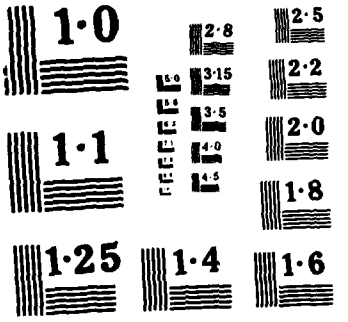
AD-A163 425 INTERPROCESSOR COMMUNICATION ISSUES IN FAT-TREE ARCHITECTURES(U) MASSACHUSETTS INST OF TECH CAMBRIDGE DEPT OF ELECTRICAL ENGIN.. A T ISHII OCT 85 1/1

UNCLASSIFIED N00014-80-C-0622 F/G 9/2 NL

END

FILMED

DTIC



NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

CAMBRIDGE MASSACHUSETTS 02139

VLSI Memo No. 85-268 October 1985

Interprocessor Communication Issues in Fat-Tree Architectures*

Alexander Toichi Ishii**

ABSTRACT

In recent years, it has become increasingly evident that conventional computer architectures will be unable to perform, in an acceptable time frame, many of the computational functions that we would desire of them. Consequently, much research has been devoted to the concept of constructing super-computers, which will be able to exploit the potential for parallel computation intrinsic to many large computational problems.

Recently, Leiserson has proposed a multiprocessor scheme based on Leighton's "tree of meshes," called a "fat-tree." Conceptually, such a multiprocessor would be comprised of a set of n processing elements each situated as a leaf in a complete binary tree. Internal nodes would be high speed switches which route messages being passed between processing elements, while edges between nodes would be bundles of constant bandwidth communication paths.

The purpose of this document will be to address and define some of the issues which effect interprocessor communication within a fat-tree multiprocessor. Specifically, we will cover the following topics:

1) Addressing in a fat-tree,
2) Generation of addresses in a fat-tree, and
3) Allocation of communication resources in a fat-tree,

*Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Bachelor of Science on May 10, 1985. This research was supported in part by the Defense Advanced Research Projects Agency under contract number N00014-80-C-0622.

**Department of Electrical Engineering and Computer Science, MIT, Room NE43-313, Cambridge, MA 02139, (617) 253-7843.

Interprocessor Communication Issues in Fat-Tree Architectures

by

Alexander Toichi Ishii

Submitted to the Department of
Electrical Engineering and Computer Science
in Partial Fulfillment of the
Requirements of the Degree of

Bachelor of Science

at the

Massachusetts Institute of Technology

May, 1985

Signature of Author_____
Department of Electrical Engineering and Computer Science
May 10, 1985

Certified By_____
Professor Charles E. Leiserson, Thesis Supervisor

Accepted By_____
Professor David Adler, Chairman, Department Committee

This research was supported in part by the Defense Advanced Research Projects Agency under Contract N00014-80-C-0622.

1 Introduction

In recent years, it has become increasingly evident that conventional computer architectures will be unable to perform, in an acceptable time frame, many of the computational functions that we would desire of them. Consequently, much research has been devoted to the concept of constructing super-computers, which will be able to exploit the potential for parallel computation intrinsic to many large computational problems.

Recently, Leiserson has proposed a multiprocessor scheme based on Leighton's "tree of meshes"[2], called a "fat-tree"[3]. Conceptually, such a multiprocessor would be comprised of a set of n processing elements each situated as a leaf in a complete binary tree. Internal nodes would be high speed switches which route messages being passed between processing elements, while edges between nodes would be bundles of constant bandwidth communication paths. For convenience, let us make the following definitions.

Definition 1 *Define a communication channel in a fat-tree to be an internodal bundle of fixed bandwidth communication paths.*

Definition 2 *Define for each communication channel c a parameter called the capacity of c, which specifies the number of communication paths within that channel, and denote this parameter as* $\mathrm{cap}(c)$.

Definition 3 *The height of a node in a fat-tree will be defined as the minimum number of edges that a path from a processor in the fat-tree to that node must traverse.*

Definition 4 *A level in a fat-tree is the set of all nodes with a given height*

Definition 5 *The height of a fat-tree will be the height of the root of the communication network. (Note: If n denotes the number of processors in a fat-tree, and h denotes the height of that fat-tree,* $n = 2^h$*)*

The purpose of this document will be to address and define some of the issues which effect interprocessor communication within a fat-tree multiprocessor. Specifically, we will cover the following topics

1. Addressing in a fat-tree.

2. Generation of addresses in a fat-tree.

3. Allocation of communication resources in a fat-tree.

2 Addressing in a fat-tree

One of the nicest features of any tree-based multiprocessor is that a message passing through the multiprocessor can specify the path it wishes to take with a string of bits, where each bit somehow encodes which communication bundle a message entering a node wishes to exit on. No ambiguity is possible, since nodes are connected by unique edges, and entering messages have only two bundles on which to possibly exit.

In general, path specification bits can be decoded in one of two ways. For convenience, I shall refer to the three communication bundles of a node as "ports". In addition, I shall refer to the port of each node which heads toward the root of the network as the "trunk" of that node, and similarly to the other two ports as the "branches". With this terminology, path specification bits can be decode as follows.

1. Imagine that one were a message entering an internal node through a particular port, much as one would enter the intersection of three abutting streets. One would then have the option of either exiting through the port on ones left or ones right, just as one would in the street analogy. We will specify a desire to go to the right by encoding the path specification bit for that node as a logical "zero". Similarly, a desire to go to the left shall be encoded as a logical "one". The particular assignments to logical "one" and "zero" are arbitrary, as long as consistency is maintained throughout the multiprocessor. One should note that the concepts of "left" and "right" used here have no absolute orientation to any global reference point. Rather, their absolute orientation changes as a message traverses a path through the network, much as the direction specified by ones own "right" changes as one moves about through a city. Exhaustive examples of all the different cases are given in Figure 1.

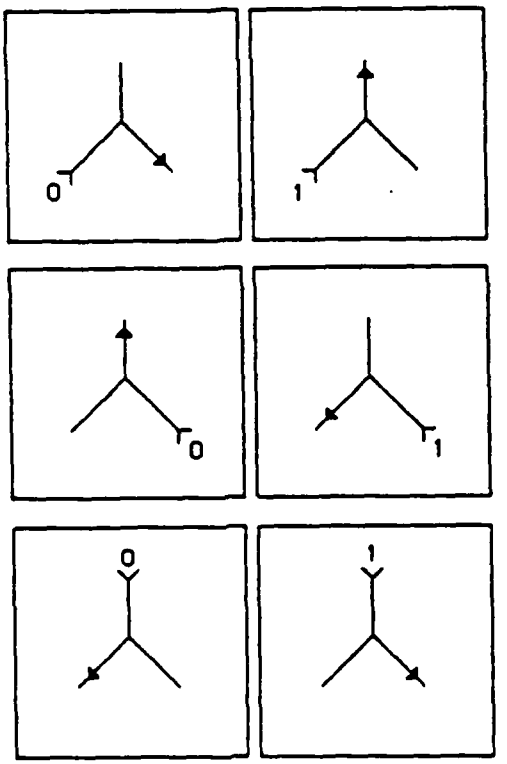


Figure 1: Routing Bit Encoding 1

2. Modify the above scheme so that when entering a node from a branch, a logical "one" encodes a desire to exit through the trunk, while a logical "zero" encodes a desire to exit through the other branch. When entering a node from the trunk, use the above scheme as stated. Exhaustive examples of all the different cases are given in Figure 2.

The differences between the two schemes may seem artificial, since they exhibit essentially identical hardware requirements. The first scheme however, seems conceptually cleaner, given that no sense of absolute directionality is required. In addition, we shall later show that under it, one must simply reverse and inverse the routing bits to retrace a path through the network. Thus, the first scheme displays some useful characteristics which the second does not, and consequently will be the assumed scheme throughout the remainder of this document.

With the encoding of individual path specification, or "routing", bits established, it is possible to describe how messages could actually be routed though a fat-tree. To begin, we note the following factors

1. Internal nodes cannot examine all the routing bits relevant to them simultaneously, since such an ability would imply a connection from each processor to each internal node, and more communication capability than even full processor interconnection does.

2. Since more messages may want to be routed through a specific port of a node than is possible, it may be necessary to not pass some messages and effectively "lose" them.

3. A node would only want to route messages which can be passed by other nodes to one of its input bundles. Otherwise one might allow an "already lost" message to prevent valid messages from using available communication channels.

4. To route messages correctly, each node needs only to see the routing bit meant for it.

It is possible to ship the routing bits to the relevant nodes bit serially. The basic strategy is as follows. Routing bits are pumped bit serially into the leaves of the network. When the bit stream reaches a node, that node

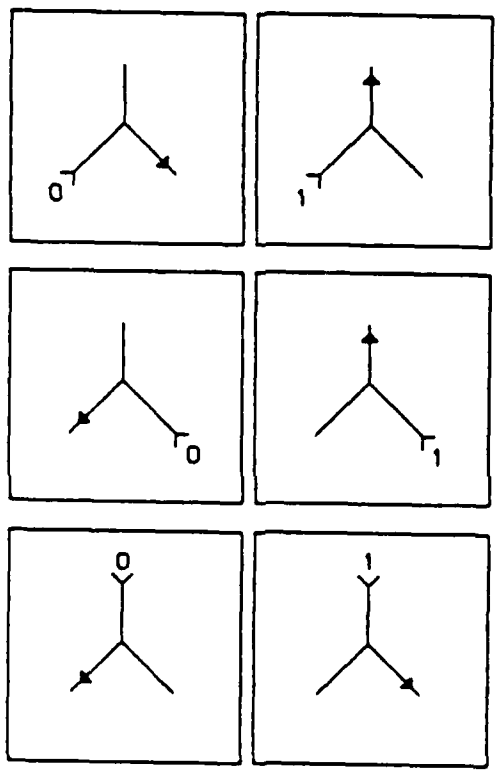


Figure 2: Routing Bit Encoding 2

will remove the first bit of the stream for the front of it and route the remainder of the stream as that bit specifies. Since the "head" of the stream will be removed, the scheme will route correctly as long as the next bit in the stream is the routing bit for the next node in the path. The message itself will then be sent bit serially, as a suffix of the routing bits. Thus one can conceptually view the internal nodes as switches, until they process the routing bit meant for them, after which they are merely cells in a distributed shift register.

In Figures 3 through 5 we give two examples of bit streams and how they are routed though a fat-tree network. Note that each shift of the bit stream is shown, and consequently the frames of the figures alternate between moving the bit stream forward to set stream paths, and stripping off the used routing bits. Stream directions are indicated as soon as a given node is able to examine its routing bit and are fixed there after. Message bits are denoted by M and are shown as they enter the network.

2.1 Synchronization and Buffering

While the serial routing scheme so far presented is essentially correct, it is also overly simplified. The situation is complicated by two factors. First, nodes will need to be signaled that the input on a given communication path is in fact a valid stream of routing bits, rather than random noise. Second, if a node wishes to give routing priority to certain kinds of messages (say ones from the trunk), it needs to have some mechanism for knowing that all the messages which wish to be routed through it have indeed made it to one of its input ports.

Signaling to a node that a given communication path has valid routing bits on it can be accomplished by simply placing a "message bit" at the front of the steam. The only modification is that one must now route on and remove the second bit of the stream, rather than the first.

Unfortunately, coordinating the arrival of different messages is more difficult. Let us assume the presence of a global shift clock, and that all bit streams start shifting in at the leaves simultaneously. Given these constraints, all the messages bound for branch inputs of nodes at a particular height in the tree will arrive simultaneously. Unfortunately, these assump-

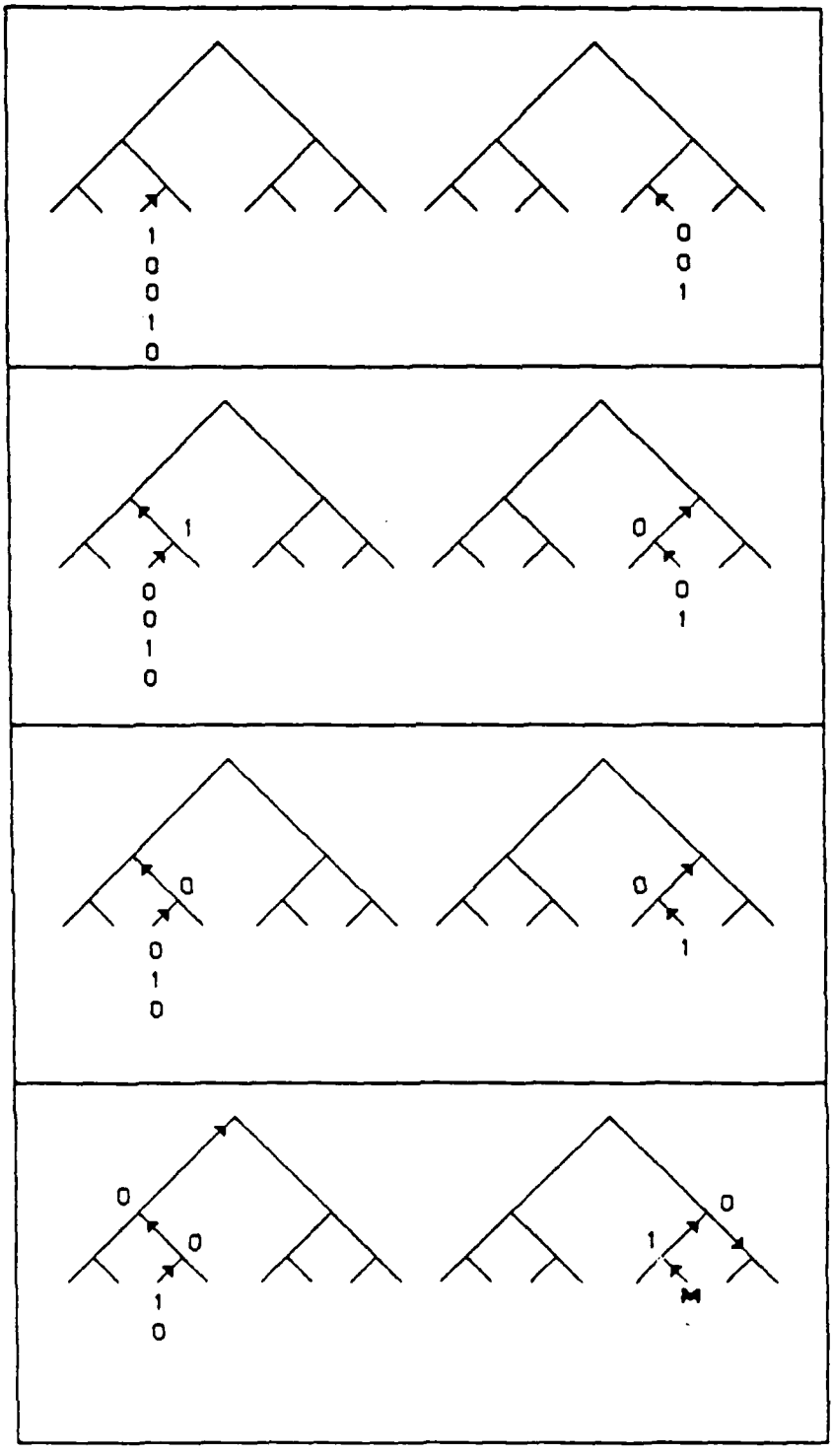


Figure 3: Bit Stream Routing 1

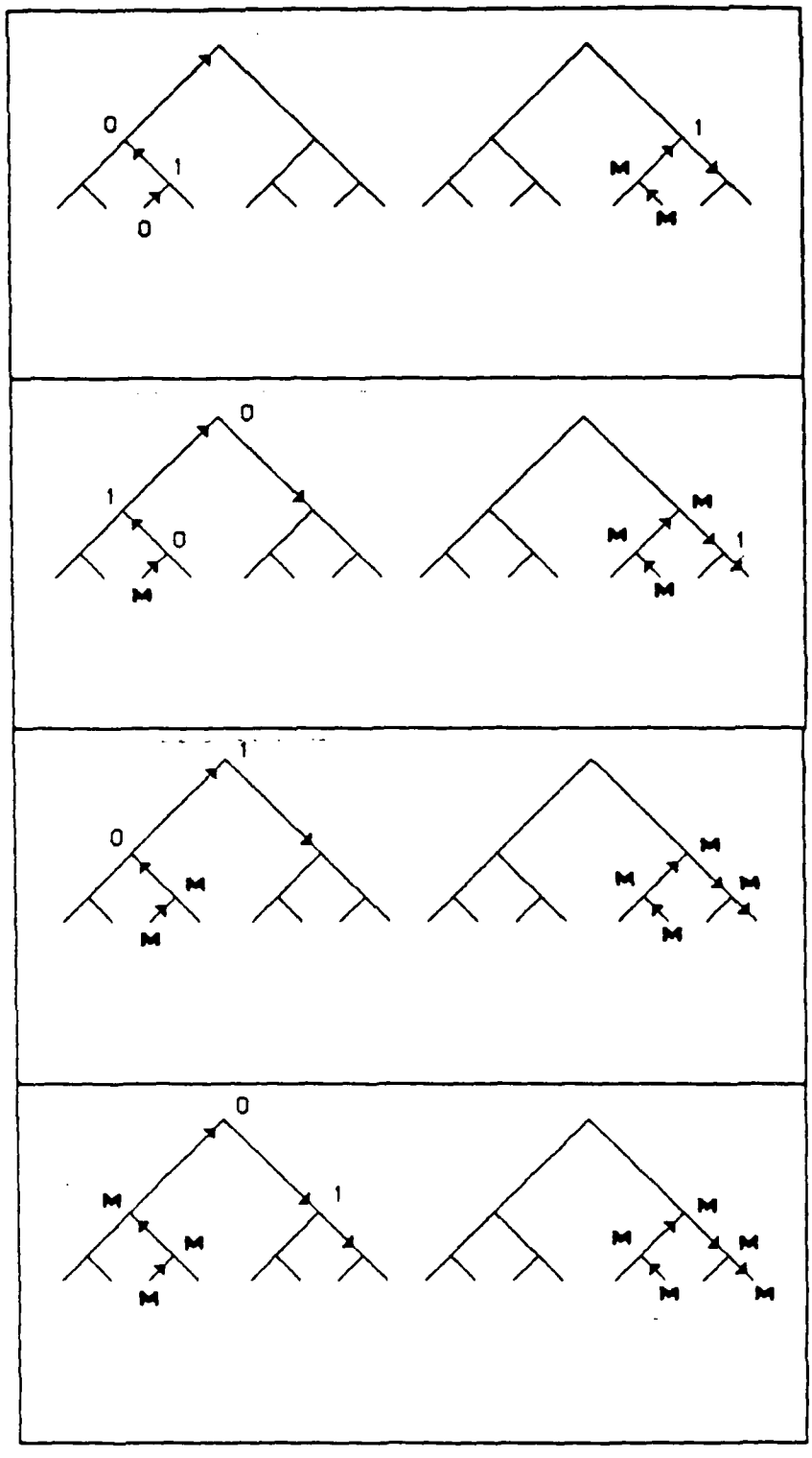


Figure 4: Bit Stream Routing 2

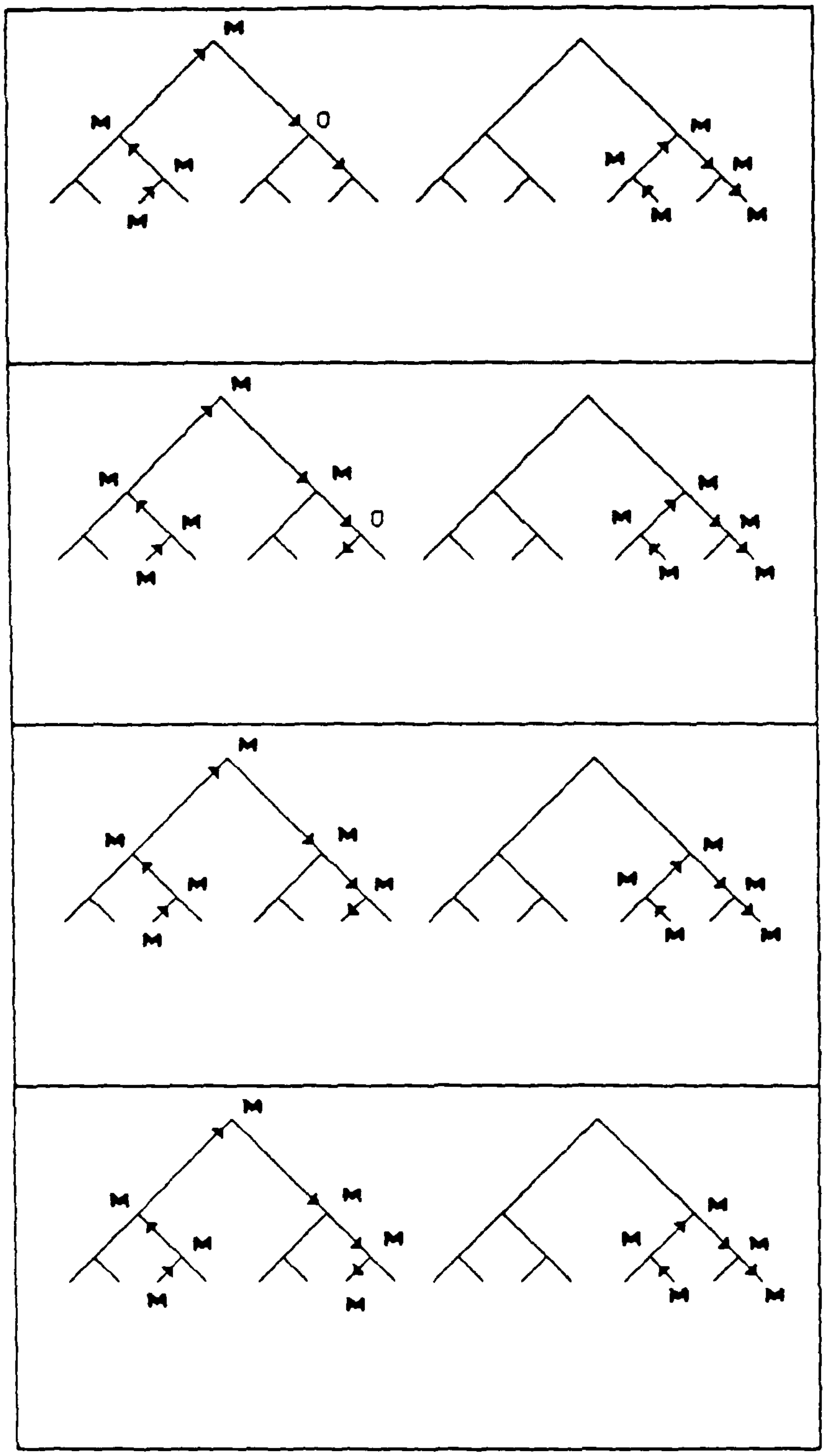


Figure 5: Bit Stream Routing 3

tions are not sufficient to solve our problem. If the reader refers back to Figure 4 to 5, they will note that the assumptions would cause the two examples given to be synchronized throughout the routing sequences just as they are grouped in the figure. Now, if the the two examples are routed on the same fat-tree rather than separate ones, the message going the shorter distance reaches the last two nodes in its path long before the other one does. Clearly, this presents a problem if the second to the last node in the path wished to give priority to messages coming down the tree. In addition, this node has no way to stop the shifting of bits coming from the message's source, since such an ability would imply a connection between each node and each processor.

Consequently, we are still in need of a method to handle the later arrival of messages entering a node through its trunk. Possible solutions would be as follows.

1. Flush low priority messages from channels as high priority ones come in.

2. Store the bits coming in the branches of a node, until the messages arrive at the trunk port.

3. Prefix routing bit streams with buffering bits, so that all messages will arrive at the nodes simultaneously.

4. Modify the nodes so that bits shifted in from a branch port, which are bound for the other branch port, are ignored until the messages coming in the trunk arrive. With this modification, it becomes possible to add "buffering" bits into the middle of the routing bit stream, and simulate the "no-shift" clock cycles that we desire.

We shall briefly examine the viability of each of these solutions in the subsections which follow.

2.1.1 Flushing Messages

Depending on the implementation of the node switches, this solution may or may not be acceptable. The constraining factor, is whether the bit

stream of the lower priority message has permanently set routing paths in subsequent nodes. If it has, then flushing messages becomes impossible. If it has not however, flushing mechanisms can also be used to allow processors to send messages into the communication network at will, rather than only at specific global times.

2.1.2 Storing Routing Bits

Unfortunately, this solution implies that a node would have to support $O(\log n)$ memory elements for each input into a branch port. If this hardware cost is acceptable, however, this scheme is perfectly viable, and conceptually very similar to mid-stream buffering.

2.1.3 Prefix Buffering

Unfortunately, buffering at the start of the routing bit stream does not provide all of the synchronization properties that we desire, since it forces nodes to commit communication resources before they are aware of all the messages which will pass through them. Consider the case where a node at a lower level in a fat-tree had messages passing through it which needed to go through the root, and other messages which only needed to go one level higher up the tree. Clearly, buffering bits at the front of the bit stream will prevent the messages going only one level higher from reaching the node until long after the ones going to the root. Consequently, without the ability to flush low priority messages from communication paths, prefix buffering does not even represent a valid solution.[1]

2.1.4 Mid-Stream Buffering

Currently, we believe that mid-stream buffering represents the most practical solution to the synchronization problem. In its favor are the following.

1. Simple node switching hardware. No buffering or flushing mechanisms need to be provided, and the concentration hardware it does require is also needed in all the other schemes mentioned.

[1]Tom Knight first suggested this strategy, in the context of a fat-tree with relaxed synchronization requirements

2. Buffered bit stream can be generated with relatively simple circuitry.

3. Conceptual simplicity. Using mid-stream buffering, it is possible to view the operation of the communication network on a level by level basis, since buffering bits conceptually make their routing streams inactive until messages have a chance to propagate down from the root of the communication network. In fact, it is even possible to view the network as packet switched, where packet passing is done in two separate phases. On the first phase packets propagate from the leaves toward the root in a wave, and are held at the nodes where they wish start heading toward the leaves. On the second phase, the packet passing wave begins at the root, and picks up packets as it propagates toward the leaves. Some nice aspects of this view, are

   (a) Makes it clear that all messages arrive at their destinations simultaneously.

   (b) Makes the parallel process of message passing appear to be sequential on a level by level basis.

   (c) Lends itself well to sequential simulation.

In Figures 6 through 8 we redo to our earlier example with message and mid-stream buffering bits added. Note that proper timing has been restored.

In Figues 6 through 8 we have given the last buffering bit the same value as the bit just before the first buffering bit, rather then the "undefined" value given the others. In the example, this is just to remind the reader about that bits value. However, in an actual implementation, one might actually enforce this convention, so that nodes do not need to actually store the value of the last valid routing bit. Depending on node implementation, this information may or may not be necessary for proper operation.

3 Generating Routing Bit Streams

In this section, we will examine the problem of actually generating routing bit streams with mid-stream buffering. We shall begin by determining

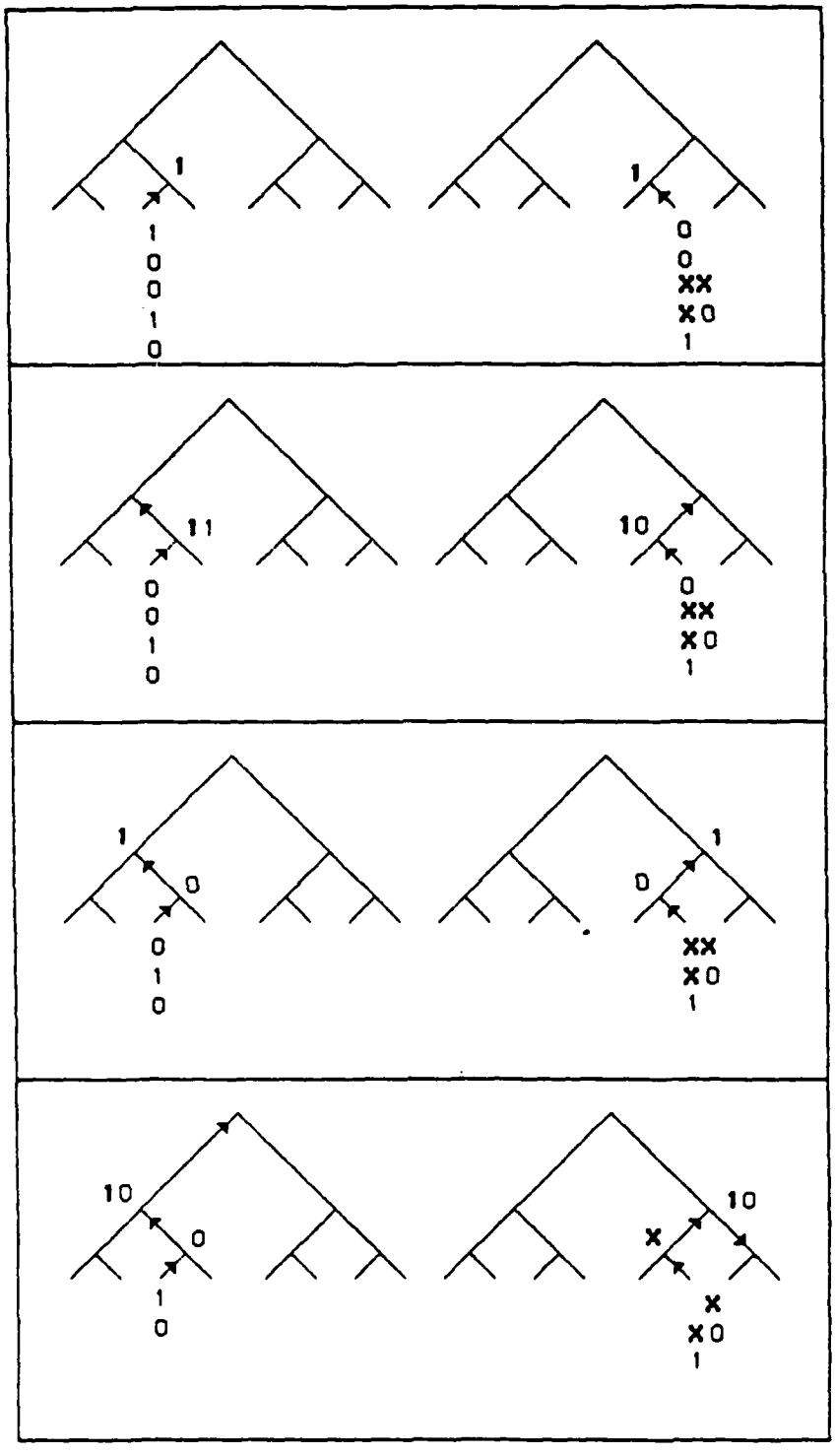


Figure 6: Improved Routing Bit Encoding 1

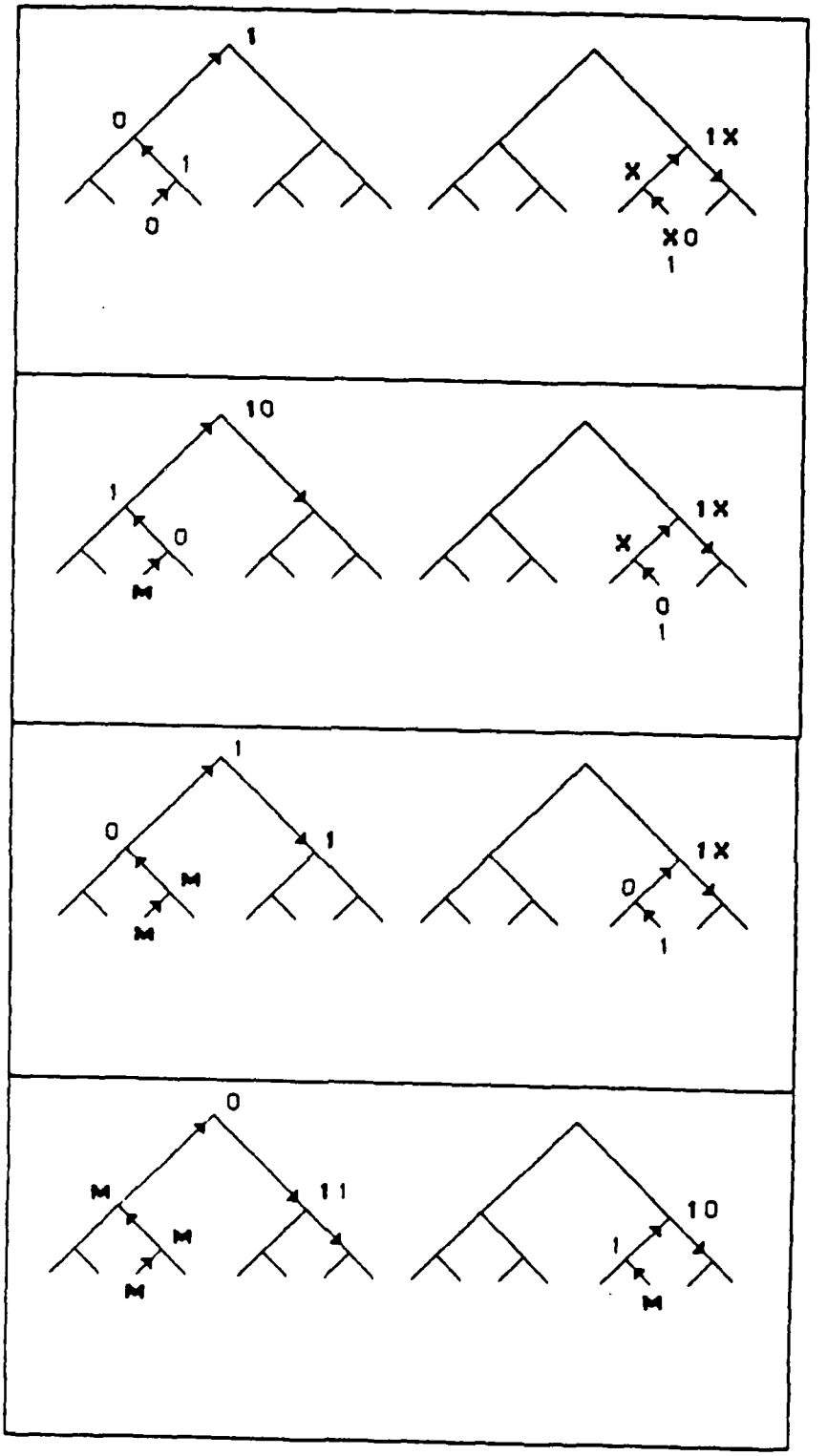


Figure 7: Improved Routing Bit Encoding 2

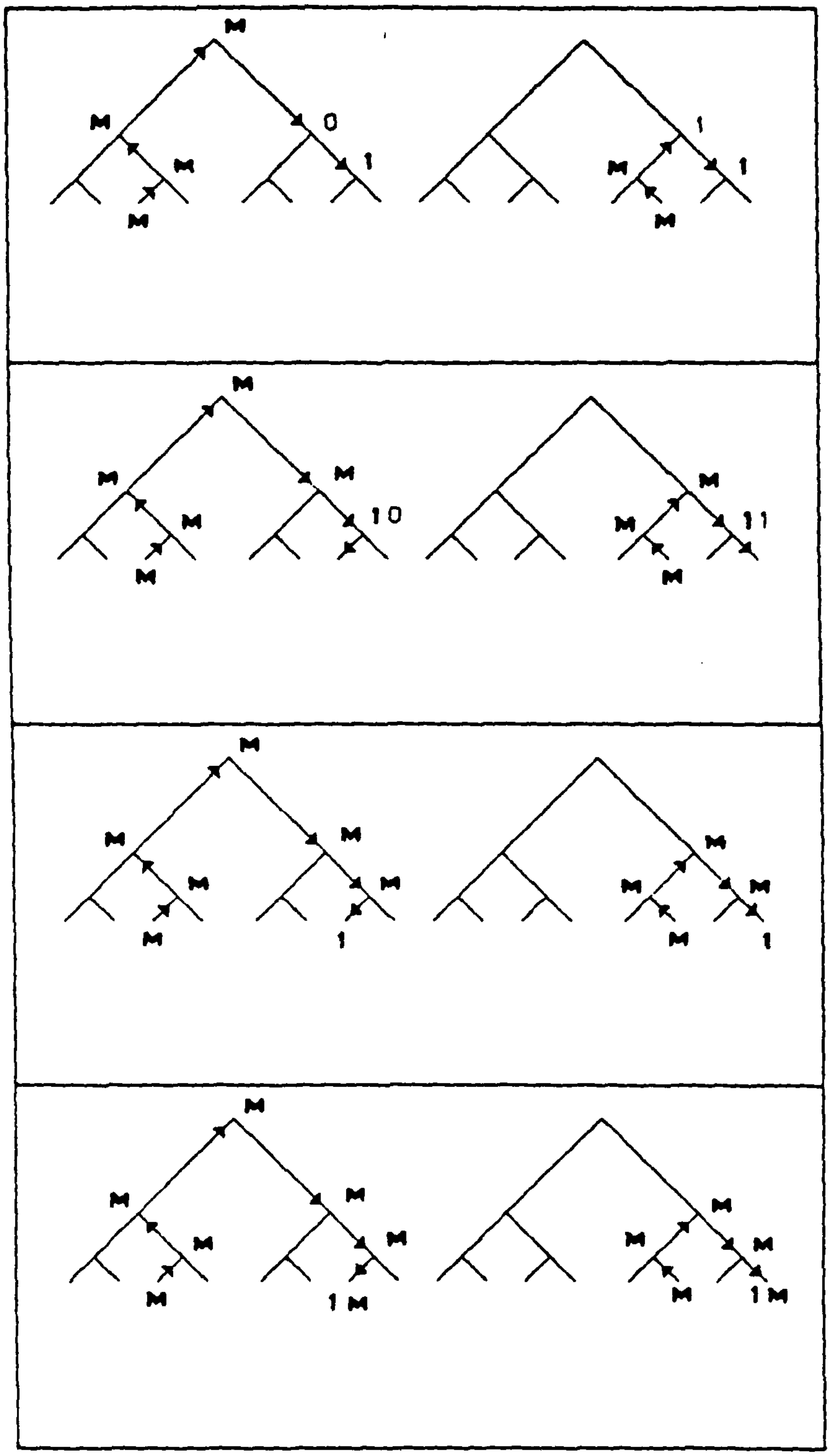


Figure 8: Improved Routing Bit Encoding 3

the number of buffering bits which must be added, and how one should assign addresses to processors. Subsequently, we shall examine how to actually generate the streams and a simple circuit capable of performing the necessary routines.

3.1 Buffering Bits

To determine the number of buffering bits which must be added to the routing bit stream, we note the following.

1. Messages waiting at a given level in the fat-tree may potentially have to wait until other messages have a chance to be routed from the current level, all the way to the root of the network, and back down.

2. For each node that messages are routed through, two shifts of the bit stream are required: one for the removal of the used route bit from the steam, and one for advancing the bit stream to the next node in the path.

3. A path through the root node of a fat-tree communication network of height H, from a node with height h to another node with height h, must pass through $(2(H-h)-1)$ nodes.

4. Messages which wait at a node, must conceptually pass through that node twice. Once when they initially enter the node, and once when other messages arrive at the trunk.

Consequently, we must insert $2(2(H-h))$ buffering bits into the middle of a "waiting" bit stream to keep the message routing scheme properly synchronized.

3.2 Assignment of Processor Addresses

It is possible to use a very simple circuit to compute the routing bit stream for a path to a specified processor, if processors are assigned addresses in the following manner. Begin at the root of the network and do an in-order traversal of the network, visiting the the branch which is the one on

the "right" from the trunk port first. Assign the first processor you visit the address zero, incrementing from there the address assigned to each processor you encounter. With processor addresses assigned in this way, we note the following properties.

Theorem 1 *If we begin from the root of the network, the address of a processor can serve as a routing bit stream to it, if we use the bits in order of descending significance.*

Proof:

We will prove this fact by inducting on the height of a fat-tree, which we will denote as h.

For $h = 1$, there are two processors, label them x and y, if x is connected to the port which is on the "right" of the trunk input, it will be visited first, and assigned an address of zero. Thus, a most significant routing bit of "0" will by our specification of node operation route to the "right" port and thus define a path to x. Alternately, a most significant bit of "1" will route to the "left" and must by elimination define a path to y.

For $h > 1$, assume that theorem is valid or all fat-trees of height $h - 1$. During address assignment, we will by the definition of in-order-traversal have visited all the processors in the subtree to the "right" of the trunk input before we visit the processors in the left subtree. Consequently, the address of every processor in the "right" subtree must be less than the address of every processor in the "left" subtree. Given this, let us examine the most significant bit of a routing bit stream of length h. If the bit is a "0" it will define a path to the right subtree. We may now remove the routing bit with no danger of mis-routing, since by our ordering observation, it is impossible for a processor in the right subtree to have a most significant address bit of "1". After removing this bit the remainder of the bit stream will by our assumption route to the correct processor. Thus, by induction, the theorem follows. □

Theorem 2 *If we invert the bits of a routing bit stream, which specify a path to a processor, and use them in reverse order, the resulting routing bit stream will define a path from the destination processor, to the first node or processor in the path.*

Proof: Let us denote the original routing bit stream as S, and the path that it specifies as P. In addition, let us associate with each node in P, the bit from S which the node used to route the path. Now consider the routing bit stream specified in the theorem. The first node in the path is defined by the physical connection to the processor, and consequently is the same node as the final one in P. In addition, the routing bit presented to the node will be the inverted least significant bit of the processor address, and consequently, must be the inverse of the bit used by the node to specify P. Thus, since port x is on the "left" of port y iff port y is on the "right" of port x, the first bit of the specified routing bit stream will specify a path which passes through the second to the last node in P. The second to the last node then routes on the inverted second to least significant bit, and the process continues until each node in P is visited in reverse order. □

Theorem 3 *If two processors in a fat-tree of height H are in a particular network subtree of height h, their $H - h$ most significant bits are identical.*

Proof: Consider the top $H - h$ levels of the fat-tree as a fat-tree in their own right. Since both processors are in the same of leaf of this fat-tree, by the first theorem, their $H - h$ high order address bits must be identical. □

3.3 Generating Routing Bits

Utilizing Theorems 1 through 3, the way to compute routing bit streams can be described as follows.

1. Compare the address of the source processor to the address of the destination processor on a bit to bit basis, and find the most significant bit pair which does not match. Call these bits the "pivot" bits.
2. Replace all the bits more significant than the pivot bits with two buffering bits.
3. Drop the pivot bit of the destination address.
4. Invert all the bits in the source's address, accept the pivot bit.
5. Reverse the bits of the modified source's address.

6. Construct a final routing bit string by prefixing the modified destination bits with the modified source bits.

By using the bits of this final bit string in order of descending significance, we can specify a path through the fat-tree to the desired destination processor.

Intuitively, the reason this construction works is as follows. Imagine that one entered the network through the trunk of the root, and followed the paths specified by the addresses of both the source processor and the destination processor, using their bits in descending order of significance. Clearly these paths would match until one tried to use the pivot bits. At that point, the paths would diverge, with each of them going down a different branch of a common "pivot" node. Thus, for a routing bit stream to specify a path from the source to the destination, it must traverse the path from the source to the root, until it gets to the pivot node. At this point, rather than continue on to the root, it must "turn down" and follow the path which leads to the destination, while inserting the proper number of buffering bits. Reversing and inverting the bits of the source address provide the bits which route a message from the source to the pivot node, while not inverting the pivot causes one to "turn down" so that the remainder of the destination address bits will route down to the destination processor. Note that, by dropping the pivot bit of the destination address and replacing high order non-pivot bits with two buffering bits, we are guaranteed to provide the right amount of buffering to keep the routing of different messages synchronized.

3.4 Generation Circuitry

A simple circuit scheme for computing the routing bit streams is hierarchically presented in Figures 9 through 17. Global control signals and clocks have been suppressed where convenient, and device sizing has been ignored.

The circuit presented is suitable for implementation in standard NMOS technology [5], and functions in the following manner. The address of the source and destination processors are loaded into their respective registers. The values of the individual bits of matching significance are compared by

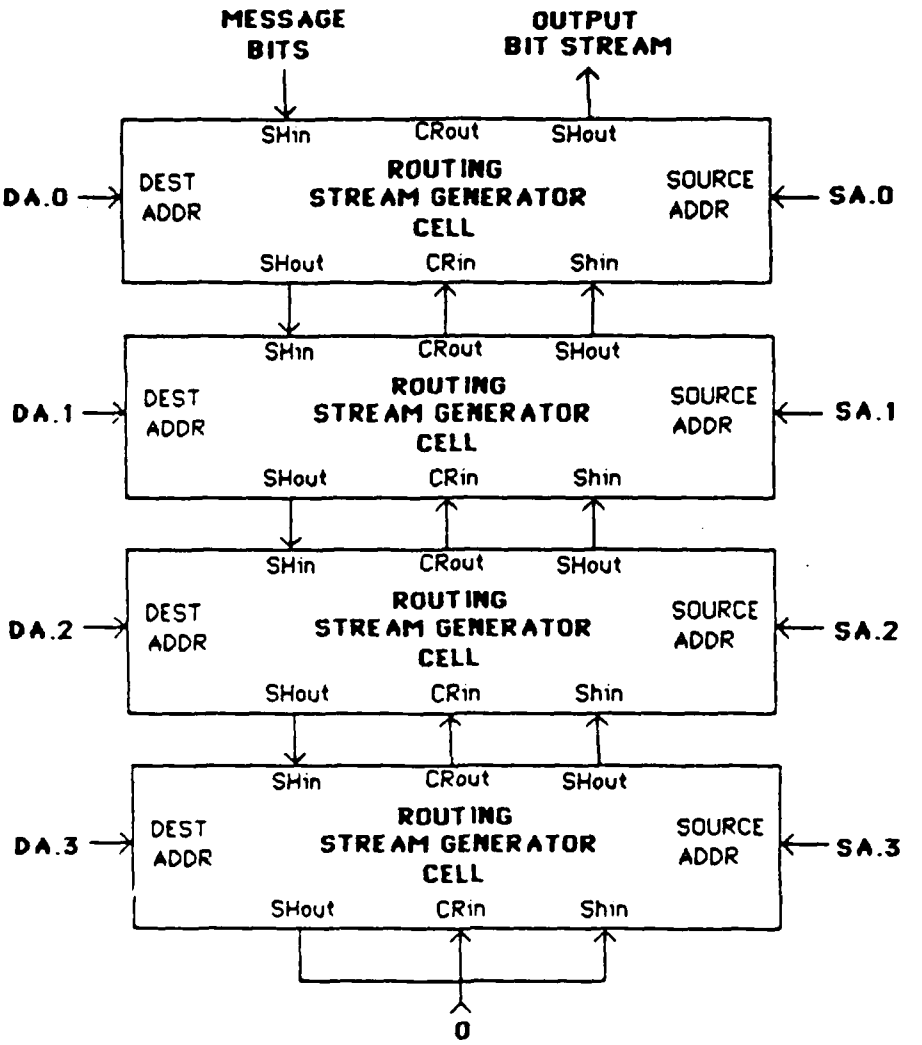


Figure 9: Routing Bit Stream Generation Register

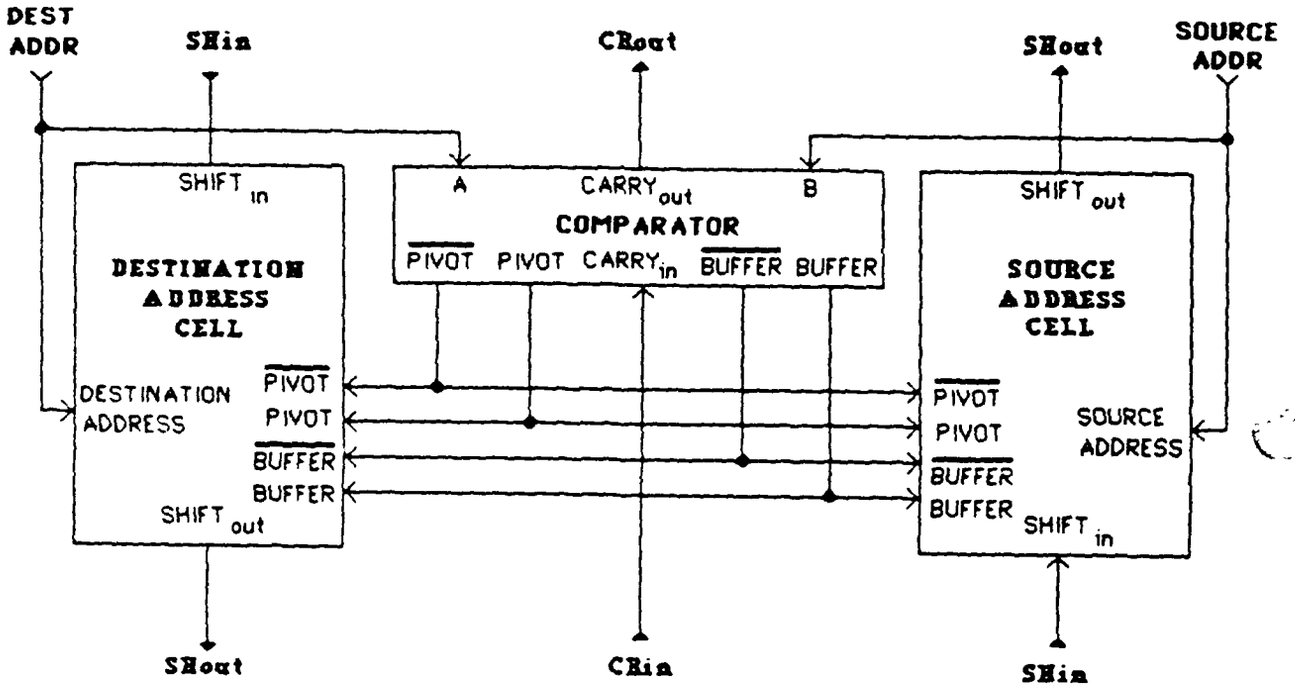


Figure 10: Routing Stream Generation Cell

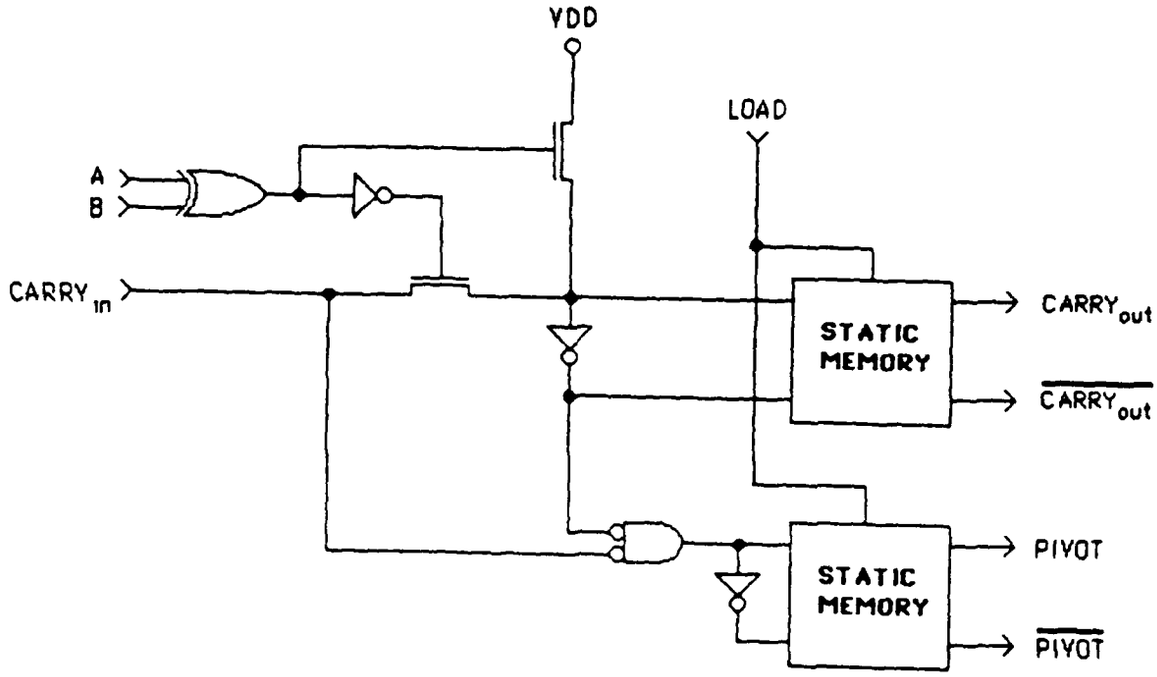


Figure 11: Comparator Cell

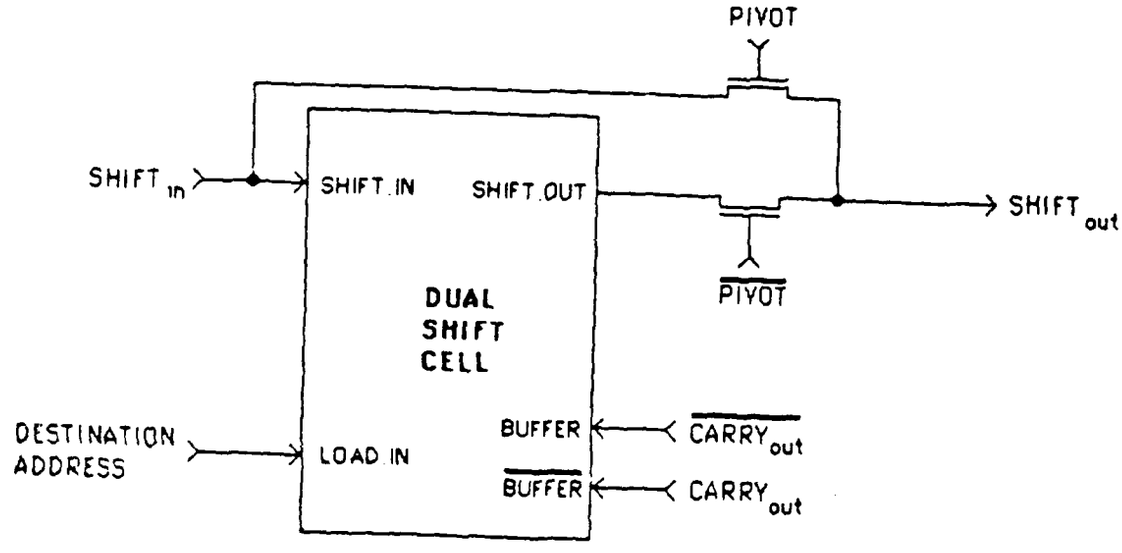


Figure 12: Destination Address Cell

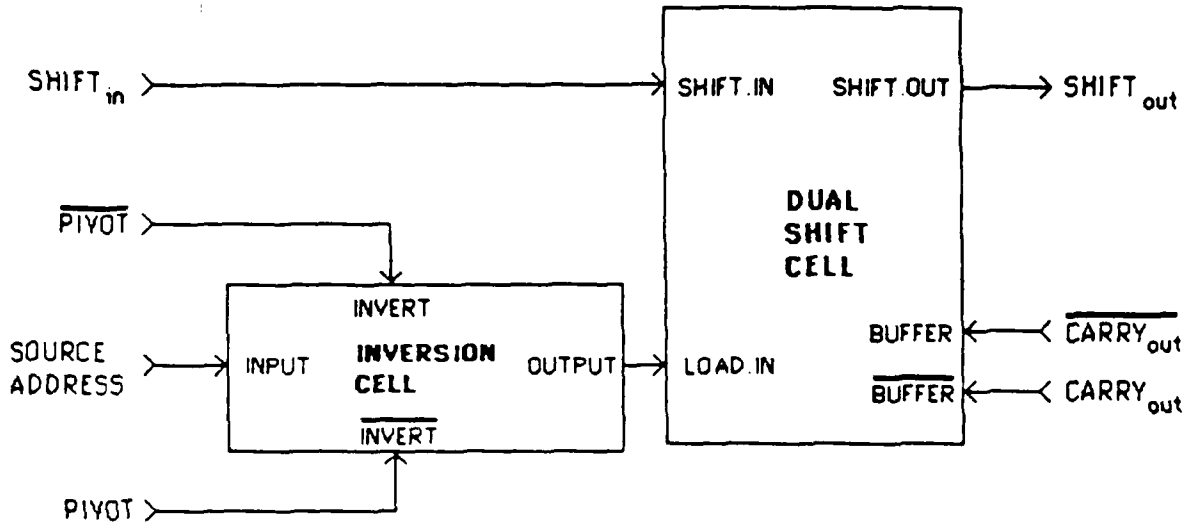


Figure 13: Source Address Cell

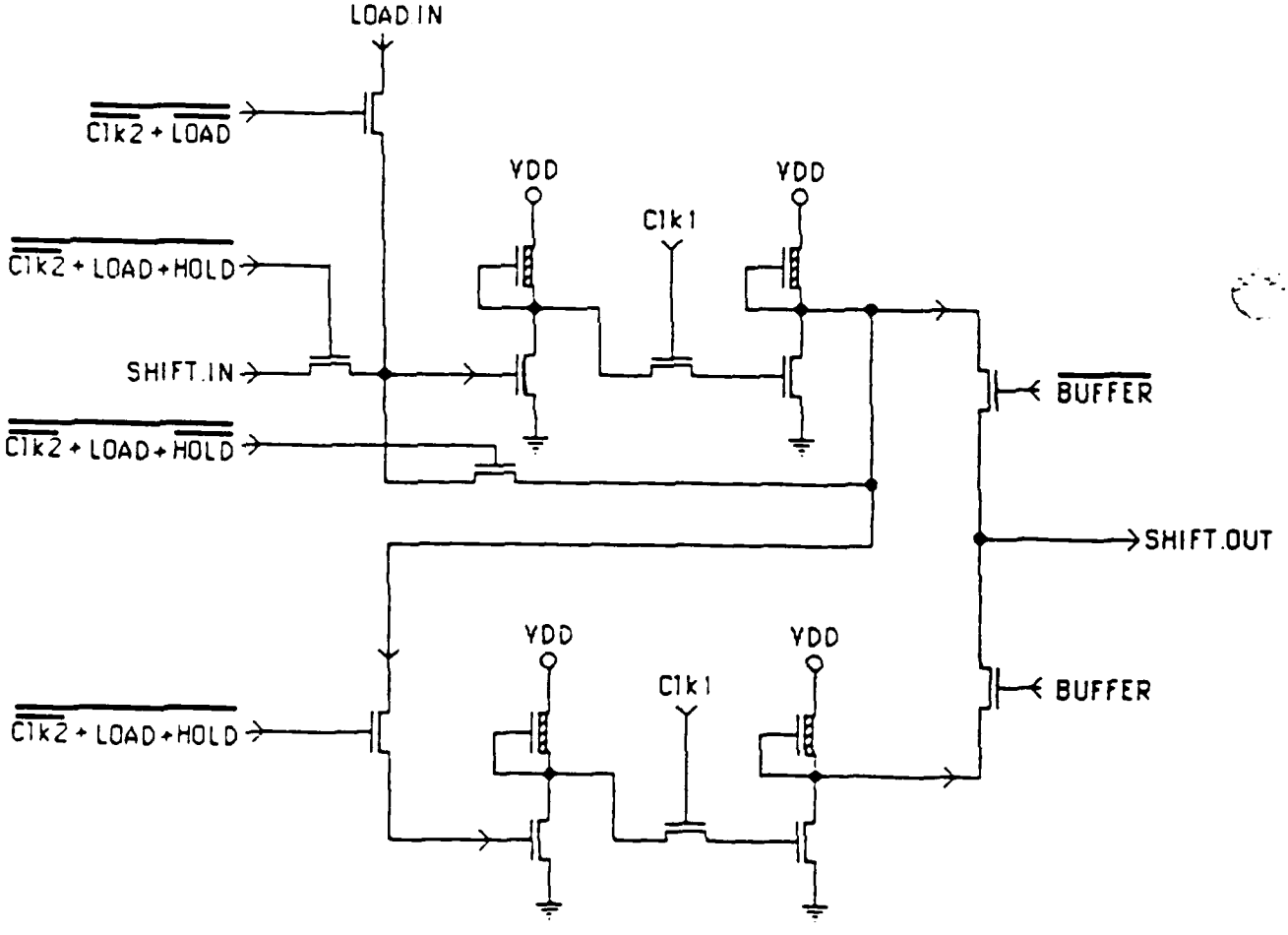


Figure 14: Dual Shift Cell

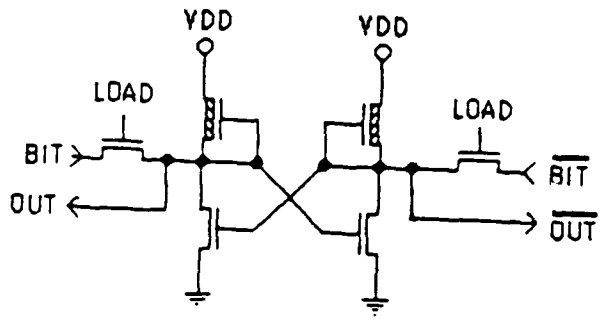


Figure 15: Static Memory Cell

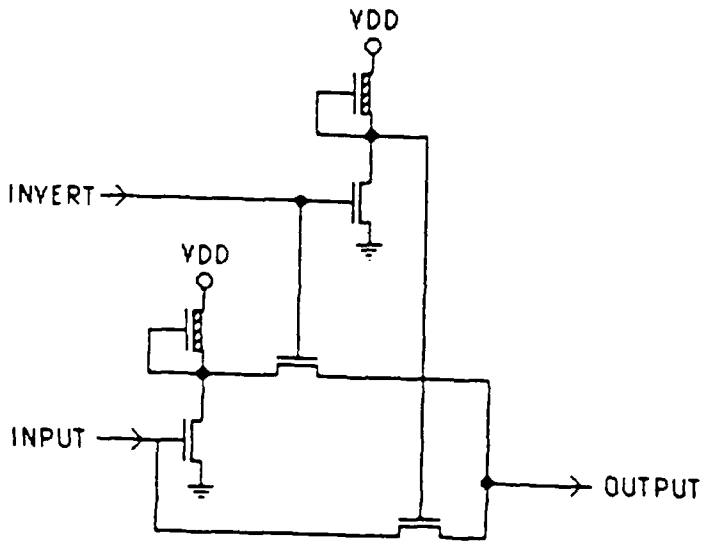


Figure 16: Inversion Cell

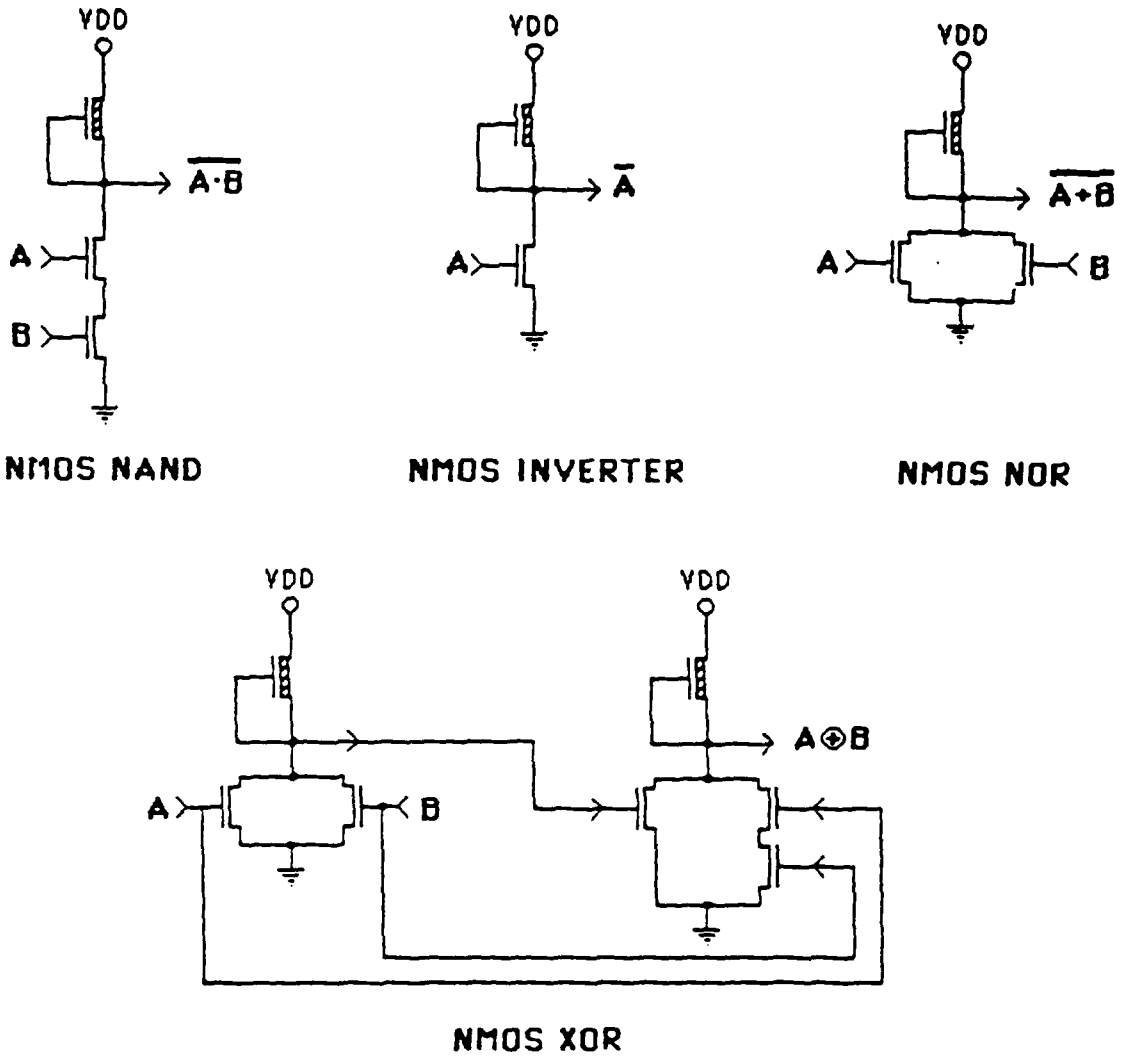


Figure 17: Logic Gate Cells

the cells in the comparator bank. If the $carry_{in}$ signal to a comparator cell is low, and the bits match, the cell will set its $carry_{out}$ signal low, otherwise it sets its $carry_{out}$ signal high. This has the following effects, bits that must be expanded into buffering bits correspond to comparator cells which have their $carry_{out}$ signals set low, while only the comparator cell for the pivot bits has its $carry_{in}$ low and its $carry_{out}$ high. All other bits correspond to comparator cells which have high $carry_{in}$ signals. Consequently, it is possible for the comparator cell to tell the register cells of the addresses which kind of bits they represent. The individual register cells can then modify their contents appropriately, by either inverting them, leaving them unmodified, or replacing them with two buffering bits. Once the contents of the registers have been properly modified, it is then possible to serially shift out their contents, and thus obtain the desired routing bit stream. The doubling of buffering bits is easily accomplished by internally configuring the register cells so that they behave as a pair of shift register cells in series. The reader is encouraged to note that, if address storage is required, this circuit produces the buffering bits without increasing the order of growth of hardware.

3.5 Network Interfaces

One of the major virtues of fat-trees is their potential graceful performance degradation when communication resources must be restricted for economic or technological reasons. This implies however, that different software applications will probably be run on fat-trees with different interprocessor communication capabilities, and possibly different sizes. If processors were to interface directly to the network however, the need for inserting buffering bits would clearly require them to know what size of fat-tree they were running on. If this were the case, transporting software to fat-trees of different sizes could pose a problem. In addition, it would be desirable for one to be able to run software designed for other types of multiprocessors on a fat-tree, and possibly use different types of special purpose processing elements as well. Consequently, it is desirable to allow processors to specify messages to be sent in some general way, and use special purpose interfaces to produce the machine specific sending format.

At present, we envision a fat-tree to be a machine whose functional structure is as shown in Figure 18. Processors utilize the communication network by specifying the address of another processor, and a message to be sent, to the network interface. The interface would in turn use circuitry like that presented in section 3.4 to generate the proper signals to send the message to the specified processor.

4 Communication Resource Allocation

Up until this point, we have only been concerned with the issues faced by an attempt to send a single message through the communication network of a fat-tree. In an actual multiprocessor though, one would expect many messages to be going through the network simultaneously. Consequently, contention for the communication resources of the network is not only possible, but likely. Given this, it is clear that some method must be devised to arbitrate communication resource contentions. It is this need for arbitration that we refer to as the "routing problem".

In this section, we will attempt to define the routing problem, spotlight some of the issues which solutions to it should consider, provide some general classifications for solutions, and finally offer some preliminary findings on one possible greedy type strategy.

4.1 The Routing Problem

Since the address mechanisms of a fat-tree have in fact taken care of all the actual routing, the "routing" problem has nothing to do with routing what so ever. Rather, the "routing" problem is an allocation problem, with a large number of interprocessor messages contending for possibly scarce communication resources. More formally, we can define the routing problem as follows.

Definition 6 *Given a set of source destination pairs M, partition M into subsets $S_0, S_1, \ldots, S_i$ such that all the source destination pairs in any S_k can be sent through the network simultaneously, without any communication resource contention.*

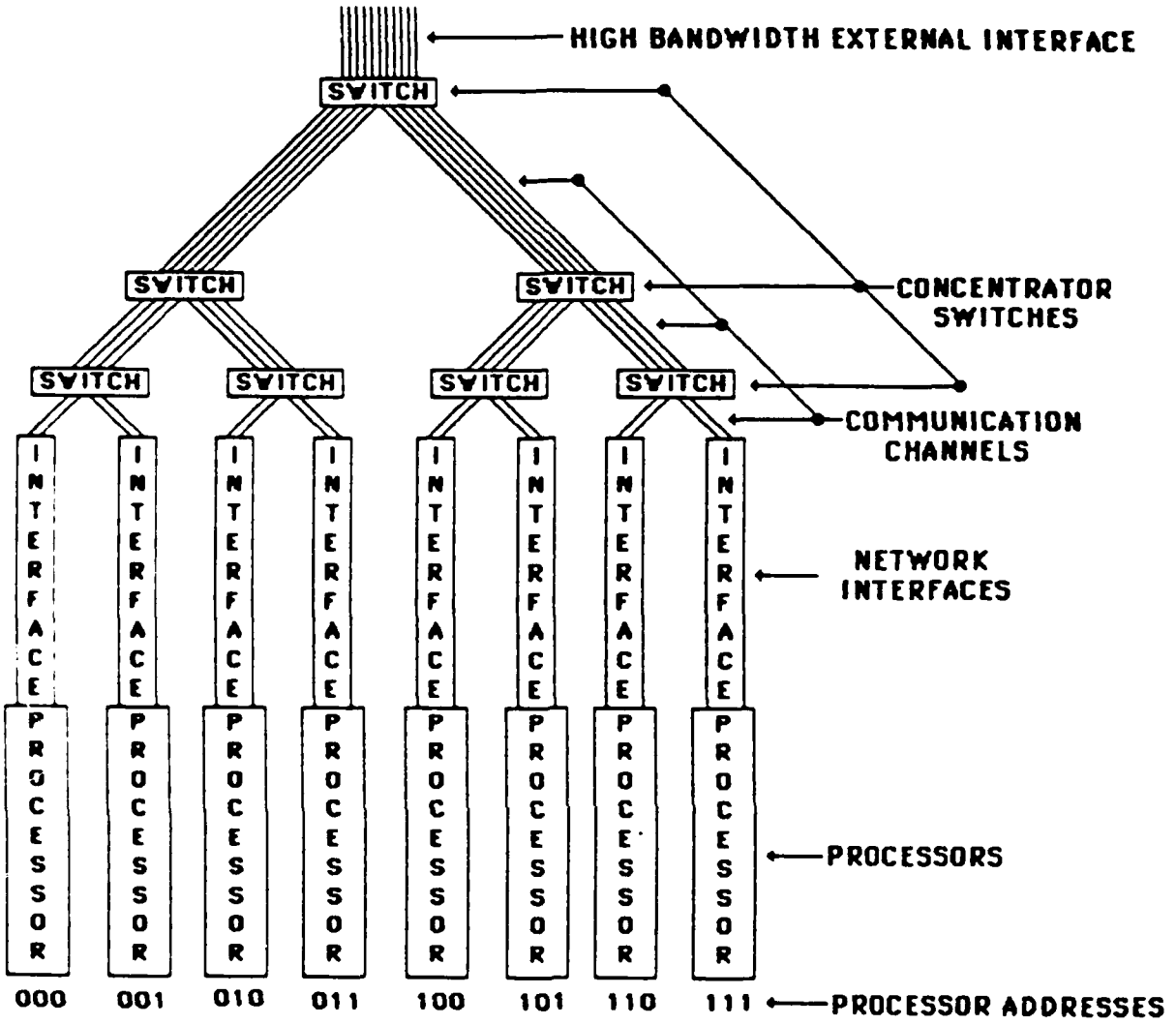


Figure 18: Basic Hardware Layout

4.2 Routing Problem Issues

In general, there are several issues which solutions to the routing problem should consider, if they are to be successful. The reader should note that "successful" is a very subjective term, and in this case is meant to be. For certain applications any or all of these criteria could be dispensed with without hindering the operation of a fat-tree. Consequently, the success of a routing problem solution will probably have to be judged by whether it strikes a balance of these criteria which is appropriate for a particular application. With this caveat in mind, we see the following to be of particular importance.

4.2.1 Guaranteed Delivery

Probably the most fundamental issue for any routing problem solution is the ability to guarantee the eventual delivery of any message in the message set being "routed". This ability would seem at first to be extraneous, however, if the switches in the nodes of the network are not able to perform perfect concentration, there is a real possibility that a particular message may be impossible to send. Consequently, it is imperative that imperfect switches be capable of establishing a path between any two processors, if guaranteed delivery is to be possible. In addition, while in the previous section we assumed that we must simply partition a fixed message set, in practice the message set may be changing, with messages being added and removed from it dynamically. If this is the case, we must be able to guarantee that a particular message cannot be perpetually denied resources.

4.2.2 Switch Complexity

Generally, one will want to minimized hardware complexity when ever possible. In the case of a fat-tree type architecture however, any additional complexity one introduces into nodes, processors, or network interfaces will be multiplied many times over, and reduced complexity could mean the difference between a tractable design and a prohibitively expensive one. In addition, increased complexity may also cause increased functioning time. Consequently, before introducing additional hardware complexity, one must

be sure that the imagined performance gains are not offset by any running time increases.

4.2.3 Global Information Restrictions

If we ignore the case of offline routing, any scheme which requires a large amount of global information is undesirable. For the availability of such information implies either communication resources from each processor to some global information source, or the use of processor time to gather the information in some distributed fashion. As was the case with hardware complexity, this does not mean that use of global information is unacceptable. Rather, it indicates that the added performance provided by any global information used, must be evaluated to insure that it is not offset by the overhead required to gather and distribute it.

4.2.4 Efficient Partitioning

In our specification of the routing problem, we required that the message set M be partitioned into subsets $S_0, S_1, \ldots, S_i$. This can be done trivially, if we place in each S_k a single message from M, thus obtaining $S_0, S_1 \ldots, S_{|m|-1}$. Such a partition is undesirable however, since almost all of the networks communication resources would be idle at any one time. Consequently, to maximize the use of the network, one should try to minimize the number of subsets.

One obvious lower bound on the number of subsets, into which M must be must be partitioned, can be defined as follows.

Definition 7 *Define the load of an internodal channel c, on a message set M, as the number of messages in M whose paths pass through that channel, and denote it as* $\text{load}(c, M)$.

Definition 8 *Define the load-factor of an internodal channel c on a message set M as the load of c on M divided by the capacity of c, and denote it as* $\lambda(c, M)$.

Definition 9 *Define the load-factor of a fat-tree FT to be the maximum load-factor of any channel in the fat-tree, and denote it as* $\lambda(FT, M)$.

Given these definitions, simple lower bound on the number of subsets required for a fat-tree FT will be $\lambda(FT, M)$.

Currently, we have as yet not been able to derive any strong lower bounds on the number of subsets which must be formed. For the present however, we shall use $\lambda(FT, M)$ as a standard metric for partitioning scheme efficiency.

4.3 Solution Classifications

In general routing problem solutions can be grouped into two categories. The first of these would be ones based on node, or switch, arbitration. In such schemes, processors would simply send all the messages they desired to, relying upon the switches to arbitrate resource contentions in some reasonable fashion. Alternately, one could utilize a scheme based on processor arbitration, where processors explicitly select some subset of their messages which they believe can be sent through the network without causing resource contention.

In practice, it would be difficult to classify any given solution to the routing problem as exclusively switch or processor arbitrated. In general however, switch arbitrated schemes will follow the following general format.

1. Begin with a set of "unsent" messages UM which is initially equal to M.
2. For each processor p_i, select a subset of the messages in UM, such that
   (a) Each message in the subset has the address of p_i as its source.
   (b) Number of messages in that subset does not exceed the number of messages a processor can send into the network at one time.
3. Let the union of the subsets for each source be an initial specification of a subset S_k.
4. Send the messages selected in step(2) into the communication network.

5. Delete from S_k any messages which are ever denied communication resources by a switch. Such messages we will term as "lost".

6. The final S_k is comprised of the messages which are granted resources all the way to their destinations.

7. Delete from UM the members of the final S_k.

8. Lost messages remain in UM and are thus made eligible for inclusion in S_{k+1}.

9. Repeat steps(1 - 8), until UM is empty.

Since all the members of a finished S_k made it to their destinations, sending these messages into the network simultaneously would not result in any resource contentions. Consequently, when UM is empty, we will have conceptually constructed $S_0, S_1, \ldots, S_i$.

Of course, by selecting which messages to initially send into the network in step(2), the above outlined scheme does utilize some processor arbitration. Consequently, most "processor arbitrated" schemes will follow the same basic format as switch arbitrated ones, except that step(2) will be much more elaborate. The extent to which a given scheme eliminates the deletions made in step(5), will determine the extent to which it implements processor arbitration.

4.3.1 Processor Arbitration Considerations

The major factor effecting processor arbitration, is the amount of global information that each processor has available to it. Global information can itself be of two types, either resource related, or message set related. Resource related global information would include things such as the internodal bandwidth of the network, and the efficiency of the switch arbitration used in the nodes. Message set related global information would include things such as the number of messages in the message set, as well as its explicit contents.

In general, resource related information is inexpensive to provide, since it is usually static for any given fat-tree. By comparison, message set related

information can be very expensive, since it changes dynamically with each different message set.

If an allocation scheme were to assume complete knowledge of global resource and message set information, it would essentially be equivalent to an "offline" allocation scheme which examines the entire message set and produces a perfect partition which eliminates any message pruning by the switches. In practice, there are situations where message sets are sufficiently static to make an offline scheme appropriate. As the message sets become more and more dynamic however, the overhead needed to collect the global data for an offline scheme may become prohibitive, and "online" schemes which utilize both processor and switch allocation would probably be more feasible.

4.3.2 Switch Arbitration Considerations

Under our current conceptual hardware model, switch arbitration schemes tend to be more constrained than ones based on processor arbitration. The main cause of this is that switches only have access to information which passes through them, rather than information which is explicitly sent to them. One should not conclude however, that switch arbitration schemes must operate "blindly," since many tricks can be used to get information to the nodes. For instance, we noted earlier that some messages will have to "wait" while other messages propagate through the network. During this waiting time, the channels which the messages are entering through will have buffering bits shifted into them. Rather than wasting these bits, it is possible to place real bits in them, and sneak information to the nodes during what would otherwise be wasted cycles.

Up until now we have been making a large assumption. Specifically, we have been assuming that switches are capable of mapping all incoming messages to available output communication paths, when ever such a mapping is possible. This ability, which we term "perfect concentration," can under certain hardware constraints be unattainable. Without perfect concentration, however, the complexity of offline routing increases enormously, since the partitioning of the message set must now take into account the imperfect functioning of the switches. In contrast, imperfect concentration

is relatively transparent to switch arbitrated schemes.

4.4 Greedy Routing

One possible solution to the routing problem, which we call the "greedy approach," is predominantly switch arbitrated, and operates under the following assumptions.

1. Given a message set M, assume that each processor knows only of the existence of messages for which it is the source.

2. Messages are delivered in "cycles," where processors send messages into the network at the start of each cycle. Individual messages are passed through the network during the cycle, and those which have been granted resources all the way to their destinations reach them at the end of the cycle.

3. The only information about the communication network available to the processors, is when each delivery cycle will start and whether a message initially sent into the network made it to its destination.

Given these assumptions, the greedy approach operates by using the following two arbitration algorithms.

Switch Arbitration

FOR each delivery cycle D_i DO

FOR each ordering of the ports (X, Y, Z) DO

Let $M_x = \{m : m \textit{ enters the node through } X \textit{ and wishes to exit through } Z\}$
Let $M_y = \{m : m \textit{ enters the node through } Y \textit{ and wishes to exit through } Z\}$
IF $\mid M_x \cap M_y \mid > \text{cap}(Z)$
THEN
Grant communication resources to a set of messages P, where
$P \subseteq (M_x \cap M_y)$
$\mid P \mid = \text{cap}(Z)$

Each message in $(M_x \cap M_y)$ has probability $(\text{cap}(Z) \div (\mid M_x \cap M_y \mid))$ of being an element of P

ELSE

Grant communication resources to all messages in $(M_x \cap M_y)$

ENDIF

ENDFOR

ENDFOR

Processor Arbitration Initialization

FOR each processor P_i DO

construct $M_i = \{m : m \text{ is an element of } M \text{ and } m = (P_i, P_j), \text{ where } j \neq i\}$

ENDFOR

Processor Arbitration Delivery

FOR each delivery cycle DC_i DO

FOR each processor P_i DO

IF $M_i = \emptyset$

THEN HALT

ENDIF

IF $\mid M_i \mid > \text{cap}(Z)$

THEN

Grant communication resources to a set of messages Q, where

$Q \subseteq M_i$

$\mid Q \mid =$ *number of messages which* P_i *can simultaneously send*

Each message in M_i has probability $\mid Q \mid \div \mid M_i \mid$ of being an element of Q

ELSE

Grant communication resources to all messages in M_i

ENDIF

DELETE from M_i all messages in Q which are granted communication resources the way to their destinations

ENDFOR

ENDFOR

Using this scheme, the number of delivery cycles needed before all processors have halted, is conceptually the number of subsets that the approach partitions M into.

Guaranteed delivery is assured by two facts

1. Cycles will continue until each processor has delivered all its messages through the communication network.

2. Switches are perfect concentrators.

In addition, essentially no global information of any kind is assumed. Finally, while the perfect concentrator property used to guarantee message delivery is a major obstacle to bounding switch complexity, the switch functionality we assume is realizable in the context of current hardware technology.

The major problem with the greedy approach, is that we have not yet been able to formulate any strong characterization of is ability to partition a message set efficiently. Currently our only bound is a result by Maley, which shows that the greedy approach cannot guarantee a partition with less than $\Omega(\gamma\lambda \log n)$ subsets[4]. This result however, provides no strong insight into how likely we are to experience partitioning which displays this type of performance. Consequently, in an attempt to experimentally examine the performance capabilities of the greedy approach, we have implemented a software simulation of a fat-tree on the Symbolics 3600 series Lisp Machines[6]. The results of these simulations, while preliminary and incomplete, provide some indication that the greedy approach may not on the average display the kind of performance that Maley's analysis shows is possible. Figures 19 through 21 show plots of our simulation results to date. All simulations are of fat-trees using the greedy approach to partition message sets which, by Maley's proof, should display the $\Omega(\gamma\lambda \log n)$ bound.

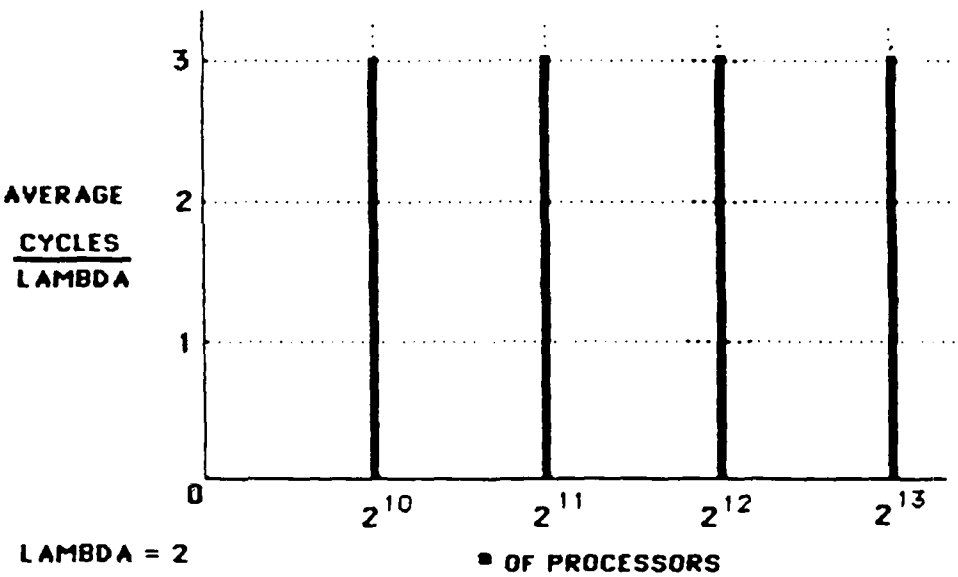


Figure 19: Simulation Results For Load-Factor of 2

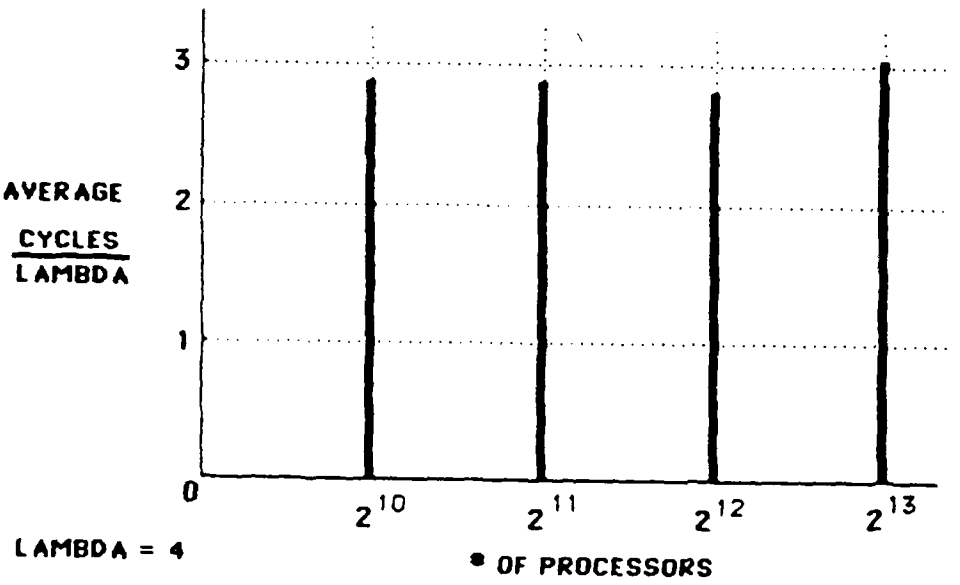


Figure 20: Simulation Results For Load-Factor of 4

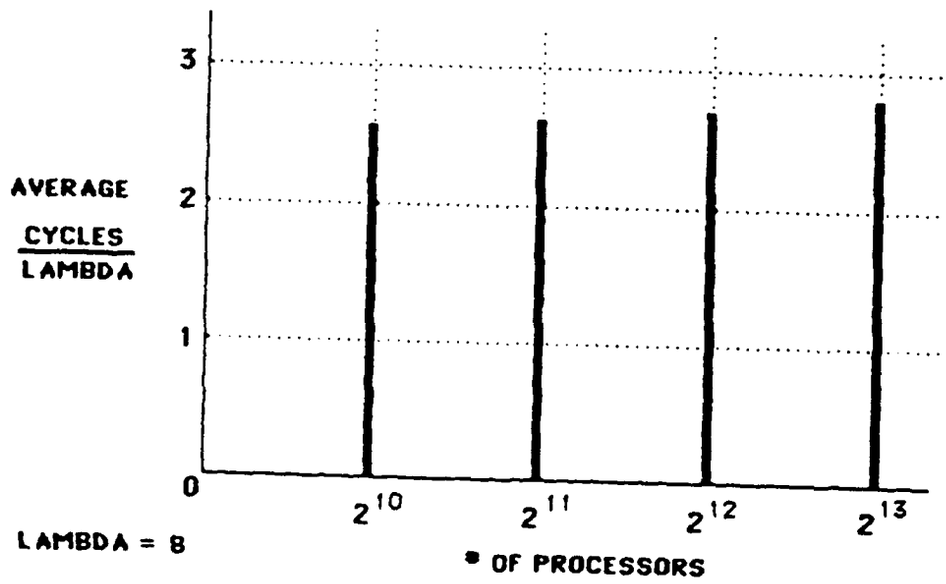


Figure 21: Simulation Results For Load-Factor of 8

While the number of simulations we have performed is insufficient to establish statistical significance, those that we have done would seem to indicate that the greedy approach performs satisfactorily, displaying an average upper bound of 3λ. This is not unexpected, since Maley's bound would utilize a $\gamma < \frac{1}{3}$ for the simulated cases, and thus be unobservable. The experimental result does indicate, however, that Maley's bound is not conservative. Consequently, we do not expect the greedy approach to display performance appreciably worse than $\Omega(\gamma\lambda \log n)$.

5 Concluding Remarks

We have, in this document, attempted to address some of the issues which effect interprocessor communication in a fat-tree. We have ignored issues of node and processor implementation, and centered more on addressing and some of the general ideas of communication resource arbitration. The findings reported here are preliminary, but would seem to indicate that efficient mechanisms exist for supporting all the vital aspects of interprocessor communication on fat-trees.

6 Acknowledgments

Greatest thanks must go to Charles Leiserson, whose ideas and support are responsible for all the work presented here. Thanks also to Tom Cormen, Ron Greenberg, Tom Knight, Bill Long, Bruce Maggs, Miller Maley, Cindy Phillips and Rich Zippel for their willingness to share both their expertise and computational resources. Finally, special thanks to Apple Computer Company, for producing the first affordable computer capable of generating the diagrams in this document.

7 References

1 G. Gordon, *System Simulation*, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1978.

2 F. T. Leighton, "A layout strategy for VLSI which is provably good", *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, May 1982, pp. 85-98.

3 C. E. Leiserson, "Fat-Trees: universal networks for hardware-efficient supercomputing", *1985 International Conference on Parallel Processing*, IEEE, to appear.

4 F. M. Maley, personal communication, documentation available on request.

5 C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, Massachusetts, 1980.

6 D. Weinreb, D. Moon and R. Stallman, *Lisp Machine Manual*, M.I.T. A.I Lab, 1984.

END

FILMED

2-86

DTIC