C.vmp: The Analysis, Architecture and Implementation of a Fault Tolerant Multiprocessor

Dan Siewiorek
Mark Canepa
Steve Clark

December 1976

Departments of Electrical Engineering
and
Computer Science

Carnegie-Mellon University
Pittsburgh, Pa. 15213

This research was supported in part by Digital Equipment Corporation and by the Advanced Research Projects Agency under contract no. F44620-73-0074, monitored by the Air Force Office of Scientific Research.

TABLE OF CONTENT

Abstract

The architecture of a multiprocessor with a fault tolerant operating mode is described and analyzed. A bus level voter is used to satisfy the stringent design constraints of software transparency (programs from non-redundant versions will execute in a fault tolerant manner without modification), modularity, use of off-the-shelf components, and dynamic trading of performance for reliability. Bus level voting also allows handling of diverse system components (processors, memories, floppy disks, teletypes, etc.) in a uniform way. Models of performance degradation (20% slower than non-redundant on instruction execution rate, 50% slower on expected disk latency) and reliability improvement (both permanent and transient failures) are presented as well as experience in redundant system debugging, system initialization and switchover software, and initial performance measurements. The system, which is nearing completion, will be used to measure the occurrence of transient failures and to test fault tolerant bus protocols.

1. INTRODUCTION

The following trends have fostered an increased concern for highly reliable computer systems:

.Computer systems are becoming more complex and sophisticated. Their complexity is growing at a faster rate than the increase in the component reliability, thus leading to decrease in overall system reliability [GoldJ75].

.Computers are being used for more critical applications where the external environment cannot be precisely controlled and at the same time little or no maintenance can be performed.

.More and more small computer systems are operated by users who either cannot or do not want to provide their own maintenance. In small systems the cost of repair and maintenance is quickly dominating the original cost of the hardware.

Therefore the correct operation of computer systems in the presence of permanent or transient failures is increasingly important.

The design presented here concentrates on toleration of hardware failures. The design goals may be summarized as follows:

. Permanent and transient fault survival
. Software transparency to user
. Real time operation capability
. Modular design
. Off-the-shelf components
. Dynamic Performance/Reliability tradeoffs

Design Goals

1) Permanent and transient fault survival.

The system has the capability to continue operation in the presence of a permanent hardware failure- i.e. a component or subsystem failure- and in the presence of transient errors- i.e. a component or subsystem is lost for a period of time due to the superposition of noise on the correct signal.

2) Software transparency to the user.

The user should not know that he is programming a fault tolerant computer. All fault tolerance is achieved in the hardware. This also implies that if a user wants to upgrade to a fault tolerant system in steps - as will be described later - he can still maintain software compatibility.

3) Capable of real time operation.

A fault must be detected and corrected within a short period from the time the fault actually occurs. In real time applications the machine cannot pause too long to restore itself after an error. The method used to detect and correct errors depends on the error response time required by the system. This, in turn, depends on the application.

4) Modular design to reduce down time.

The hardware must be able to operate without certain sections activated. Hence, maintenance can be performed without having to halt the machine. Modularity includes the design of separate power distribution networks to be able to deactivate selected sections of the machine. The use of modules in the design also has the virtue of allowing the user to upgrade from a non-redundant, to a fully fault tolerant computer, in steps.

5) Off-the-shelf components.

To decrease the amount of custom designed hardware, to be able to rely on an established software library, and to allow systematic upgrading to a fault tolerant system, the computer primarily employs off-the-shelf components. In our case the LSI-11 computer was chosen as the primary building block.

6) Dynamic performance/reliability tradeoffs.

The fault tolerant computer has the capability, under operator or program control, to dynamically trade performance for reliability. Three computers executing three separate tasks can, by switching to fault tolerant mode and thereby working on the same task, achieve an increase in reliability at the expense of a performance degradation by a factor of three.

Section 2 describes the architecture that met the design goals, including multiprocessor organization, extensions to the architecture, and instrumentation for transient fault measurements. From experience gained in the design and implementation of C.vmp (for Computer, voted multi-processor) some conclusions about

fault tolerant bus protocols are given in Section 3. Section 4 outlines experiences in debugging, initialization, and software. Models of reliability (both hard and transient faults) and performance are derived in Sections 5 and 6. The paper concludes with a discussion of current status in Section 7.

2. SYSTEM ARCHITECTURE

The following techniques were explored to decide the best design for a fault tolerant computer [SiewD71].

1) Code and Hardware duplication

Code duplication techniques put fault tolerance on the software level. Fault tolerance is no longer user transparent and existing software can no longer be used without radical modifications. SIFT [WensJ72] and Randell's caching scheme [RandB75] are examples of software level fault tolerance. Hardware duplication allows the detection of error but cannot provide error correction.

2) Coding

Hardware error codes, such as parity, Hamming code and self-checking circuits provide error detection/correction, and are widely used to increase reliability (IBM 360 [IBM72]). Coding was a leading candidate, however an initial design study* showed it was not modular nor did it allow dynamic performance/reliability tradeoffs.

3) Periodic Diagnostics.

Periodic diagnosis is valuable in maintaining a system's availability. It is a powerful failure detection mechanism if the failure is permanent, but it assists little in the detection and correction of transient errors. This technique also interferes with real time applications.

4) Instruction retry.

Instruction retry techniques can be used in conjunction with error detection circuits to correct transient errors (IBM 360 [IBM72], STAR [AvizA71]),

*Coding on a bus with asynchronous protocol requires breaking the bus and buffering the signal. In addition coding would not mask processor errors.

but such methods would be of little use in the case of a permanent hardware failure.

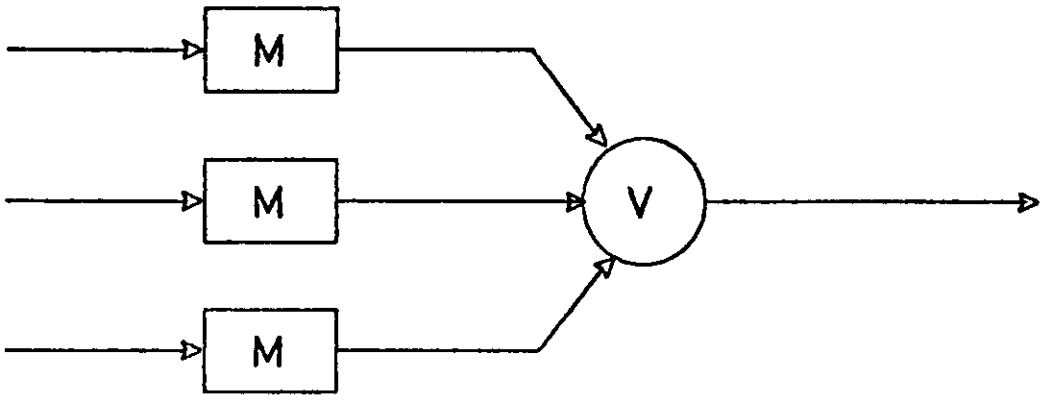
5) Voting Techniques.

> Voting techniques have the potential for fulfilling all the stated design goals. Triplicated majority voting logic is a classic method to achieve an increase in reliability for critical applications. First postulated by von Neumann [VonnJ56] voting requires a set of modules, M, all identical. The output of each module is compared in the voter. The result of the vote is then sent to the next stage of computation (see Figure 1). This is the simplest voting scheme and assumes a fault free voter.

Voting was thus selected for further study. What remained was to determine at what level voting was to occur. At the highest level of voting there are three independent computers each executing the same program. At certain points during program execution the three machines, through some common communication channel, compare results. If the three machines agree, they continue to the next checkpoint. If two of the three machines agree, the result of the vote is forced upon the disagreeing machine and the program continues execution. This level of voting, called "software voting", has been used in SIFT [WensJ72]. This method's clear advantage is the simplicity of the interprocessor communication links.

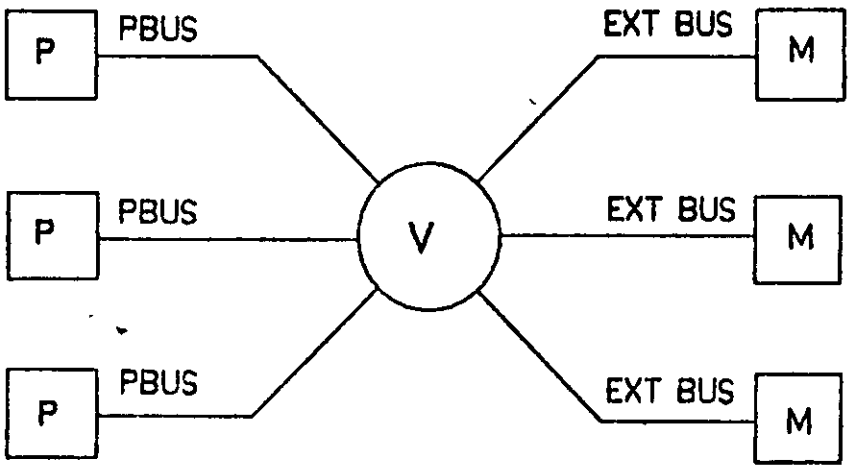
Software level voting also has some disadvantages. First of all the software necessary to do the voting is not transparent, and the user has to decide when and how often to vote. Second, the "voter" is not "memoryless". The three computers are independently executing the same task. If one computer makes an error, this error will not be caught and will be allowed to propagate until the next checkpoint is reached. When an error is finally detected there is still the problem of restoring the faulty computer's memory. The problem can be further compounded if the computers have three different answers.

At the other end of the spectrum, voting can be done at a very low level in the



TRIPLICATION WITH VOTING (FAULT FREE VOTER)

FIG. 1



BUS LEVEL VOTING (SINGLE VOTER)

FIG. 2

organization of the computer. The boxes in Figure 1 can be simple units such as single integrated circuits. Voting at this level solves the problem of propagation of errors and multiple disagreements. However it leaves the user with reduced flexibility in the area of reliability versus performance and in the area of gradual upgrade to fault tolerance. The best solution, at the present level of technology, is somewhere between these two extremes.

System Configuration

To be consistent with the design goals of modularity and software transparency, voting is performed at the bus level. That is, voting occurs every time the processors access the bus to either send or retrieve information. There are three processor-memory pairs, each pair connected via a bus as depicted in Figure 2. A more precise definition of C.vmp would therefore be: a multi-processor system capable of fault tolerant operation. C.vmp is in fact composed of three separate machines capable of operating while independently executing three separate programs. Under the control of an external event or under the control of one of the processors, C.vmp can synchronize its redundant hardware, and start executing the critical section of code.

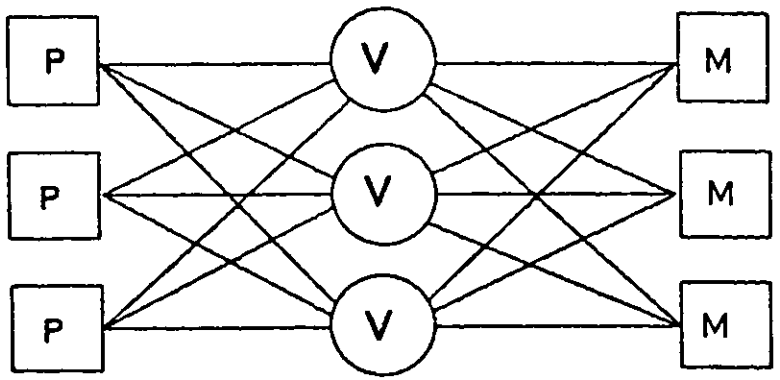
With the voter active, the three buses are voted upon and the result of the vote is sent out. Any disagreements among the processors will, therefore, not propagate to the memories and vice versa. Since voting is a simple act of comparison, the voter is memoryless. Disagreements are caught and corrected before they have a chance to propagate. If the voter is not faultless, triplicated

voters can be used (Figure 3). With this scheme any single element can have either a transient or a hard failure and the computer will remain operational. In addition, provided that the processor is the only device capable of becoming bus master, only one bidirectional voter is needed regardless of how much memory or how many I/O modules are on the bus. Voting is done in parallel on a bit by bit basis. A computer can have a failure on a certain bit in one bus, and, provided that the other two buses have the correct information for that bit, operation will continue. There are cases, therefore, where failures in all three buses can occur simultaneously and the computer would still be functioning correctly.

Bus level voting works only if information passes through the voter. Usually the processor registers reside on the processor board and so do not get voted upon. The PDP11, for example has six general purpose registers, one stack pointer, and one program counter. However, after tracing over 5.3 million instructions over 41 programs written by five different programmers and using five different compilers, the following average program behaviour was discovered [LundA74]:

> .On the average a register gets loaded or stored to memory every 24 instructions.
>
> .A subroutine call is executed, on the average, every 40 instructions, thus saving the program counter on the stack.
>
> .The only register that normally is not saved or written into is the stack pointer. To maintain fault tolerance the user must periodically save and reload the pointer.

This bus level voting scheme can be contrasted with the Draper Laboratory Symmetric Fault Tolerant Multiprocessor [HopkA75]. In SFTMP, memory and processor triads are interconnected by a triplicated serial bus. Program tasks are read from a memory triad into local memory in a processor triad where execution takes place.



BUS LEVEL VOTING (TRIPLICATED VOTER)

FIG 3

After execution the results are transferred back to memory triads. The major architectural differences from C.vmp are:

.Serial bus rather than parallel bus, thus degrading performance.

.Voting only takes place on transfers from and to memory triads. Errors in the processors may accumulate to the point that their results are not comparable.

.Programmer has to partition problem into tasks and provide for transfer to processor triads.

.SFTMP has up to 14 processors that can be dynamically assigned to four triads (two are spares). When a processor fails it can be replaced in its triad by another processor. However processors cannot operate independent of triads to improve throughput.

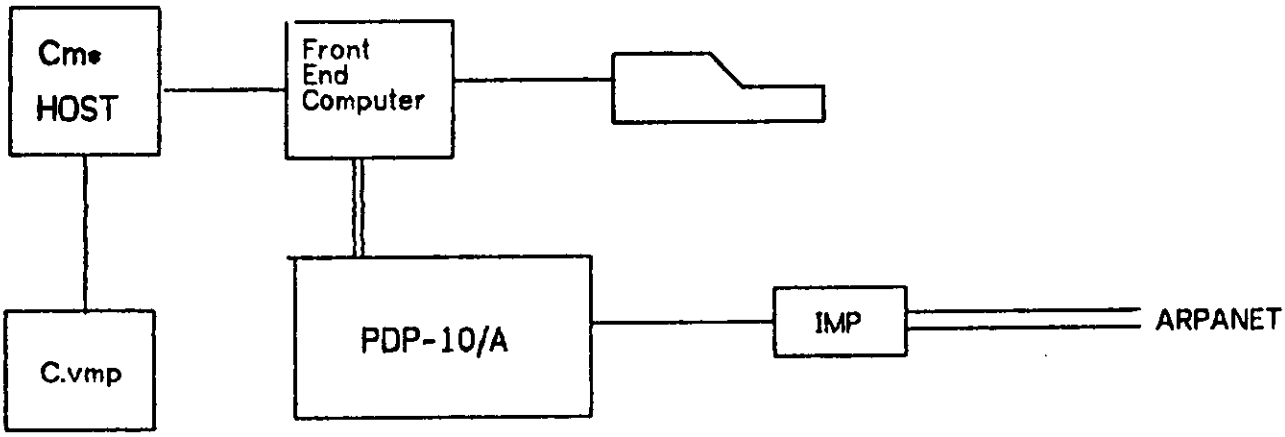
Another voting design is described by Wakerly [WakeJ75]. The major difference from C.vmp is that a unidirectional voter is employed and only the information flow from memory to processor is voted upon. Multiple devices on the bus require separate voters.

The connection of C.vmp to the external world and the system configuration presently under development are shown in Figure 4a and 4b respectively. The notation in Figure 4b is: P-processor, V-voter, L-parallel interface, M-memory, and SLU-terminal interface. To present a detailed description of the voter a brief digression to explain the DEC® LSI-11 Qbus is necessary [LSI]. The bus uses a hybrid of synchronous and asynchronous protocols.

Every bus cycle begins synchronously with the processor placing an address on the time multiplexed Data/Address Lines (DAL).

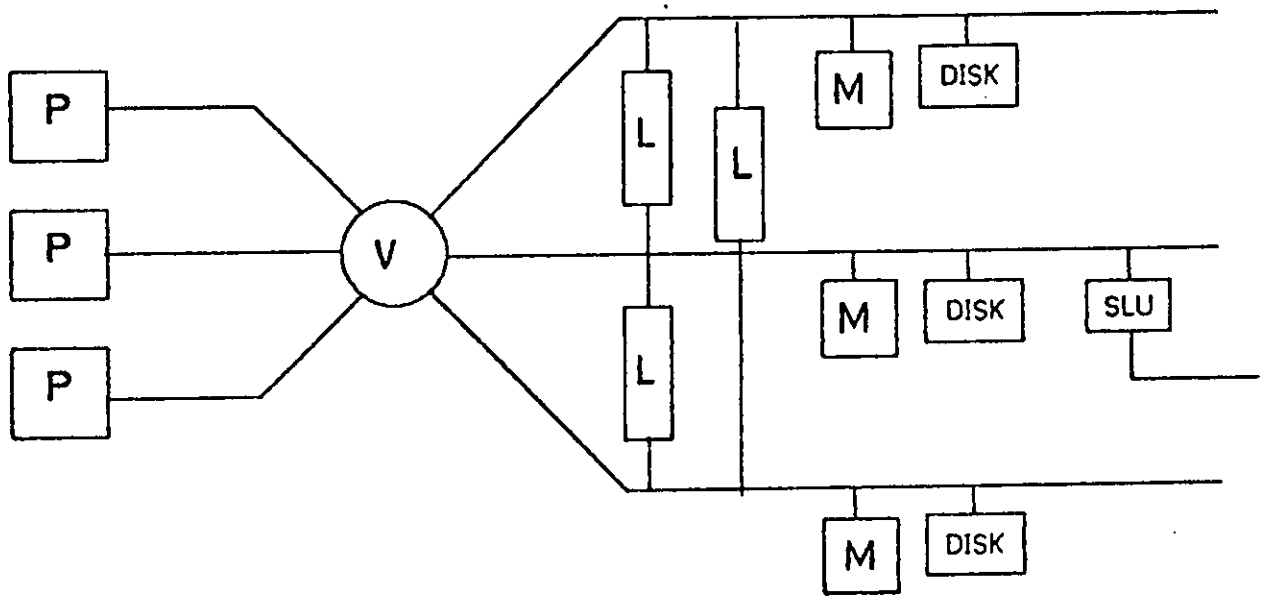
.The SYNC line goes high and all the devices on the bus latch the address from the DAL lines. The address is then removed by the processor. This terminates the synchronous portion of the bus cycle.

.In the event of an input cycle (DATI, shown in Figure 5) the processor activates DIN on the bus.



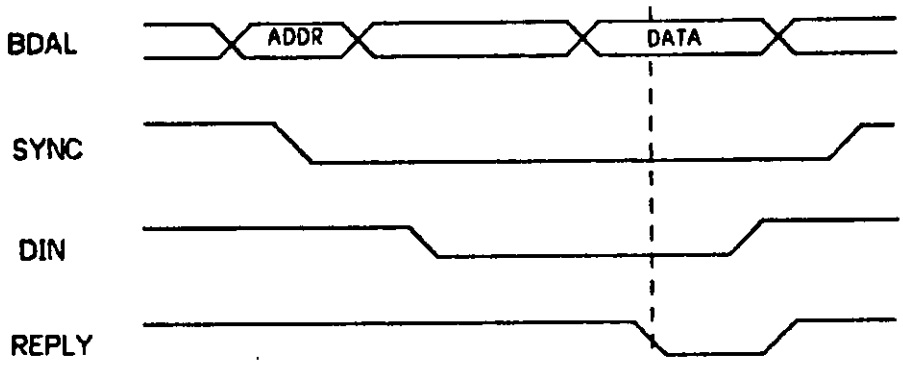
C.vmp connection to existing facilities

FIG 4a



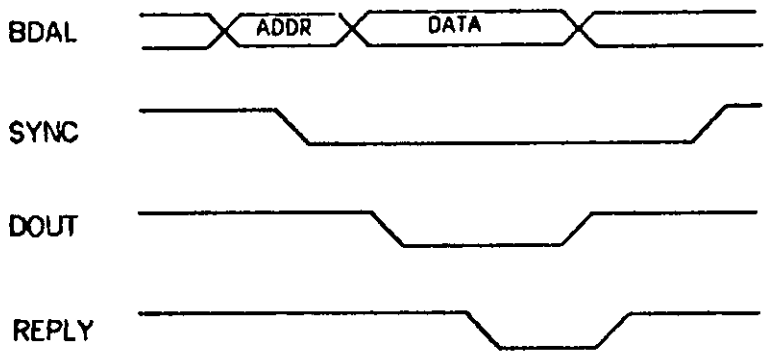
C.vmp System configuration

FIG 4b



DATI cycle for LSI-11 computer

FIG 5



DATO cycle for LSI-11 computer

FIG 6

.The addressed slave responds by placing a data word on the DAL lines and asserting REPLY.

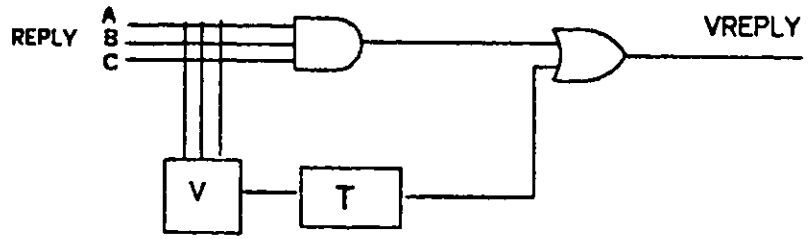
.The processor latches the data word and terminates DIN and SYNC.

.In the event of an output cycle (DATO, shown in Figure 6), after removing the address the processor places a data word on the bus and activates DOUT.

.When the slave device has read the word it activates REPLY.

.The processor responds by terminating DOUT and SYNC.

The three processors are kept synchronized by two separate means. A master clock synchronizes the microinstruction fetch-execute cycle in the three processors. Since bus transactions are asynchronous, they too must be synchronized in order to keep the processors in microinstruction lock step. The voter uses the REPLY signal to synchronize the external bus. In steady state, REPLY is issued by an external device at least once every bus cycle. At a certain point in time triplicated elements will issue REPLYs on the buses. These REPLY signals will all be on their respective buses within a window time t. The voter delays the REPLY signals to the processors until all the REPLYs have arrived. Then a common VOTED REPLY signal is routed to the three processors (Figure 7). If a REPLY fails to arrive within a time T>t then that REPLY signal is considered failed for that cycle and a VOTED REPLY based on the other two buses is sent to the processors. T is the maximum delay in REPLY a device on the bus can exhibit and still be within specifications. This method allows the three processors to receive REPLY within five nanoseconds of each other and stay synchronized*.



Voting circuit for REPLY and DONE

FIG 7

Voter modes of operation

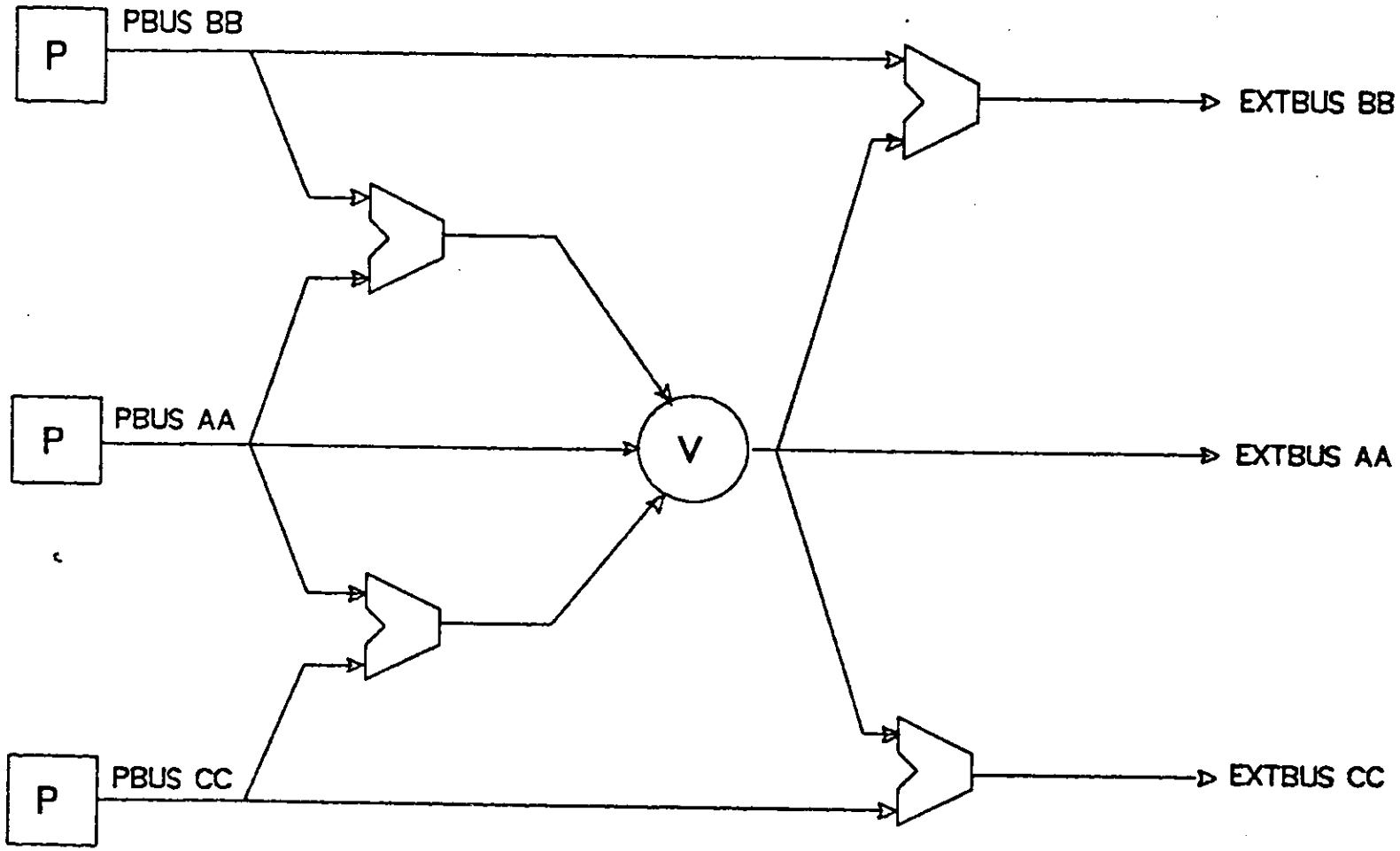
The Voter (Figure 8) can operate in three modes: independent (Figure 9), voting (Figure 10) and broadcast (Figure 11).

. Independent mode. Buses BB and CC are routed around the voting hardware. Bus AA is routed to feed its signals to all three inputs of the voting elements. In this mode C.vmp is a multiprocessor. Switching between independent and voting modes allows the user to perform a performance/reliability tradeoff.

. Voting mode. The transmitting portion of the three buses are routed into the voter, and the result of the vote is then routed out to the receiving portions of all three buses. In addition to the voting elements the voter has a set of disagreement detectors. These detectors, one for each bus, activate whenever that bus has "lost" a vote. By monitoring these disagreement detectors, one can learn about the kinds of failures the machine is having.

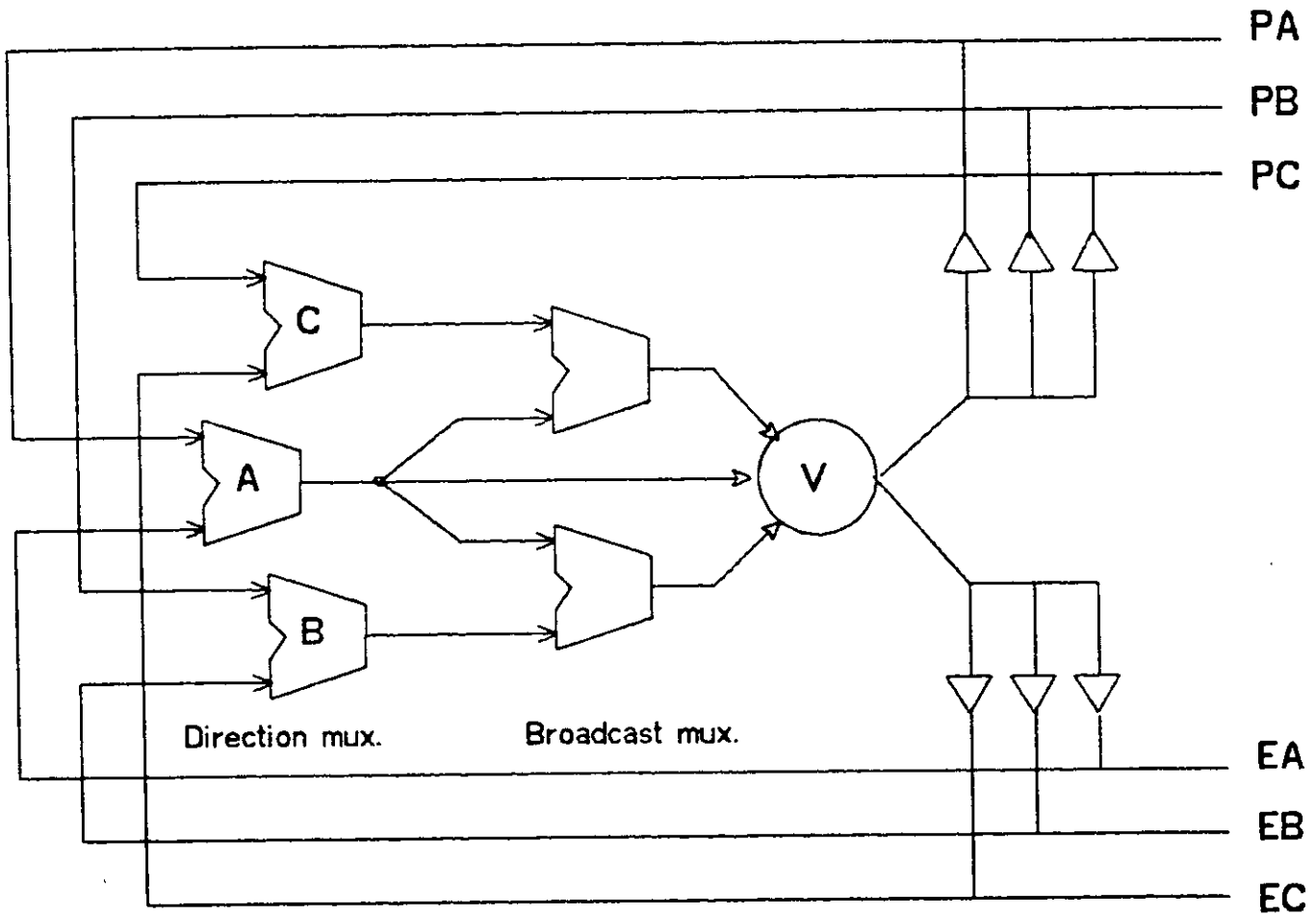
. Broadcast mode. Only the transmitting portion of bus AA is sampled and the content of bus AA is broadcast to the receiving portions of all three buses. This mode of operation is used for system initialization as well as allowing for selective triplication and non-triplication of I/O devices. This feature was added to provide reliability/cost tradeoffs so that a user can triplicate only those devices he deems necessary. The voter has no idea which devices are triplicated and which are not. The only requirement is that all non-triplicated devices be placed on bus AA. To handle non-triplicated devices an extra line is added to bus AA. Any non-triplicated device asserts this line during the addressing portion of the cycle to inform the voter to switch to broadcast mode.

* The processor samples external events, like REPLY, on the first phase of specific microinstructions (microinstructions take a multiple of four clock phases to execute) The voter blocks external signals that arrive within ten nanoseconds of the end of phase four. This insures that there will be no runt pulses [Chan72] with sufficient energy to allow some processors to recognize and others to miss the signal. Such divergence would quickly cause the processors to lose microinstruction synchronism.



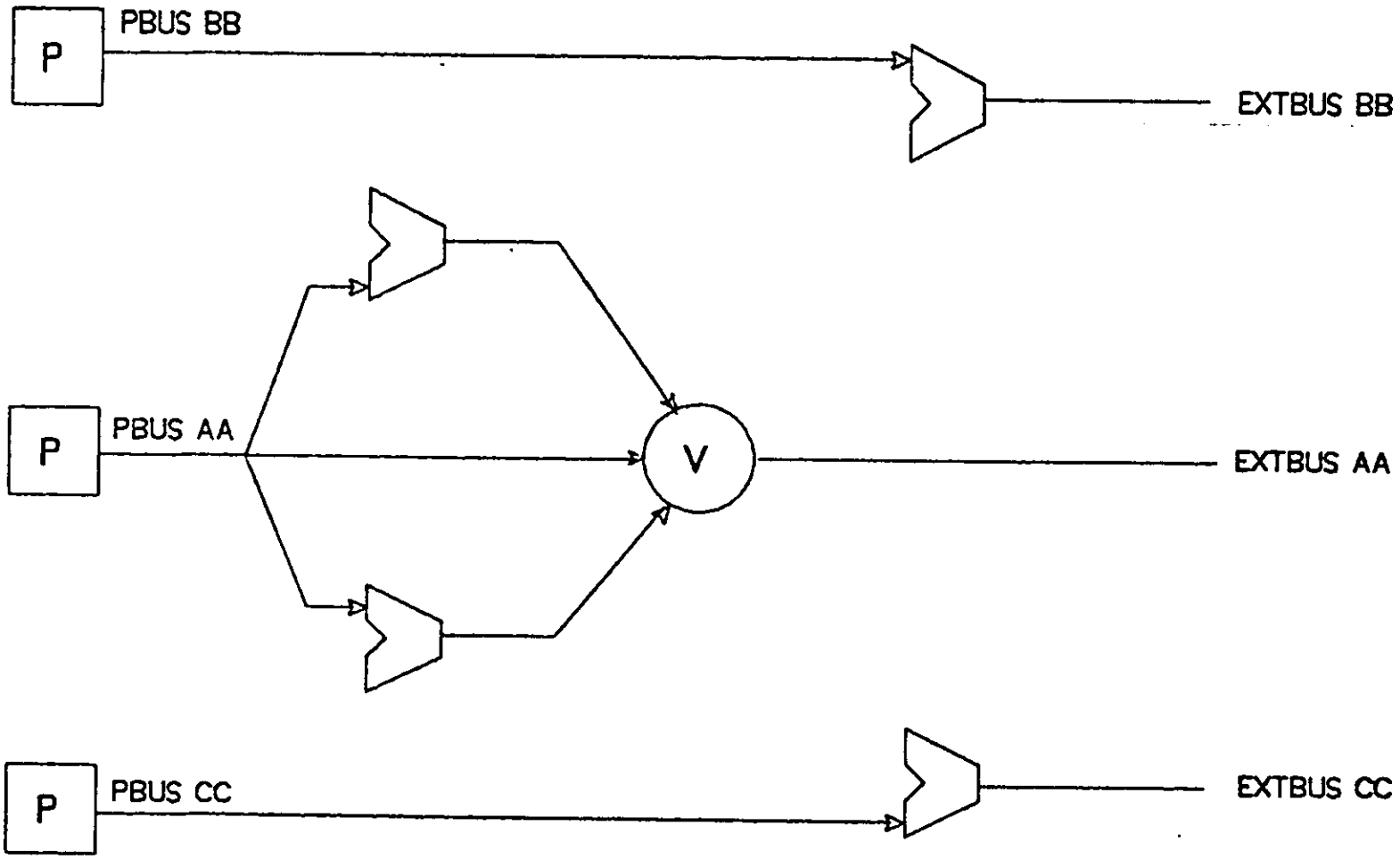
Unidirectional Voter data paths

FIG 8a.



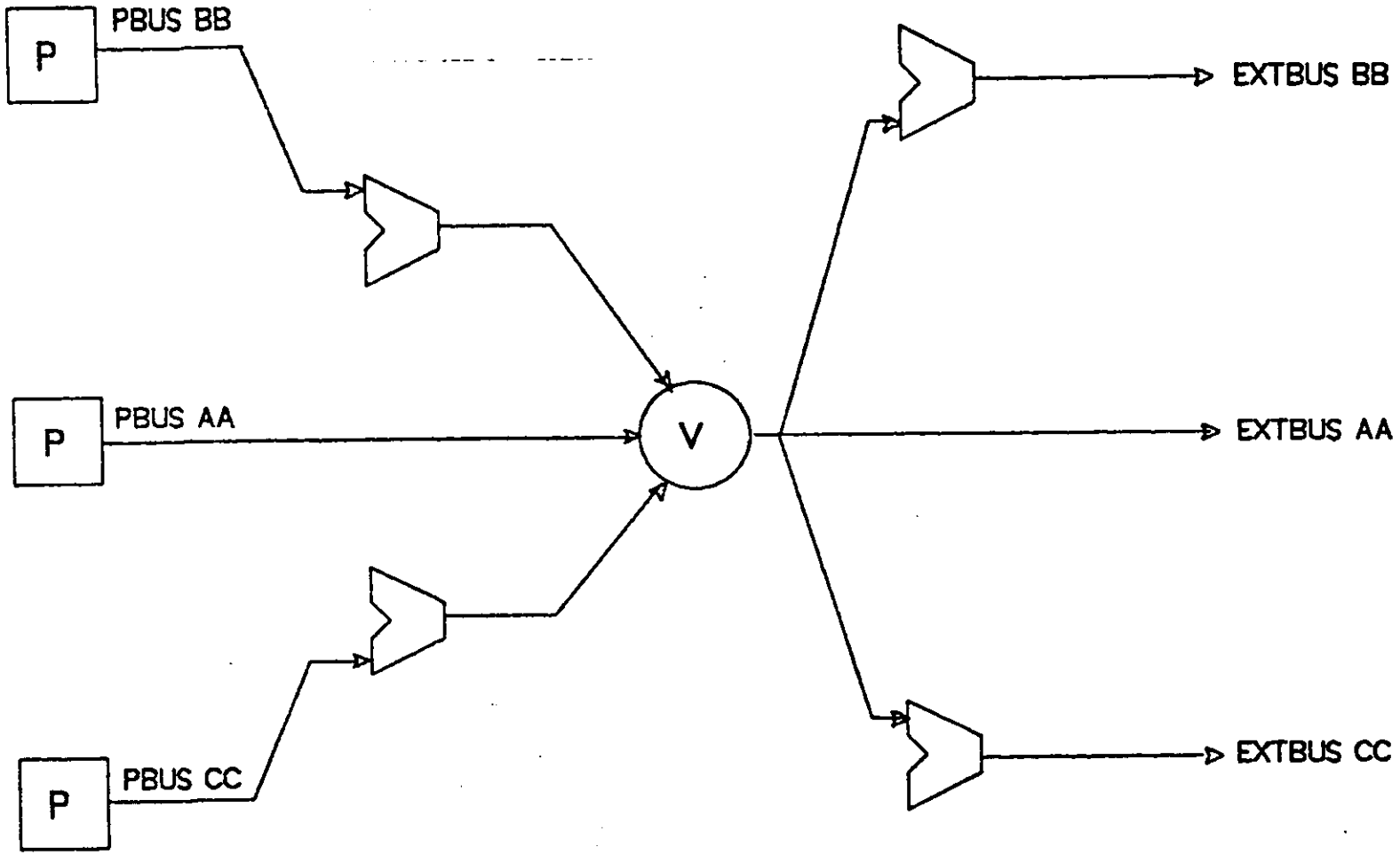
Bidirectional voter data paths

Figure 8b.



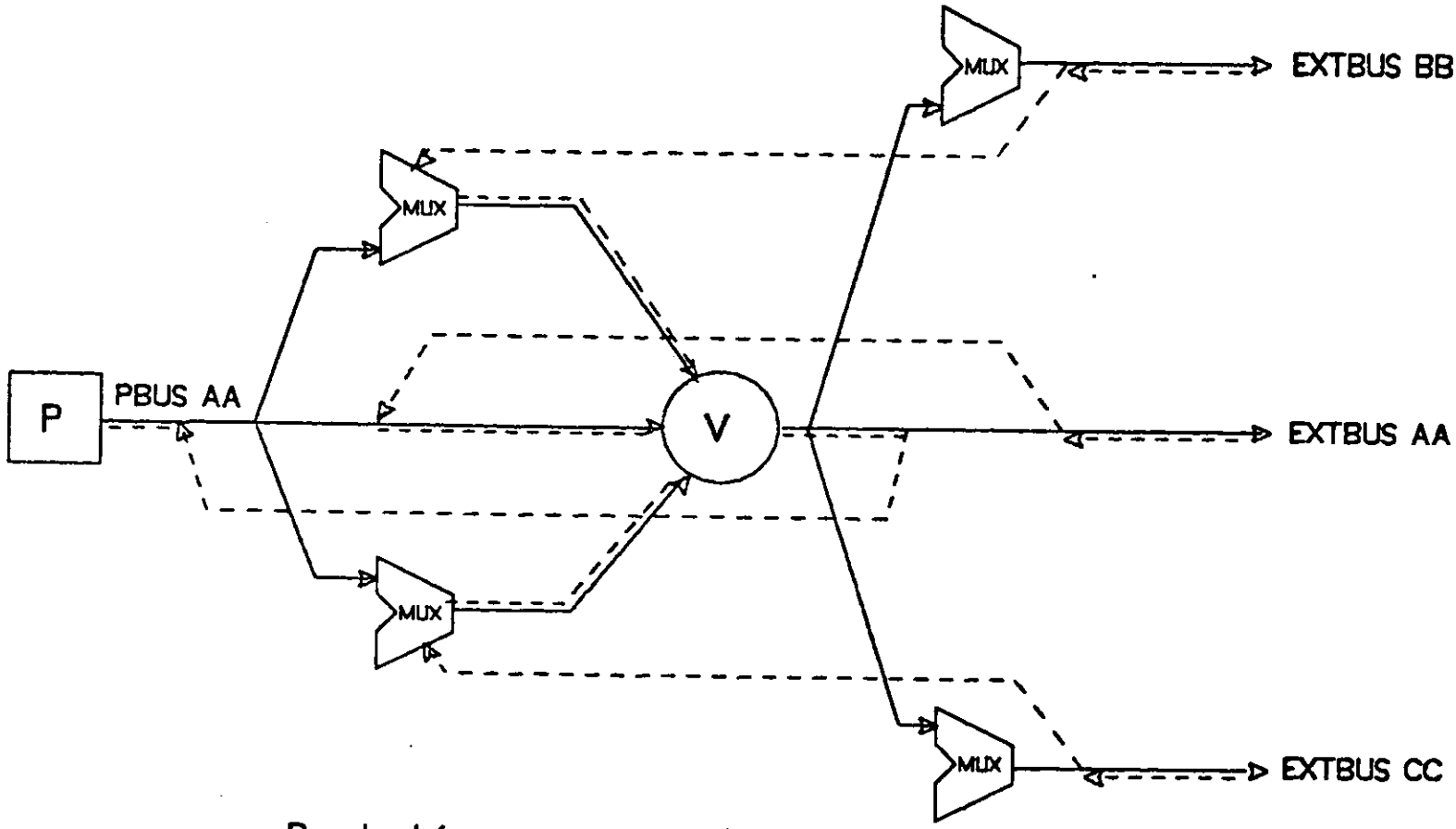
Unidirectional Voter Independent Mode

FIG 9



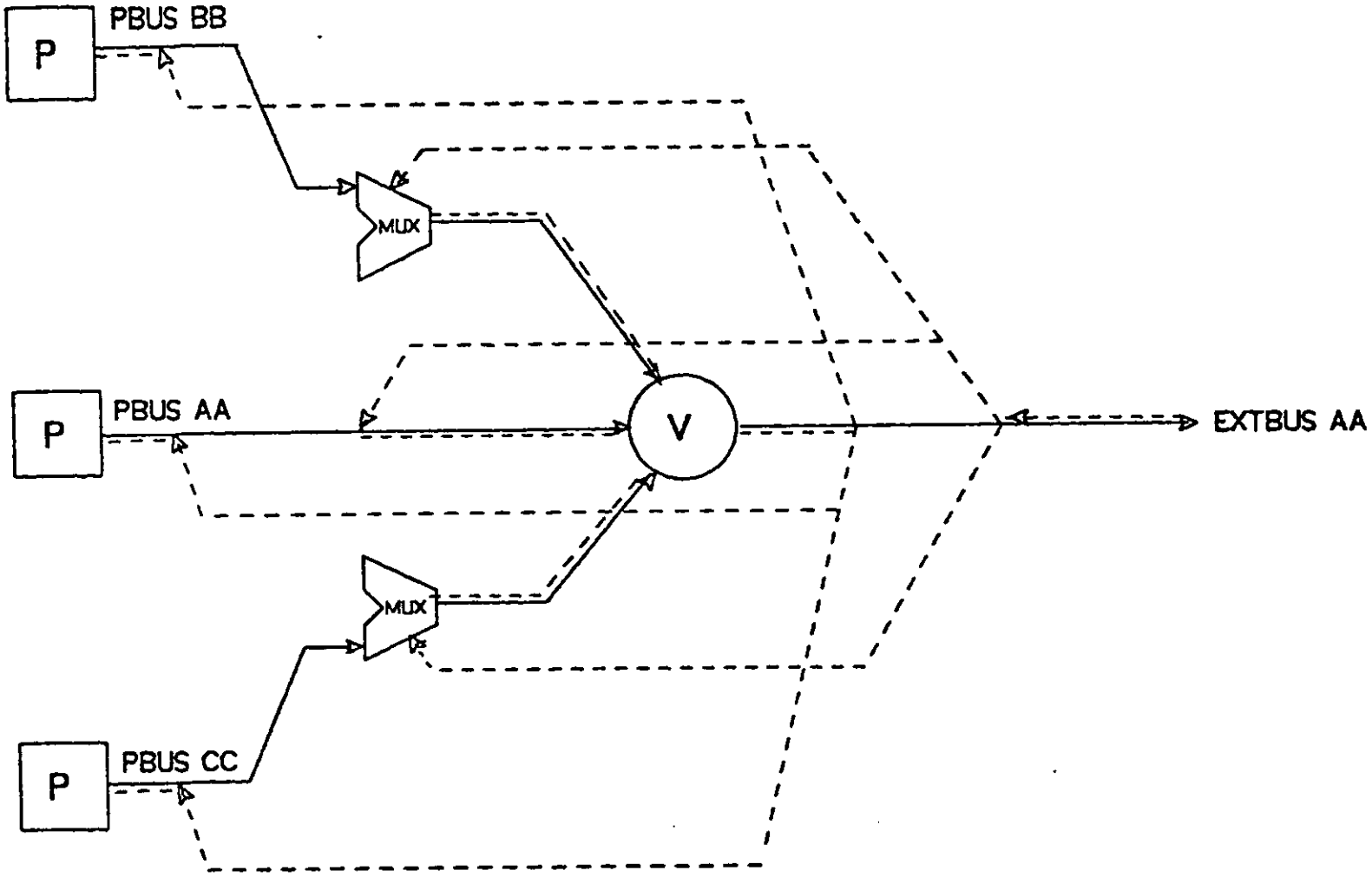
Unidirectional Voter Voting mode

FIG 10



Broadcast from one processor to three external buses

FIG 11a



Broadcast from a non-triplicated element to the three processors

FIG 11b

Architecture Extension

In most cases, triplicating a device just means plugging standard boards into the backplane, as is the case with memory. In some cases, however, the solution is not quite so simple. An example of a device that has to be somewhat modified is the RX01 FLOPPY DISK drive. The three FLOPPYs run asynchronously. Therefore there can be as much as a 360 degree phase difference in the diskettes. Since the information does not arrive under the read heads of the three FLOPPYs simultaneously, the obvious solution to this problem is to construct a buffer whose size is large enough to accomodate the size of the sectors being transferred. A disk read READ operation would then occur as follows [RXV1].

> .The track and sector number to be read are loaded into the three interfaces and the "READ" command is issued.
>
> .The three FLOPPYs load their respective buffers asynchronously.
>
> .The processors wait until the three buffers are loaded and then synchronously empty the buffers into memory.

A write operation would be executed in a similar fashion.

The main synchronization problem is to find out when all three FLOPPYs have completed their task or when one of the FLOPPYs is so out of specification that it can be considered failed. Once this is determined the "DONE" signals are transmitted to the three buses simultaneously.

There are times, in the case of a switchover from independent to voting mode, when the three processors must be able to communicate to each other. For this reason there are three full duplex single word transfer fully interlocked parallel interfaces in the system (labeled L in Figure 4b). The switchover protocol is implemented in four steps.

1. The processor requesting fault tolerance interrupts the other two via setting a bit in the appropriate parallel interface control register. All three load a fault tolerant program into memory, from the disk.

2. When each processor has finished loading the program it sets a bit in the Voter and enters a "wait-for interrupt-state". In a "wait-for-interrupt" state each processor relinquishes control of its bus.

3. The voter waits for all three processors to set the bit and then switches to voting mode. Switching to voting mode is accomplished by the circuit of Figure 7 (ie. a vote is taken after a specified time period if all three processors have not responded).

4. The voter, through the parallel interfaces, posts simultaneous interrupts to the three processors. Since the voter is in voting mode the following bus cycle will be voted upon. From then on the processors stay synchronized.

The process of leaving fault tolerant operation is simpler, because the processors are synchronized. A bit is set in the voter allowing it to switch to independent mode at the end of the current cycle. Operation then proceeds with three independent processors.

C.vmp is being built primarily for experiments on how computers behave in the presence of noise. A Statistics Board has been designed which straddles the three buses. On command from certain bus and voter signals, the statistics board latches and stores selected information from the bus. In addition, a unique time reference is also stored so that the collected data can be analyzed. By exposing certain sections of the C.vmp to a noisy environment, and protecting the rest of the machine, it is hoped that a model of how transient failures affect computers can be constructed.

The Transient Analysis Experiment

A search through the literature reveals little or no experimentation in the area of noninduced transient fault measurements. The main experiments that we hope to perform on this machine are the following.

The first experiment consists of exposing one of the external buses to a controlled noise environment, either directly coupled through the power supply, or radiated by a noise source. The rest of the computer would be kept in a shielded environment.

With the statistics board operating, we can find out how often we get a failure, where the failure is most likely to occur, and how long a failure lasts. By repeating the experiment with different noise frequencies and different noise intensities, we can map the noise susceptibility of components in the computer. By replacing components and repeating the experiment we can determine the variation in noise susceptibility as a function of component variation due to construction.

For C.vmp to prove successful, the smallest possible correlation between a component failing and a corresponding component failing at the same time is desirable, since these correlated failures cause a system failure. In theory, we would like to prove independence between failures in similar sections of the computer. Once we know the probability of a non-fatal failure, we can expose two sections of the system that perform the same task, and record fatal failures in the system. From the first experiment we hope to compute the mean and standard deviation of a non-fatal failure. From the second experiment we hope to compute the mean and standard deviation for a fatal failure. We can then measure the independence of two sections of the system to a noise source.

3. DESIGN OF BUS LEVEL VOTERS

From the experience gained in designing and implementing C.vmp some suggestions about fault tolerant bus protocol and voter design have developed. These are presented in the following subsections.

Synchronous versus Asynchronous Buses

The bus protocol issue is one of the most important things to settle before trying to build an effective fault tolerant computer. With different protocols various cost/reliability/performance tradeoffs can be obtained. A few commonly used protocols will be outlined and compared.

A computer with an asynchronous bus, and separate address and data lines, would yield a voter less complicated, but using more hardware than C.vmp. The voter would be less complicated because any delays in the voter itself could be ignored and no latching of information would be necessary. The extra hardware would be needed because separate voting elements would be required to vote on the address and data lines. The extra hardware would also make the voter less reliable. This bus protocol would therefore yield a higher performance, higher cost, lower reliability voter than C.vmp.

A computer employing a synchronous bus with separate data and address lines requires a slowing down of the processor clock to wait for the voter delay on the bus. The slower clock would then unnecessarily degrade processor performance during internal processor cycles. A fast clock with a longer waiting period during a bus access can be used, but this would require modifications to processor hardware and/or microcode. The full bus requires separate voting elements and a similar cost/performance/reliability tradeoffs as the asynchronous bus.

By multiplexing data and address lines in a fully asynchronous bus, considerable hardware savings can be realized because the same voting hardware can be used during the address and data portion of the cycle. The PDP-11 bus uses 56 bus lines while the LSI-11, with a multiplexed bus, needs only 36. This cuts the hardware requirements by nearly a third. This bus protocol would yield a faster, more reliable, lower cost voter than C.vmp. As a comparison with a full-width asynchronous bus this protocol would yield a more reliable less costly voter because less hardware would be needed. There would be a resulting performance degradation due to multiplexing. A synchronous multiplexed bus incurs the same problems as the full synchronous bus.

The bus that is by far the hardest to handle is a part synchronous/part asynchronous multiplexed bus. This is exactly what the LSI-11 has. Since the address is synchronous (and since we did not want to slow down the processor any more than necessary) it was decided to latch the address in the voter so that it can be held long enough to vote on it and send it through to the external side, and then open the latches so that the asynchronous portion of the cycle can take place. These observations led us to suggest the following fault tolerant design.

An efficient bus level voting architecture.

In defining a protocol suitable for a fault tolerant computer, two points should be kept in mind. First the bus should be kept as narrow as possible since the complexity of the voter increases, and hence the reliability decreases, with the number of data paths within it. Second, the bus should not be split to insert the voter. The ideal solution would be to have the voter as a guard, passively monitoring the three buses. When there is no disagreement, the bus runs at its normal speed (i.e. there is

no degradation due to signal buffering etc.) As soon as a fault is detected the voter would take control and provide the corrected information on the bus. The only problem with this protocol is that information gets to the voter no faster than it gets to all the other devices on the bus. It is impossible to vote and have the corrected information before other devices on the bus get it. The voter, once a mistake is detected, must suspend the bus master, gain control of the bus and broadcast the corrected information. This idea can be implemented using both synchronous and asynchronous buses whether or not they are multiplexed, but yields the best cost/reliability/speed performance for a multiplexed bus. For an asynchronous, multiplexed bus the protocol could be implemented as follows.

The proposed protocol does not split the bus. The bus lines needed for a memory or I/O reference are:

16 bus Data/Address lines: BDAL<15:0>
Master Synch Line: MSYS
Data in Line: DATI
Data out Line: DATO
Processor Synchronization Line: VSYNC
Bus Master Hold Line: BHOLD

If there is no error, or if there is no voter, the protocol for a single processor-memory pair would proceed as follows for a DATA IN bus cycle:

.The Bus Master asserts the address.

.After the address has settled MSYS is activated.

.The Slave receives and decodes the address. The address is removed by the Bus Master and DATI is asserted.

.The Slave puts data on the bus and the Master retrieves it.

.The Bus Master terminates DATI and MSYS.

.The bus cycle terminates.

The voter controls the Bus Master using the VSYNC and BHOLD lines. VSYNC is needed to keep the processors synchronized. It is issued by the voter at the end of each bus cycle. BHOLD is issued by the voter in response to a MSYS, DATI, DATO if a disagreement has been generated on the bus. The Bus Master responds by releasing control of the bus and entering a "wait" state until BHOLD is disabled by the voter. The voter, once a disagreement has been detected, becomes bus Master and broadcasts out the voted information back on the bus. The receiving device can then latch the information a second time. The only critical delays are the times between an MSYS and DATI, MSYS and DATO and the time a Slave responds to a DATI and the time MSYS terminates. The time between these signals has to be sufficiently long so that the voter has time to assert BHOLD if necessary. After the voter has finished the broadcast operation BHOLD is released and the processors can continue with the next step in the bus cycle. Operation in independent mode is achieved by disabling BHOLD and enabling VSYNC. In this fashion the bus master cannot be stopped and will not be forced in synchrony with the others. Another major advantage is that no data paths cross the voter. This makes the debugging of the voter easier since a complete working voter is not required to operate the multiprocessor.

Voter Simplifications.

One of the major problems with splitting the bus to insert the voter is that the memory on the LSI-11 card can no longer be used because it is on the wrong side of the voter. Since this board space can no longer be used for memory, it can be used for the voter. If the voter can be made compact enough, and if the size of the processor board could be made HEX instead of QUAD*, then the voter could be fitted

* HEX: board containing 108 16 pin ICs and having 216 edge connectors.

QUAD: board containing 72 16 pin ICs and having 114 edge connectors.

directly on the processor board. To triplicate the voter, and preserve operation in independent and broadcast mode, a total of 300 chips would be needed. By customizing some of the circuitry, however, large hardware savings can be achieved. One design involves the customization of two chips.

1. A four bit multiplexer with output latches and input bus receivers.

2. A pair of four bit multiplexers with four one bit voters, disagreement detectors and bus drivers.

This design would reduce the number of ICs for a triplicated voter to 195.

A second, alternative design, involves putting a four bit bidirectional voter and disagreement detectors and part of the control logic in a 40 pin package. This would reduce a triplicated voter to 30 chips. Table 1 summarizes these findings.

Table 1. Chip count for different implementations

	Single Voter	Triplicated Voter
Current design	250	300
Current design on processor card	84*3	100*3
Integration level 1	60*3	63*3
Integration level 2	10	10*3

4. C.VMP IMPLEMENTATION EXPERIENCE

Hardware implementation and modifications

Figure 8a. shows a single unidirectional line in the voter, such as an interrupt line. The multiplexer and output gating select the operating mode. In Figure 8b., a bidirectional data line is shown. A second set of multiplexers is added to select the direction of voting (toward or away from the bus master). The propagation delay of the voter necessitates latching the address, which extends the address time on the bus. Error detection hardware (not shown) compares each input with the output of the voting element for use in gathering statistics on bus errors.

Since one of the goals of the project is to construct a fault tolerant computer from standard components, few changes have been made to most boards. The processor clocks were combined into a single clock in the initial design, but this has been rejected in favor of having three clocks with a common reset line for synchronizing. Eventually, a true fault tolerant clock (as in [DalyW]) will be implemented.

The Fault Tolerant Computer is assembled in a standard DEC cabinet. It is composed of the following parts:

1) Power Switch
2) An H720 DEC power supply rated at 20a. @+5v,7a@+21v, 7a.@-21v
3) 3 Backplanes
4) 3 RX01 FLOPPY DISK DRIVES
5) 3 LSI-11 Processors
6) 1 Voter
7) 36K of semiconductor dynamic memory (3 sets of 12K)
8) 6 Parallel Line Units (PLU)
9) 1 Serial Line Unit
10) 3 RX01 interfaces
11) 3 +12v regulators

Bus, Processor, Voter and Peripherals modifications

Processor modifications were kept to a minimum and amounted to the addition of a few wires and the breaking of a few etch points. No new circuits have been added. The clock on processor AA (PAA) is being used as the Master Clock.

The following modifications are required on PAA.

1) 4 wires to feed the four clock phases to the Voter.

2) 1 wire to generate the memory refresh clock on all three processors.

3) 1 wire to feed the clock to processors BB and CC.

4) 2 wires to synchronize the clocks on processors BB and CC

Processors BB and CC (PBB, PCC) require the following modifications.

1) 1 wire to receive the memory refresh clock.

2) 1 wire to receive the Master clock.

3) 2 wires to receive the clock synchronization signals

The following modifications must be made in the disk interface to achieve operation in fault tolerant mode. Two bits are used by the disk to communicate with the processor [RXV1]. They are TR and DONE. TR specifies that the disk interface needs or has available a byte of information. DONE specifies that a disk operation has been accomplished. To operate in fault tolerant mode these bits have to appear together on the three buses. Therefore some circuit must wait until TR or DONE has appeared from all three disk controllers before releasing TR or DONE on the buses. This circuit must also sense when one disk has failed and continue operation without it. A circuit very similar in concept to that of the REPLY voter (Figure 7) is implemented for the disk.

The Serial Line Unit (SLU), or teletype interface has been our example of a non-triplicated device on the bus. To operate in a non-triplicated mode the SLU must

communicate with the voter through the Non Triplicated Element Line (NTEL). The SLU asserts this line whenever its address is on the bus and holds the line asserted for the whole bus cycle.

The parallel interfaces (PLU) have not at present been installed. However, no known modifications are required for triplicated operation.

The memory needs no modifications to operate in triplicated mode.

The voter is the only totally custom designed circuit in the C.vmp. It is composed of three HEX sized boards called VAA, VBB and VCC. VAA and VCC contain most of the data paths, while VBB contains most of the control logic.

Triplicated Power Supplies

The three buses need independent power supplies so that any section of the system can be powered down for maintenance or repair without having to halt the whole computer. Independent power supplies also reduce the chance of noise crosscoupling on the three buses. In our case we decided not to triplicate the power supplies. Only the +12v regulators have been triplicated. The +5v and the +12v lines leading to the three buses have been provided with separate on-off switches.

The Statistics Board

The Statistics board has been implemented as follows. There are 48x1k shift registers. Twelve are allocated to each bus. Twelve are used to store a unique time reference. For each bus, four shift register positions are allocated for the SYNCH, REPLY, DIN and DOUT bus signals and eight are allocated for half of the data/address lines. The bottom or the top half of the data/address lines can be chosen under manual or program control. The processors can communicate with the Statistics board in Broadcast mode through the bus as a regular I/O device. The Statistics Board is

capable of posting interrupts to the processor. The Statistics board receives the output of the disagreement detectors and certain control lines directly from the voter. Based on manually presettable switches the Statistics board can store disagreements in conjunction with the occurrence of SYNCH, DOUT and REPLY. These signals mark that there is useful information on the bus.

A twelve bit counter counts bus cycles. The value of the count is stored with every disagreement providing a unique time signature to the occurrence of a fault. Every time the counter completes one full cycle an empty frame is stored in the shift registers to provide further timing information. When the storage capacity of the Statistics board has been exceeded an interrupt to the processor is posted so that information can be transferred from the shift registers to the disk where it can later be retreived for data analysis (See Section 4). However at any time the processor can reset, stop, or empty a partially filled Statistics board.

Debugging Experience

Fault tolerant computers cannot be debugged and tested as regular computers. The computer can be executing correct instructions and performing a required task and still some of its components might not be functioning. In some designs, where three processors stay synchronized by virtue of some external timing signal which depends on the system being operational, the initial phase to bring the the system to an operating state can be tedious and lengthy. It is important that some features that could ease the debugging effort be designed into the computer. The use of off-the-shelf components with a minimum of modifications helps to reduce the amount of hardware to build and test. However careful attention should be paid to the kind of hardware used. For example, the use of dynamic memory is not recommended during

the debugging phase if the processor microcode is in charge of doing the refresh. If a glitch causes a failure, the content of memory is lost. In our case the capability to operate the three computers in independent mode helped considerably. Almost 90% of the data paths and virtually all the control logic could be tested with a single processor operating at a time. Once the three processors work independently, the task of synchronizing the three computers is greatly simplified.

Power-up, bootstrap sequence and diagnostics

RT-11 is the operating system currently used. It is stored on Floppy disks. The Serial Line Unit that normally communicates with the console teletypewriter is connected, through a high speed link, to the HOST computer for Cm* [SwanR75]. The HOST is a DEC PDP 11/10 computer and has a link connecting it to the Front End computer which services the PDP-10 (Figure 4a). The use of the HOST and the Front End computer make down line loading and bootstrapping from files on the PDP-10 quite easy. The LSI-11 lacks a front console with the lights and switches. Instead microcode has been provided to execute ODT with the teletypewriter. A paper tape bootstrap loader has also been provided in the microcode. These facilities remove the inconvenience of having to hand toggle bootstrap programs into C.vmp. Bootstrapping is done in the following way. The computer is initialized in fault tolerant mode by manually generating a single initialization signal to the three processors. This brings C.vmp up in voting mode executing console ODT to the SLU. The single SLU is now capable of broadcasting to the three processors. A link from the HOST, through the Front End, to the PDP-10 is opened to permit file transfer for down line loading C.vmp. Using a Front End connected terminal we then log on the HOST and request the use of C.vmp (called Computer Module Fault Tolerant (CMF1) in Cm* notation). From the video

terminal we are then able to communicate to CMF1 through ODT. Using this mechanism programs can be hand typed in the machine and executed directly. However using the BLISS11 compiler and the MACN11 assembler, programs can be written and assembled on the PDP-10. A binary file is transmitted to the HOST and loaded in CMF1. The HOST returns control of CMF1 to the terminal in ODT and we proceed from there. The program loaded is usually the floppy disk bootstrap routine which is used to load the RT-11 operating system.

Diagnostics for C.vmp

A fault tolerant computer is designed to operate even if certain sections of the machine are failed. Therefore standard diagnostic methods cannot be used. If a computer is capable of operating only in fault tolerant mode then the only method to run diagnostics is to disconnect certain sections of the system until the computer is no longer fault tolerant. Diagnostics can then be run on sections of the machine at one time. If, as is the case for C.vmp the three processors can operate in independent mode then the diagnostic process, although more complicated than for a regular computer can, in part, be automated. The basic idea is that while the three processors are operating independently each machine can check itself through regular diagnostic routines and can then check the other two machines through the parallel interfaces. This set of diagnostics tests most of the machine. Some of the voter data paths (see Figure 8) that are not tested by operation in independent mode can be tested by selectively powering down that section of the system.

Software protocol for switch between independent and voting mode.

C.vmp would, under normal applications, spend most of its time in independent mode thus maximizing the performance capability of the system. The three processors

would be executing three totally unrelated tasks, with the possible exeption of having to share some peripherals, or could be working on different section of the same task. This is the multiprocessor environment. A typical example would be the onboard computer of a space vehicle where each computer would be assigned a separate task during noncritical portions of the flight. When the need arises, however, the three processors must be synchronized in a short time to execute a task in fault tolerant mode. This case might arise during the calculation of a critical burn, where an error might lead to a catastrophe during the subsequent burn, and during the burn itself when the lack of real time response can be just as fateful. The switchover protocol always terminates with the steps describes in Section 2. We get there as follows. Assume that the three computers are executing separate tasks and that one of the processors requests a switch to fault tolerant mode.

1. The requesting processor interrupts the other two and through some handshaking sequence requests the switchover. At this point the other two processors should make sure of the validity of the request before blindly accepting it.

2. When the validity of the request is proven the three processors should determine system resources.

3. The sequence in Section 2 (page 11) is performed.

5. RELIABILITY MODELS

There are two ways to improve the hardware reliability of a computer: improve the component reliabilities, or employ redundancy. The first method has been effectively applied in complete screening and testing programs. But there is a limit, set by the law of diminishing returns, beyond which such methods cannot be economically justified. Redundant design, when coupled with the first technique, can give improvement in reliability otherwise impossible to obtain.

In any redundant system, results must be obtained by taking a weighted sum of the outputs of each replicated portion.

$$Y = a_1Y_1 + a_2Y_2 \ldots + a_nY_n$$

where a_1 is the weight of output Y_1. When equal weights are given to all outputs, the method is called voting. Each weight is thus:

$$a = n^{-1}$$

In order to break ties, n must be an odd number. Therefore, n=3 is the least non-trivial voting configuration.

The size of the portion which is replicated determines the type of voting which takes place. At one extreme, the entire computer can be reproduced, requiring only "process-level" voting. This places the burden of voting on the software designer, since convenient points for saving or comparing results must be inserted in all code. At the other extreme, voting on the component level would require immense effort on the part of the hardware designer, as well as an inordinate number of voting devices [LyonW]. Replication at the board level facilitates maintenance, but would require a different voter design for the outputs of each different type of board.

Ideally, the voter should be placed at a level where all parts of the system meet in common. With a computer based on a general bus structure, the choice is obvious. A bus-level voter will observe all system-wide transactions in a computer without requiring either extensive redesign or software changes. When the bus protocol is the master/slave type, one voter is required for each bus master. This ensures that every bus transaction will be voted [SiewD76].

This section presents a reliability analysis of C.vmp for both permanent and transient faults. The result is compared with an equivalent non-redundant computer, as well as the effect of the redundant architecture on performance.

Permanent fault models

To calculate the reliability of C.vmp, we assume a hardware configuration which consists of three processors, 28K of memory per processor, the voter, the console serial interface (SLU) and appropriate power supplies.

The reliability calculated for permanent (stuck-at) faults is based on a parts count model of the system [Mil74]. For each component, a failure rate is calculated. Reliability is assumed to be exponential with failure rate and time, in the form:

$$R = \exp(- L * T)$$

where L is failure rate in failures per million hours, and T is time in millions of hours.

The reliability of a non-redundant module is the product of its component reliabilities, which is found by summing the failure rates. The total system reliability (redundant or non-redundant) is the sum of the reliabilities for each correctly operating state.

Non-redundant

The non-redundant system has a failure rate found by simple summation of the individual failure rates:

$$Rnon = \exp(- (Lp + Lm + Ls + Lw) * T)$$

Substituting the failure rates from Table 2 yields:

$$Rnon = \exp(-844.1 * T)$$

Table 2. Failure rates for system components

Lp	= 383.4	LSI-11 processor module.
Lm	= 61.1	Memory module (4K semiconductor RAM)
Ls	= 4.4	Console terminal serial interface
Lw	= 25.0	Master power supply
Lr	= 3.6	Replicated section of power supply
Lvp	= .8	Triplicated portion of voter on the processor bus
Lvm	= .9	Triplicated portion of voter on the external bus
Lv	= 3.1	Triplicated portion of voter on both buses
Ln	= 3.7	Non-triplicated portion of voter

Triplicated

The reliability of the triplicated system shown in Figure 4b is found by summing the reliabilities for all states in which the system is correctly operating. Each state is a combination of working and failed units.

The various parts are:

Rp = reliability of processor bus elements
$= \exp(-(Lp + Lvp + Lr) * T)$
$= \exp(-387.8 * T)$

Rm = reliability of a single 4K memory module
$= \exp(-Lm * T)$
$= \exp(-61.1 * T)$

Rv = reliability of the triplicated part of the voter
$= \exp(-Lv * T)$
$= \exp(-3.1 * T)$

Rn = reliability of the non-triplicated part of the voter
= exp(-(Ln + Lr + Lw + Ls) * T)
= exp(-36.7 * T)

Re = reliability of the part of the voter associated with the external bus
= exp(-(Lvm + Lr) * T)
= exp(4.5 * T)

The reliabilities for each operational state are:

1) At most one processor failed, at most one memory module per 4K address range failed, voter and buses all working.

$$R1 = (3Rp^2 - 2Rp^3) * (3Rm^2 - 2Rm^3)^7 * Rv^3 * Rn * Re^3$$

2) At most one processor failed, single memory bus failed, voter and all memory on the other two buses working.

$$R2 = 2 * (3Rp^2 - 2Rp^3) * Rm^{14} * Rv^3 * Rn * Re^2(1 - Re)$$

3) One third of voter failed, all processors and memories on the other two buses working.

$$R3 = 2 * Rp^2 * Rm^{14} * Rv^2(1 - Rv) * Rn * Re^2$$

The coefficient of two in 2) and 3) represents the two possible configurations for this error. If the error were to occur on the third bus (bus A), then the system would be considered failed, since the console could no longer be accessed.

Note that in the last case, the failure of a third of the voter masks the operation of one processor-memory pair. Since it no longer matters whether that pair operates, the Rp and Re terms are only squared.

When added together, the triplicated reliability reduces to:

$$Rt = (3*Rm^2 - 2Rm^3)^7 * Rv^3 * Re^3 * Rn * (3Rp^2 - 2Rp^3) + (Rm^7 * Rv * Re * Rp)^2 * Rn * (Rv(4 - 6Re - 4Rp + 4Re*Rp) + 2)$$

Substituting the values from Table 2 yields:

$$Rt = (3 \exp(-122.2 * T) - 2 \exp(-183.3 * T))^7 * (3 \exp(-835.1 * T) - 2 \exp(-1222.9 * T)) + 4 \exp(-1649.3 * T) - 6 \exp(-1653.8 * T)$$

$$- 4 \exp(-2037.1 * T) + 4 \exp(-2041.6 * T) + 2 \exp(-1646.2 * T)$$

Note that triplicating all elements of the voter would give theoretically complete single fault tolerance, but would have little effect on the actual system reliability, due to the small size of the failure rates Lv and Lm.

Independent mode

In independent mode, the reliability of a single processing unit is similar to the non-redundant system, with the addition of the voter failure rate.

$$Rind = \exp(-(Lp + (Lvp + Lv + Lvm + Ln) + 7Lm + Ls + Lw + 3Lr) * T)$$

this yields:

$$Rind = \exp(-859.8 * T)$$

Broadcast mode

In broadcast mode, the processor side (Rps) is triplicated but the external side (Res) is not. This gives a reliability of:

$$Rbrd = (3Rps^2 - 2Rps^3) * Res$$

where the reliability of the processor side is:

$$Rps = \exp(-(Lp + Lvp + Lv + Lr) * T)$$

and the external side (including the non-triplicated parts of the voter) has a reliability:

$$Res = \exp(-(Ln + Lvm + Lm17 + Ls + 2Lr + Lw) * T)$$

When the numbers are substituted, this reduces to:

$$Rbrd = 3 \exp(-1250.7 * T) - 2 \exp(-1641.6 * T)$$

Figure 12 shows a graph of reliability vs. time for each of the above models.

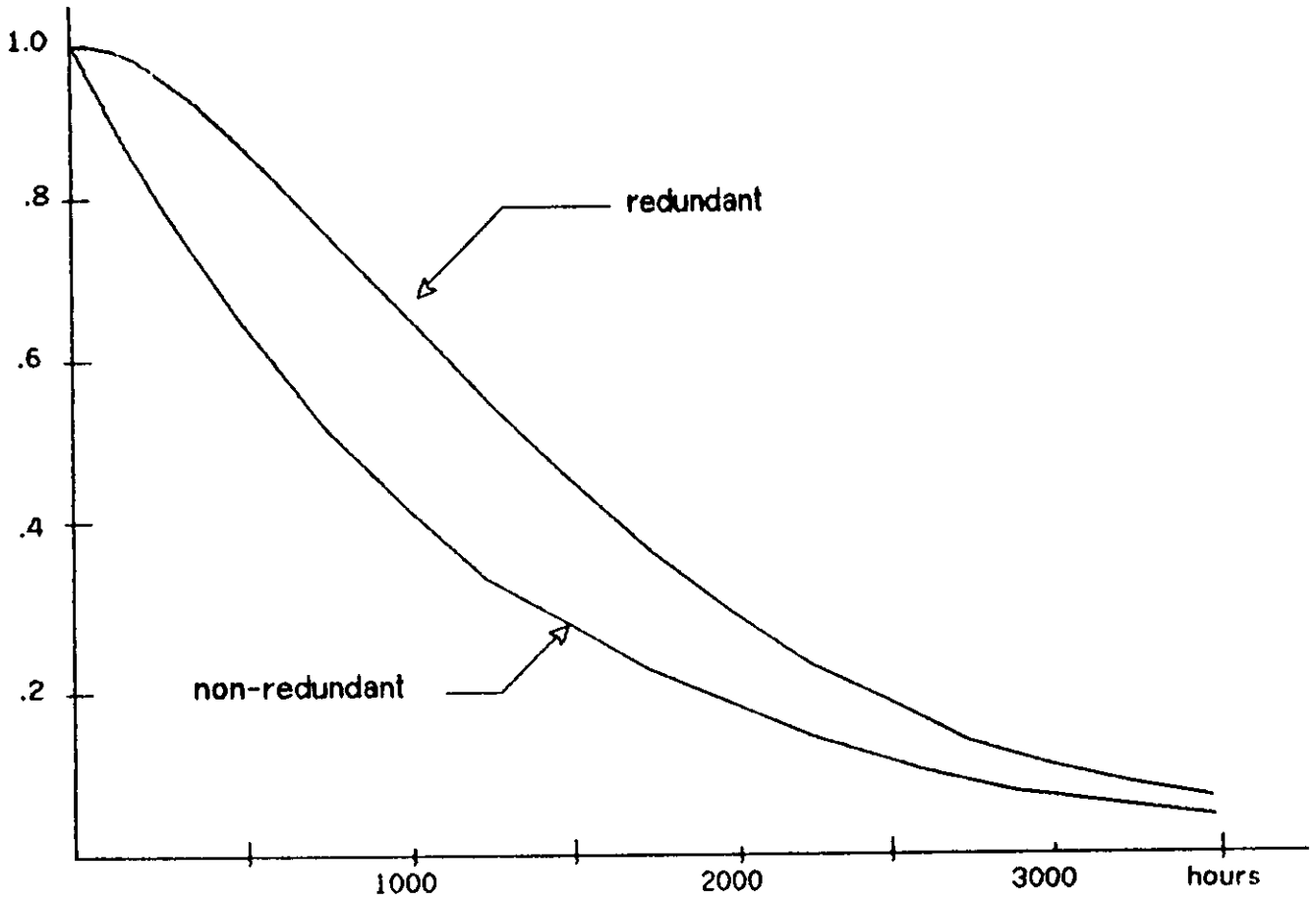


Figure 12. Permanent fault reliability versus time.

The reliability of C.vmp in independent mode is indistinguishable from that of the non-redundant LSI-11. Table 3 shows the mission time improvement factors derived from the graph.

Table 3. Mission time improvement.
Hours of operation above reliability

Rel.	LSI-11	C.vmp	MTI
.99	12	103	8.6
.95	60	286	4.7
.90	125	440	3.5
.80	264	694	2.6

Transient models

To model as system for transient faults, assumptions must be made about the form that transients will take. The complexity of the system is reduced by assuming that transients take the form of noise signals added to the bus signals. Since it is also assumed that bus noise affects all modules connected to that bus, no generality is lost by ignoring transients which are local to a board.

Bus noise can be characterized by two parameters. The probability that the noise causes the bus line to be incorrectly read by the receiving device is F. The duration (in bus cycles) of a transient is N. The measure used for reliability will be the probability that the system survives a given transient.

Faults are assumed to be signal independent (that is, the fault probability is not affected by the value of the signal). Faults are also assumed to be well-behaved [SiewD75]. A well-behaved fault will affect all points on the bus identically.

Non-redundant system reliability

In the non-triplicated system, any bus fault will cause a system failure. Since the LSI-11 uses 32 bus lines (not counting four spares), the probability of successful operation for a bus cycle during a transient is:

$$(1 - F)^{32}$$

For the duration of the transient, this becomes:

$$(1 - F)^{32N}$$

Redundant system

The redundant system can be modeled in more than one way, depending on what we believe to be the dominant failure mode of the system. The first model considers the accumulation of disagreements in memory during the course of a transient [SiewD75]. The second model is addressed to loss of synchronization between processors [WakeJ75]. The section concludes with alternative methods for improvement of reliability, depending on the outcome of the transient analysis experiments.

Redundant system reliability - memory model

The triplicated system fails if any two control lines fail:

$$(3(1 - F)^2 - 2(1 - F)^3)^{16}$$

Failure also occurs if two data lines fail. However, data line failures have memory (an accumulative effect) in that a single failure on a write to a memory location, combined with a single failure on a read to the same location, will cause a system error. Assume that this accumulative effect only holds for the duration of the transient (i.e., that

transients are far enough apart that any wrong memory cell will be correctly rewritten, thus erasing the error, before the next transient occurs). Furthur assume a decaying model of program behavior in memory references. The probability of accessing the same location on the i^{th} bus cycle after the initial access is e^{-i}. Hence, a word written to memory has a probability of e^{-i} of being read during the same transient, and thus being vulnerable to failure. This exponential model of program behavior is intuitive, and does not, of course, preclude the use of other models.

for the data lines:

$$((3 S^2 - 2 S^3)16)^N$$

where the term for the accumulative effect of memory errors is:

$$S = (1 - \sum e^{-i}F)$$

Recalling that:

$$\sum e^{-i} = (1 - e^{-N}) * (e / (e-1))$$

yields the total equation for the reliability:

$$Rt = ((3(1-F)^2 - 2(1-F)^3) * (3(1 - F(1-e^{-N})(e/e-1))^2 - 2(1 - Fe(1-e^{-N})(e/e-1))^3)^{16N}$$

Table 4 lists the reliabilities for some typical values of F and N. Note that the accumulative effect of memory errors does not severely degrade reliability, even for extremely long transients.

Table 4. Comparison of transient fault reliabilities

	C.vmp		Non-redundant	
F =	.001	.00001	.001	.00001
N				
10	.998	1.000	.726	.997
100	.979	1.000	.041	.968
1000	.805	1.000	10^{-14}	.725
10000	.115	1.000	0	.040

Redundant system reliability - synchronization model

A single fault will fail the redundant system only if voting does not cause the system to recover before a second error. If a processor receives an incorrect instruction, it may lose step with the other two in their microcode sequence. When this occurs, the processor will either halt due to bus errors, or execute improper code until it randomly falls into synchronization with the other two.

The probability of an error occurring during a transient is:

$$1 - (1 - F)^{32}$$

The probability that the cycle in which the error occurs is an instruction fetch is:

$$(1 - (1 - F)^{32}) * I$$

Finally, the probability that the error will cause the processor to halt is:

$$p2 = (1 - (1 - F)^{32}) * I * Df$$

A count of the types of PDP-11 instructions and bus cycles in a sample of code gives a raw estimate of I and Df (see Appendix A). When this is compared to a count for repeatedly executed code (code appearing in loops) in order to weight the numbers for frequency of execution, (rather than just frequency of appearance in a

program), the two counts agree closely, and no weighting is required. The estimated values are .53 for I and .41 for Df.

Garbled data, and some erroneous instructions, will not cause the processor to fail, but the registers will contain incorrect data. Due to the voter and the presence of two correct processors, the incorrect processor will continue to execute the same instruction sequence as the others. This will eventually cause the registers to be corrected. The recovery rate is an exponential term, which decays over the course of 10 to 50 instructions due to register lifetime [LundA74]. The probability of this type of error occurring is similar to the above, but the multiplier is (1 - I*Df) to give the probability of not halting on an error.

This probability is:

$$p1 = (1 - (1 - F)^{32}) * (1 - I * Df)$$

Hence, there are two system failure modes. A processor can halt when it receives an incorrect instruction, or it can have a temporary failure from which it will recover within a few cycles. The system will fail whenever two processors have simultaneously failed.

We can represent the system as being in one of four states:

S0 - all processors operating.
S1 - a single processor has a temporary failure.
S2 - a single processor has halted.
S3 - system has failed.

A state matrix M can be made of the conditional probabilities of a transition between any two states. The reliability of the system is the probability that the system is not in state S3 at the end of N bus cycles. A vector of state probabilities after N cycles can be found from:

$$P(N) = S0 * M^N$$

Although no solution can be found in closed form, we can compare the two systems for particular values of F and N (Table 5). The recovery rate from temporary failures is assumed to be .03, which gives a mean recovery time of about 33 bus cycles.

Table 5. Comparison of transient reliabilities

F	N	$(1 - F)^{32N}$	$(1 - P_3(N))$
.001	10	.726	.839
.00001	10	.997	1.000
.001	100	.041	.020
.00001	100	.968	.999

Alternatives

It has been shown that accumulative memory errors will not adversely affect the reliability of the system. The danger of a processor becoming unsynchronized is more acute. Altering the microcode of the LSI-11 can, when combined with appropriate software, eliminate the possibilities of halts and temporary losses of a processor. First, the HALT instruction and all bus errors should trap, instead of entering ODT (the console service routine). A trap routine would resynchronize the processor which has halted. If an actual halt state is needed, the branch instruction would serve (octal '000777). This is an infinite loop consisting of one instruction. A console switch connected to the HALT lines of the three microprocessors can be used to enter ODT when necessary (for maintenance, or cold starts etc.), but it should never be possible to enter ODT under program control, since it is not possible to get out

without external assistance. Additional hardware may be needed on each processor board to sense the beginning of the voted instruction cycle for use in resynchronizing.

Systems employing triplication are characterized by an initially high reliability due to the probability of all three sections being operative. This is paid for by a lower reliability later in the life of the sytem when the probability of a module failure has increased. To restore the system to its original reliability, periodic maintenance should be employed. This technique is greatly facilitated by the ability to power down sections of C.vmp without interrupting operation in the other two sections.

6. PERFORMANCE MODELS

Performance degradation is calculated for two activities: the memory cycle time, and the disk access time.

Memory cycle time

Two properties of the voter cause an increase in bus cycle time over the non-redundant system. The propagation delay of the voter requires some of the control signals to be delayed to insure correct data reception. Secondly, the voter prevents the processors from losing synchronization by latching inbound control signals (e.g. reply line and interrupt requests) during the clock phase in which they are sampled by the processors. This insures that all three processors will see the reply during the same clock phase, but also causes receipt of these signals to occasionally slip by one clock cycle (400 ns.). Since the window during which the signals are latched lasts 40 ns. out of every bus cycle, the average degradation per cycle is: .5 * (40/400) * 400 = 20 ns.

The address part of the bus cycle is lengthened 200 ns. by the delay on the SYNC control line. This is the sum of the voter propagation delay, the bus settling time and the worst-case processor clock skew (assumed to be half of a clock phase, or 50 ns.). Because 200 ns. is longer than the time that the address is present on the processor bus, latches are used to hold the address until it can be seen by the slave. Other data signals are allowed to appear only after the address is gone, so read and write cycles are delayed an additional 50 ns. for propagation delay. Finally, the end of the cycle is delayed to match the start. The typical time degradations are listed in Table 6.

Table 6. Performance degradation due to the voter.

	typ	latch	delays	total	degradn
DATO	1600	20	450	470	29 %
DATI	2000	20	450	470	23 %
DATIO	3600	40	650	690	19 %

Disk access time

Access time to a particular position on a rotating memory is assumed to be directly proportional to the initial position of the disk. Since the hardware makes no attempt to synchronize disk rotation, access to the triplicated disks will take the maximum of the three times. In general, for n disks, the access time is given by [LevnD74]:

$$Tn = MAX (t1, t2, \ldots tn)$$

Assuming that each access time t is uniformly distributed over the normalized range [0,1], the expected value for access time is:

$$Tn = n / (n+1)$$

This means that for a single disk (n=1), we can expect to wait .5 rotations; for the triplicated disk (n=3), .75 rotations. This gives a 50% degradation in access time for the triplicated disks over the non-triplicated disk.

7. SUMMARY AND CONCLUSIONS

The goal of the project was to propose, analyze and synthesize a computer capable of operation in spite of transient and hard faults. Various design techniques were explored to fulfill the goals of software transparency, modularity, real time capability, and performance/reliability tradeoffs. Triplicated modular redundancy at the bus level was selected as the architecture best suited to the task. C.vmp is capable of operation in three modes: independent, each processor communicates with its own bus; voting,the fault tolerant mode; broadcast, selective devices may be left non-triplicated. Off-the-shelf components were used with only minor modifications: in the LSI-11 processor, four wire changes; in memory, no changes; and to the floppy disks, three chips added. Synchronization was achieved by both an internal clock synchronizing the microinstruction execution and an external timing signal synchronizing the bus cycles. Early performance measurements show a degradation of 20-30% over a non-redundant system. The main use of C.vmp will be to perform experiments on the occurrence of transient faults in computers.

Acknowledgements

C.vmp was designed using the Stanford Drawing Package which produces hardcopy logic drawings and wirewrap lists. Al Dunlop assisted in the design, documentation and production of the voter and statistics boards. Ed Snow contributed numerous hours in the debugging phase. Also, special help was offered throughout the project by William Avery, Lloyd Dickman, Steve Teicher, Rick Olsen, and Mike Titlebaum of Digital Equipment Corporation. The faculty and students of the Computer Science Department provided a cheerful ambience during the production of this report.

8. APPENDIX A

Instruction mix in a sample of PDP-11 code

Assuming that the code as a whole represents the same mix as the code executed, a count of 200 instructions from a real-time processing system [AlsuM74] can be broken down as follows:

Table A.1 Breakdown of code by bus cycles.

Total	In Loops	Type of cycles
91	40	Fetch only
38	9	Fetch + DIN
5	2	Fetch + DOUT
1	13	Fetch + RMW
49	0	Fetch + DIN + DOUT
15	0	Fetch + DIN + RMW
1	0	Fetch + DIN + DIN

where DIN = data in, DOUT = data out and RMW = read-modify-write.

To justify the assumption about execution, a second count was made of just that portion of the code which was repetitively executed (appeared in loops). This method is admittedly ad hoc; however, it was felt that if the two counts agreed then it did not matter how often sections of code were traversed, the overall execution would still remain homogeneous with respect to the types of bus cycles being executed. The counts for the loops appear in the second column of Table A.1.

The resulting breakdown is shown in Table A.2.

Table A.2 Frequency of bus cycle types

Type	Total	In loop
Fetch	53 %	63 %
Data in	28	22
Data out	15	15
Read-mod-write	4	0

9. REFERENCES

[AlsuM74] Alsup, Clark, Hackwelder, McConnell, Price, Talmadge, "A Computer - Controlled Vehicle with Sonar Obstacle Detection", Electrical Eng. Dept., C-MU, 1974.

[AvizA71] Avizienis, A., G. D. Gilles, F. P. Mathur, D. A. Rennels, J. S. Rohr, D. K. Rubin, "The STAR (Self-Testing-And-Repairing) Computer: An Investigation of the Theory and the Practice of Fault Tolerant Computer Design", IEEE Transactions on Computer, Vol. c-20, no. 11, November 1971, pp. 1312-1321

[Bart067] Bartos, O.J., "Simple Models of Group Behaviour", Coloumbia Press, 1967, pp 23-38.

[BourW69] Bouricius, W.G., W.C. Carter, P.R. Schneider, "Reliability Modeling Techniques for Self-Repairing Computer Systems", IBM Watson Research Center, Yorktown Heights, N.Y. 1969.

[BourW71] Bouricius, W.G., W.C. Carter, D.C. Jessep, P.R. Schneider, A.B. Wadia, "Reliability Modeling for Fault Tolerant Computers", IEEE Transactions on Computers, Nov. 1971.

[Cane76] Canepa, M.A., "C.vmp: The Implementation of a Fault Tolerant Multiprocessor", E.E. Dept., C-MU, 1976.

[Chan72] Chaney, T.J., Ornstein, S.M., Littlefield, W.M., "Beware the Synchronizer", Compcon 1972, pp. 317, 319.

[ClarS76] Clark, S.D., "Design and Analysis of a Fault Tolerant Computer", E.E. Dept., C-MU, 1976.

[DalyW] Daly, W.M., Hopkins, A.L.Jr., McKenna, J.F., "A Fault Tolerant Digital Clocking System", MIT C.S., Draper Laboratory, Cambridge Ma.

[DunlA76] Dunlop, A., "Reliability Calculation for a Fault Tolerant Computer", C-MU Memo, 1976.

[GoldJ75] Goldberg, J., "New Problems in Fault Tolerant Computing", International Symposium on Fault Tolerant Computing, 1975.

[HopkA75] Hopkins, A.L., Jr, Smith, T.B., "The Architectural Elements of a Symmetric Fault Tolerant Multiprocessor", IEEE Transactions on Computers Volume C-24, no. 5, May 1975, pp. 498-505.

[IBM72] "A Guide to the IBM System/360 Model 145", IBM Corporation, Technical Publication Department, 1133 Westchester Avenue, White Plains, New York, Third Edition, August 1972.

[LevnD76] Levner, D., "The N-Disk Problem Solved", C-MU Memo, Nov 1976.

[LogA] "1600A Logic State Analyzer Operating and Service Manual" Hewlett Packard Colorado Springs division.

[LSI1] "LSI-11 PDP11/03 Users Manual", Digital Equipment Corporation.

[LundA74] Lunde, A., "Evaluation of Instruction Set Processor Architecture by program tracing", Department of Computer Science, Carnegie Mellon University, July 1974.

[LyonR62] Lyons, R.E., Vanderkulk, W., "The use of Triple-Modular Redundancy to Improve Computer Reliability", IBM Journal of Research and Development April 1962, pp 200-209.

[Mil74] Mil-Std-Hdbk-217B, "Military Standardization Handbook: Reliability Prediction of Electronic Equipment", Sept 1974.

[RandB75] Randell, B., "System Structure for Software Fault Tolerance", IEEE Transactions Software Engineering, June 1975, pp 220-232.

[RXV1] "RXV11 Users manual", Digital Equipment Corporation.

[SiewD71] Siewiorek, D.P., "A Unifying Perspective of Fault Tolerant Computer Techniques",Technical note no 61, Digital Systems Laboratory, Stanford University, 1971.

[SiewD75] Siewiorek, D.P., "Transient Fault Model for a Triplicated Fault Tolerant LSI-11", Research & Development Group, Digital Equipment Corp., 1975.

[SwanR75] Swan, R.J., Fuller S.H., Siewiorek D.P., "The Structure and Architecture of Cm*", Computer Science Departement Research Review, 1975, 76, Carnegie-Mellon University.

[VonnJ56] von Neumann, J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components", Automata Studies, from Annals of Mathematics Studies no 34, Princeton University Press, pp43-99, 1956.

[WakeJ75] Wakerly, J.F., "Reliability of Microcomputer Systems Using Triple modular Redundancy", Technical note no. 41, Digital Systems Laboratory, Department of Electrical Engineering and Computer Science Stanford University, April 1975.

[WensJ72] Wensley, J.H., "SIFT - Software Implemented Fault Tolerance" AFIPS Conference Proceedings, Vol. 41, Part 1, AFIPS Press, Montvale, New Jersey 1972, pp. 243-254.