Bolt Beranek and Newman Inc.

Report No. 4563

Development of a Voice Funnel System

Quarterly Technical Report No. 6
1 November 1979 to 31 January 1980

November 1980

Prepared for:
Defense Advanced Research Projects Agency

(9) Quarterly technical rept. no. 6, 1 Nov 79-31 Jan 80,

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A093 513 / 3. RECIPIENT'S CATALOG NUMBER
(6) 4. TITLE (and Subtitle) DEVELOPMENT OF A VOICE FUNNEL SYSTEM, QUARTERLY TECHNICAL REPORT NO. 6	5. TYPE OF REPORT & PERIOD COVERED Quarterly Technical
	(14) 6. PERFORMING ORG. REPORT NUMBER BBN-4563
7. AUTHOR(s) (10) R. Rettberg	(15) 8. CONTRACT OR GRANT NUMBER(s) MDA903-78-C-0356, ARPA Order-3653
9. PERFORMING ORGANIZATION NAME AND ADDRESS Bolt Beranek and Newman Inc. 50 Moulton Street, Cambridge, MA 02138	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order No. 3653
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd., Arlington, VA 22209	(11) 12. REPORT DATE November 1980
	13. NUMBER OF PAGES 44 (12) 54
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Voice Funnel, Digitized Speech, Packet Switching, Butterfly Switch, Multiprocessor

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This Quarterly Technical Report covers work performed during the period noted on the development of a high-speed interface, called a Voice Funnel, between digitized speech streams and a packet-switching communications network.

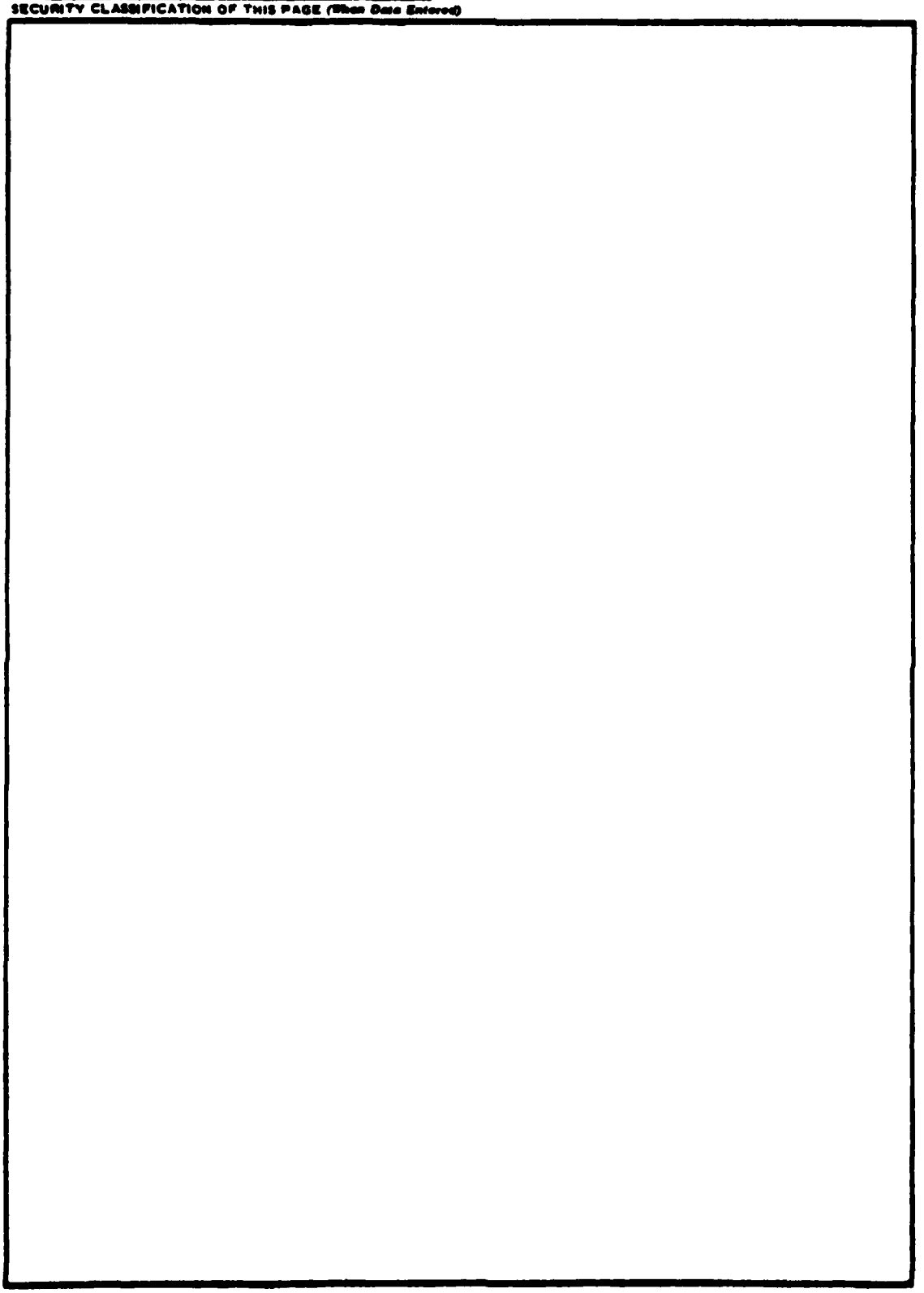
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

JOB

060100



Report No. 4563 Bolt Beranek and Newman Inc.

DEVELOPMENT OF A VOICE FUNNEL SYSTEM

QUARTERLY TECHNICAL REPORT NO. 6
1 November 1979 to 31 January 1980

November 1980

This research was sponsored by the
Defense Advanced Research Projects
Agency under ARPA Order No.: 3653
Contract No.: MDA903-78-C-0356
Monitored by DARPA/IPTO
Effective date of contract: 1 September 1978
Contract expiration date: 31 January 1980
Principal investigator: R. D. Rettberg

Prepared for:

Dr. Robert E. Kahn, Director
Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, VA 22209

Table of Contents

FIGURES

1. *Introduction*

This Quarterly Technical Report, Number 6, describes aspects of our work performed under Contract No. MDA903-78-C-0356 during the period from 1 November 1979 to 31 January 1980. This is the sixth in a series of Quarterly Technical Reports on the design of a packet speech concentrator, the Voice Funnel.

This report describes the hardware design of the primary processing resource of the Voice Funnel, the Processor Node.

2. Processor Node Hardware Description

The Butterfly Multiprocessor upon which the Voice Funnel is being built consists of Processor Nodes which are interconnected by Switch Nodes. The designs of the Switch and the Switch Node have been described previously.* A preliminary design for the Processor Node was described in the Design Report. The current report describes the Processor Node design as it now exists. The description in this report is not intended to be a reference manual for the Processor Node, but is rather a description of the design and the considerations and tradeoffs that went into the design.

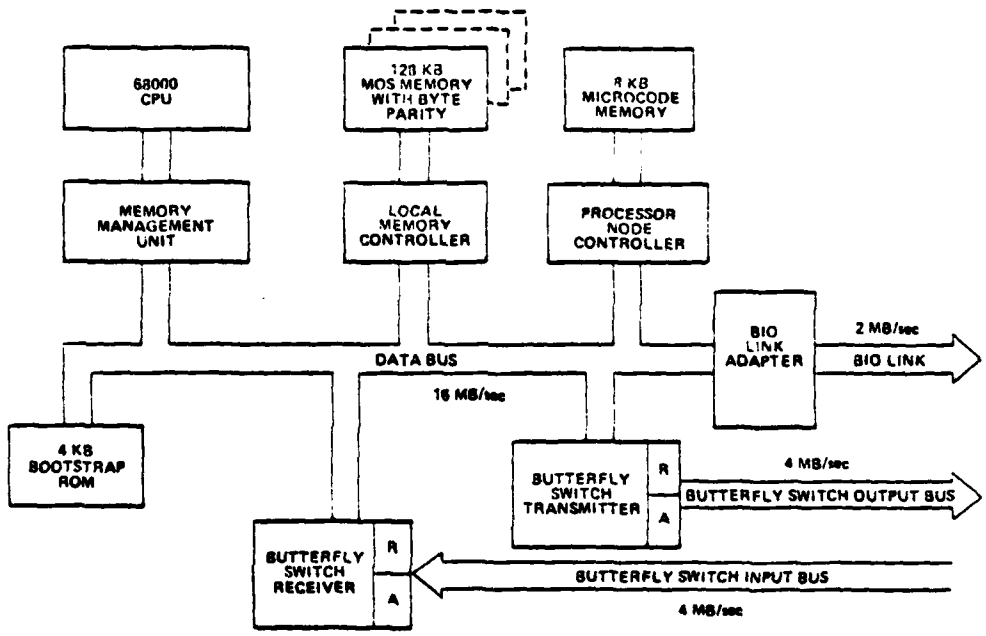
The Processor Node is a collection of hardware components which together provide the resources that support the processing of the Butterfly Multiprocessor. Its physical components are:

- a Motorola MC68000 16-bit microprocessor
- a memory management unit
- main memory and main memory controller
- Read Only Memory
- Processor Node Controller
- peripheral bus adapter
- interface to the Butterfly Switch

These components communicate over a high speed internal bus called the Data Bus. Figure 1 illustrates the interconnection of the major Processor Node components and gives the bus

---

* in "Development of a Voice Funnel System: Design Report," BBN Report No. 4098, and "Development of a Voice Funnel System: Quarterly Technical Report No. 2," BBN Report No. 4143.



Processor Node Components
Figure 1

communication rates in megabytes per second.

The MC68000 is a VLSI microprocessor from Motorola which combines state-of-the-art technology and advanced circuit design techniques to achieve an architecturally advanced 16-bit microprocessor. Its primary features include:

- 32-bit data and address registers
- 16 megabyte direct addressing range
- wide variety of instruction types
- five data types
- I/O access through the memory address space
- 14 addressing modes.

Augmenting the MC68000 is a custom designed Memory Management Unit which separates the virtual address space seen by an executing process from the actual memory configuration. In addition to address translation, the MMU provides protection by allowing each process only certain access privileges.

Up to four main memory modules can be attached to the Processor Node. Each module has 128K bytes of semiconductor memory and is implemented with 16K bit memory chips. Byte parity provides single-bit error detection. Battery backup assures no loss of state information on system power failure.

A 16-bit wide 2901-based microprocessor with 1K words of 64-bit wide microcode ROM is used to control the various Processor Node resources and provide those functions which, because of throughput or indivisibility requirements, cannot be

provided by the MC68000. The Processor Node Controller (PNC) also acts as the controlling agent in Butterfly Switch transactions.

4K bytes of Read Only Memory (ROM) are provided to allow execution of instructions immediately upon power on. The ROM contains the loader and diagnostics and serves as a permanent residence for such programs as the system debugger.

Also on the Data Bus are two kinds of bus adapters: an adapter to an input and an output port of the Butterfly Switch, and an adapter to the BIOLINK which connects I/O peripherals such as local net interfaces and the interface to the PSAT to the Processor Node.

Physically, the Processor Node consists of a 12" X 18" 4-layer printed circuit board which holds all non-memory elements and one or more 10" X 11" 4-layer PC daughter boards for the memory units. Each PC board has its own switching power supply. Four mass-terminated cables provide power, connection to the Butterfly Switch input and output ports, and connection to the I/O boards.

2.1 Processor Node Controller

The elements of the Processor Node execute many simultaneous operations. As an example, the following operations could be

executing simultaneously:

1. the MC68000 adding two numbers,
2. an incoming message from another Processor Node being assembled,
3. a block of data being transmitted across the switch, and
4. an arbitration being made as to which I/O controller is next to access local memory.

For high throughput environments (such as the Voice Funnel) multiple concurrent operations are normal and desirable. An important design issue in the Processor Node is how to provide communications paths between the various autonomous hardware resources while controlling and serializing their interactions. The problem is complicated by the use of ICs from different families which have incompatible timing requirements; for example, the Bootstrap ROM is much slower than main memory.

In addition, the Butterfly architecture requires many functions which are not provided by the MC68000. These functions include:

1. a 32-bit time-of-day clock and timer which can interrupt the MC68000 at a given time,
2. refreshing the dynamic memory chips,
3. converting virtual to physical addresses,
4. moving a block of memory from one Processor Node to another,
5. restarting another processor node,

6. supporting inter-process communications with the dual-queue mechanism,
7. handling timeouts for both messages being transmitted and solicited replies,
8. controlling interrupt acknowledges from the MC68000,
9. controlling error reporting from events such as timeouts, checksum errors, and parity errors,
10. posting events for the operating system,
11. generating power-down interrupts and power-up sequencing,
12. interacting with I/O device controllers on background jobs,
13. initializing the MMU's segment attribute registers after a restart.

Also, as the software impact of a multiprocessor environment becomes clearer, additional previously unrecognized functions may need to be added to the Butterfly architecture. Control flexibility thus becomes an important design consideration.

These motivations led to the inclusion of a high speed processing element called the Processor Node Controller (PNC). To reduce complexity, a single bus is used to interconnect the various Processor Node elements. This approach satisfies the need for flexible control hardware and reduces the interconnection complexity to a minimum.

2.1.1 Microinterrupt Requirements

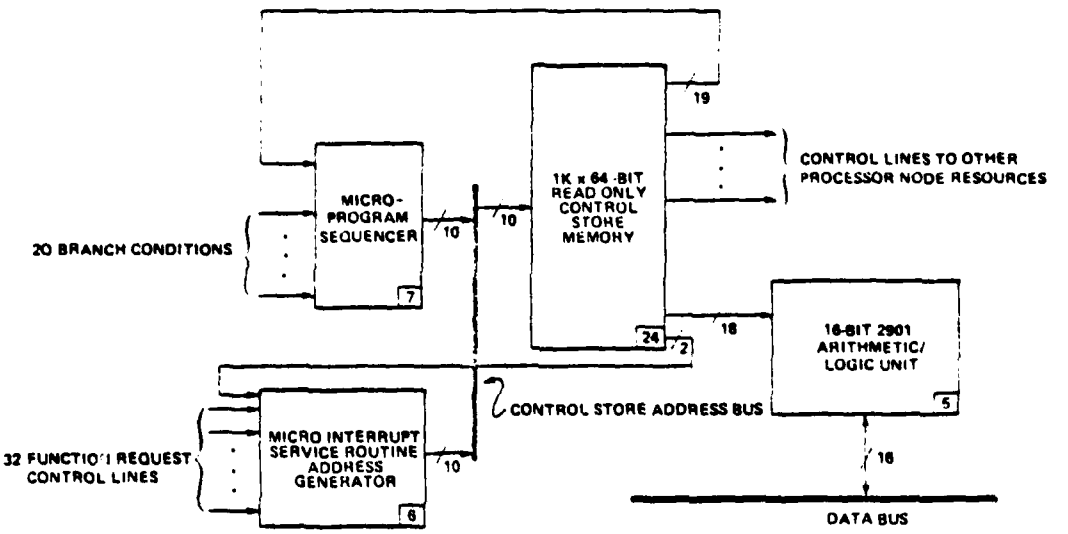
The execution of all Processor Node functions involves a sequence of transfers from one bus element to another and/or activation of bus element control lines. Each function occurs through the PNC's execution of a sequence of microinstructions from its 1K-word, 64-bit wide, Control Store memory.

Several of the data transactions must have very low delay. For example, the delay required when the MC68000 accesses its local memory has a dramatic impact on system performance. Other issues such as low DMA latency, prioritization of switch transaction events, and handling of memory refresh have led to a microinterrupt-driven PNC. In this architecture, a hardware element generates a microinterrupt service routine address corresponding to the highest priority function request. When the PNC enables microinterrupts, the service routine begins executing almost immediately.

2.1.2 PNC Hardware Implementation

The PNC has four components interconnected as shown in Figure 2. (The numbers in each box indicate how many ICs are used for that function. The other numbers are bus widths.)

A 16-bit bipolar microprocessor unit using the AMD 2901 provides a means for the PNC to perform data manipulations and



Processor Node Controller Components
Figure 2

provides storage for 13 variables used in various Processor Node functions. Four additional 16-bit registers are provided for use during a microinterrupt service routine. Distinctive characteristics of the microcoded processor are:

- seventeen 16-bit registers,
- an eight-function ALU,
- two-address architecture which allows simultaneous access to two working registers,
- flexible data source selection which allows ALU data to be selected from five source ports for a total of 203 source operand pairs for every ALU function,
- left/right 32-bit rotates,
- carry, zero, and negative status flags which can be used to control conditional branches.

The 1K 64-bit read only Control Store memory consists of 16 high-speed PROMS with edge-triggered output registers. In this architecture, the next microinstruction is fetched from the Control Store in parallel with the execution of the current microinstruction. This pipelining approach allows a 9 MHz PNC microinstruction rate. A writable Control Store would have been the preferred choice if not for cost (a factor of five over non-writable Control Store) and the additional complexity required to load the Control Store in a multiprocessor environment, where the PNC controls all switch transactions.

The 10-bit Control Store address is sourced by a conventional microprogram sequencer within a microinterrupt

service routine or by the Microinterrupt Service Routine Address Generator (MSRAG) just before entry to a microinterrupt service routine. The microprogram sequencer is implemented using the AMD 2911, augmented by two multiplexers which allow two-condition four-way branching. Other microsequencer features include:

- a pushdown stack for saving up to 4 return addresses,
- a means for returning to the zero microcode word,
- an internal address register usable as the address for commonly used routines.

The MSRAG is implemented using Programmable Logic Arrays to allow for future function additions.

2.2 Processor and Memory Management

The processor we have selected for use in the Voice Funnel is the 8 Mhz Motorola MC68000. Lower speed versions of this machine have been available since Q3 1979 and we expect full speed parts at the beginning of 1981. We have selected this processor because: 1) it is a 32-bit microprocessor, 2) it has a reasonably high instruction rate, and 3) it can manipulate a 24-bit virtual address space.

2.2.1 Processor Candidates

There were two candidates for the processor in the Voice Funnel: a commercial microprocessor or a custom microprogrammed processor. The advantages of a commercial microprocessor are its small size, low cost, and multiple sourcing. The primary advantage of a custom microprogrammed processor is that it can be designed to provide all those functions which are required in a multiprocessor environment.

We have selected use of a commercial microprocessor because it reduces the engineering risk and the MC68000 seems to satisfy our requirements in most of the critical areas. The choice is not without drawbacks since it was necessary to add the PNC co-processor to correct for defects (from a multiprocessor standpoint) in the MC68000. Use of a commercial microprocessor did allow us to concentrate on those aspects which are unique to our multiprocessor.

2.2.2 MC68000 Shortcommings

It is our opinion that the MC68000 is the most suitable device for the Butterfly Multiprocessor, and we are grateful to Motorola for producing a 32-bit machine which enabled us to avoid many serious problems. However, the design is lacking in a few areas -- we hope that by pointing out the weaknesses, future designs will be improved.

The MC68000 executes a bus error exception when the MMU signals that some protection attribute has been violated or a remote memory reference has timed out. During the bus error exception, seven words of state information are saved on the stack. Unfortunately, this state information is not sufficient to allow instruction retry. This prevents graceful recovery and makes demand paging unworkable. With the machine's 24-bit virtual address space, this is a serious oversight.

While advertising a bus structure whose architecture is easily interfaced, the MC68000 has substantial deficiencies when throughput is a primary goal. For example, in its four-cycle read operation, the first two cycles are used to keep the address stable for the previous memory operation, ungate the old address from the bus, regate the new memory operation address onto the bus and provide a delay before informing the world that a memory access is about to start. The MC68000 then expects an acknowledge signal indicating that the memory access can be terminated one cycle later. All decisions on what to do with the memory access (e.g., access local or remote memory, access an I/O peripheral register, interrupt acknowledge, etc.) must be made within that one-cycle period or the MC68000 will add additional cycles to the memory access, thus lowering throughput. The principal problem is that while the MC68000 knows it is going to do a memory access, it keeps that fact hidden from the outside world for half of the minimum memory access period. Because of

this, memory reads from local memory require five cycles instead of four. Local memory write operations also require five cycles although no loss of throughput is incurred since the minimum MC68000 write operation takes five cycles.

2.2.3 The MC68000 Interrupt System

The MC68000 provides a vectored interrupt system. An interrupt vector sourced by an element requesting service is used to find the address of the appropriate service routine. Multiple interrupt requests can be pending simultaneously on seven priority levels. The highest priority request having a priority level greater than or equal to a 3-bit code in the processor status register will trigger an interrupt. The highest priority element requesting an interrupt then supplies an 8-bit vector to the processor. We are allocating the 7 priority levels as follows, with level 7 being the highest:

7) PNC error or remote processor node interrupt message
6) memory parity error
5) microcode settable interrupt
4) I/O system high level interrupt request
3) I/O system low level interrupt request
2) microcode settable interrupt or system timer
1) microcode settable interrupt

The MC68000 provides 256 interrupt vectors which at first glance appear to be ample (the PDP-11 allows only 56). However, this poses serious problems. The MC68000 itself uses 47 of the interrupt vectors for various exceptions (e.g., bus error, trace,

spurious interrupt, etc.). Many of the LSI I/O devices use interrupt vectors freely, causing a shortage of interrupt levels with even a small number of devices.

For example, the Zilog SIO uses 8 interrupt vectors. If this chip is employed, only 26 I/O channels can be supported. In order to provide the 100-200 channels which could be connected to a single Processor Node, we will have to allocate unique interrupt vectors to only those devices which require low latency response. This requires that some interrupt service routines determine the cause of the interrupt themselves. Here the PNC provides a valuable service. During the interrupt acknowledge sequence, the PNC first determines if the I/O system is to source the interrupt vector. If so, the highest priority interrupting device provides both an 8-bit interrupt vector and an 8-bit device number/error code. The PNC stores both bytes in a special memory location reserved for each priority level. The interrupt service routine can then easily determine the cause of the interrupt if more than one event has the same interrupt vector.

2.2.4 Memory Management Unit

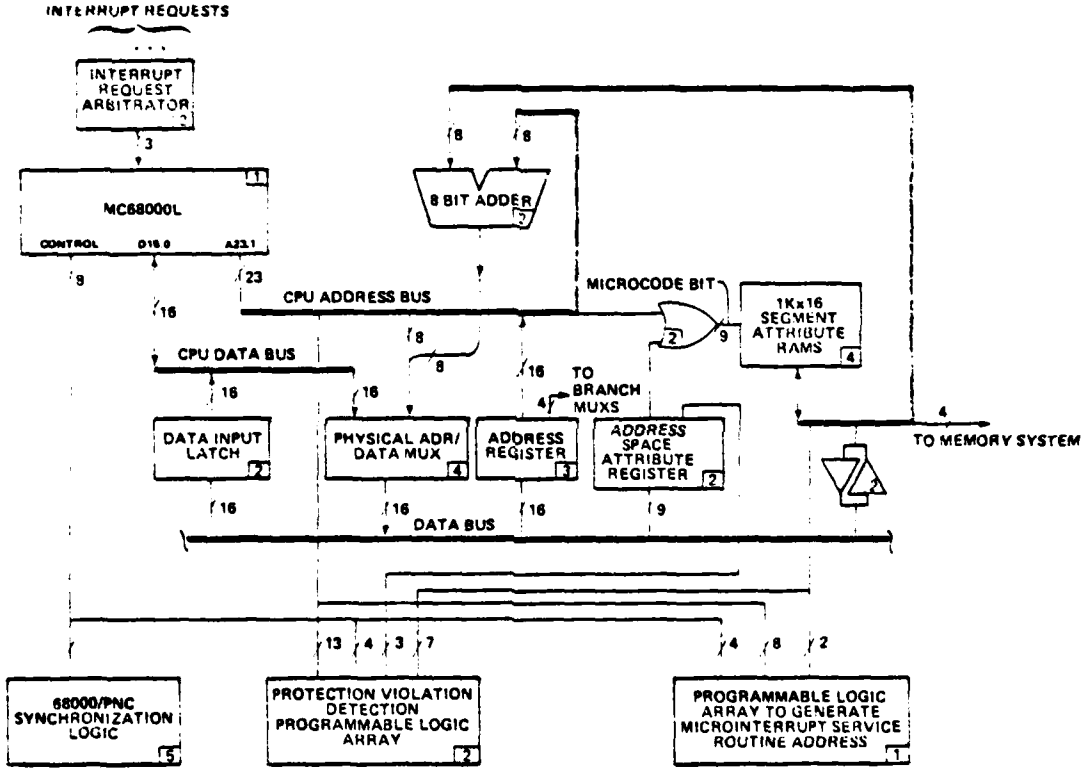
Integrating the MC68000 into the Processor Node requires several interesting design techniques which are briefly described. The need for non-standard interface techniques arises out of the design goals of the interface. The more important

design goals are:

1. the MC68000 local memory access delay should be minimized,
2. the PNC should be able to convert virtual to physical addresses,
3. the PNC should be able to test for protection violations,
4. context switching should be fast,
5. all MMU registers should be both read and write,
6. cost and complexity should be minimized, and
7. physical memory space should be uniform.

To set the context for the discussion, the reader should be familiar with the MMU functional description given in Quarterly Technical Report No. 5, BBN Report No. 4396. Figure 3 shows the final MC68000/MMU architecture. A SAR refers to one of the 512 32-bit Segment Attribute Registers and ASAR refers to the 16-bit Address Space Attribute Register.

The first design technique is to set up the MC68000 to access local memory, except when the PNC is executing a microinterrupt service routine. This, coupled with the use of a delayed version of the Processor Node clock to clock the MC68000, allows fine tuning of the synchronization between the MC68000 and PNC. During this period of synchronization, the MMU is converting the virtual address to the physical address and the PNC is loading the physical address into a latch in the memory subsystem. Only a small delay penalty is paid using this



MC68000 Interface to Processor Node
Figure 3

approach since determining what to do overlaps the first step of the most probable MC68000 operation.

The second technique is to recognize that relocation and protection are really separate functions. This allows time multiplexing of the 1K by 16-bit memory containing the SARs. During the synchronization microstep, the relocation portion of the SAR is gated to the AD bus. The low 4 bits of the Data bus become bits 16-19 of the physical address. Bits 8-15 of the Data bus are added to bits 8-15 of the virtual address to form bits 8-15 of the physical address. Bits 6-7 of the Data bus enter a Programmable Logic Array (PLA) which, during the synchronization process, generates the microinterrupt service routine address associated with the particular MC68000 access (e.g., local or remote memory, I/O space access, or segment zero access). In the microinterrupt service routine, a bit in the microword selects the protection portion of the SAR. The Protection PLA generates a signal which can be tested or used to cause a bus error to the MC68000. The multiplexing of these MMU memories reduces the cost of the MMU by approximately 35%. The penalty of this multiplexing is the reduction of the total number of SARs from 1024 to 512.

The third design technique is to map virtual segment FF (decimal 255) into physical segment zero, in which the PNC control registers are located. The MC68000 always sign-extends a short address offset. Thus the instruction "MOVE RO,8000"

actually causes R0 to be loaded into byte 8000 of segment FF because of the sign extension. By mapping virtual segment FF into physical segment zero, all MC68000 access to PNC control registers (a reasonably high frequency event) can use the short addressing mode.

The other element in the MC68000/MMU interface to the PNC is the Address Register. This register is multipurpose; it serves as a means for converting virtual addresses generated by the PNC to physical addresses, and checks for page and segment protection violations. It also allows accessing the SAR registers by the PNC. To access a SAR register, the PNC first requests the MC68000 to give up its command of the address and control lines. The PNC then loads the SAR address into both the Address register and the ASAR. Since the SAR address is the logical OR of these two registers, the correct SAR is selected. The third utilization of the Address register is a mechanism for allowing microcode branching on various bit values. Bits 0, 1, 2, and 3 can be used in two- or four-way microcode branching. Thus, to test if some address is even or odd, it is first loaded into the Address register. The next microinstruction can then branch to an even or odd microword depending on whether the address is even or odd.

2.3 Bootstrap Read Only Memory

The Read Only Memory (ROM) consists of one 4K x 8 Erasable Programmable ROM. This ROM is needed to allow execution of instructions immediately upon power on. In addition to the bootstrap, other programs such as the system debugger and some diagnostics will reside in the ROM.

Accesses to the ROM are triggered by the MC68000 reading from location 8000-8FFF of segment zero. The PNC reads two successive bytes from the ROM to form the 16-bit word given to the MC68000. The byte manipulation plus the slow access time of the ROM result in a 2 microsecond ROM access time. This slow access time creates no problems, however, since the programs executing out of ROM need not execute quickly.

A special mapping of locations 0-7 of segment zero into location 0-7 of the ROM allows the MC68000 to pick up the stack pointer and program counter to be used on startup from the ROM.

2.4 Butterfly I/O Link Adapter

A single processor subsystem consists of a Processor Node and up to four I/O modules. These I/O modules communicate with the Processor Node via a bus protocol called the BIOLINK protocol. An adapter on each Processor Node assists the PNC in satisfying the bus protocol.

Three types of transactions are supported by this protocol:

- MC68000 read from or write into an I/O device control or data register,
- I/O device controller read from or write into the Processor Node's local memory,
- I/O device controller request for an interrupt and source of an interrupt acknowledge response word to the MC68000.

The finest granularity of signal transitions in the BIOLINK protocol occurs at the Processor Node clock rate. Thus the BIOLINK is synchronous. Transactions require an acknowledgement signal from the address's bus element to signal completion of the transaction. This type of protocol allows devices of different latencies to be accessed. Thus, when the MC68000 reads a particular device register, it will be forced to wait until the device's controller is able to access the device on behalf of the MC68000.

In development of the BIOLINK adapter, five design goals were initially established; they are listed below in ascending importance.

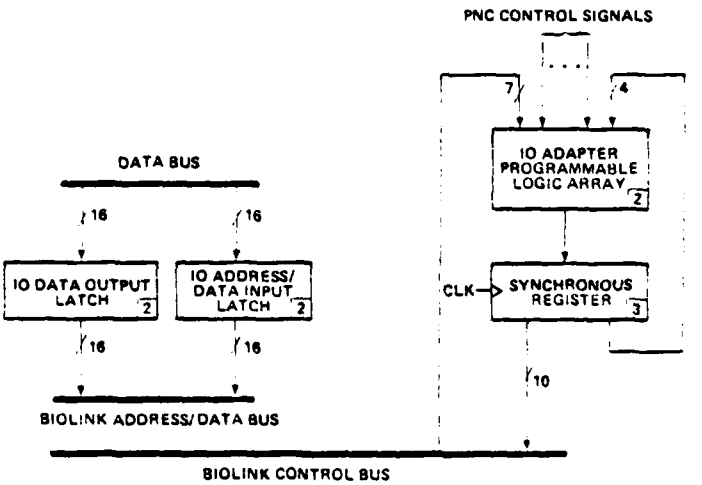
1. PNC bandwidth utilization should be minimized in I/O controller accesses to local memory,
2. the cost and complexity of the hardware should be minimal,
3. multiple I/O controllers should be given equal access privileges to local memory,
4. the PNC should be free immediately following a data transaction to/from the I/O adapter, and

5. interrupt vector capture should be a very simple transaction.

2.4.1 I/O Link Adapter Implementation

The hardware implementation of the I/O adapter consists of two parts as shown in Figure 4. A pair of back-to-back 16-bit latches pass data and address information between the PNC's Data bus and the BIOLINK's Address/Data bus, and a small finite state machine controls the autonomous aspects of the BIOLINK protocol. The finite state machine handles such tasks as: BIOLINK bus capture, round robin arbitration of the I/O controllers requesting access of local memory, microinterrupt request generation, data acknowledge synchronization, and bus enabling of the I/O Address/Data latch to the BIOLINK Address/Data bus.

The finite state machine implements only those BIOLINK protocol functions which must be handled immediately or which cannot be handled by the PNC. For example, arbitration of the multiple I/O controller requests to access memory could be done by the PNC on a polling basis. However, the resulting PNC bandwidth utilization and delay would be unacceptable. Furthermore, arbitration of who is to gain control of the BIOLINK cannot be done by the PNC by simple polling since the arbitration and locking must be accomplished in one microstep. During I/O device register accesses and interrupt vector capture sequences the PNC does assert and negate the BIOLINK Control signals since



I/O Adapter
Figure 4

these operations have relatively low duty cycles and the PNC must be dedicated to the transaction anyway.

2.5 PNC Memory Interface and Memory Modules

Although the processor is often more interesting, the memory system of a machine provides the same amount of computing power through its effective cycle time and its size and organization. In our current design there are several design points which are different from the original conception of the Processor Node's memory section.

In many systems, Error Detection and Correction (EDAC) logic is supplied with the hope of substantially improving the soft error rate of the memory. This logic requires a large number of chips (both control and memory). It is questionable whether it is worth doubling the number of chips to include EDAC. In the Processor Node memory, we chose to provide simple parity instead of EDAC, which may or may not be a realistic point of view. This uncertainty has reinforced the idea of placing the memory on a card by itself so that if a modification is necessary, only one card needs to be changed. While 64K dynamic memory chips are now appearing, their cost, soft error statistics, and availability make their use in the memory system an engineering risk at this time.

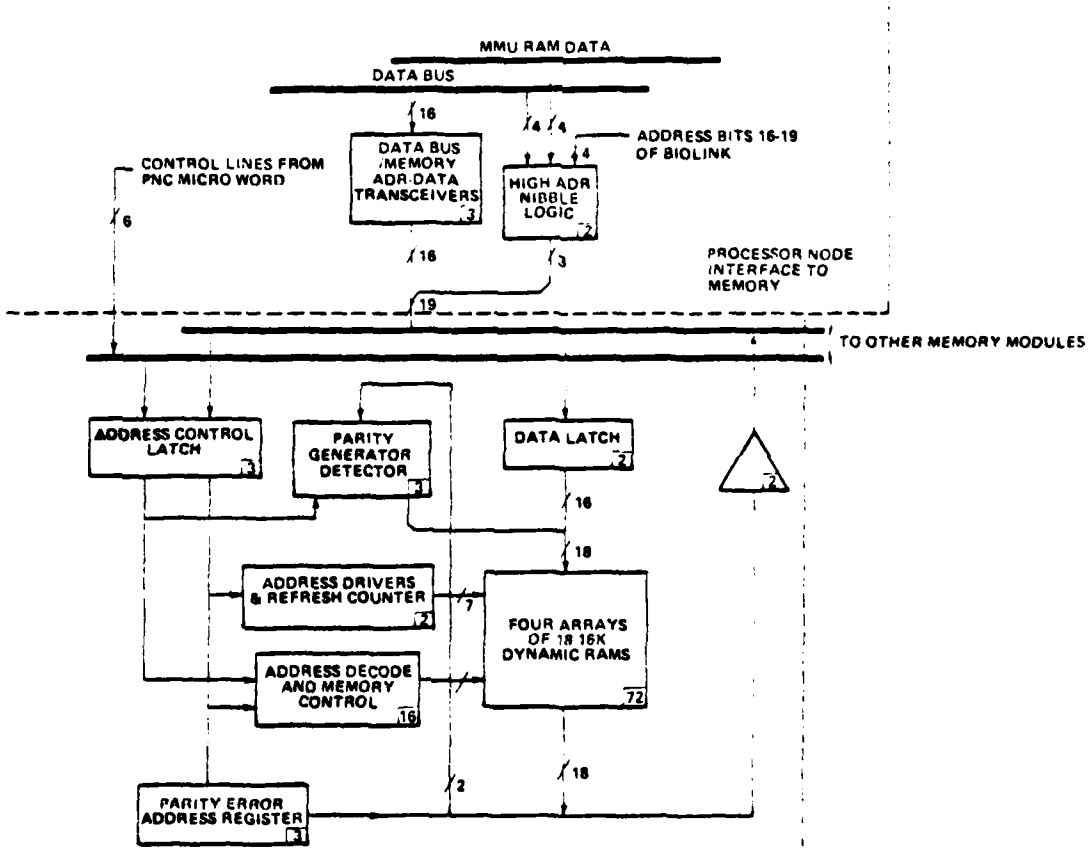
We have selected a basic memory size of 128K bytes on the basis of the expectation that 64K memory chips will make it unreasonable to provide a smaller memory. It also seems that around 80K bytes will actually be needed in the Voice Funnel application. Expansion of the local memory can be accomplished in the future by changing to denser chips in the same space allocated for memory, by adding memory boards, or by a demand paging system.

2.5.1 Memory Implementation

The memory has two parts: an interface on the Processor Node, and the memory module. The interface on the Processor Node is quite simple, consisting of a transceiver to allow gating the Processor Node's Data bus to/from the Memory Address/Data bus, and a multiplexer for sourcing the high 4 bits of the local memory address. The memory module is more complex, in part because of the need for automatic refresh upon power loss, but primarily due to error control and reporting.

The memory module is illustrated in Figure 5. Its major components are:

1. the memory, which includes the 72 16K memory chips, their line drivers, and the delay line required to develop critical control signal edge relationships,
2. latches, decoders, and line drivers for latching and buffering PNC data to/from the memory module,



Processor Node Memory Interface and Memory Module
Figure 5

3. parity generation and parity error detection logic,
4. a means for refreshing the memory chips and locking out the PNC from affecting memory state when the PNC is powered down.

Whenever a Processor Node reset is detected, all memory modules will automatically enter the self-refresh mode. In this mode, the module locks out all PNC interactions with the memory module except for that access which will change the memory module to the normal mode. A reset can occur when the system-wide reset button is activated, when a restart message is received, or when one of the power supplies detects that the power bus input voltage is too low.

In the normal case of power cycling, the PNC enables and inhibits the auto-refresh mode in a controlled manner. That is, all memory chip control signal timing relations are satisfied. When either of the two asynchronous reset events occurs, the possibility exists that the memory chip timing will be incorrect. We could have included extra hardware to prevent this occurrence but did not because it is not required. In a multiprocessor environment, activation of the system-wide reset occurs only as a result of a catastrophic system-wide failure. If the Processor Node receives a restart message, it is considered to be broken by the community of other Processor Nodes so that potential loss of memory contents is not a concern.

Each memory board has a 19-bit register in which is stored the address of the memory location which had the earliest parity error. Since a parity error generates a level six interrupt request, eventually the MC68000 will get around to reading this register (thereby allowing the register to be loaded again). What a particular Processor Node will do in the event of parity error detection is not clear at this moment. Most probably, we will invoke a soft restart and inform the community of Processor Nodes of a Processor Node malfunction. Persistent memory parity errors may cause the failing node to be disconnected.

All memory accesses require three cycles. The first microstep loads the address and type of operation (read, word write, or byte write) into a latch on each memory board. In a write operation, the PNC broadcasts the data to be written to all memory boards. The selected memory board latches the data, thereby freeing the PNC to do some non-memory-related operation during the third microstep. In a read operation, the PNC reads the data during the third microstep. The second microstep is thus free to be used to do something else. Thus memory bandwidth is 43MHz and uses the PNC during only two of the 3 cycles.

2.6 Switch Receiver Section

The function of the Receiver is to collect and assemble incoming messages from the Butterfly Switch in an input buffer.

Sometime during the assembly, the receiver requests microinterrupt service of the PNC. In the resulting service routine, the PNC uses the message type in the input buffer to determine what to do with the input buffer's data.

We have elected to implement the receiver as a microcoded finite state machine. This approach has the advantage of allowing a wide variety of message formats and optimization of the Receiver/PNC synchronization. More importantly, the approach frees the PNC for other service functions while the message is being assembled. In addition, it allows recognition of a Restart message without PNC intervention.

If a Restart message arrives, the Receiver checks a 16-bit password and, if it is correct, issues a restart to the rest of the Processor Node. The receipt of a Restart message indicates that the community of other Processor Nodes has agreed that the Processor Node receiving the Restart is in serious trouble (i.e., all other means for communication with it have failed). Since there are few software failures which will prevent one Processor Node from communicating with another, the community will assume a hardware failure has occurred. The advantage of having the Receiver interpret the message is that only a small section of the hardware need be working to initialize all finite state machines of a Processor Node (including the Receiver).

2.6.1 Preventing Deadlock

Two input buffers are provided. Assembled in one buffer are those message types which require the Processor Node to send out a reply message. Assembled in the other buffer are those messages which can be processed without using the switch. Two buffers are needed to prevent deadlock.

The Receiver peeks at each incoming message and makes the decision either to assemble the message in one of the two input buffers or to reject the message. Message rejection occurs when some resource needed to assemble the message is currently in use. For example, if the message is a "read-request", and the input buffer for messages requiring a reply response is full, the message will be rejected.

A Restart message cannot be rejected. If one arrives, the Receiver checks a 16-bit password and if the password is correct, issues a restart to the rest of the Processor Node.

2.6.2 Supporting Long Messages

The overhead involved in crossing the switch, coupled with the extra message data associated with routing, checksumming, and source Processor Node number inclusion, suggest that throughput can be substantially improved by sending more data in each message.

There are two ways we could read long messages. The first is the one used with short messages: accept the whole message into a large input buffer and poke the PNC just before reading the last part of the message. This approach makes the Receiver very simple but has some very undesirable characteristics. For instance, it requires the Receiver to have a very large input buffer. Complexity is introduced using a large buffer because of the MSI architectures of larger memory chips, the need to go to multiplexing schemes (the PNC has to read the data), and the extra address bits required. In addition, since the PNC is activated at the end of the message assembly, the input buffer will be locked by the PNC's loading the data into local memory for a duration almost equal to the assembly time.

The second and preferred way of handling long messages is to have an input FIFO. In the current implementation, a 16-byte input FIFO is used. This allows the PNC to start loading data into memory very soon after the beginning of the message is detected so that overlap of memory load and message assembly occurs. The FIFO approach also reduces the hardware required by the Receiver by reducing the number of state variables and allowing use of dual-port memories for the input buffers.

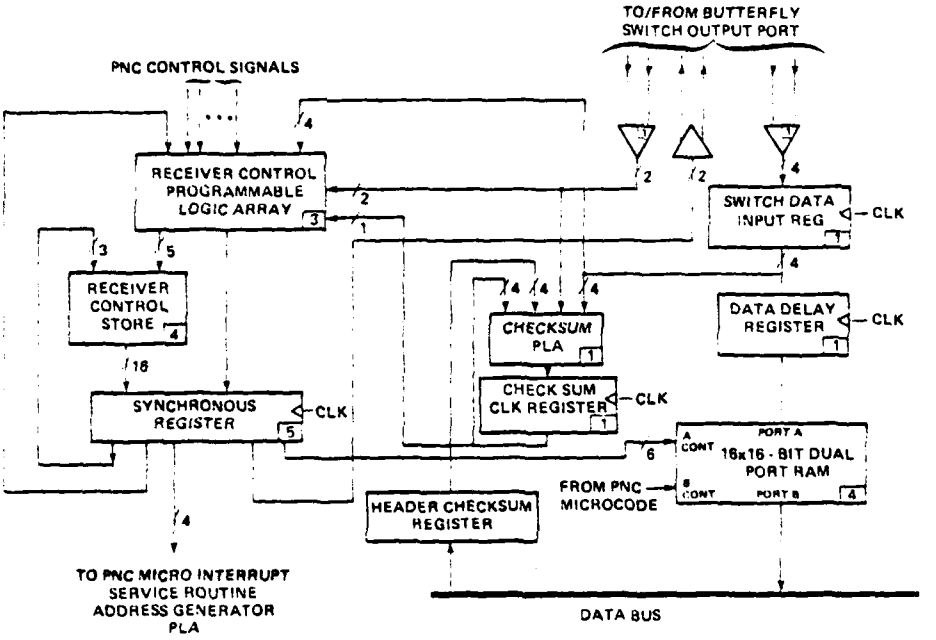
2.6.3 Flow Control

A characteristic which we have come to expect of communications networks is a requirement for flow control. This is necessary whenever we cannot guarantee that the receiver has sufficient resources to accept what the transmitter may wish to send. With the use of a Receiver input FIFO (and as we shall see in the next section, a corresponding Transmitter output FIFO), flow control is required. The cost of flow control in the MSI implementation was surprisingly low; the only cost was for the extra wires in the interconnecting cables and a small increase in Switch Node complexity. Since for all Butterfly systems of practical interest there are more Processor Nodes than Switch Nodes, moving the complexity from the Processor Node to the Switch Node is a good choice. The use of flow control in larger Butterfly systems (presumably using VLSI implementations) may not be justified. Instead, large dual-port memories or even multiple memories may be the better solution.

2.6.4 Receiver Implementation

A block diagram of the Receiver is shown in Figure 6. It consists of four parts:

1. a set of ECL to TTL line drivers and receivers,
2. a 16-word 16-bit/word dual-port memory,



Receiver Components
Figure 6

3. a checksum generator, and
4. a ROM/PLA-controlled finite state machine.

The finite state machine which drives the register control and state update is implemented with a general structure of a programmable logic array sourcing the address lines of a ROM control store.

We plan to run the Receiver FSM at the same clock frequency as the PNC to remove the synchronization problems which would arise from using autonomous clocks. While the Receiver's FSM structure allows a somewhat higher upper frequency of operation, the small potential benefit of using different clocks does not outweigh the additional complexity for synchronization.

Differential line drivers and receivers are being used because of past experience in interconnecting large systems. These receivers have a common mode noise rejection of at least one volt which allows system integration to be done with little worry about grounding and inter-cable crosstalk. ECL transceivers are used instead of TTL transceivers because the higher common mode rejection of the TTL transceivers does not offset the lower power consumption, lower edge speeds, and substantially higher speed of the ECL transceivers. Thus all communications between Butterfly Switch Nodes and between Processor Nodes and Switch Nodes use balanced ECL compatible signal protocols.

A dual-port memory provides a means by which the Receiver can be assembling a message in one of the input buffers and the PNC can be accessing data from a buffer during the same microstep. The Receiver FSM controls where the next nibble of Butterfly Switch output port data is to be loaded (i.e., one of the 64 possible locations), and the PNC can gate one of the 16 Receiver memory words onto its Data bus.

Calculation of the checksum is accomplished via a PLA which can either load the 4-bit contents of the Header Checksum Register (which is loaded by the MC68000) into the Checksum register, or update the Checksum register with the data from the Butterfly Switch output port.

2.7 Switch Transmitter Section

The approach of using PNC bandwidth with a few shift registers was initially thought to yield the simplest means for interfacing to the Butterfly Switch input port. The PNC always initiates a message transmission and therefore has nothing to do until message transmission is complete. This assumption turned out to be incorrect since the PNC interacted with other Processor Node resources, many of which were doing something while the message was being transmitted. Moreover, several Processor Nodes reading one Processor Node's memory could require 100% of its PNC bandwidth in returning replies, thus effectively halting the

node.

Moreover, a significant amount of additional hardware was required to provide those high speed functions which could not be implemented with a serial decision machine like the PNC. For example, the switch implementation expects the message to be terminated immediately upon giving a rejection to the Processor Node. Sensing this rejection cannot be done on a polling basis by the PNC, so special hardware must be provided to microinterrupt the PNC. This microinterrupt must also have a shorter latency than other requests. Handling the flow control in long messages requires sensing the "stop sending data" line. This has longer latency requirements but still must be handled very quickly because of the small input FIFO at the destination.

We have chosen to implement the Transmitter as a finite state machine similar to that used in the Receiver. This architecture satisfies the goal of minimizing PNC usage and requires very little additional hardware compared to the approach discussed above. The separate transmit and receive FSMs also mirror the full duplex nature of the interface and separate the PNC and transmitter functions at a well-defined boundary. The robustness of the transmitter structure also facilitates the addition of messages which have formats different from those originally planned.

2.7.1 Transmitting Long Messages

As was the case in the Receiver, the existence of long message formats to improve switch bandwidth utilization suggests that an output FIFO be implemented. An output FIFO allows the Transmitter's FSM to begin sending the message even before the first data byte is loaded into the output FIFO. This greatly reduces the delay that is normally encountered in store-and-forward techniques. The two flow control lines that were needed because of the Receiver's input FIFO now also provide flow control on the transmit side. Hence when the PNC becomes so busy (e.g., responding to I/O controller access requests to local memory and/or servicing the Receiver) that the Transmitter's output FIFO becomes empty, the Transmitter simply asserts the flow control line which indicates "this nibble should be ignored." Eventually the PNC will be available and will complete transmission of the message.

2.7.2 Two Output Buffers

The Receiver required two buffers to avoid deadlocks. The Transmitter, however, could be implemented with only one buffer. When a reply message must be sent, the PNC could simply ask the Transmitter to stop transmitting the message, assemble the reply, wait for the reply to be transmitted, and then reassemble the original message in the one output buffer. This works because

the PNC had to create the preempted message in the first place and presumably has not lost the means to reassemble the message.

We chose to provide two output buffers not only to mirror the receiver structure but to reduce PNC bandwidth utilization and introduce fairness. Fairness becomes an issue when a Processor Node must transmit both a request and a reply message at the same time. With the one buffer structure, the reply message must always be transmitted or deadlock will result. This could be resolved by having the PNC reassemble the reply and request messages alternately in the one output buffer as rejections are sensed. Unfortunately, this requires almost as much PNC intervention as does the first design approach.

The final motivation for providing two output buffers was that the FSM structure and size of the Transmitter RAM memory made it very easy, and the resulting interactions with the PNC became very clean. The PNC now sets a flip-flop in the Transmitter telling it that the corresponding output buffer is not empty. In the Transmitter's idle state, which occurs after transmission of a message or when a rejection is sensed, the flip-flop assertion is sensed and the message transmission begins. At the end of transmission, the Transmitter clears the flip-flop, thus enabling another message to be assembled by the PNC. The Transmitter has full responsibility for retransmission on rejection, checksum generation, and alternate path selection. If both output buffers are non-empty, the Transmitter will

alternate their transmission upon rejection responses, thereby guaranteeing fairness.

2.7.3 Alternate Path Selection

In the minimum Butterfly Switch configuration, the switch network is vulnerable to switch node failures. Intuitively, the failure of an individual switch node can be seen to affect a wedge of switch nodes and entry points, emanating in both directions from the failed node.

A good way to make the switch network more resilient against failures is to introduce some redundancy by adding an extra column of switching elements. This provides four paths from source to destination and brings up the problem of alternate path selection.

There are two ways of dealing with these alternate paths to a given destination. They can be handled either at a software or a hardware level. At the software level, the alternate paths would manifest themselves as several addresses for any given destination. As a processor notices that it can no longer access a destination by one address, it switches over to an alternate. The total software approach has two deficiencies. First, it requires the software to "know" about the switch structure on all transactions instead of only when the switch breaks. Second, the

benefits arising from spreading the messages amongst the alternate paths when simple conflicts occur in the switch could not be supported by a software-only means.

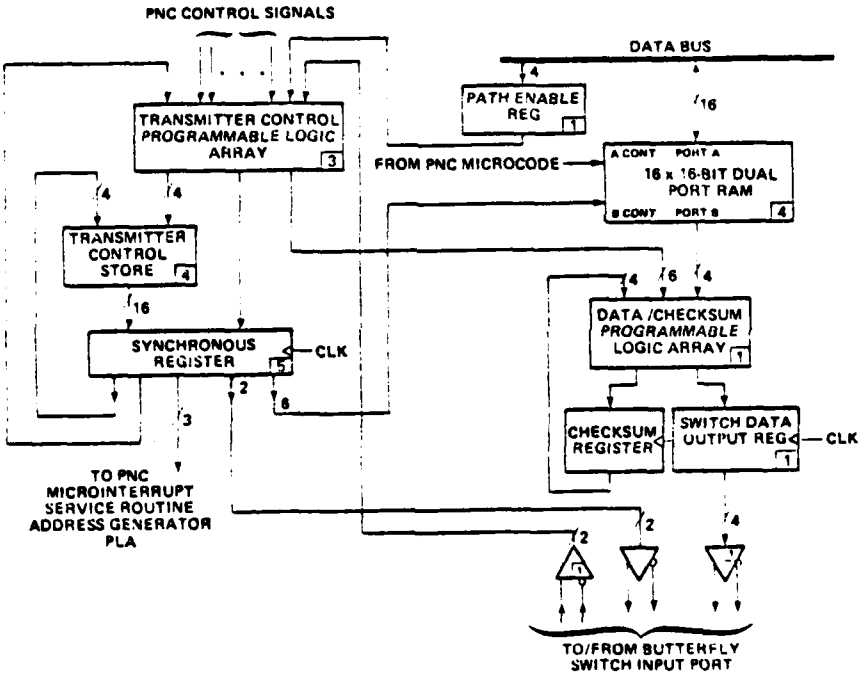
We have chosen a combined hardware/software approach. The Transmitter will automatically choose a new alternate path for each message transmission. This removes the need for the normal software to have an awareness of the switch structure. To prevent broken paths from being used, a four-bit register called the Path Enable register allows the MC68000 to enable or inhibit one or more of the alternate paths.

2.7.4 Transmitter Implementation

A block diagram of the Transmitter is shown in Figure 7. It consists of four parts:

1. a set of ECL to TTL line drivers and receivers,
2. a 16-word 16-bit/word dual-port memory,
3. a checksum generator/data multiplexer, and
4. a ROM/PLA-controlled finite state machine.

The Transmitter is very similar to the Receiver in its structure. The PNC can gate one of the 16-bit words onto the Data bus and load the low and/or high bytes of any word. Only 12 of the 16 RAM words are used for the two output buffers, so 4 words of temporary storage are added to the PNC's resources. The



Transmitter Components
Figure 7

switch data is sourced from a programmable logic array to provide the various sources (e.g., zero, RAM data, shifted RAM data, alternate path routing, checksum, ones). Alternate path selection occurs via part of one of the programmable logic arrays.

2.8 Performance and Statistics

Summaries of Processor Node hardware statistics and performance numbers are given in this section. It is hoped that they might give some insight into the expense and relative complexity of the various Processor Node elements on an absolute basis and relative to one another. The following table gives the component count, IC count, and cost of each of the major Processor Node sections. The costs are unburdened parts cost only for a run of 25 Processor Nodes. It is assumed that each Processor Node will include 128K bytes of local memory. The number in parentheses is the percentage of the total for the item.

Processor Node Element	Component Count	IC Count	Parts Cost
MC68000/MMU	44 (6.9)	33 (13.1)	366.8 (21.3)
Processor Node Controller	61 (9.6)	40 (15.9)	256.1 (14.9)
PC boards	2 (0.3)		244.5 (14.2)
Memory chips	72 (11.3)	72 (28.6)	223.2 (13.0)
Power supplies	327 (51.3)	4 (1.6)	163.6 (9.5)
Receiver	25 (3.9)	21 (8.3)	124.4 (7.2)
Transmitter	24 (3.8)	20 (7.9)	114.4 (6.6)
Memory Controller & Memory Interface	45 (7.1)	35 (13.9)	91.9 (5.3)
ROM	2 (0.3)	1 (0.4)	63.9 (3.7)
I/O Adapter	15 (2.4)	9 (3.6)	39.4 (2.3)
Miscellaneous	21 (3.3)	17 (6.7)	32.6 (1.9)
Total	638 (100.2)	253 (100.0)	1720.7 (99.9)

The following table gives execution times of many of the important Processor Node functions. To simplify the generation of the numbers it is assumed that:

1. there is no switch contention,
2. I/O device controllers are not accessing local memory,
3. remote transactions are between two distinct Processor Nodes,
4. the clock frequency is 8MHz,
5. the Butterfly Switch is two columns deep,
6. the MC68000 is executing code out of local memory when block transfers are in progress,
7. an addressed I/O device register adds two wait states to an access, and
8. the MC68000 does not have to wait for a switch transaction to start because of a previously initiated switch transaction.

Processor Node Function	Execution Time (microseconds)
local memory read	.625
local memory write	.625
remote memory read	3.750
remote memory write	1.250
I/O register read	1.625
I/O register write	1.625
block transfer initiation	8.000
interrupt acknowledge	1.750
set interval timer	1.250
read real time clock	1.500
set real time clock	.875
ROM read	2.000
read SAR	1.875
write SAR	1.625
write misc register	.625
read Processor Node Number	.625
write Processor Node Number	.625
read ASAR	.625
write ASAR	1.000
read Memory control register	.750
write Memory control register	.625
read and clear PNC status reg	.625
check user mode write	1.625
average instruction execution	1.625
memory refresh	2.4%
block transfer data rate	32 MHz for very long blocks

DISTRIBUTION OF THIS REPORT

Defense Advanced Research Projects Agency

Dr. Robert E. Kahn (2)

Defense Supply Service -- Washington

Jane D. Hensley (1)

Defense Documentation Center (12)

Bolt Beranek and Newman Inc.

Library

Library, Canoga Park Office

R. Bressler

R. Brooks

P. Carvey

P. Castleman

G. Falk

F. Heart

M. Hoffman

M. Kraley

W. Mann

J. Pershing

R. Rettberg

E. Starr

E. Wolf