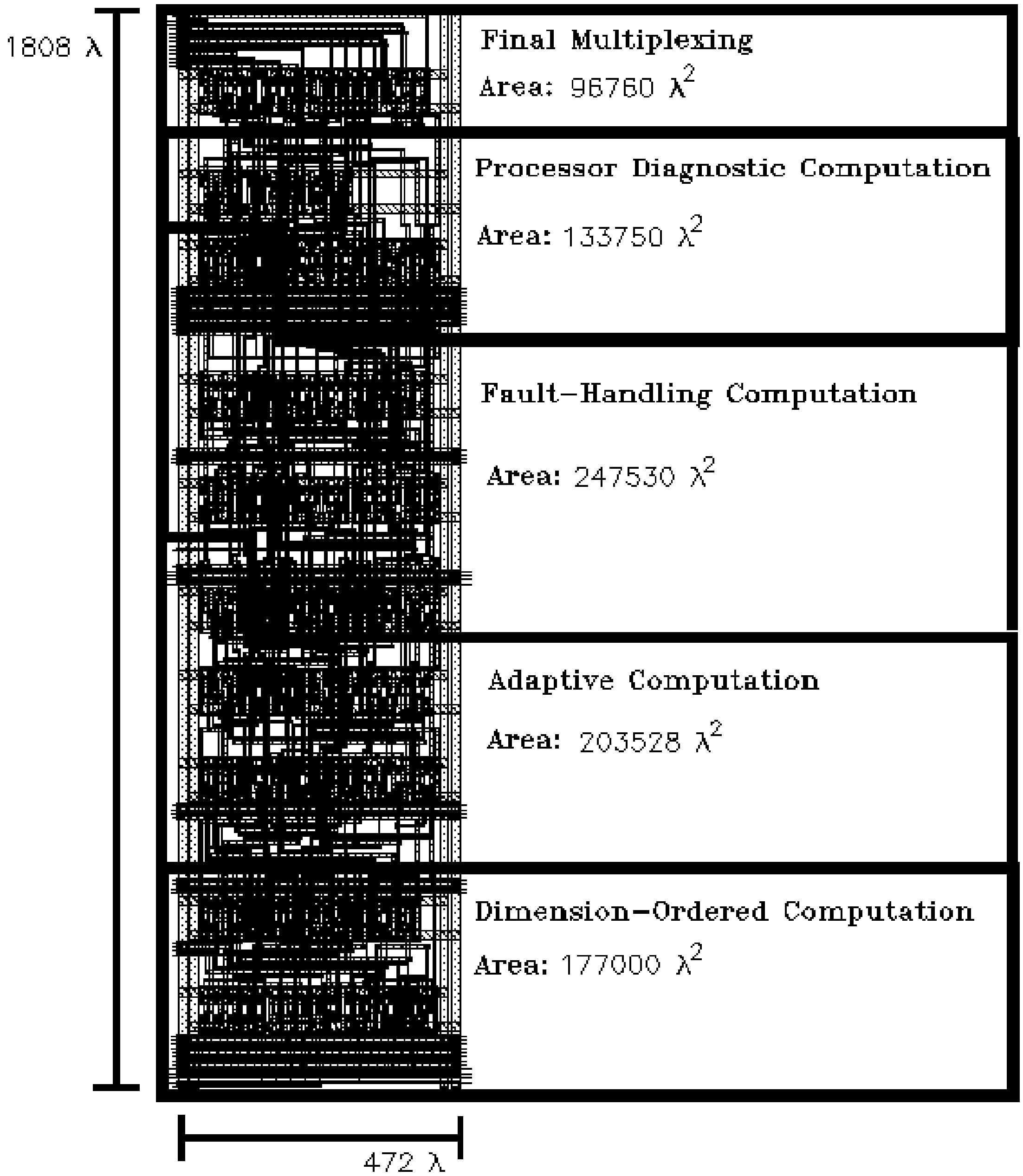

Figure 7.6: Optimistic Router layout

Number of transistors	734
Number of nfets	367
Number of pfets	367
Heighth (λ)	$1,808$
Width (λ)	472
Area (λ^2)	$853,376$

Table 7.9: Optimistic Router physical characteristics.

Process Corner	**Rise Delay (ns)**	**Fall Delay (ns)**
Nominal	3.67	1.63
Fast Speed	2.78	1.29
Slow Speed	6.00	2.60
Fast N Slow P	3.53	1.58
Slow N Fast P	3.34	1.57

Table 7.10: Extracted layout rise and fall delays across all process corners.

7.6.1 Determining Latency Using Extracted Layout

Critical path Hspice circuit simulation has been run again on a netlist produced from the extracted circuit layout. These results are slightly more accurate because the netlist includes interconnect capacitances and exact values for device source and drain parasitic capacitances. The input rise and fall time was again assumed to be 0.5 ns and the output load is a 2x inverter. The results are summarized in Table 7.10. As expected, there is minimum deviation from the schematic results of Table 7.8.

7.7 Layout Verification

The circuit layout has been verified using Cadence DIVA for both Design Rule Checking (DRC) and Layout vs. Schematics (LVS). DRC has been checked using the MOSIS λ-based design rules [MC80]. The LVS part of the verification involves extracting a textual netlist from both the schematic view and the extracted layout view. Then DIVA checks for matching nets, devices and parameters. DIVA has a number of advanced features including full device permutability.

The Optimistic Router has passed both stages of layout verification.

7.8 Summary

This chapter has presented the logic and physical design of the Optimistic Router, a block of combinational logic that implements Reliable Adaptive Routing. The Optimistic Router is part of the Reliable Router chip, a high-performance network switching element. The design has undergone exhaustive logic simulation. The delay of the circuit has been fully characterized across all process corners. Finally, the circuit has been laid out in silicon and has passed the necessary verification steps.

Chapter 8

The Big Picture

An adaptive and fault-tolerant routing algorithm is a necessary but not sufficient ingredient for the construction of a reliable and high-performance multicomputer interconnection network. This chapter briefly describes how Reliable Adaptive Routing fits into the Reliable Router chip. The Reliable Router (RR) is a high-performance network switching element targetted to two-dimensional mesh topologies. It is designed to run at 100 MHz. The Reliable Router uses Reliable Adaptive Routing coupled with the Unique Token Protocol [Den91] to increase both performance and reliability. The RR can handle a single non-transient node or link failure anywhere in the network without interruption of service. The first section of this chapter briefly describes the architecture of the chip.

Reliable Adaptive Routing cannot handle by itself a fault in the network at all times. A retransmission protocol is also necessary to ensure message delivery when a message (worm) is cut in half as a result of a link fault. The second section of the chapter describes the Unique Token Protocol, an efficient retransmission and duplicate detection mechanism which uses network buffer storage to keep duplicate copies of messages.

8.1 Brief Architectural Description

The top level organization of the Reliable Router is shown in Figure 8.1. There is one Input Controller and one Output Controller for every direction. Moreover, there is a processor input/output and a diagnostic input/output. Communication between an input and an output controller occurs through a crossbar switch. The switch is a full crossbar and allows each input controller to connect to every output controller. Non-adaptive routing systems that implement more restrictive routing algorithms do not need a full crossbar switch.

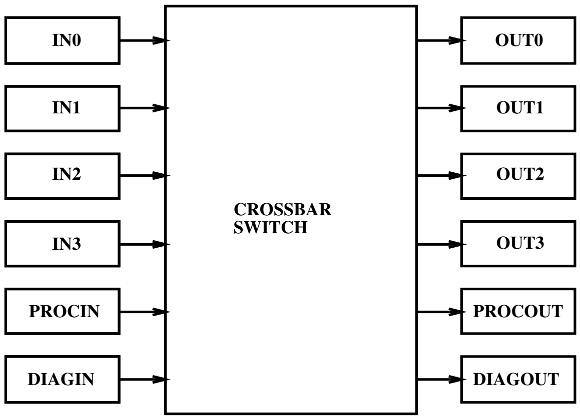

Figure 8.1: Organization of the Reliable Router.

The architecture of the chip is more accurately depicted by Figure 8.2. The input and output controller that correspond to a given network direction have been grouped into a block called a Port. The system consists of six ports including the processor and the diagnostic. The four ports that correspond to the four network directions are bidirectional. The processor and the diagnostic ports use separate unidirectional pins for inputs and outputs. The crossbar switch has been distributed among the six ports. Each port contains a six-way multiplexer. This multiplexer allows the outputs of all six input controllers to be connected to each one of the six output controllers, thus ensuring full connectivity.

Each input controller includes five FIFO buffers for each one of the five virtual channels that are implemented. Moreover, it includes five copies of the Optimistic Router logic, one per virtual channel. Once a message reaches an input port, the head flit is fed into the corresponding Optimistic Router, and the data flits are stored in the corresponding FIFO. The Optimistic Router will decide which output port and output virtual channel the incoming message is going to arbitrate for. The multiplexers of Figure 8.2 are controlled by arbiters that serialize the requests for the same output controller. This is the reason that we call the routing logic Optimistic: Even though it can produce an answer in terms of an output port and an output virtual channel, it is not certain whether the message will actually get to that port during the next cycle because other messages from other ports may want to go there too. Each arbiter makes a randomized decision among the messages that request the same output controller.

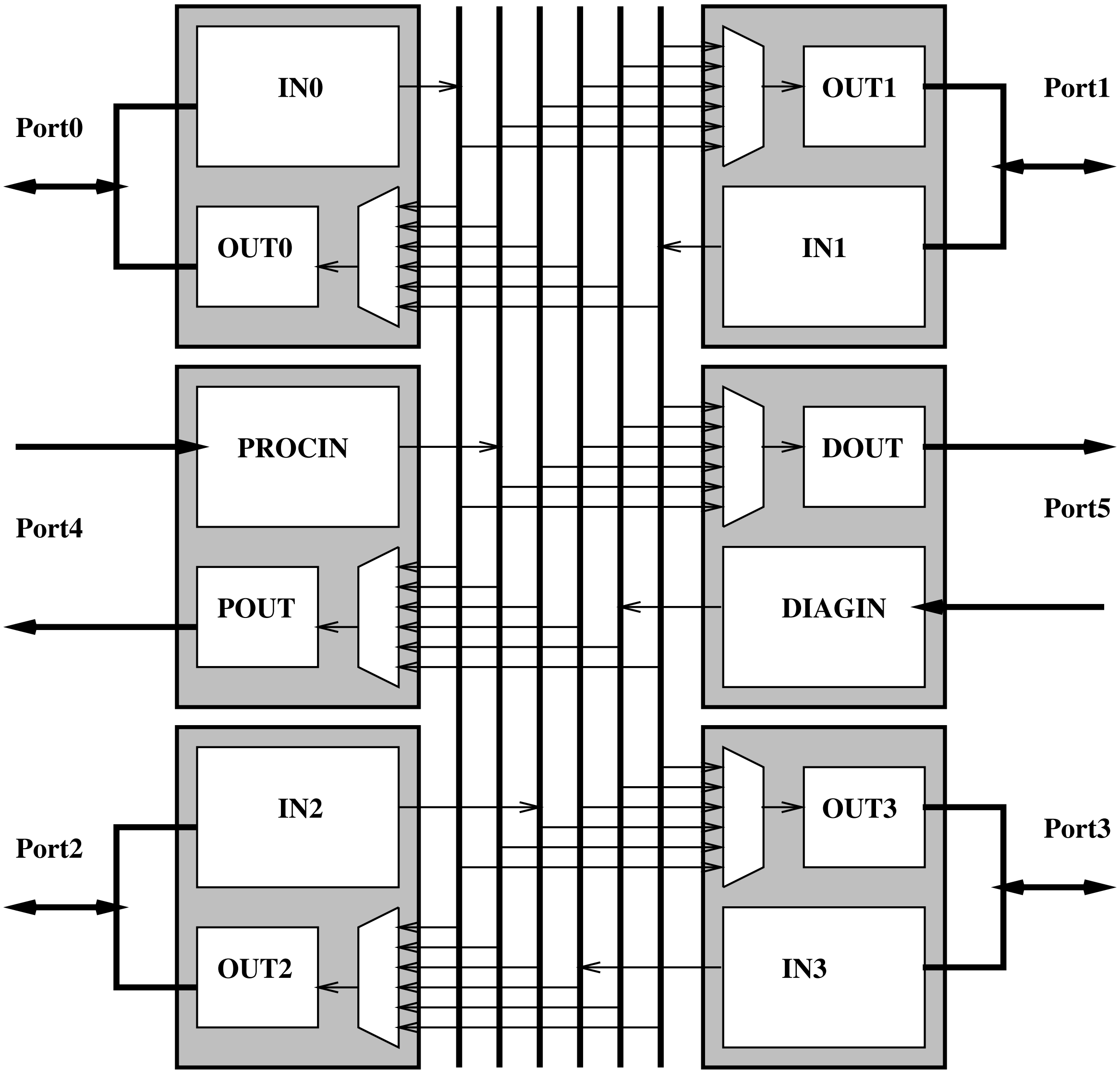

Figure 8.2: Architecture and rough floorplan of the Reliable Router chip.

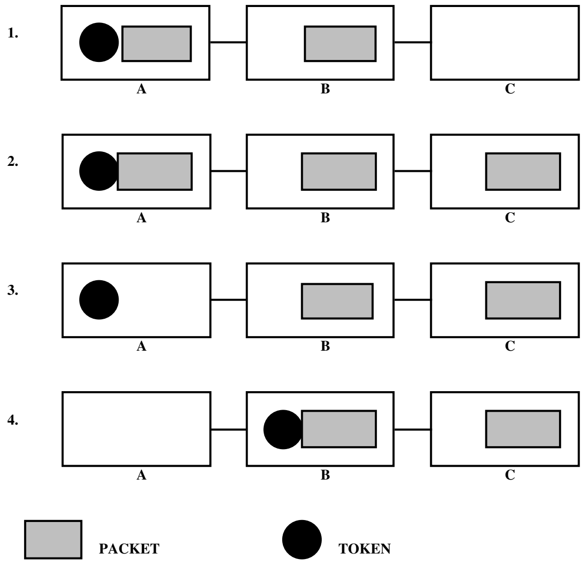

Figure 8.3: Buffering and forwarding under the UTP.

8.2 The Unique Token Protocol

Adaptive routing by itself is not enough to solve the reliability problem. Routing around faulty links cannot guarantee delivery of messages that are half-way transmitted through the physical channel at the tome the fault occurs. End-to-end protocols may solve the reliability problem when coupled with adaptive routing, but require extra overhead and the necessary resources do not scale linearly with the size of the machine [Den91]. For this reason, the Reliable Router uses link-level retransmission in combination with a unique-token protocol (UTP) [Den91] to guarantee fault-tolerant exactly-once delivery of all packets in the network. This link-level protocol offers significant advantages over end-to-end protocols because it does not require acknowledgment packets and does not keep copies for possible retransmissions at the packet source. In this way, effective network bandwidth is increased and storage requirements at the nodes decrease. Moreover, the protocol reduces the amount of storage required at the destination nodes for duplicate message detection. These properties allow the protocol resources to scale linearly with the number of network nodes as opposed to end-to-end protocols.

An example of packet forwarding under the UTP is shown in Figure 8.3 where source node A sends a packet to destination node C. The packet is buffered and forwarded through switching node B. The process must ensure that at least two copies exist in the path between

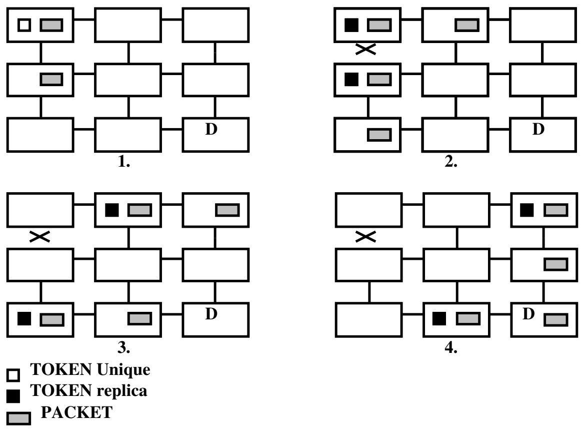

Figure 8.4: Fault Handling under the UTP.

the source node and the destination node at all times. This can be achieved by first copying the packet forward one node, and then allowing the release of the storage in the rearmost node as shown in Figure 8.3. When the packet is first injected into the network, a token is injected right behind the packet. The invariant that no copies of the packet exist behind the token is always preserved. Packet copying and token passing are carried out exactly as shown in Figure 8.3. *Note that although multiple copies of the packet are kept in the network, every node receives a packet only once.* Thus, in the absence of faults, the arrival of the token at the destination implies that the packet has been delivered exactly once. For simplicity, the UTP was described at the packet level only. The Reliable Router implements the UTP at the flit level, and the protocol is a bit more involved.

8.2.1 Fault Handling

When a node in the network fails, communication between the advance and rear copies of the packet may be severed. Each copy now must make its way to the destination without knowing the fate of the other copy. When packets arrive at the destination they must be marked in such a way so that the destination knows that it needs to look for duplicates. For this reason, two types of tokens are defined: *Unique*, and *Replica*. If the network needs to use multiple paths while forwarding the packet, the token is changed to type Replica for all copies of the packet. After the token is changed, forwarding proceeds in the usual way of always keeping two copies of the packet per path.

Such an example is shown in Figure 8.4. Due to a faulty link, communication between the two copies of the packet has been broken. As a result, each copy changes its token to

Replica, or generates a Replica token and proceeds to the destination using different paths. When the destination receives a packet with a Replica token, it knows that it should be looking for duplicates. This scheme is based on the assumption that packets have unique identifiers so that duplicates can be detected and eliminated.

8.2.2 Flit-Level Implementation of the UTP

The actual implementation of the Unique Token Protocol occurs at the flit level rather than at the packet level. A long packet may span a number of nodes. The flit-level UTP guarantees that each flit of the packet has a copy for retransmission purposes in the neighboring node. A snapshot of a packet in flight under the flit-level UTP is shown in Figure 8.5. The figure shows a packet that consists of one head flit, seven data flits, and one tail flit. There are a number of things to notice from that figure:

1. There exists a second copy of every data or tail flit in the preceding node.
2. There is only one copy of the token flit. There exist no flits of the specific packet behind its token.
3. The head flit is stored at the head of the flit queue in every node spanned by the packet. It is deallocated only when the token flit leaves the node.

Every node needs a copy of the head flit to ensure retransmission of the partial packet when a link fails after only part of a packet has been transmitted to the next node. The head flit is used to encapsulate the partial packet in the regular packet format and send it to the destination through an alternate route. The head flit of the trailing piece of a partial packet is tagged as a special kind of head flit – in the Reliable Router terminology it is called a *head:restart* flit as opposed to a *head:original* flit. This is necessary for the destination to reconstruct the original packet. It is also assumed that the tail flit of each packet contains the length of the message in flits. Given two partial pieces of a packet, the packet length and the the relative order of the two pieces, the destination can reconstruct unambiguously the original packet.

8.3 Summary

This chapter has briefly covered some system design issues for a routing chip that uses Reliable Adaptive Routing. First, a top level architectural description of such a switching

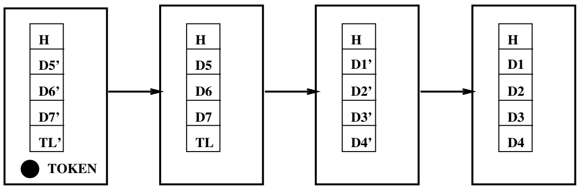

Figure 8.5: The UTP at the flit-level.

element has been presented. Second, a retransmission protocol necessary for guaranteed message delivery has been desribed and explained.

Chapter 9

Conclusion

Adaptive routing has been extensively studied. It has been concluded that it is ideally suited for multicomputer networks because it can take advantage of the rich connectivity of such topologies [Nga89]. Researchers have amassed simulation results that indicate that adaptive routing can increase interconnection network performance. The benefits of adaptive routing are especially profound for non-uniform traffic patterns. The ability of adaptive routing to assign multiple paths between each source and destination pair can diffuse hot spot traffic and achieve better utilization of the network bisection bandwidth.

There are two main disadvantages with adaptive routing. The first is that it introduces significant complexity to the design and implementation of the network switching element. Such complexity is introduced by the need for virtual channels for deadlock prevention, more complicated combinational routing logic, and the need for a full crossbar switch connecting each input port with each output port. Virtual channels are probably the most expensive (in terms of circuit area and latency) not only because of the extra flit memory required but also because of various arbitration layers necessary for the management and allocation of shared resources. Adaptive routing can significantly increase system cycle time and system silicon area compared to simpler routing schemes. Increased cycle time is one of the main reasons that we do not find adaptive routing in commercial multicomputers [Dua94b]. Unfortunately there are no widely accepted benchmarks for multicomputer interconnection networks. As a result, multicomputer manufacturers advertise their machines using peak values (peak throughput and minimum switching node latency.) The increased cycle time of adaptive routing increases the minimum switching node delay although it may also decrease the average message latency and standard deviation. As long as manufacturers advertise their machines with peak values, we will probably not see adaptive routing being used in commercial multicomputers.

Yet, trends in VLSI technology will benefit adaptive routing in the long run [Cas94]. Continuous decreases in the minimum mask feature size (2λ) make possible to fit more and more transistors in a single silicon die. As a result, the extra area required for adaptive routing and virtual channels will soon be available at negligible cost. The fact that most routing chips are pad limited due to a large number of I/O pins also contributes to the argument of free silicon area.

Another effect of decreasing MOS transistor channel length is the reduction of the carrier transit time across the channel. This will increase the speed of future logic gates. In a few years, the speed of routing chips won't be limited by the delay of the routing logic. It will be limited by how fast on-chip circuitry can drive package pins and external interconnection wires. Unfortunately the speed of external wiring does not follow the dramatic increase in device speed. In such a case, the extra delay required by adaptive routing logic and virtual channel management will also be available at negligible cost.

The second disadvantage of adaptive routing is the significant design time involved. The designer of an adaptive routing chip is faced with the daunting task of selecting an adaptive routing algorithm to implement. Oblivious router designers do not face such a task. Dimension-Ordered routing is well studied, well understood and very easy to implement. A number of hardware implementations of Dimension-Ordered routing are available for designers to use as a valuable reference. Although numerous adaptive algorithms have been proposed in the literature none of these have been extensively studied and understood. Adaptive algorithms only exist in the form of numerous computer simulators which have different and inconsistent assumptions regarding traffic patterns, size of node buffering, deadlock resolution, packet size, cycle time, result presentation etc. The task of picking the right adaptive routing algorithm to implement in hardware can be both difficult and time consuming. Hardware design time can depend greatly on the underlying routing algorithm. Design complexity is the main reason that there are no adaptive routing chips today.

The main contribution of this work is that it fills the existing gap between algorithm design and hardware implementation. This thesis has proposed Reliable Adaptive Routing, a fully adaptive algorithm that can also handle a single fault at a time. RAR has been designed with hardware implementation in mind. It is not optimal in that it does not allow the maximum possibe routing freedom given the resources that it requires [Dua94a]. Yet, it can be easily implemented in hardware without significant increases in chip area and system latency. A proof-of-concept implementation ranging from gate-level schematics to silicon layout has been presented and discussed. The proposed implementation does not require significant chip area although it supports five separate virtual channels. A channel state predecoding scheme (chapter 7) has helped reduce the gate count and circuit area. Moreover, the proposed implementation does not add considerably to the total system latency. Extensive lookahead techniques (chapter 7) have helped reduce the circuit delay to acceptable levels. Finally, a top level system design that utilizes Reliable Adaptive Routing is briefly presented.

The author hopes that this work will motivate further attempts to implement adaptive routing in hardware by suggesting a simple and easy fully adaptive algorithm geared to hardware implementation and also by presenting to a full extent a circuit realisation.

9.1 Future Work

All performance evaluations of adaptive routing have relied on software simulators. The Reliable Router chip will act as an excellent research vehicle in helping us understand the performance of adaptive routing on real hardware. The Reliable Router has a number of features that can help us run a number of interesting experiments comparing adaptive and oblivious routing.

The chip allows the user to turn off adaptive routing and use the adaptive virtual channels as if they were Dimension-Ordered channels. This feature enables a fair comparison of the two routing schemes on real hardware, something that has never been done before.

Moreover, the Reliable Router allows the user to control the number of virtual channels that are actually used. in the network. This feature enables a number of interesting experiments which will relate the number of virtual channels in the network with the resulting performance. Experiments measuring incremental performance gained by the addition of adaptive channels vs. Dimension-Ordered channels can also be easily set up. Such experiments have never before been run on real hardware.

Appendix A

Circuit Schematics

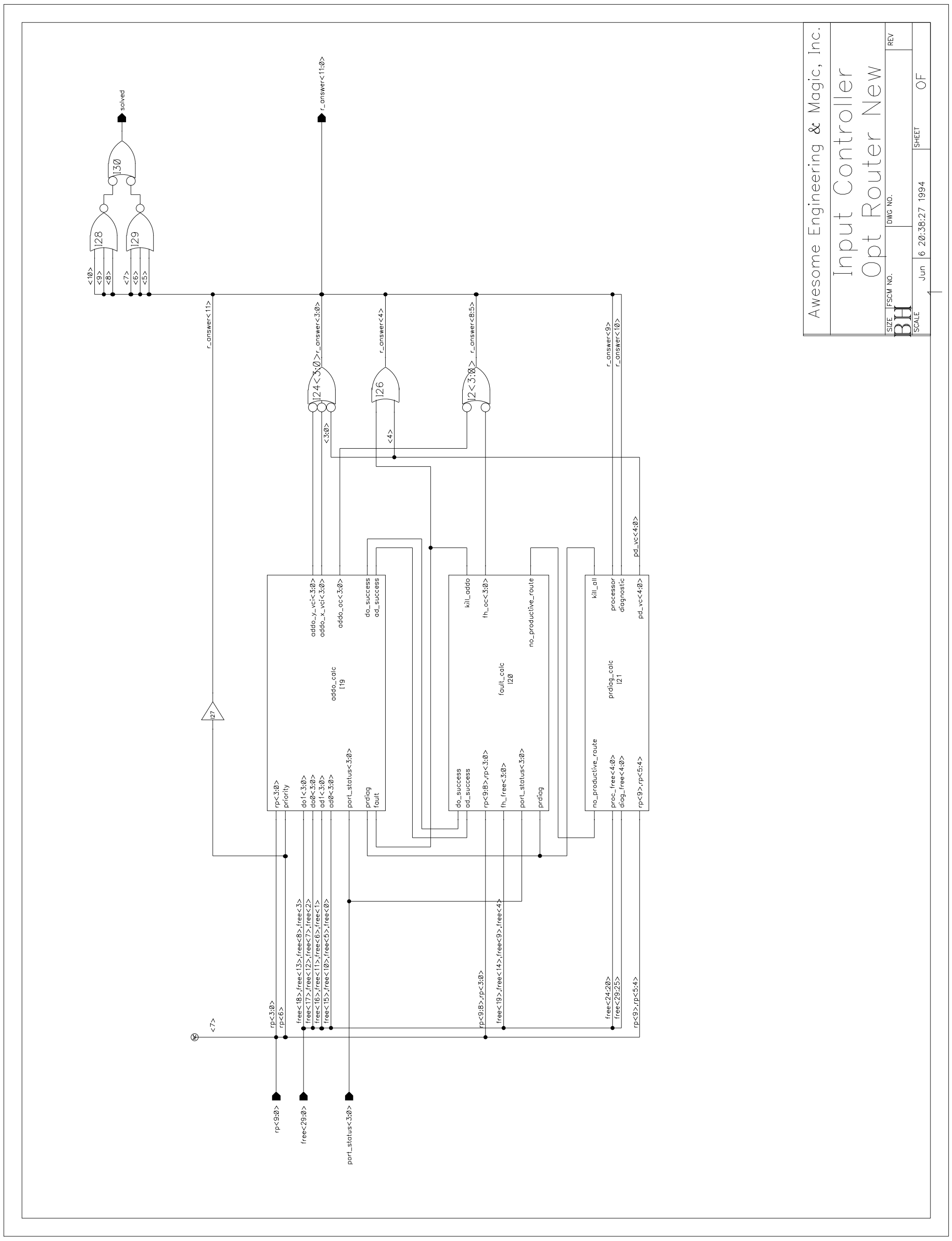

Figure A.1: Top level schematic

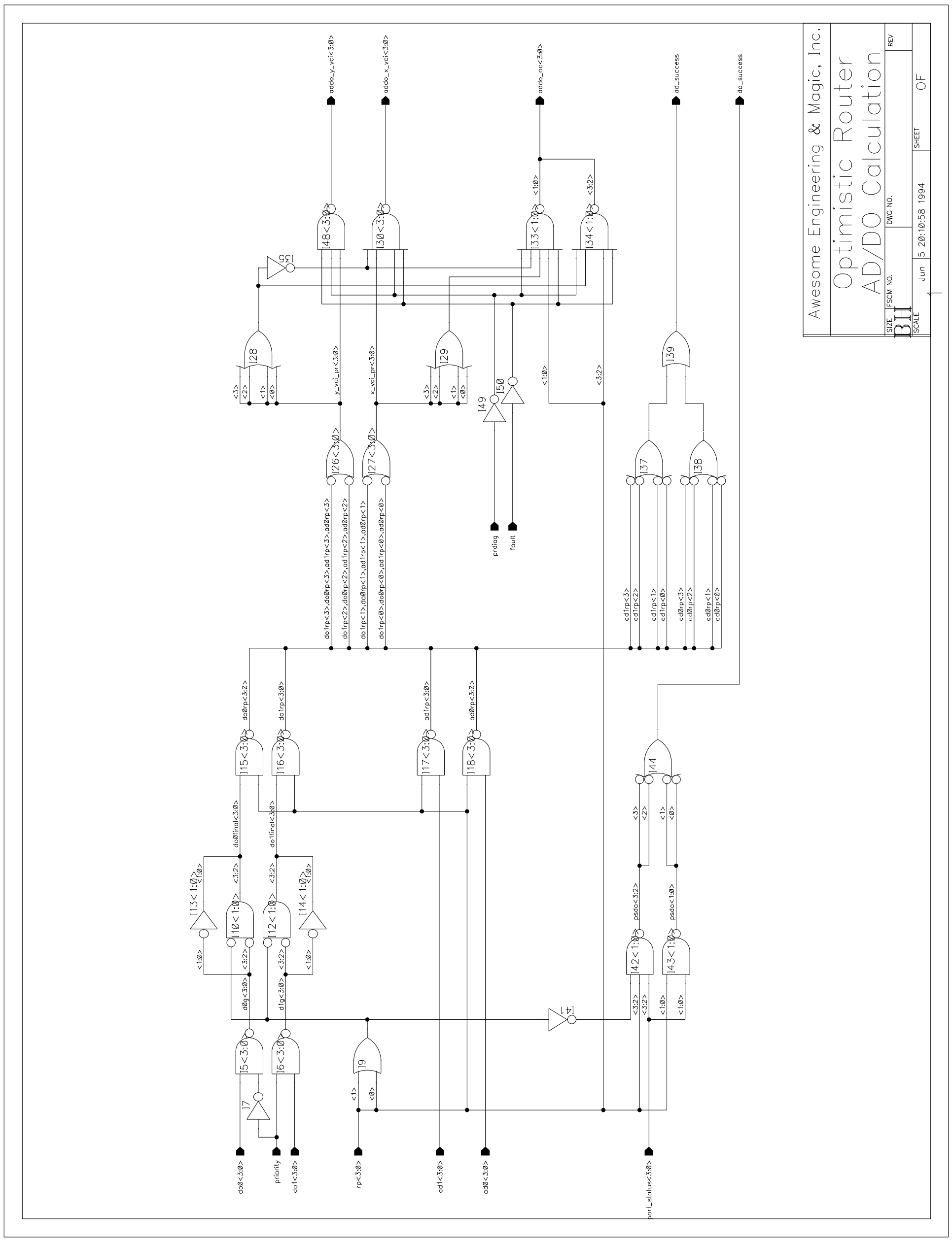

Figure A.2: Adaptive and Dimension-Ordered computation

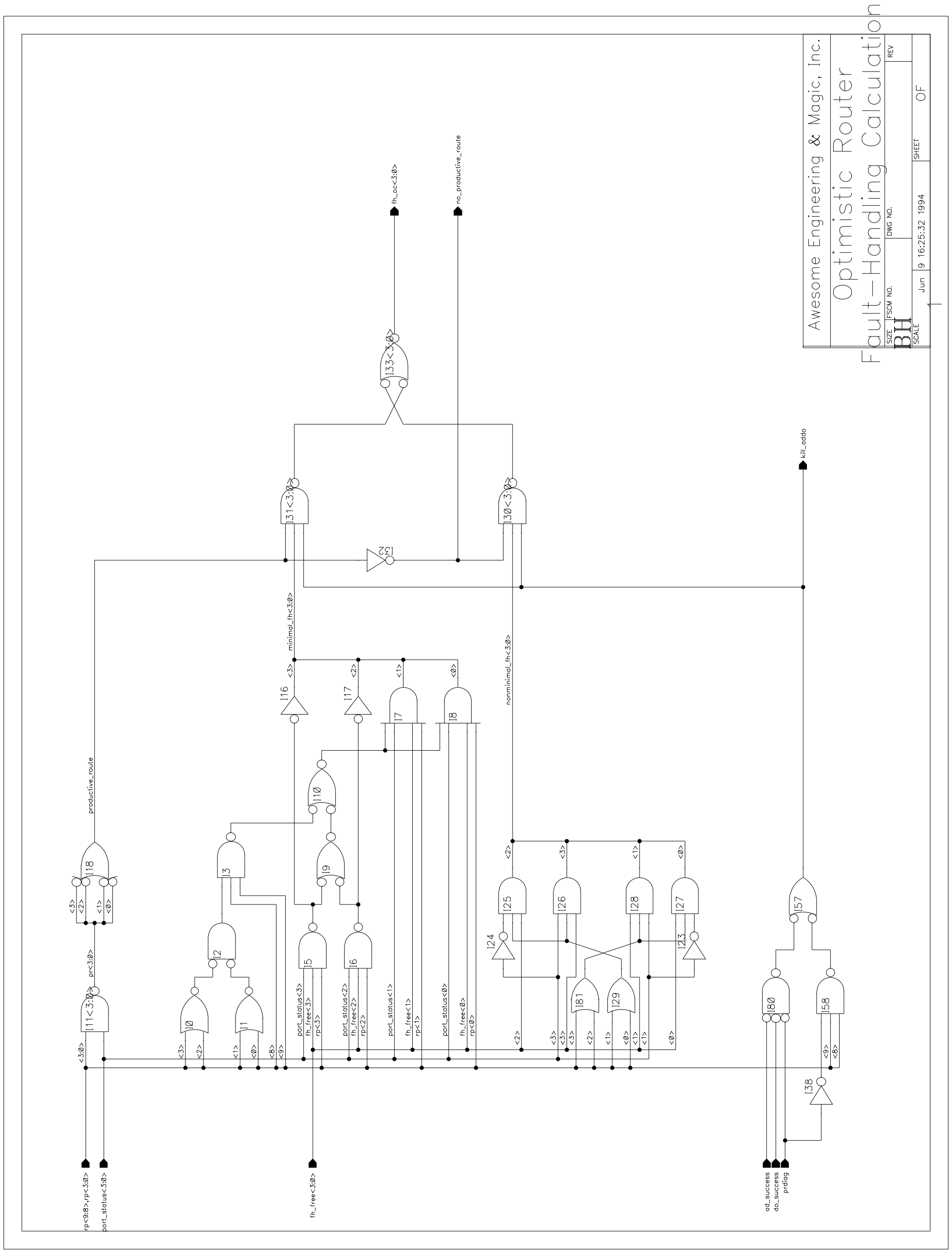

Figure A.3: Fault-Handling computation

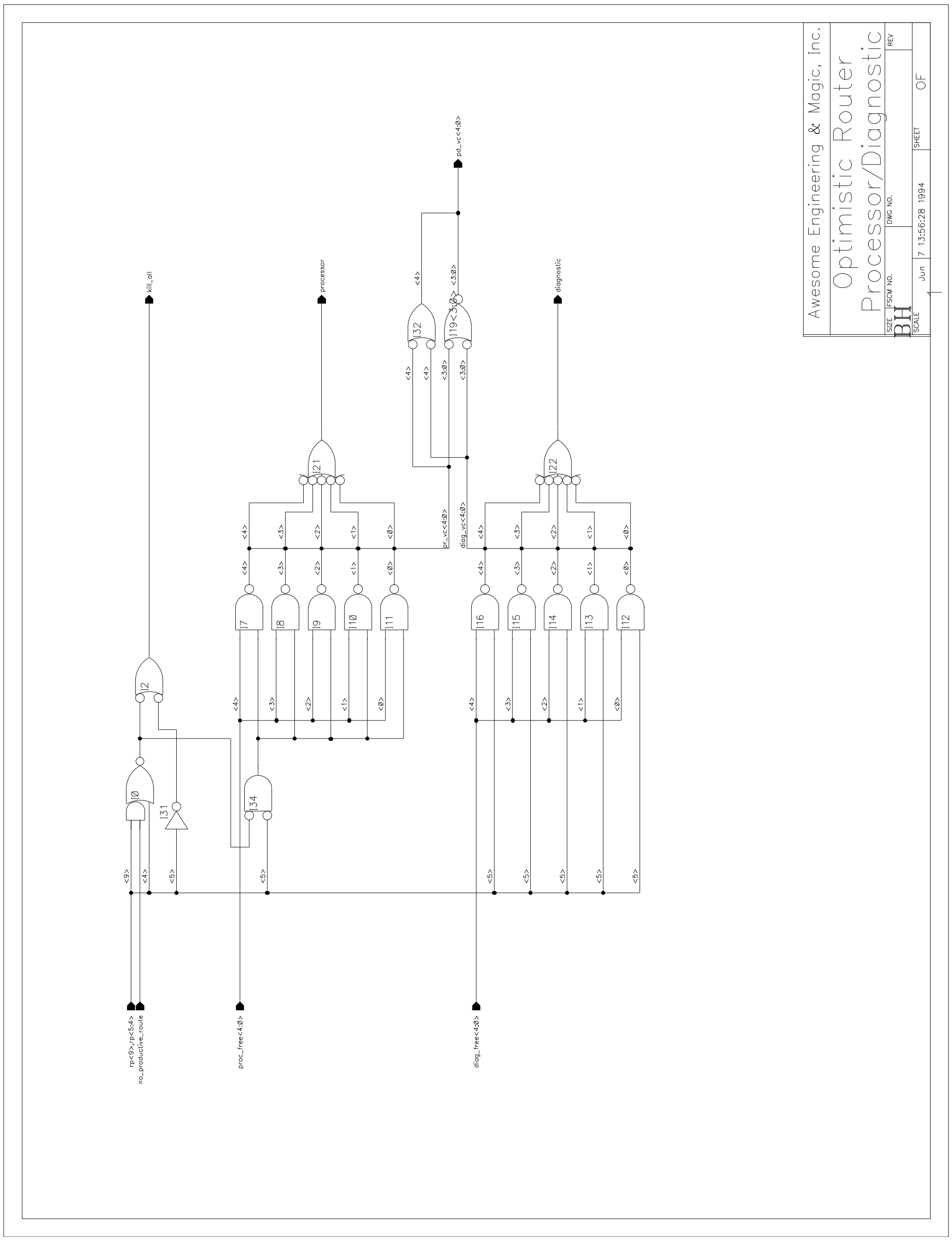

Figure A.4: Processor and Diagnostic computation

Appendix B

Verilog Schematic Validation Wrapper

```
`timescale 1ns / 10ps

module test;


  wire  solved;
  wire [11:0]  r_answer;
  wire [29:0]  free;
  reg [3:0]  port_status;
  reg [9:0]  rp;
  opt_router top(r_answer[11:0], solved, free[29:0],
 port_status[3:0], rp[9:0]);


  // Comprehensive testing of the entire optimistic router.

    // Try to cover the whole design space.
  // Possible channel status:

  reg [4:0] pcs [11:0];
  reg [4:0] pfree, dfree;

  initial
    begin
      pcs[0] = 5'b00000;
```

```
    pcs[1] = 5'b00001;
    pcs[2] = 5'b00010;
    pcs[3] = 5'b00100;
    pcs[4] = 5'b01000;
    pcs[5] = 5'b01100;
    pcs[6] = 5'b10000;
    pcs[7] = 5'b10001;
    pcs[8] = 5'b10010;
    pcs[9] = 5'b10100;
    pcs[10] = 5'b11000;
    pcs[11] = 5'b11100;
  end

// Possible proc_free values:
reg [4:0] ppf [5:0];
initial
  begin
    ppf[0]=5'b00000;
    ppf[1]=5'b00001;
    ppf[2]=5'b00010;
    ppf[3]=5'b00100;
    ppf[4]=5'b01000;
    ppf[5]=5'b10000;
  end

// Possible diag_free values:
reg [4:0] pdf [5:0];
initial
  begin
    pdf[0]=5'b00000;
    pdf[1]=5'b00001;
    pdf[2]=5'b00010;
    pdf[3]=5'b00100;
    pdf[4]=5'b01000;
    pdf[5]=5'b10000;
  end
```

```
reg [4:0] port3, port2, port1, port0;

// Now do the actual assignment of free bits:

assign free[29:0]={pfree,dfree,port3,port2,port1,port0} &
    {10'b1111111111,port_status[3],port_status[3],
     port_status[3],port_status[3],port_status[3],
     port_status[2],port_status[2],port_status[2],
     port_status[2],port_status[2],
     port_status[1],port_status[1],
     port_status[1],port_status[1],port_status[1],
     port_status[0],port_status[0],port_status[0],
     port_status[0],port_status[0]};

wire [3:0] ad0, ad1, do0, do1, fh_free;
wire [4:0] proc_free, diag_free;
wire priority;

assign ad0={free[15],free[10],free[5],free[0]};
assign ad1={free[16],free[11],free[6],free[1]};
assign do0={free[17],free[12],free[7],free[2]};
assign do1={free[18],free[13],free[8],free[3]};
assign fh_free={free[19],free[14],free[9],free[4]};

assign proc_free[4:0]=free[24:20];
assign diag_free[4:0]=free[29:25];

assign priority=rp[6];

// Possible routing problems:

reg [7:0] prp70 [19:0];
initial
  begin
    prp70[0]  = 8'b00010000;
    prp70[1]  = 8'b00000001;
    prp70[2]  = 8'b00000010;
    prp70[3]  = 8'b00000100;
```

```
    prp70[4]  = 8'b00001000;
    prp70[5]  = 8'b00001001;
    prp70[6]  = 8'b00001010;
    prp70[7]  = 8'b00000110;
    prp70[8]  = 8'b00000101;

    prp70[9]  = 8'b00100000;
    prp70[10]  = 8'b01100000;
    prp70[11]  = 8'b01010000;

    prp70[12] = 8'b01000001;
    prp70[13] = 8'b01000010;
    prp70[14] = 8'b01000100;
    prp70[15] = 8'b01001000;
    prp70[16] = 8'b01001001;
    prp70[17] = 8'b01001010;
    prp70[18] = 8'b01000110;
    prp70[19] = 8'b01000101;

  end

reg [1:0] prp98 [3:0];
initial
  begin
    prp98[0]=2'b00;
    prp98[1]=2'b01;
    prp98[2]=2'b10;
    prp98[3]=2'b11;
  end


// Possible port status:

reg [3:0] pps [10:0];

initial
  begin
    pps[0]  = 4'b1111;
    pps[1]  = 4'b1110;
```

```
      pps[2]  = 4'b1101;
      pps[3]  = 4'b1011;
      pps[4]  = 4'b0111;
      pps[5]  = 4'b0011;
      pps[6]  = 4'b0101;
      pps[7]  = 4'b0110;
      pps[8]  = 4'b1001;
      pps[9]  = 4'b1010;
      pps[10] = 4'b1100;
    end

  integer i0, i1, i2, i3, irp70, irp98, ips, ipf, idf, count;
  integer total, state_space, antipode;

  initial
      total=0;

  reg ad_success, do_success, fh, productive_route, kill_all_others;
  reg [3:0] do_rp;

  reg [11:0] c_answer;
  reg c_solved;

  reg stopint;


  // This is the main test loop:
  initial
    begin
      // Channel status state space (527076 cases)
      for (i0=0;i0<=11;i0=i0+1)
begin
  $display("i0= %d", i0);
  port0=pcs[i0];
  for (i1=0;i1<=11;i1=i1+1)
    begin
      $display("  i1= %d", i1);
      port1=pcs[i1];
      for (i2=0;i2<=11;i2=i2+1)
```

```
begin
  $display("    i2= %d", i2);
  port2=pcs[i2];
  for (i3=0;i3<=11;i3=i3+1)
    begin
      $display("      i3= %d", i3);
      port3=pcs[i3];
      for (ipf=0;ipf<=5;ipf=ipf+1)
begin
  pfree=ppf[ipf];
  for(idf=0;idf<=5;idf=idf+1)
    begin
      dfree=pdf[idf];

      // Port status state space (11 cases)
      for (ips=0;ips<=10;ips=ips+1)
begin
  port_status=pps[ips];

  // Routing problem state space (80 cases)
  for (irp70=0;irp70<=19;irp70=irp70+1)
    begin
      rp[7:0] = prp70[irp70];


      if (!(((rp[3:0] == 4'b1001) && (port_status[3:0] == 4'b0110)) ||
    ((rp[3:0] == 4'b1010) && (port_status[3:0] == 4'b0101)) ||
    ((rp[3:0] == 4'b0110) && (port_status[3:0] == 4'b1001)) ||
    ((rp[3:0] == 4'b0101) && (port_status[3:0] == 4'b1010))))
begin


  for (irp98=0;irp98<=3;irp98=irp98+1)
    begin
      rp[9:8]=prp98[irp98];
      #30; // Wait for inputs to settle.

      // Initialize some state;
```

```
      c_solved=0;
      c_answer[11:0]=11'b0;
      do_rp[3:0]=rp;
      // Set the dimension-ordered routing problem.
      if (|rp[1:0])
  do_rp[3:0]={2'b0,rp[1:0]};

      // Set some flags
      ad_success=|(rp[3:0] & (ad1[3:0] | ad0[3:0]) & port_status[3:0]);

      do_success=|(do_rp[3:0] & port_status[3:0]);
      productive_route=(|(port_status[3:0]&rp[3:0])) || (rp[3:0]==4'b0000);
      kill_all_others=(rp[5] || rp[4] || (rp[9] & (~productive_route)));
      fh=((~do_success) & (~ad_success) & (~kill_all_others)) |
  (rp[9] & rp[8] & (~kill_all_others));


      // ROUTING ALGORITHM IMPLEMENTATION

      // First set the priority bit.
      c_answer[11]=rp[6];

      // First check for diagnostic:
      if (rp[5] && (rp[3:0] == 4'b0000))
begin
  c_answer[4:0]=diag_free;
  if (|diag_free[4:0])
    begin
      c_answer[10]=1;
      c_solved=1;
    end
end // end checking for diagnostic

      // Now check for processor:
      if (rp[4])
begin
  c_answer[4:0]=proc_free;
  if (|proc_free[4:0])
    begin
```

```
      c_answer[9]=1;
      c_solved=1;
    end
end //end checking for processor

      // Check whether we should route to the neighbor.
      if (rp[9] & (~productive_route))
begin
  c_answer[4:0]=proc_free;
  if (|proc_free[4:0])
    begin
      c_answer[9]=1;
      c_solved=1;
    end
end //end checking for neighbor routing


      // Check for fault-handling
      if (fh)
begin
  c_answer[4]=1;
  // Check for non-minimal fh first
  if (~productive_route)
    begin
      stopint = 0;
      count = 3;
      while ((stopint != 1) && (count>=0))
begin
  if (count % 2 == 0)
      antipode = count + 1;
  else
      antipode = count - 1;
  if (port_status[count] && ~rp[count] && ~rp[antipode])
    begin
      c_answer[5+count] = fh_free[count]; //oc assignment
      //c_answer[4]=1; // ovc assignment
      c_solved=fh_free[count];
      stopint = 1;
```

```
    end
  count = count-1;
end //while
    end //if (~productive_route)

  // Now check for minimal fh
  else
    begin
      count = 3;
      stopint = 0;
      while ((stopint != 1) && (count>=0))
begin
  if (rp[count] & fh_free[count] &port_status[count])
    begin
      c_answer[5+count] = 1; //oc assignment
      //c_answer[4]=1; //ovc assignment
      c_solved=1;
      stopint = 1;
    end
  if ((c_solved==1) && (rp[3:2] !== 2'b00) &&
      ((c_answer[5]==1)||(c_answer[6]==1)))
    begin
      c_solved=0;
      c_answer[6:5] = 2'b00;
      stopint=0;
    end
  count = count-1;
end //while
    end // else (productive_route)

end // if (fh)

      // Now check for adaptive channels
      if ((~fh) & (~kill_all_others))
begin
  // First try the adaptive channels. y has priority over x.
  stopint=0;
  count = 3;
  while ((count>=0) && (stopint ==0))
```

```
    begin
      if (port_status[count] && (ad0[count] || ad1[count]) &&
  rp[count])
begin
  c_answer[5+count]=1;
  c_answer[4:0] = {3'b0,ad1[count],ad0[count]};
  c_solved=1;
  stopint=1;
end
      count = count-1;
    end // ad while loop

  // Now try the dimension ordered channels if the
  // adaptive channels have failed.

  if (!ad_success)
    begin
      // Reset some state
      stopint=0;
      count=0;

      // Check for priority 1
      if (priority)
begin
  while ((count<=3) && (stopint==0))
    begin
      if (port_status[count] && (do1[count]) &&
  do_rp[count])
begin
  c_answer[5+count]=1;
  c_answer[4:0] = {1'b0,do1[count],3'b0};
  c_solved=1;
  stopint=1;
end
      count = count+1;
    end // do1 while loop
end   // checking for priority 1
      // Check for priority 0
```

```
      if (!priority)
begin
  while ((count<=3) && (stopint==0))
    begin
      if (port_status[count] && (do0[count]) &&
  do_rp[count])
begin
  c_answer[5+count]=1;
  c_answer[4:0] = {2'b0,do0[count],2'b0};
  c_solved=1;
  stopint=1;
end
      count = count+1;
    end // do0 while loop
end  // checking for priority 1

    end  // if (!ad_success)

end // if ((~fh) & (~kill_all_others))

      // Now do the comparisons:

      if ((c_solved !== solved) ||
  (c_answer[11:0] !== r_answer[11:0]))
begin
  $display("Error !!!!!!!!!!!!!");
  $display("c_solved=%b, solved=%b", c_solved, solved);
  $display("c_oc=%b, oc=%b", c_answer[10:5], r_answer[10:5]);
  $display("c_ovc=%b, ovc=%b", c_answer[4:0], r_answer[4:0]);
  $display("c_pri=%b, pri=%b", c_answer[11], r_answer[11]);
  $display(" ");
  $display("Inputs:");
  $display(" ");
  $display("routing problem=%b", rp[9:0]);
  $display("port status=%b", port_status[3:0]);
  $display("ad0=%b", ad0);
  $display("ad1=%b", ad1);
  $display("do0=%b", do0);
```

```
  $display("do1=%b", do1);
  $display("fh_free=%b", fh_free);
  $display("proc_free=%b", proc_free);
  $display("diag_free=%b", diag_free);
  $display(" ");
  $display("---------------------------------");
  $display(" ");
  $stop;
end


    end //irp98
end //big if
    end //irp70
end // ips
    end //idf
end //ipf
    end //i3
end //i2
    end //i1
end //i0
    end // main initial

endmodule
```

Appendix C

Hspice Decks

C.1 Wrapping Deck and Input Generation

```
* Optimistic Router- Nominal Corner
.include '/projects/abacus/lib/hspice/hp26/Nominal.model'
.include 'netlist'

.options post nolist nomod

vdd  vdd gnd 3.3v

Vport_status3 port_status3 0 vdd
Vport_status2 port_status2 0 vdd
Vport_status1 port_status1 0 vdd
Vport_status0 port_status0 0 vdd

Vrp9 rp9 0 0
Vrp8 rp8 0 0
Vrp7 rp7 0 0
Vrp6 rp6 0 0
Vrp5 rp5 0 0
Vrp4 rp4 0 PWL(0n 3.3v 20n 3.3v 20.5n 0v 40n 0v 40.5n 3.3v)
Vrp3 rp3 0 0
Vrp2 rp2 0 0
Vrp1 rp1 0 0
Vrp0 rp0 0 PWL(0n 0v 20n 0v 20.5n 3.3v 40n 3.3v 40.5n 0v)
```

```
Vfree29 free29 0 0
Vfree28 free28 0 0
Vfree27 free27 0 0
Vfree26 free26 0 0
Vfree25 free25 0 0
Vfree24 free24 0 0
Vfree23 free23 0 0
Vfree22 free22 0 0
Vfree21 free21 0 0
Vfree20 free20 0 0

Vfree19 free19 0 0
Vfree18 free18 0 0
Vfree17 free17 0 0
Vfree16 free16 0 0
Vfree15 free15 0 0
Vfree14 free14 0 0
Vfree13 free13 0 0
Vfree12 free12 0 0
Vfree11 free11 0 0
Vfree10 free10 0 0

Vfree9 free9 0 0
Vfree8 free8 0 0
Vfree7 free7 0 0
Vfree6 free6 0 0
Vfree5 free5 0 0
Vfree4 free4 0 vdd
Vfree3 free3 0 0
Vfree2 free2 0 vdd
Vfree1 free1 0 0
Vfree0 free0 0 0


.trans .5ns 60ns


.end
```

C.2 Optimistic Router Netlist

```
****************************************************************************
* HSPICE Netlist:
*
* Block: test_router
* Netlist Time: Tue Dec 13 00:00:44 EST 1994
****************************************************************************


****************************************************************************
* GLOBAL Net Declarations
****************************************************************************
.global gnd vdd


****************************************************************************
* MODEL Declarations
****************************************************************************


****************************************************************************
* Sub-Circuit Netlist:
*
* Block: inv
* Last Time Saved: Jun 21 12:06:20 1994
****************************************************************************
.subckt inv a y
mx1 y a vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
mx0 y a gnd gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
.ends inv


****************************************************************************
* Sub-Circuit Netlist:
*
* Block: nor3
* Last Time Saved: Aug  2 13:09:10 1994
****************************************************************************
.subckt nor3 a b c y
mx5 net120 a vdd vdd pmos w=1.2e-05 l=1e-06 as=3e-11 ad=3e-11 ps=1.7e-05
+pd=1.7e-05
mx3 y c net117 vdd pmos w=1.2e-05 l=1e-06 as=3e-11 ad=3e-11 ps=1.7e-05
```

```
+pd=1.7e-05
mx4 net117 b net120 vdd pmos w=1.2e-05 l=1e-06 as=3e-11 ad=3e-11 ps=1.7e-05
+pd=1.7e-05
mx2 y c gnd gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
mx1 y b gnd gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
mx0 y a gnd gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
.ends nor3

******************************************************************************
* Sub-Circuit Netlist:
*
* Block: inv3x
* Last Time Saved: Jun 17 10:54:44 1994
******************************************************************************
.subckt inv3x a y
mxp8 y a vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
mx1 y a vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
mxp10 y a vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
+
mxn11 y a gnd gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
mxn9 y a gnd gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
mx0 y a gnd gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
.ends inv3x

******************************************************************************
* Sub-Circuit Netlist:
*
* Block: buffer
* Last Time Saved: Aug 16 11:41:24 1994
******************************************************************************
.subckt buffer a y
xi11 net5 y inv3x
xi1 a net5 inv
.ends buffer

******************************************************************************
* Sub-Circuit Netlist:
*
```

```
* Block: or2
* Last Time Saved: Aug 22 10:54:42 1994
****************************************************************************
.subckt or2 a b y
mxn16 net99 a gnd gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
+
mxn17 net99 b gnd gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
+
mxp15 net99 b net118 vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05
+pd=1.3e-05
mxp14 net118 a vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05
+pd=1.3e-05
xi13 net99 y inv
.ends or2

****************************************************************************
* Sub-Circuit Netlist:
*
* Block: nand2
* Last Time Saved: Aug  2 13:09:33 1994
****************************************************************************
.subckt nand2 a b y
mx3 y a vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
mx2 y b vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
mxn6 y a net9 gnd nmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
+
mx0 net9 b gnd gnd nmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05
+pd=1.3e-05
.ends nand2

****************************************************************************
* Sub-Circuit Netlist:
*
* Block: nand3
* Last Time Saved: Aug  2 13:09:23 1994
****************************************************************************
.subckt nand3 a b c y
mx5 y a vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
mx4 y b vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
```

```
mx3 y c vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
mx2 y a net17 gnd nmos w=1.2e-05 l=1e-06 as=3e-11 ad=3e-11 ps=1.7e-05
+pd=1.7e-05
mx1 net17 b net14 gnd nmos w=1.2e-05 l=1e-06 as=3e-11 ad=3e-11 ps=1.7e-05
+pd=1.7e-05
mx0 net14 c gnd gnd nmos w=1.2e-05 l=1e-06 as=3e-11 ad=3e-11 ps=1.7e-05
+pd=1.7e-05
.ends nand3

*****************************************************************************
* Sub-Circuit Netlist:
*
* Block: nor2
* Last Time Saved: Aug 21 15:33:39 1994
*****************************************************************************
.subckt nor2 a b y
mx3 net105 a vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05
+pd=1.3e-05
mx2 y b net105 vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05
+pd=1.3e-05
mx1 y b gnd gnd nmos w=2e-06 l=1e-06 as=5e-12 ad=5e-12 ps=7e-06 pd=7e-06
mx0 y a gnd gnd nmos w=2e-06 l=1e-06 as=5e-12 ad=5e-12 ps=7e-06 pd=7e-06
.ends nor2

*****************************************************************************
* Sub-Circuit Netlist:
*
* Block: and2
* Last Time Saved: Nov  3 17:16:16 1993
*****************************************************************************
.subckt and2 a b y
mx5 y net235 gnd gnd nmos w=2e-06 l=1e-06 as=5e-12 ad=5e-12 ps=7e-06 pd=7e-06
mx2 net235 a net240 gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06
+pd=9e-06
mx3 net240 b gnd gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
mx4 y net235 vdd vdd pmos w=2e-06 l=1e-06 as=5e-12 ad=5e-12 ps=7e-06 pd=7e-06
mx1 net235 b vdd vdd pmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
mx0 net235 a vdd vdd pmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
```

```
.ends and2

************************************************************************
* Sub-Circuit Netlist:
*
* Block: aoi12
* Last Time Saved: May 26 19:41:22 1994
************************************************************************
.subckt aoi12 a b c y
mxp5 y b net11 vdd pmos w=1.6e-05 l=1e-06 as=4e-11 ad=4e-11 ps=2.1e-05
+pd=2.1e-05
mxp4 y a net11 vdd pmos w=1.6e-05 l=1e-06 as=4e-11 ad=4e-11 ps=2.1e-05
+pd=2.1e-05
mxp3 net11 c vdd vdd pmos w=1.6e-05 l=1e-06 as=4e-11 ad=4e-11 ps=2.1e-05
+pd=2.1e-05
mxn2 y c gnd gnd nmos w=2e-06 l=1e-06 as=5e-12 ad=5e-12 ps=7e-06 pd=7e-06
mxn1 net20 b gnd gnd nmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05
+pd=1.3e-05
mxn0 y a net20 gnd nmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05
+pd=1.3e-05
.ends aoi12

************************************************************************
* Sub-Circuit Netlist:
*
* Block: nand5
* Last Time Saved: Aug  3 15:23:38 1994
************************************************************************
.subckt nand5 a b c d e y
mxp17 y e vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
+
mx8 y d vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
mx7 y c vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
mx6 y b vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
mx5 y a vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
mxn18 net376 e gnd gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06
+pd=9e-06
mx3 y a net295 gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
```

```
mx2 net295 b net302 gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06
+pd=9e-06
mx1 net302 c net301 gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06
+pd=9e-06
mx0 net301 d net376 gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06
+pd=9e-06
.ends nand5

******************************************************************************
* Sub-Circuit Netlist:
*
* Block: prdiag_calc
* Last Time Saved: Jun  7 13:56:28 1994
******************************************************************************
.subckt prdiag_calc diag_free4 diag_free3 diag_free2 diag_free1 diag_free0
+diagnostic kill_all no_productive_route pd_vc4 pd_vc3 pd_vc2 pd_vc1 pd_vc0
+proc_free4 proc_free3 proc_free2 proc_free1 proc_free0 processor rp9 rp5 rp4
xi34 net5516 rp5 net5554 nor2
xi193 pr_vc3 diag_vc3 pd_vc3 and2
xi192 pr_vc2 diag_vc2 pd_vc2 and2
xi191 pr_vc1 diag_vc1 pd_vc1 and2
xi190 pr_vc0 diag_vc0 pd_vc0 and2
xi31 rp5 net5535 inv
xi0 rp9 no_productive_route rp4 net5516 aoi12
xi22 diag_vc4 diag_vc3 diag_vc2 diag_vc1 diag_vc0 diagnostic nand5
xi21 pr_vc4 pr_vc3 pr_vc2 pr_vc1 pr_vc0 processor nand5
xi11 proc_free0 net5554 pr_vc0 nand2
xi16 diag_free4 rp5 diag_vc4 nand2
xi15 diag_free3 rp5 diag_vc3 nand2
xi14 diag_free2 rp5 diag_vc2 nand2
xi13 diag_free1 rp5 diag_vc1 nand2
xi9 proc_free2 net5554 pr_vc2 nand2
xi7 proc_free4 net5554 pr_vc4 nand2
xi8 proc_free3 net5554 pr_vc3 nand2
xi10 proc_free1 net5554 pr_vc1 nand2
xi12 diag_free0 rp5 diag_vc0 nand2
xi32 pr_vc4 diag_vc4 pd_vc4 nand2
xi2 net5516 net5535 kill_all nand2
```

```
.ends prdiag_calc

*****************************************************************************
* Sub-Circuit Netlist:
*
* Block: or3
* Last Time Saved: Nov  5 12:57:50 1993
*****************************************************************************
.subckt or3 a b c y
xi12 a b c net99 nor3
xi13 net99 y inv
.ends or3

*****************************************************************************
* Sub-Circuit Netlist:
*
* Block: and3
* Last Time Saved: Nov  5 10:46:25 1993
*****************************************************************************
.subckt and3 a b c y
xi12 net154 y inv
xi11 a b c net154 nand3
.ends and3

*****************************************************************************
* Sub-Circuit Netlist:
*
* Block: nand4
* Last Time Saved: Aug  2 17:06:09 1994
*****************************************************************************
.subckt nand4 a b c d y
mx8 y d vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
mx7 y c vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
mx6 y b vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
mx5 y a vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05 pd=1.3e-05
mx3 y a net295 gnd nmos w=1.2e-05 l=1e-06 as=3e-11 ad=3e-11 ps=1.7e-05
+pd=1.7e-05
mx2 net295 b net302 gnd nmos w=1.2e-05 l=1e-06 as=3e-11 ad=3e-11 ps=1.7e-05
+pd=1.7e-05
```

```
mx1 net302 c net301 gnd nmos w=1.2e-05 l=1e-06 as=3e-11 ad=3e-11 ps=1.7e-05
+pd=1.7e-05
mx0 net301 d gnd gnd nmos w=1.2e-05 l=1e-06 as=3e-11 ad=3e-11 ps=1.7e-05
+pd=1.7e-05
.ends nand4

*************************************************************************
* Sub-Circuit Netlist:
*
* Block: and4
* Last Time Saved: Nov  5 11:33:29 1993
*************************************************************************
.subckt and4 a b c d y
xi15 net204 y inv
xi14 a b c d net204 nand4
.ends and4

*************************************************************************
* Sub-Circuit Netlist:
*
* Block: fault_calc
* Last Time Saved: Jun  9 16:25:32 1994
*************************************************************************
.subckt fault_calc ad_success do_success fh_free3 fh_free2 fh_free1 fh_free0
+fh_oc3 fh_oc2 fh_oc1 fh_oc0 kill_addo no_productive_route port_status3
+port_status2 port_status1 port_status0 prdiag rp9 rp8 rp3 rp2 rp1 rp0
xi80 ad_success do_success prdiag net674 or3
xi57 net674 net714 kill_addo nand2
xi27 fh_free0 net683 net707 nonminimal_fh0 and3
xi26 port_status3 net698 fh_free3 nonminimal_fh3 and3
xi28 fh_free1 net683 port_status1 nonminimal_fh1 and3
xi25 net705 net698 fh_free2 nonminimal_fh2 and3
xi29 rp1 rp0 net698 or2
xi81 rp3 rp2 net683 or2
xi38 prdiag net5813 inv
xi32 productive_route no_productive_route inv
xi23 port_status1 net707 inv
xi24 port_status3 net705 inv
xi18 pr3 pr2 pr1 pr0 productive_route nand4
```

```
xi113 rp3 port_status3 pr3 nand2
xi112 rp2 port_status2 pr2 nand2
xi111 rp1 port_status1 pr1 nand2
xi110 rp0 port_status0 pr0 nand2
xi0 rp3 rp2 net726 nor2
xi1 rp1 rp0 net724 nor2
xi2 net726 net724 net727 nor2
xi303 no_productive_route nonminimal_fh3 kill_addo net7330 nand3
xi302 no_productive_route nonminimal_fh2 kill_addo net7331 nand3
xi301 no_productive_route nonminimal_fh1 kill_addo net7332 nand3
xi300 no_productive_route nonminimal_fh0 kill_addo net7333 nand3
xi58 net5813 rp9 rp8 net714 nand3
xi313 productive_route minimal_fh3 kill_addo net6770 nand3
xi312 productive_route minimal_fh2 kill_addo net6771 nand3
xi311 productive_route minimal_fh1 kill_addo net6772 nand3
xi310 productive_route minimal_fh0 kill_addo net6773 nand3
xi6 port_status2 fh_free2 rp2 net737 nand3
xi5 port_status3 fh_free3 rp3 net753 nand3
xi3 net727 rp8 rp9 net750 nand3
xi333 net7330 net6770 fh_oc3 and2
xi332 net7331 net6771 fh_oc2 and2
xi331 net7332 net6772 fh_oc1 and2
xi330 net7333 net6773 fh_oc0 and2
xi9 net753 net737 net752 and2
xi10 net750 net752 net760 and2
xi7 net760 port_status1 fh_free1 rp1 minimal_fh1 and4
xi8 net760 port_status0 fh_free0 rp0 minimal_fh0 and4
xi16 net753 minimal_fh3 inv
xi17 net737 minimal_fh2 inv
.ends fault_calc

*****************************************************************************
* Sub-Circuit Netlist:
*
* Block: nor4
* Last Time Saved: Aug  3 14:22:34 1994
*****************************************************************************
.subckt nor4 a b c d y
mxp9 net3 a vdd vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05
```

```
+pd=1.3e-05
mx5 net159 b net3 vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05
+pd=1.3e-05
mx3 y d net156 vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05
+pd=1.3e-05
mx4 net156 c net159 vdd pmos w=8e-06 l=1e-06 as=2e-11 ad=2e-11 ps=1.3e-05
+pd=1.3e-05
mxn8 y a gnd gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
mx2 y d gnd gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
mx1 y c gnd gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
mx0 y b gnd gnd nmos w=4e-06 l=1e-06 as=1e-11 ad=1e-11 ps=9e-06 pd=9e-06
.ends nor4


****************************************************************************
* Sub-Circuit Netlist:
*
* Block: or4
* Last Time Saved: Nov  5 12:59:48 1993
****************************************************************************
.subckt or4 a b c d y
xi12 a b c d net99 nor4
xi13 net99 y inv
.ends or4


****************************************************************************
* Sub-Circuit Netlist:
*
* Block: addo_calc
* Last Time Saved: Jun  5 20:10:58 1994
****************************************************************************
.subckt addo_calc ad03 ad02 ad01 ad00 ad13 ad12 ad11 ad10 ad_success addo_oc3
+addo_oc2 addo_oc1 addo_oc0 addo_x_vci3 addo_x_vci2 addo_x_vci1 addo_x_vci0
+addo_y_vci3 addo_y_vci2 addo_y_vci1 addo_y_vci0 do03 do02 do01 do00 do13 do12
+do11 do10 do_success fault port_status3 port_status2 port_status1
+port_status0 prdiag priority rp3 rp2 rp1 rp0
xi331 net5572 tmp net5541 rp1 net5570 addo_oc1 nand5
xi330 net5572 tmp net5541 rp0 net5570 addo_oc0 nand5
xi341 net5572 net5546 rp3 net5570 addo_oc3 nand4
xi340 net5572 net5546 rp2 net5570 addo_oc2 nand4
```

```
xi303 tmp x_vci_pr3 net5572 net5570 addo_x_vci3 nand4
xi302 tmp x_vci_pr2 net5572 net5570 addo_x_vci2 nand4
xi301 tmp x_vci_pr1 net5572 net5570 addo_x_vci1 nand4
xi300 tmp x_vci_pr0 net5572 net5570 addo_x_vci0 nand4
xi421 net5574 rp3 port_status3 psdo3 nand3
xi420 net5574 rp2 port_status2 psdo2 nand3
xi483 net5570 net5572 y_vci_pr3 addo_y_vci3 nand3
xi482 net5570 net5572 y_vci_pr2 addo_y_vci2 nand3
xi481 net5570 net5572 y_vci_pr1 addo_y_vci1 nand3
xi480 net5570 net5572 y_vci_pr0 addo_y_vci0 nand3
xi37 ad1rp3 ad1rp2 ad1rp1 ad1rp0 net5557 nand4
xi44 psdo3 psdo2 psdo1 psdo0 do_success nand4
xi38 ad0rp3 ad0rp2 ad0rp1 ad0rp0 net5536 nand4
xi29 x_vci_pr3 x_vci_pr2 x_vci_pr1 x_vci_pr0 net5541 or4
xi28 y_vci_pr3 y_vci_pr2 y_vci_pr1 y_vci_pr0 net5546 or4
xi273 do1rp1 do1rp0 x_vci_pr3 nand2
xi272 do0rp1 do0rp0 x_vci_pr2 nand2
xi271 ad1rp1 ad1rp0 x_vci_pr1 nand2
xi270 ad0rp1 ad0rp0 x_vci_pr0 nand2
xi263 do1rp3 do1rp2 y_vci_pr3 nand2
xi262 do0rp3 do0rp2 y_vci_pr2 nand2
xi261 ad1rp3 ad1rp2 y_vci_pr1 nand2
xi260 ad0rp3 ad0rp2 y_vci_pr0 nand2
xi141 d1g1 do1final1 inv
xi140 d1g0 do1final0 inv
xi131 d0g1 do0final1 inv
xi130 d0g0 do0final0 inv
xi39 net5557 net5536 ad_success or2
xi9 rp1 rp0 net5573 or2
xi121 net5573 d1g3 do1final3 nor2
xi120 net5573 d1g2 do1final2 nor2
xi101 net5573 d0g3 do0final3 nor2
xi100 net5573 d0g2 do0final2 nor2
xi49 prdiag net5572 inv
xi50 fault net5570 inv
xi41 net5573 net5574 inv
xi35 net5546 tmp inv
xi7 priority net5578 inv
```

```
xi431 rp1 port_status1 psdo1 nand2
xi430 rp0 port_status0 psdo0 nand2
xi183 rp3 ad03 ad0rp3 nand2
xi182 rp2 ad02 ad0rp2 nand2
xi181 rp1 ad01 ad0rp1 nand2
xi180 rp0 ad00 ad0rp0 nand2
xi173 rp3 ad13 ad1rp3 nand2
xi172 rp2 ad12 ad1rp2 nand2
xi171 rp1 ad11 ad1rp1 nand2
xi170 rp0 ad10 ad1rp0 nand2
xi163 do1final3 rp3 do1rp3 nand2
xi162 do1final2 rp2 do1rp2 nand2
xi161 do1final1 rp1 do1rp1 nand2
xi160 do1final0 rp0 do1rp0 nand2
xi153 do0final3 rp3 do0rp3 nand2
xi152 do0final2 rp2 do0rp2 nand2
xi151 do0final1 rp1 do0rp1 nand2
xi150 do0final0 rp0 do0rp0 nand2
xi63 priority do13 d1g3 nand2
xi62 priority do12 d1g2 nand2
xi61 priority do11 d1g1 nand2
xi60 priority do10 d1g0 nand2
xi53 do03 net5578 d0g3 nand2
xi52 do02 net5578 d0g2 nand2
xi51 do01 net5578 d0g1 nand2
x9 do00 net5578 d0g0 nand2
.ends addo_calc

************************************************************************
* Sub-Circuit Netlist:
*
* Block: opt_router
* Last Time Saved: Jun  6 20:38:27 1994
************************************************************************
.subckt opt_router free29 free28 free27 free26 free25 free24 free23 free22
+free21 free20 free19 free18 free17 free16 free15 free14 free13 free12 free11
+free10 free9 free8 free7 free6 free5 free4 free3 free2 free1 free0
+port_status3 port_status2 port_status1 port_status0 r_answer11 r_answer10
```

```
+r_answer9 r_answer8 r_answer7 r_answer6 r_answer5 r_answer4 r_answer3
+r_answer2 r_answer1 r_answer0 rp9 rp8 rp7 rp6 rp5 rp4 rp3 rp2 rp1 rp0 solved
xi28 r_answer10 r_answer9 r_answer8 net2519 nor3
xi29 r_answer7 r_answer6 r_answer5 net2509 nor3
xi27 rp6 r_answer11 buffer
xi26 net2545 pd_vc4 r_answer4 or2
xi30 net2519 net2509 solved nand2
x10 net25220 net25440 r_answer8 nand2
x11 net25221 net25441 r_answer7 nand2
x12 net25222 net25442 r_answer6 nand2
xi20 net25223 net25443 r_answer5 nand2
xi243 net25550 net25560 pd_vc3 r_answer3 nand3
xi242 net25551 net25561 pd_vc2 r_answer2 nand3
xi241 net25552 net25562 pd_vc1 r_answer1 nand3
xi240 net25553 net25563 pd_vc0 r_answer0 nand3
xi21 free29 free28 free27 free26 free25 r_answer10 net2536 net2532 pd_vc4
+pd_vc3 pd_vc2 pd_vc1 pd_vc0 free24 free23 free22 free21 free20 r_answer9 rp9
+rp5 rp4 prdiag_calc
x13 net2542 net2541 free19 free14 free9 free4 net25440 net25441 net25442
+net25443 net2545 net2532 port_status3 port_status2 port_status1 port_status0
+net2536 rp9 rp8 rp3 rp2 rp1 rp0 fault_calc
xi19 free15 free10 free5 free0 free16 free11 free6 free1 net2542 net25220
+net25221 net25222 net25223 net25560 net25561 net25562 net25563 net25550
+net25551 net25552 net25553 free17 free12 free7 free2 free18 free13 free8
+free3 net2541 net2545 port_status3 port_status2 port_status1 port_status0
+net2536 rp6 rp3 rp2 rp1 rp0 addo_calc
.ends opt_router


****************************************************************************
* Main Circuit Netlist:
*
* Block: test_router
* Last Time Saved: Dec 12 23:56:32 1994
****************************************************************************
xi211 r_answer11 net100 inv
xi210 r_answer10 net101 inv
x14 r_answer9 net102 inv
x15 r_answer8 net103 inv
x16 r_answer7 net104 inv
x17 r_answer6 n18 inv
```

```
x19 r_answer5 net106 inv
x20 r_answer4 net107 inv
x10 r_answer3 net108 inv
x11 r_answer2 net109 inv
x12 r_answer1 net1010 inv
xi20 r_answer0 net1011 inv
xi1 solved net12 inv
xi0 free29 free28 free27 free26 free25 free24 free23 free22 free21 free20
+free19 free18 free17 free16 free15 free14 free13 free12 free11 free10 free9
+free8 free7 free6 free5 free4 free3 free2 free1 free0 port_status3
+port_status2 port_status1 port_status0 r_answer11 r_answer10 r_answer9
+r_answer8 r_answer7 r_answer6 r_answer5 r_answer4 r_answer3 r_answer2
+r_answer1 r_answer0 rp9 rp8 rp7 rp6 rp5 rp4 rp3 rp2 rp1 rp0 solved opt_router
```

Bibliography

[AGSY94] James D. Allen, Patrick T. Gaughan, David E. Schimmel, and Sudhakar Yalamanchili. Ariadne – an adaptive router for fault-tolerant multicomputers. In *Proceedings of the 21st International Symposium on Computer Architecture*, pages 278–288, 1994.

[BC93] Rajendra V. Boppana and Suresh Chalasani. A comparison of adaptive wormhole routing algorithms. In *Proceedings of the 20th International Symposium on Computer Architecture*, pages 351–360, 1993.

[BC94] Rajendra V. Boppana and Suresh Chalasani. Fault-tolerant routing with non-adaptive wormhole algorithms in mesh networks. In *Proceedings of Supercomputing '94*, pages 693–702, November 1994.

[Cas94] Dan Cassiday. Slow [routers] with features or fast and simple? Panel Discussion during the 1st International Parallel Computer Routing and Communication Workshop, May 1994.

[Chi93] Andrew A. Chien. A cost and speed model for k-ary n-cube wormhole routers. In *Proceedings of Hot Interconnects '93*, August 1993.

[CK92] Andrew A. Chien and Jae H. Kim. Planar adaptive routing: Low-cost adaptive networks for multiprocessors. In *Proceedings of the 19th International Symposium on Computer Architecture*, pages 268–277, 1992.

[DA93] William J. Dally and Hiromichi Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–475, April 1993.

[Dal90] William J. Dally. Network and processor architecture for message-driven computers. In Suaya and Birtwhistle, editors, *VLSI and Parallel Computation*. Morgan Kaufmann, 1990.

[Dal92] William J. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, March 1992.

[Dal93] William J. Dally. A universal parallel computer architecture. *New Generation Computing*, 11:227–249, June 1993.

[DDH+94a] William J. Dally, Larry Dennison, David Harris, Kinhong Kan, and Thucydides Xanthopoulos. Architecture and implementation of the reliable router. In *Proceedings of Hot Interconnects 1994*, 1994.

[DDH+94b] William J. Dally, Larry R. Dennison, David Harris, Kinhong Kan, and Thucydides Xanthopoulos. The reliable router: A reliable and high-performance communication substrate for parallel computers. In *Proceedings of the 1st International Parallel Computer Routing and Communication Workshop*, pages 241–255. Springer-Verlag, 1994.

[Den91] Larry R. Dennison. Reliable interconnection networks for parallel computers. Technical Report AI-TR 1294, MIT Artificial Intelligence Laboratory, 545 Tech. Sq., Canbridge MA 02139, October 1991.

[DFK+92] William J. Dally, J.A. Stuart Fiske, John S. Keen, Richard A. Lethin, Michael D. Noakes, Peter R. Nuth, Roy E. Davison, and Gregory A. Fyler. The Message-Driven Processor: A multicomputer processing node with efficient mechanisms. *IEEE Micro*, 12(2):23–39, April 1992.

[DL94] José Duato and Pedro López. Performance evaluation of adaptive routing algorithms for k-ary n-cubes. In *Proceedings of the 1st International Parallel Computer Routing and Communication Workshop*, pages 45–59. Springer-Verlag, 1994.

[DS87] William J. Dally and Charles L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–53, May 1987.

[Dua91] José Duato. On the design of deadlock-free adaptive routing algorithms for multicomputers: Design methodologies. In *Proceedings of PARLE '91*, pages 390–405. Springer-Verlag, June 1991.

[Dua93] José Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, December 1993.

[Dua94a] José Duato. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. In *Proceedings of the International Conference on Parallel Processing*, August 1994.

[Dua94b] José Duato. Why commercial multicomputers do not use adaptive routing? *IEEE Computer Society Technical Committee on Computer Architecture Newsletter*, pages 20–22, Summer-Fall 1994.

[GN92] Cristopher J. Glass and Lionel M. Ni. The turn model for adaptive routing. In *Proceedings of the 19th International Symposium on Computer Architecture*, pages 278–287, 1992.

[GY93] Patrick T. Gaughan and Sudhaka Yalamanchili. Adaptive routing protocols for hypercube interconnection networks. *Computer*, 26(5), May 1993.

[KLC94] Jae H. Kim, Ziqiang Liu, and Andrew A. Chien. Compressionless routing: A framework for adaptive and fault-tolerant routing. In *Proceedings of the 21st International Symposium on Computer Architecture*, pages 289–300, 1994.

[LH91] Daniel H. Linder and Jim C. Harden. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Transactions on Computers*, 40(1), January 1991.

[MC80] Carver A. Mead and Lynn A. Conway. *Introduction to VLSI Systems*. Addison-Wesley, 1980.

[Nga89] John Y. Ngai. A framework for adaptive routing in multicomputer networks. Technical Report Caltech-CS-TR-89-09, Caltech, Pasadena, CA 91125, May 1989.

[NM93] Lionel M. Ni and Philip K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2), February 1993.

[Ous94] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[Per92] Michael J. Pertel. A critique of adaptive routing. Technical Report Caltech-CS-TR-92-06, Caltech, Pasadena, CA 91125, April 1992.

[SS93] Charles R. Seitz and Wen-King Su. A family of routing and communication chips based on the mosaic. In *Proceedings of the Symposium on Research on Integrated Systems*, pages 320–337. MIT Press, March 1993.