intellitech

The Intelligent Use of Technology

(2)

DESIGN AND ANALYSIS OF FAULT TOLERANT ARCHITECTURES FOR MULTI-MICROPROCESSOR SYSTEMS

INT-84-45

(2)

DESIGN AND ANALYSIS OF FAULT TOLERANT ARCHITECTURES FOR MULTI-MICROPROCESSOR SYSTEMS

DECEMBER 1984

Prepared By: S. Boucouris (1)
Approved By: Dr. S.A. Mahmoud

INTELLITECH CANADA LIMITED

352 MacLaren Street
Ottawa, Ontario
K2P 0M6

Government of Canada / Gouvernement du Canada

Department of Communications

DOC CONTRACTOR REPORT DOC-CR-SP -84-051

DEPARTMENT OF COMMUNICATIONS - OTTAWA - CANADA

SPACE PROGRAM

TITLE: Design and Analysis of Fault Tolerant Architectures for Multi-Microprocessor Systems

AUTHOR(S): Spiros Boucouris
Dr. S.A. Mahmoud

ISSUED BY CONTRACTOR AS REPORT NO: INT-84-45

PREPARED BY: S. Boucouris
Dr. S.A. Mahmoud

DEPARTMENT OF SUPPLY AND SERVICES CONTRACT NO: OER 83-05075

DOC SCIENTIFIC AUTHORITY: Michel Savoie
Communications Research Centre
Ottawa, Ontario

CLASSIFICATION: Unclassified

DATE: December 1984

SUMMARY

This report presents the results of a study conducted to design, simulate and evaluate a fault-tolerant multi-microprocessor system for spacecraft on-board processing applications. The fault-tolerant processor architecture can also be used in general applications requiring high degree of reliability over a specific processor life cycle.

The design approach proposed in this study is novel in the sense that fault-tolerant features and supporting mechanisms are embedded both in the hardware architecture and in the operating system software. The hardware has several redundant components that are controlled by fault detection mechanisms to isolate the sources of error and prevent the proliferation of these errors from the faulty component to the remainder of the system. The operating system software contains all the intelligence needed to detect errors, identify their sources, take the necessary action to remove faulty units, reallocate the processing tasks and reconfigure the system to adapt to the new operational state.

Results of the fault tolerant study are presented in two reports, both of which are deliverables under contract OER-83-05075. This report presents the hardware fault tolerant architecture. The second report, entitled "Conceptual Design of a Fault Tolerant Multiprocessor Operating System and the Implementation of a Prototype Kernel", presents the conceptual design and simulation of the basic functions (the kernel) of an operating system with fault-tolerant characteristics.

Further research intended to complete the description of the fault-tolerant operating system and to integrate it with the underlying hardware structure is currently being conducted. Results of this research will be presented at the conclusion of the current phase of the contract (March 1985).

This report is a deliverable under the terms of DSS contract OER 83-05075, to the Communications Research Centre of the Department of Communications, Government of Canada.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Objectives Of This Study

The main objective of this study is the design and simulation of a processing system architecture capable of graceful recovery from component failures or other errors, so that it can function in a normal mode for a specific, relatively long, time duration.

This study has been motivated by the ever increasing on board processing requirements in future spacecraft missions. One of the essential requirements for satellite on-board processing is the ability of the processing system to recover from component failures, redistribute the processing load among the remaining components and continue its normal functioning throughout the intended mission life span.

To achieve these characteristics in an unattended mode, the fault tolerant features of the system must be embedded both in its hardware structure and in the software operating system which manages the fault recovery mechanisms.

This study was conducted for the Communications Research Centre, Department of Communications under a DSS contract.

1.2. A Typical Satellite Bus Configuration

The block diagram of a typical present day satellite system is shown in Figure 1.1. The system consists of a number of dedicated subsystems each of which controls a set of devices that serve one particular function. The most important of these subsystems are:

- Telemetry Tracking and Command subsystem (TT&C)

  This subsystem handles the communications with the ground station and the control of the other subsystems. Any on board

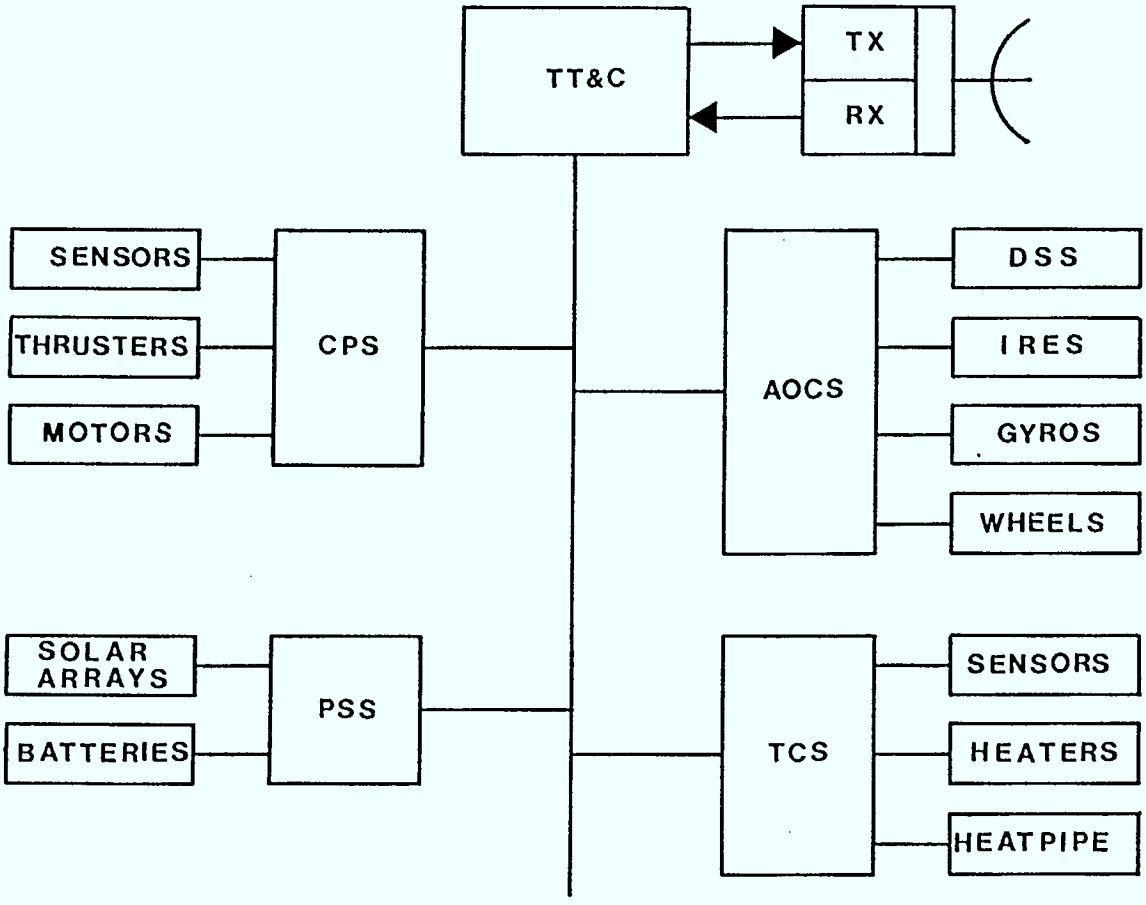


FIGURE 1-1: Typical Satellite Subsystems

intelligence is, at present, very limited and is included in this subsystem. Aside from controlling the other subsystems this subsystem directly controls the communication devices of the satellite, namely the Transmitter (TX) and the Receiver (RX).

- Attitude and Orbit Control Subsystem (AOCS)

  This subsystem monitors the stability of the satellite and ensures its proper orientation. To accomplish this it takes its data from Digital Sun Sensors (DSS), Infra-Red Earth Sensors (IRES) and Gyros. After suitable processing these data can be transmitted to the ground station or they can be used to control the wheels in order to change the orientation of the satellite.

- Combined Propulsion Subsystem (CPS)

  This subsystem monitors the speed and acceleration of the satellite and performs any necessary corrections under the commands of the TT&C subsystem.

- Thermal Control Subsystem (TCS)

  The function of this subsystem is to maintain the temperature of the satellite within certain prespecified limits in order to ensure proper operation of the on-board systems. This is accomplished by monitoring heat sensors (HS) and operating either a heater or a cooler as appropriate.

- Power Subsystem (PSS)

  This subsystem ensures that all systems in the satellite are provided with sufficient electrical power. It controls the connection of the Solar Arrays (SA) to the batteries and the flow of energy from the batteries to the various subsystems of

the satellite.

1.3. Fault Tolerant Features

The processing system for the typical satellite example described in section 1.2 does not employ any fault tolerant features. Instead designers relied mainly on component screening and thorough testing in order to ensure the reliable operation of the system.

Although this approach was feasible for very simple satellite systems, the ever increasing complexity of the various subsystems dictated the inclusion of certain fault tolerant features for the processing system of the described architecture. These consisted mainly of duplicate devices intended to become operational in case of a failure of the main device. However, no change in the architecture of the system was attempted. The main reason for that is the simplicity of this architecture which made it usable in the resource limited satellite environment.

As the state of the art of electronic components progresses, it becomes feasible to employ more complex architectures in a satellite processing system. These architectures enable the satellite processing system to function on its own without any supervision from the ground and to automatically make any corrections necessary in case of a component failure.

To accomplish this, a satellite processing system must be capable of detecting any failure. As well, such failure should not be allowed to interfere with the normal operation of the on-board processing system. This is accomplished by employing more redundancy in the processing system. For example three or more processors running in parallel can both detect and mask any single failure, thus allowing the

system to continue its operation in a normal fashion despite the failure.

Once the error is detected the processing system must take steps for the isolation of the faulty component and possibly for its replacement with a functional unit. This capability is provided to the system by a suitable integration of hardware support mechanisms and appropriate control software. Typically the hardware will first detect the fault, and signal the software about it. Then the software will determine if a certain device must be isolated and/or the system must be reconfigured. Appropriate commands are then issued to the hardware which then implements the isolation/reconfiguration.

1.4. Structure of this Report

In this report a processor architecture will be presented which meets the requirements mentioned in section 1.3. Many elements of the proposed architecture have been adopted from fault tolerant architectures used in various control applications, namely flight control for aircraft, train junction control etc. These architectures were considered too complex to be employed in the satellite environment. The study reported here investigates the feasibility of adapting such fault tolerant architecture to meet the on-board processing requirements of satellite systems.

The control applications mentioned above require a very high degree of reliability since in most cases human lives are in immediate jeopardy. However the time period over which this reliability is required is small since flight time is in the order of few hours. Thus repairs and service are accessible every few hours. A satellite system, by contrast, requires a relatively lower degree of reliability but for

significantly longer periods of time, namely the intended life span of the satellite which can be in the order of 10 years.

In chapter 2 of this report the various available architectural alternatives are described and compared to each other. From this comparison an optimal architecture is selected and described in detail in chapter 3. Chapter 4 presents a reliability analysis of the selected architecture. This analysis is similar to the analyses available for existing systems although the different requirements dictate different weighting of the analysis parameters. Some factors that were considered insignificant in other application turn out to be playing a primary role in the satellite application.

Chapter 5 contains a detailed description of one possible hardware realization of the selected architecture presented in chapter 3. The description is general and is not hardware specific. However, it is detailed enough that it can be used directly for an implementation of the architecture. Finally, Chapter 6 contains a summary of the work completed so far along with some concluding remarks.

2. SELECTION OF A MULTIPROCESSOR ARCHITECTURE

2.1. Introduction

From a logical point of view the control processes of a satellite system can be divided into central processes and peripheral processes. The central processes perform the actual processing while the peripheral processes are simply device drivers which interface to peripheral devices. Although the logical separation does not necessarily dictate physical separation, in most cases the device drivers are actually embedded in the devices as firmware. In this report, the processors that run this firmware will be referred to as peripheral processors while the processors that run the actual control software will be called central processors. Peripheral devices may either be connected to an external (peripheral) processor or they may be directly controlled by the central processors.

An on-board processing system for a typical satellite can be modelled as a composite of two sets of processors. One set does the actual processing while the other set consists of intelligent peripheral controllers, sensors and actuators. The latter set of processors can be considered as slave processors and are controlled by the former set. This arrangement is shown in Figure 2.1. The central processors share the computational load and are the highest authority. They are built as a fault tolerant structure and employ redundancy to ensure reliability. The peripheral processors usually operate alone and may have stand-by replacements.

The processor network interconnects the central processors while the peripheral network connects the central processors to the peripheral processors which they control. Once again logical separation does not dictate physical separation. In the detailed descriptions of the

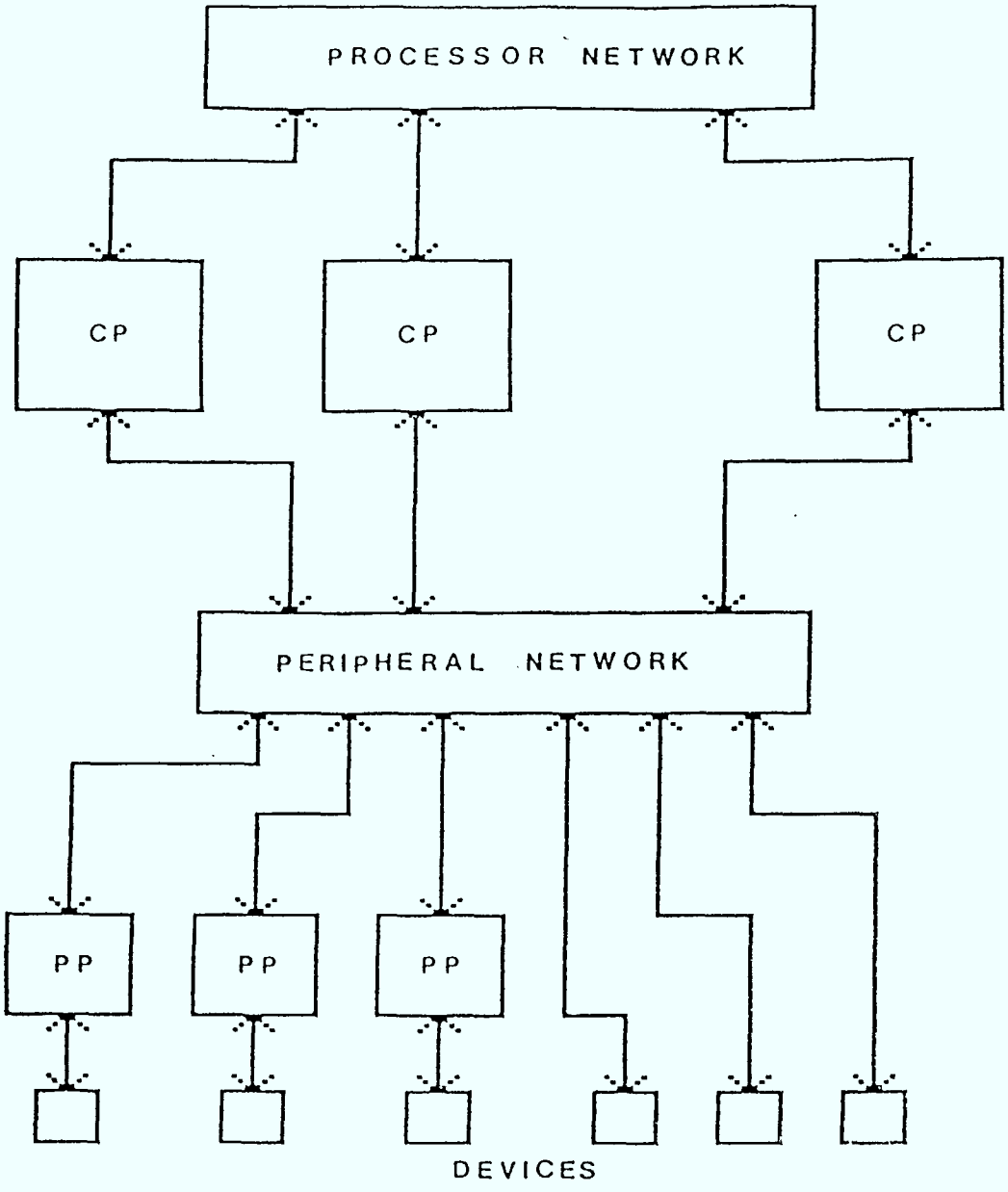


FIGURE 2-1: Conceptual Logical Description of a Satellite Processing System

various possible configurations we shall see that for some configurations the two networks may be physically one and the same.

2.2. Central Processors

The central processors handle the processing load of the satellite. Since they constitute the highest authority they must be made fault tolerant and fail safe. Besides, the whole system must be able to degrade gracefully so as to maximize its total useful life. To accomplish that it is necessary that any faulty part of the system can be completely isolated. This ensures that accumulation of faults will not cause a system crash as might happen if the fault was simply masked out of the system.

To accomplish this some redundancy must be built into the system so that there will be no single point of failure. A single point of failure is defined as a part of the system whose malfunction would be irrecoverable. Redundancy may be employed either microscopically or macroscopically. Microscopic redundancy is achieved by duplication of individual devices in a processor and is more suitable for processors built out of discrete devices. Macroscopic redundancy is achieved by employing redundancy in the number of processors executing a segment of code while providing for flexible reconfiguration of the system for the isolation of faulty parts. Considering the rapid advances in microprocessor technology, macroscopic redundancy seems preferable.

In order to ensure that the results of any program will be available in time independent of any hardware failure, the program must be run on at least three processors and the results voted. Different implementations of this idea have been used, each with its own advantages and disadvantages. These implementations differ in the way

the processors are interconnected and synchronized. The various processor interconnection schemes will be presented and discussed in the section devoted to the processor network. Two different synchronization methods are available. These are tight coupling (or tight synchronization) and loose coupling.

In a tightly coupled system all processors run synchronously (i.e. they are all driven by the same clock). Different implementations of the clock circuit ensure that it will not become a single point of failure [1]. Any program that has to be executed is run on three processors simultaneously and the results of the processors are voted upon by hardware majority voters. This voting may take place either on a per instruction basis or on a task basis. Figure 2.2 showns a system with instruction synchronization. Any data fetched from memory are actually fetched from three separate memories and voted upon before they are distributed to the processors. Similar voting takes place for data sent to the memory from the processors. Figure 2.3 shows a system with task synchronization. Each processor accesses its own memory as an independent system. However any I/O is voted upon before it is passed out to the rest of the system.

Although voting on an instruction by instruction basis minimizes the time during which an error remains undetected, it requires a large hardware overhead and therefore is not suitable for a satellite system. On the other hand task synchronization requires much less hardware overhead while, by an appropriate choice of checkpoints, it ensures an acceptable level of security. A classic example of a tightly coupled system with task synchronization is FTMP [1].

In a loosely coupled system there is no phase synchronization between the clocks of the various processors in the system. Instead

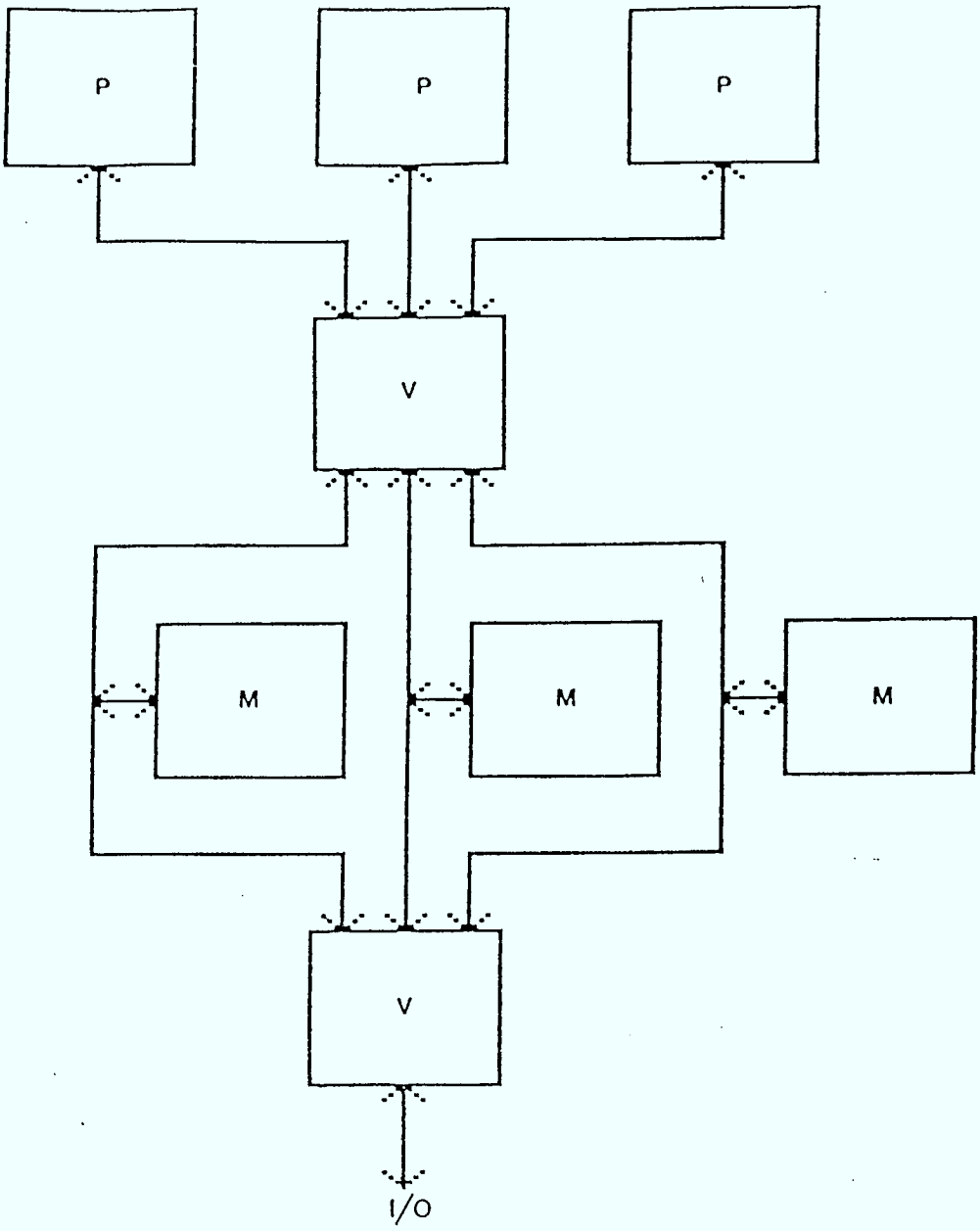


FIGURE 2-2: Tightly Coupled System with Instruction Synchronization

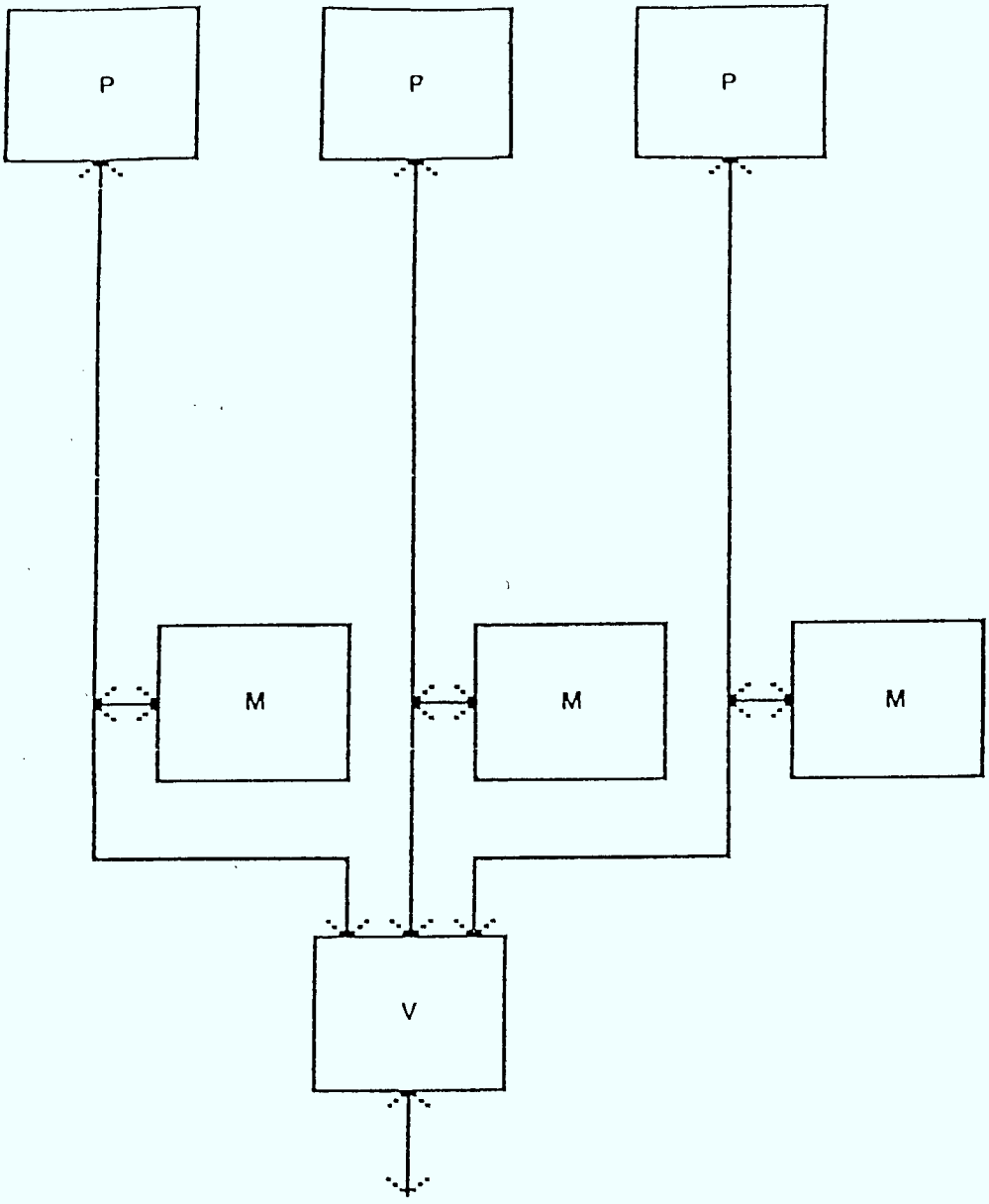


FIGURE 2-3: Tightly Coupled System with Task Synchronization

synchronization is effected by software. Any task runs on three independent processors. At certain predefined points the three copies of the task exchange information. This information is then voted upon in software. If an error is detected it is masked by the voting and logged. When a task is completed it distributes its results to the three copies of its successor(s). Each of the copies will vote the received results and proceed. Figure 2.4 shows the passing of messages between three copies of a task and one of the copies of its successor through message buffers. A classic example of a loosely coupled system is the SIFT system [2,3,4].

Both tight and loose coupling seem to provide a similar level of fault tolerance. In both cases some sort of error logging and handling is required so that transient errors will not affect the operation of the system. This may be done by some operating system level tasks (error handlers) running in parallel with the application tasks.

Whenever a fault is detected an appropriate message is passed to these tasks. Then these tasks will decide if there is a hardware failure or the fault is simply a transient one and can be ignored for the time being. The exact nature of the message depends on the fault and the fault detection mechanism of the particular system whereas the way the message is passed depends on the hardware and software implementation of the system.

In terms of error recovery the loosely coupled system is superior because all processors are autonomous. Therefore the load of a failed processor can be transferred to another processor, or it may be distributed among several processors. In a tightly coupled system a similar distribution is not possible. The load of a processor must be transferred to a spare which must then be brought in synchronization

with the other two members of the triad. As a result, the processing power of tightly coupled systems, degrades less gracefully than that of loosely coupled systems as shown in Figure 2.5. This is to be expected since in tightly coupled systems any processor that is not a member of a triad remains idle whereas in loosely coupled systems all processors can share the processing load.

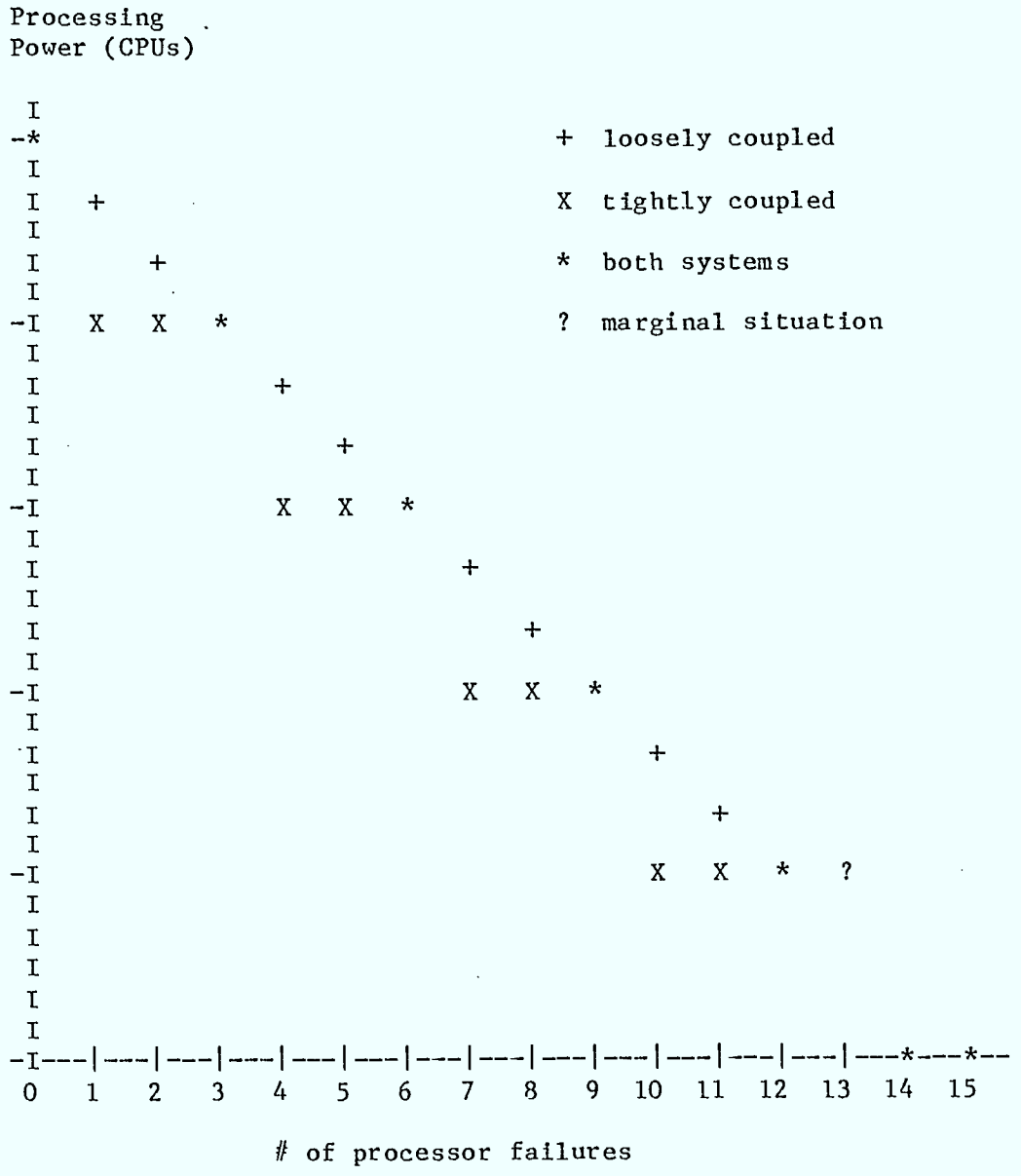


FIGURE 2-5: System Degradation for Tightly and Loosely Coupled Systems of 15 Processors

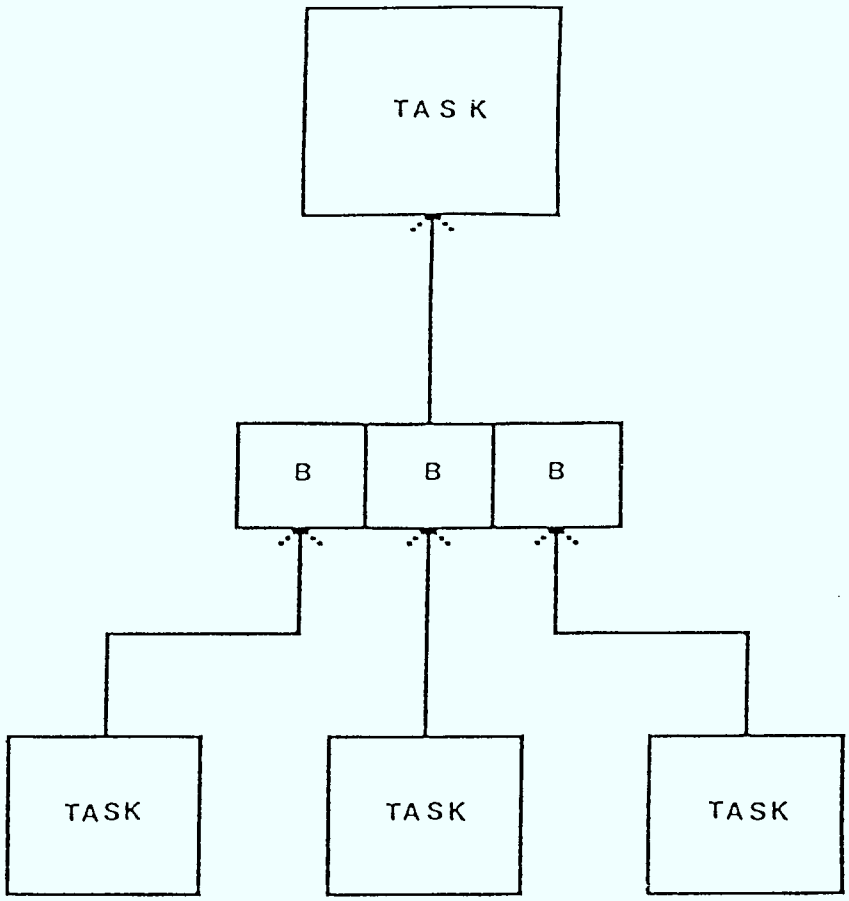


FIGURE 2-4: Task Communication in a Loosely Coupled System

2.3. Peripheral Processors

The peripheral processors are little more than intelligent device controllers. Besides controlling the actual devices and interfacing with sensors/actuators they have to be able to handle some communication protocol in order to assure reliable communication with the central processors. The nature of the protocols available will be discussed in the section for the peripheral network.

Triple redundancy is not necessary for these processors. Instead the operation of the processor is monitored by the central processors and, in case of a failure, a cold stand-by unit is activated and brought in to replace a faulty unit. The monitoring of the peripheral processors can be done either by employing a suitable transaction protocol or by periodic polling of the peripheral processors by the central processors.

2.4. Processor Network

The function of the processor network is to interconnect the central processors of the system. The nature and requirements of this network may vary greatly depending on the memory organization of the system. The two schemes that can be used are common memory and private memory. If common memory is used then each processor must also be equipped with an on-board cache in order to minimize bus and memory contention.

The two main schemes used for processor interconnection are common bus and full interconnection. The fully interconnected system inherently offers a greater degree of fault tolerance due to its redundancy whereas the common bus offers simplicity and lower cost. Clearly care must be taken so that the common bus will not become a

single point of failure. This is accomplished by using several busses in parallel and providing spare busses and the ability to switch busses. The most obvious implementation is again triple redundancy. Data is sent along three separate busses and if one bus fails it is replaced by a spare.

Intermediate bus structures do not seem very promising for fault tolerant applications because they contain single points of failure. For example in a crossbar arrangement the failure of a switch may incapacitate both busses it connects. If triple redundancy is employed then the number of busses in the system will become comparable to that of a fully interconnected system without any added advantage. Besides a crossbar allows connections between any member of one set and any member of another set. Therefore it is usable for connecting processors with memories but not for interconnecting processors. One example of such a system is the Pluribus system [5] which uses a modified crossbar structure. Although this modified structure avoids single points of failure it is very complicated.

Simpler interconnection schemes like rings or stars have obvious single points of failure. If redundancy is employed in the implementation of one of these schemes the possibility of single point failure could be removed. However, such an implementation would have no advantage over a common bus configuration.

In summary, we can combine the previously mentioned configurations (i.e. common or private memory, common bus or fully interconnected and tight or loose coupling) in almost every possible way. Each possible combination will now be discussed briefly in order to determine the most suitable configuration.

1) Common Memory - Common Bus

The block diagram of this configuration is shown in Figure 2.6 for both tight and loose coupling.

1.a) Tightly Coupled

This configuration is the one employed in the FTMP system [1]. Effectively each processor gets assigned a memory and a bus and operates with these as a separate system. However both memories and busses are common and may be assigned to any processor. Thus it provides a fine component granularity. A serious drawback of this configuration is that all processors must run in perfect synchronization. This requires extra hardware for reliable clock distribution and processor synchronization. However its main disadvantage is that either the interface to the memory will be serial and therefore slow or a very large bus will be required.

1.b) Loosely Coupled

This configuration has the disadvantage over the corresponding configuration with tight coupling that the busses that form the common bus will be asynchronous to each other. Therefore some extra synchronization and arbitration will be necessary. On the other hand it does not require clock distribution. However, it shares the disadvantage of either slow memory interfaces or expensive bus.

2) Private Memory - Common Bus

The block diagram of this configuration is shown in Figure 2.7 for both tight and loose coupling.

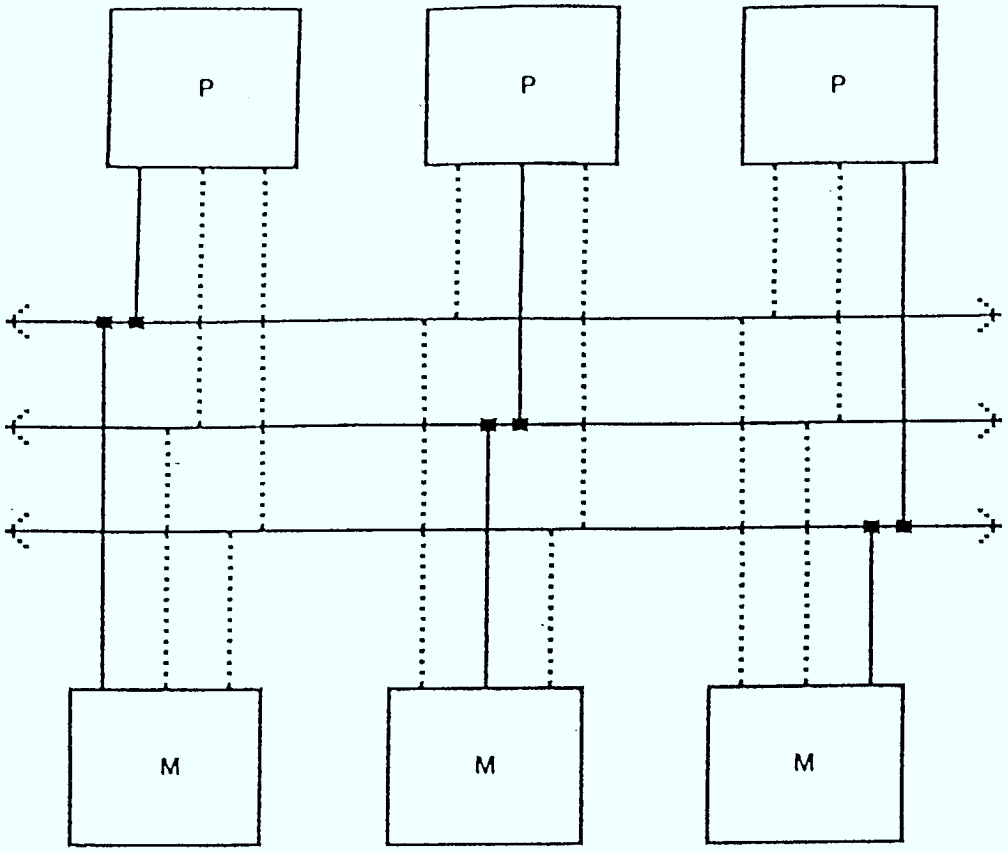


FIGURE 2-6: Common Memory Common Bus System

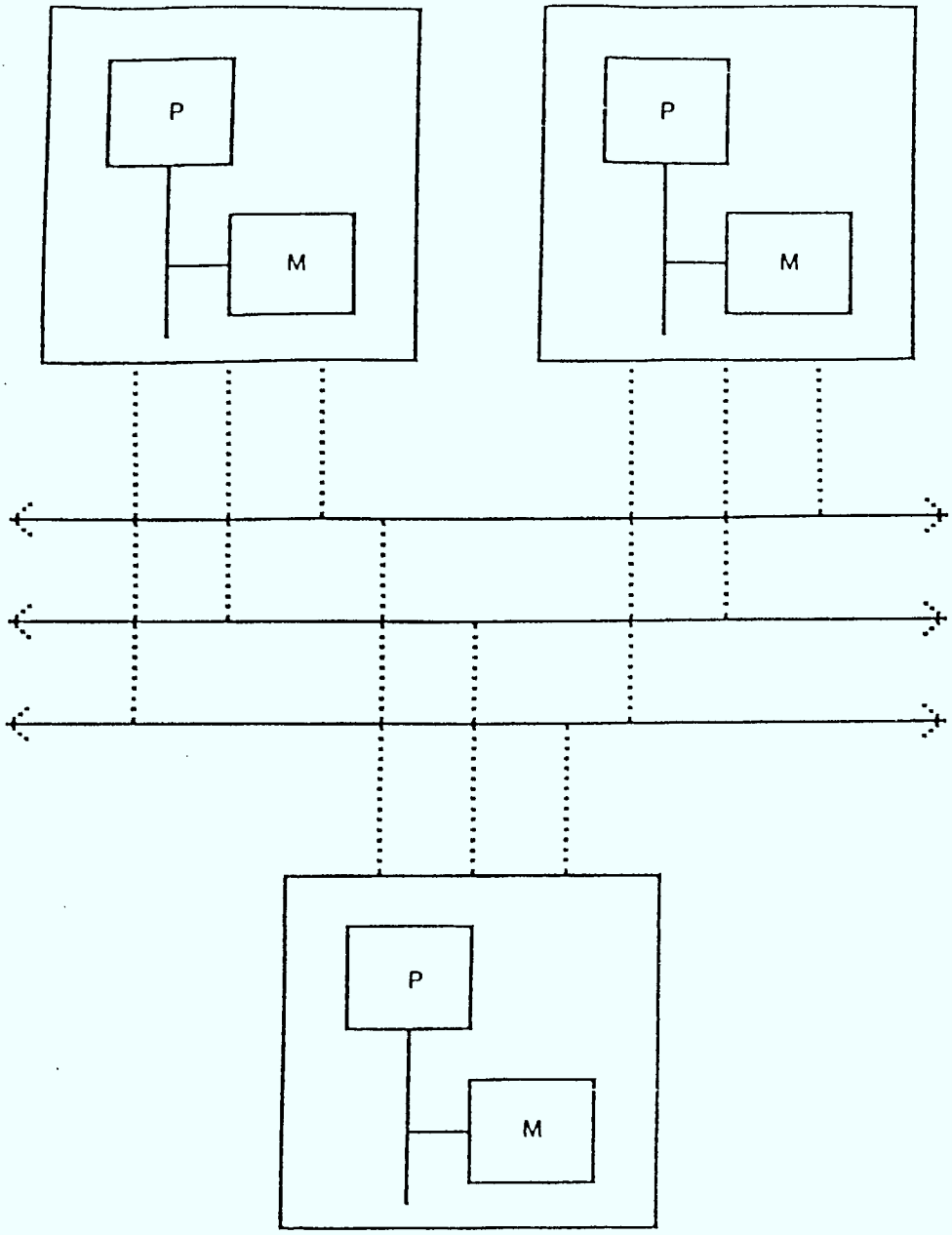


FIGURE 2-7: Private Memory Common Bus System

2.a) Tightly Coupled

This configuration seems to have no particular advantage over the corresponding system with loose coupling. Given the relatively independent nature of each processor there is no reason justifying the extra hardware cost of tight coupling.

2.b) Loosely Coupled

This configuration is very much like the SIFT [2,3,4] configuration with the difference that the common bus is used for message passing. The disadvantage of this approach is that a faulty processor may end up overwritting messages (in the receivers input) from another processor. This is because there is no physical separation of the communications of the various processors. Besides, the sharing of a common bus between asynchronous processors imposes some requirements for reliable and fault tolerant arbitration.

3) Common Memory - Fully Interconnected

The block diagram of this configuration is shown in Figure 2.8 for both tight and loose coupling.

3.a) Tightly Coupled

This configuration has several drawbacks. In order to allow for reconfiguration each memory must be connected to all the processors in the system. This means a significant increase in hardware complexity. Besides, memory contention might make it impossible for tightly coupled processors to operate with any acceptable performance.

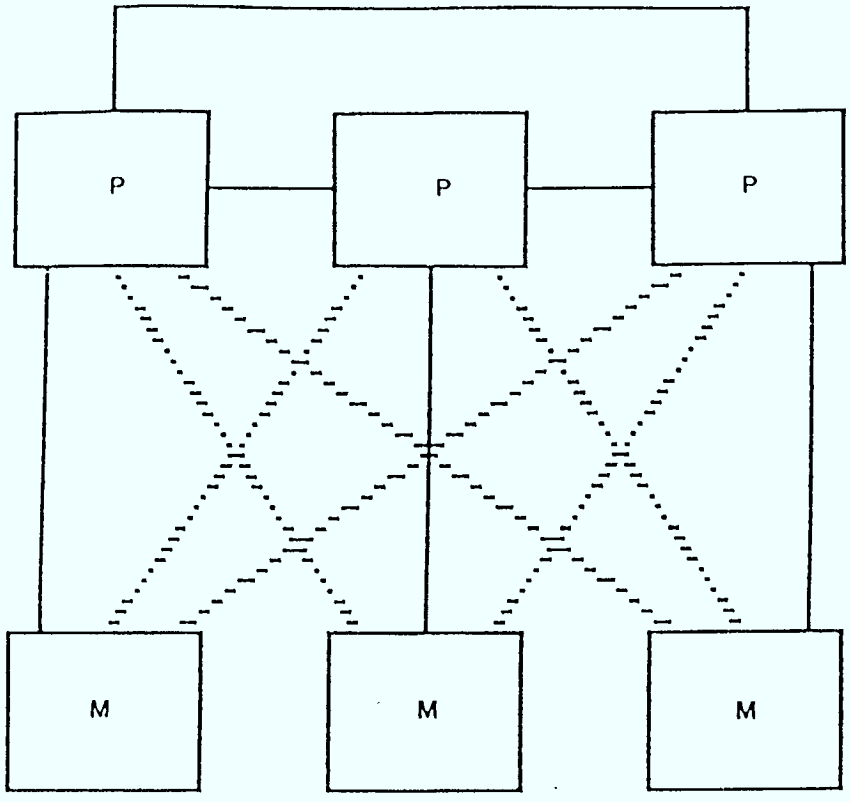


FIGURE 2-8: Common Memory Fully Interconnected System

3.b) Loosely Coupled

This configuration serves as an alternative to a private memory - fully interconnected system. Its advantage is that it has more independent subsystems that can be isolated and therefore it degrades more gracefully than the corresponding configuration with private memories. The requirement for multiport memories, along with the large number of connections required, makes it impractical for large systems.

4) Private Memory - Fully Interconnected

The block diagram of this configuration is shown in Figure 2.9 for both tight and loose coupling.

4.a) Tightly Coupled

This configuration has the advantage over the corresponding loosely coupled system that it can directly drive peripherals without any extra overhead. However, it requires some extra hardware overhead for fault detection and degrades less gracefully than the loosely coupled system.

4.b) Loosely Coupled

This is the best alternative for large systems. The hardware overhead is minimized and all fault tolerant functions are handled in software. This makes future changes and upgrades to the system extremely easy to implement without any requirement for hardware changes. Therefore the system can be upgraded without being physically changed which is an important feature for satellite systems. A classic example of this configuration is the SIFT system [2,3,4].

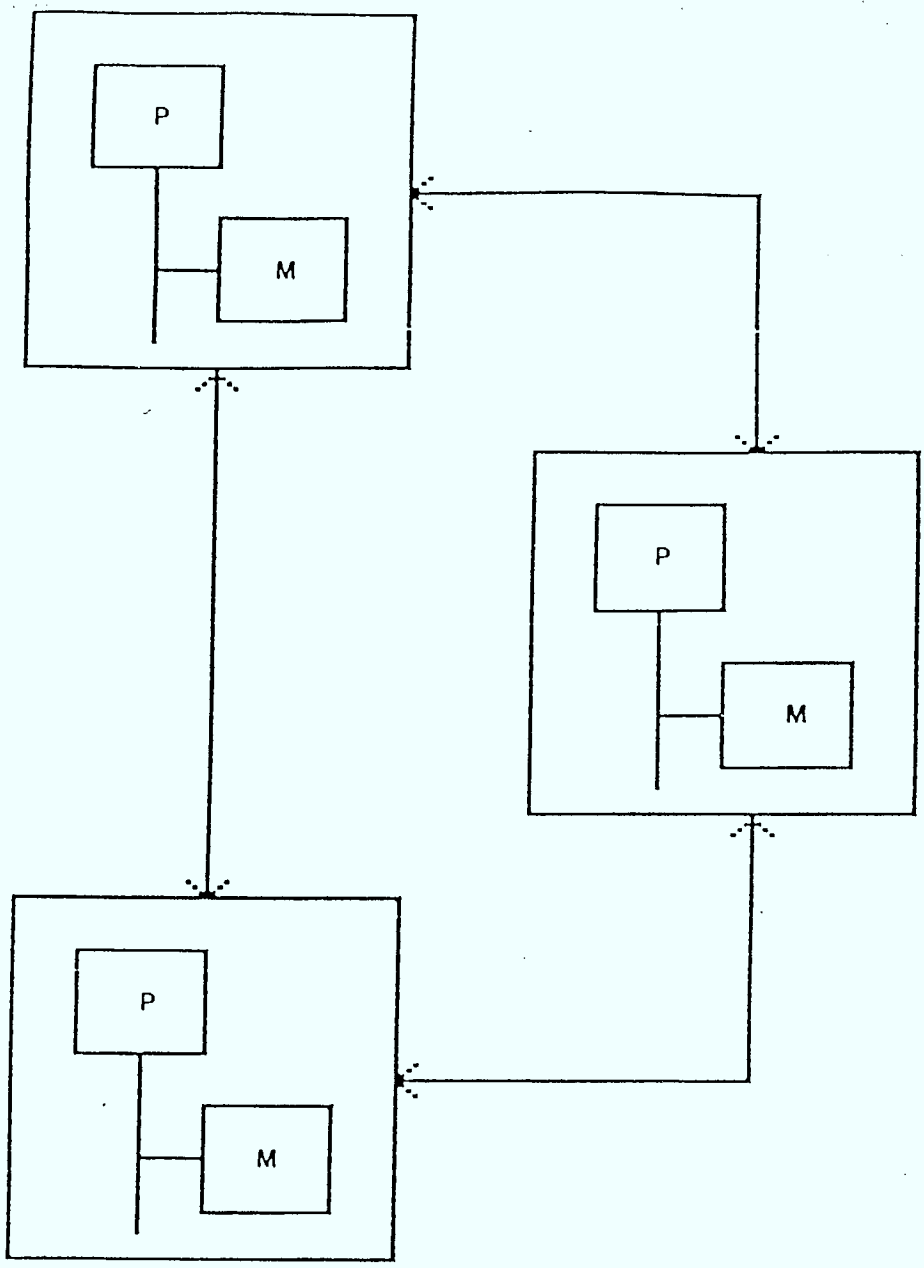


FIGURE 2-9: Private Memory Fully Interconnected System

2.5. Peripheral Network

The peripheral network will connect the central computer system to the peripheral system which will be spread throughout the satellite. The bandwidth requirements of this network depend on whether it will be used strictly for control purposes or there will be some high speed data transfers as well. If the network is only used for control purposes then its only function will be to pass control messages to actuators and receive readings from sensors. If high speed data transfers are also required it may be better to employ a dedicated path for high speed data transfers rather than giving the whole network a high bandwidth. These factors seem to dictate a serial realization of the data paths of this network.

The large number of devices connected to this network makes a common bus the only viable alternative. Therefore the only decision that remains to be taken is the actual realization of the bus. Since not all devices connected to this bus will have intelligence it seems that three busses carrying the same information is the only viable alternative. The final decision is whether the three busses will be synchronous to each other or not. If the busses are not synchronous then every peripheral device will have to buffer its messages until all three copies are available and only then vote on them. This implies the need for timeout in the case of missing messages along with some protocol for dealing with this case. On the other hand a synchronous bus can drive a voter directly thereby simplifying the device interface. Given the large number of devices a synchronous bus is the best alternative.

The term synchronous busses is used above to mean that the three or more parts of the bus will be synchronous to each other. However the

operation of the bus as a unit may be synchronous or asynchronous in the conventional sense. Synchronous operation yields a higher bandwidth but requires distribution of the clock to all peripherals. On the other hand asynchronous operation of the bus does not have this requirement. If the extra bandwidth offered by synchronous operation is not essential, asynchronous bus operation is preferable.

3. FAULT TOLERANT ARCHITECTURE

3.1. Introduction

In this chapter the architecture of a fault tolerant hierarchical multiprocessor system will be presented. The overall block diagram of the system, shown in Figure 2.1, is similar to the general block diagram presented in Figure 1.1. However, some of the blocks in the diagram differ radically from the corresponding blocks described in chapter 2.

In the following sections only the blocks that differ from the "standard" blocks described in chapter 2 will be discussed. For reasons of simplicity the central processors and the processor network are presented in a single section (section 3.2) as the central control system. No specific requirements are placed upon the peripherals. As well, no assumption is made about the peripherals, therefore the system is completely general.

The peripheral interface system (peripheral network) is described in section 3.3. Section 3.4 describes an extension to the system which could be added if there is a requirement for large computational power. Finally section 3.5 describes the devices used for system reset and in the removal of malfunctioning units.

3.2. Central Control System

This consists of the central processors and the processor network. Given the requirements for expandability, modularity and flexibility of the architecture along with the requirements for an easily reconfigurable, gracefully degrading system, a loosely coupled, fully interconnected system with private memories was chosen.

The key element of the central control system is the processor module (Figure 3.2). A processor module consists of the CPU, memory,

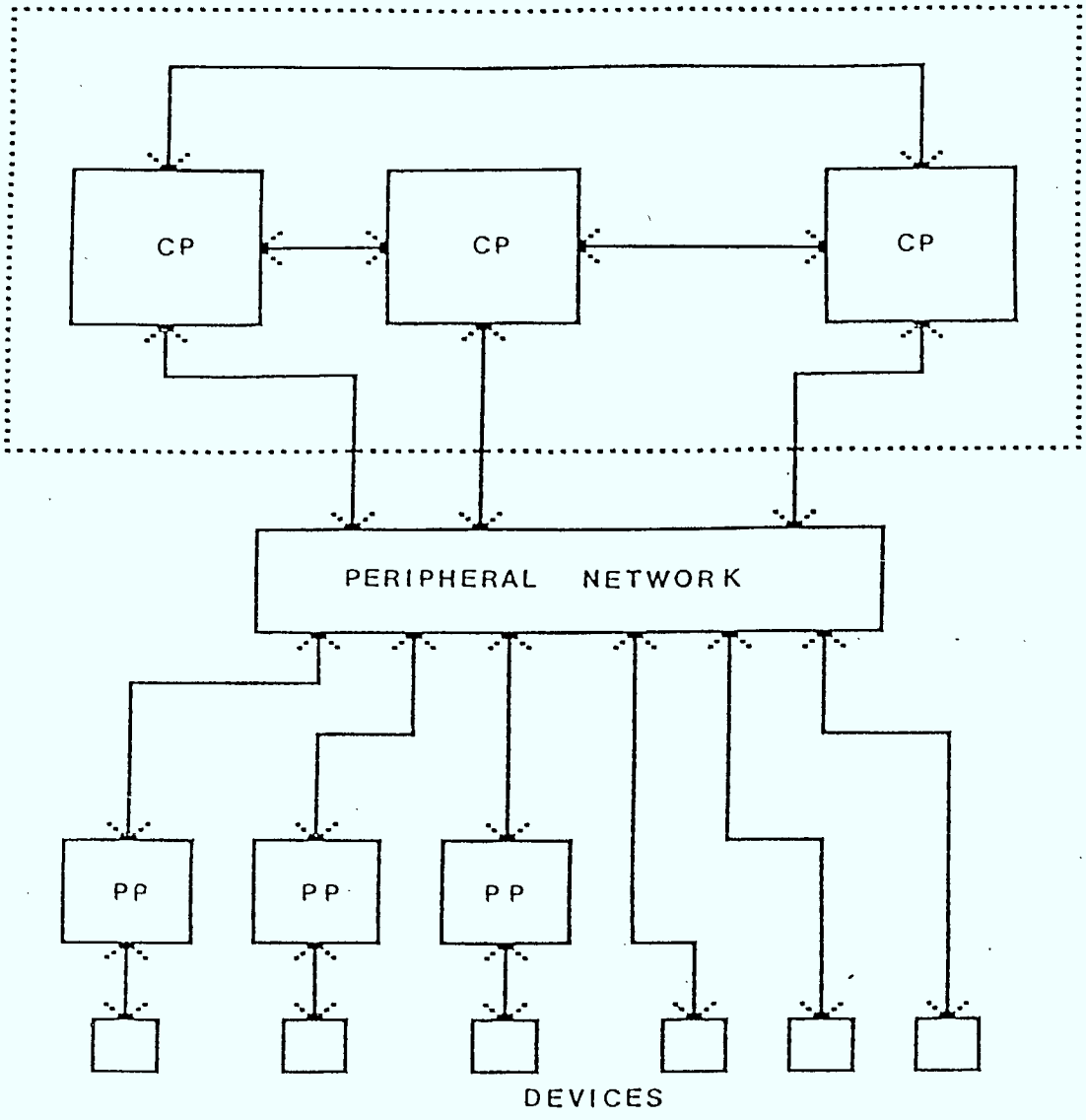


FIGURE 3-1: Block Diagram of the Proposed System

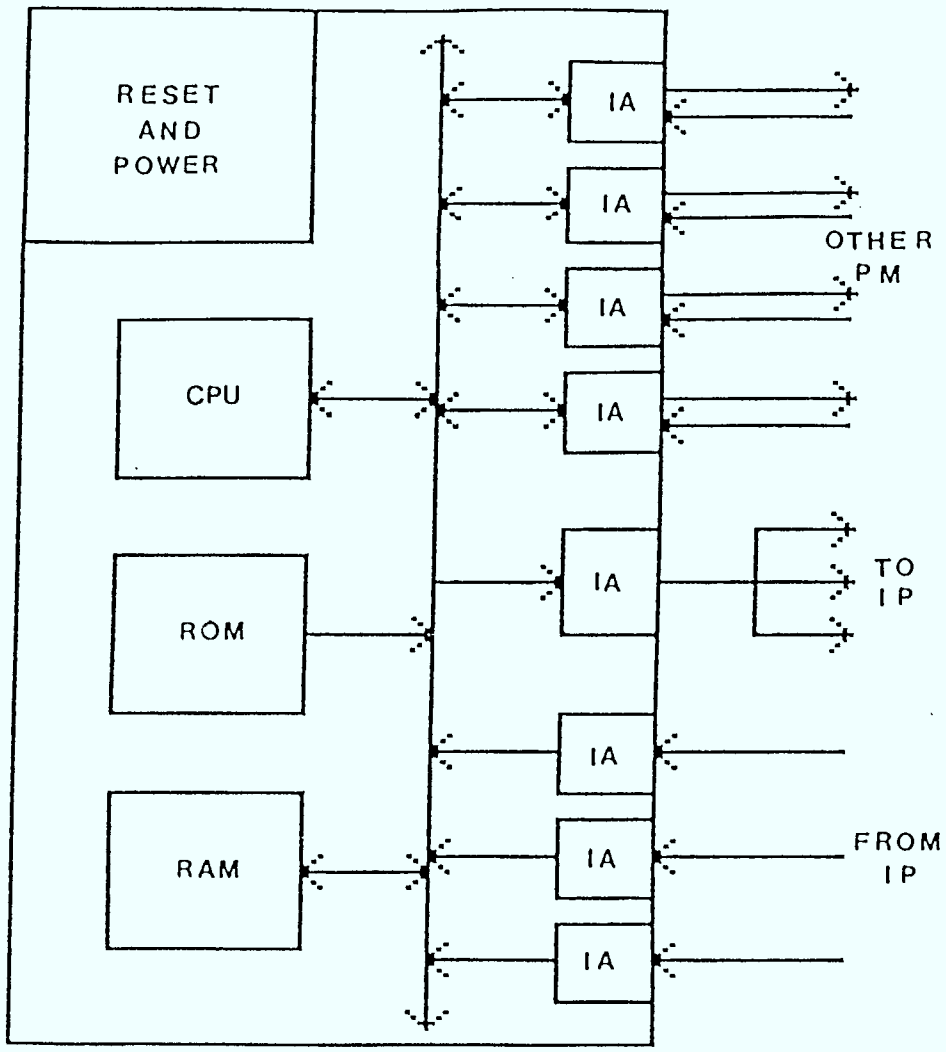


FIGURE 3-2: Processor Module

hardware support for an operating system (timers, interrupts, etc.) and appropriate interfaces for the interconnection with the other processors in the system. The module includes both RAM and ROM memory and is completely self contained. The on board ROM contains a basic operating system that handles the communication with other processors, and a number of self checking routines.

The interfaces to the other processor modules consist of a transmitter/receiver pair. The interface to the interface processors consists of a single transmitter that broadcasts the message to all interface processors and a number of receivers, one for each interface processor. The reason for this different interface to the interface processors will be explained in the following section that describes the peripheral interface system. The exact nature of the transmitters and receivers will be explained in the chapter about the detailed description of the hardware.

In addition, the board has a power control and reset block which will be discussed in detail in the section about the system reset mechanism. The function of this block is to force a reset on a board which is suspected of malfunctioning, thereby causing it to run its self checking routines. Also if a board is determined to be irrecoverably lost, then, it is powered down to conserve energy. All these functions will be carried out following instructions originating from some supervisory tasks running on the processor cluster. From that point of view the system can be though of as self maintaining.

The relatively simple structure of the hardware has to be complemented by reliable and intelligent software so that the system can fulfill its specifications. At a bare minimum the operating system must be able to :

1) Keep a record of errors in order to determine if some module needs to be tested. The record must be complete and the software must be intelligent enough so that transient errors will not cause unnecessary tests. In other words a processor that makes a single mistake must be given the benefit of the doubt. Only if the number of errors within a certain time period exceeds a threshold should the processor be declared as faulty.

2) Provide for some sort of a timeout mechanism so that a processor will not be tied forever waiting for a message from another processor. This will prevent a crashed processor from delaying or even deadlocking the system.

3) Isolate and ignore any single faulty communication channel and reconfigure the system around the problem. If, for example, the channel between two processors fails, then by appropriately rearanging the distribution of tasks it may be possible to keep the system operating at full capacity.

The implementation of an operating system that covers the above and several other requirements for fault tolerance will be presented and discussed in detail in the report on the system software [6].

3.3. Peripheral Interface System

Basically the peripheral interface consists of a common redundant bus (a number of serial busses running in parallel). In section 2.5 it was determined that such a bus should be tightly coupled. However the central control system, as described in section 3.2, is loosely coupled. The interface system must provide an interconnection between the two without introducing a single point of failure.

To accomplish this a number of interface processors are used. The interface processors are tightly coupled and provide an interface between the loosely coupled central processors and the tightly coupled peripheral network as shown in Figure 3.3. The connections of an interface processor are shown in Figures 3.3 and 3.4.

On one side an interface processor is connected to the fully interconnected central control system. This interface is especially designed to accomodate the conflicting operating modes of the central and interface processors. To the other side it is connected to the redundant bus. At any given time an interface processor drives at most one of the busses that form the redundant bus. However, it is possible for any interface processor to drive any one of the buses.

Three interface processors will be active at any time. These will run in an infinite loop polling both the central processors and the peripheral devices. Whenever a message is received from a device it is passed to the appropriate central processors. When a message is received from a central processor it is stored and, after a prefixed timeout period has elapsed, a software voting takes place and the correct(ed) message is passed to the appropriate peripheral.

If one of the interface processors fails, it is disconnected and a cold stand-by is phased in to replace it. Similarly, in case of a bus failure, the corresponding processor is connected to one of the unused busses. In this event all peripheral devices must be notified of the change so that they too will switch to the new bus.

Any device connected to the peripheral bus is buffered by a fail safe interface. An interface consists of a parallel to serial converter, a tri-state gate and a number of bus guardians. The exact number of guardians is a subject of analysis. A detailed analysis will

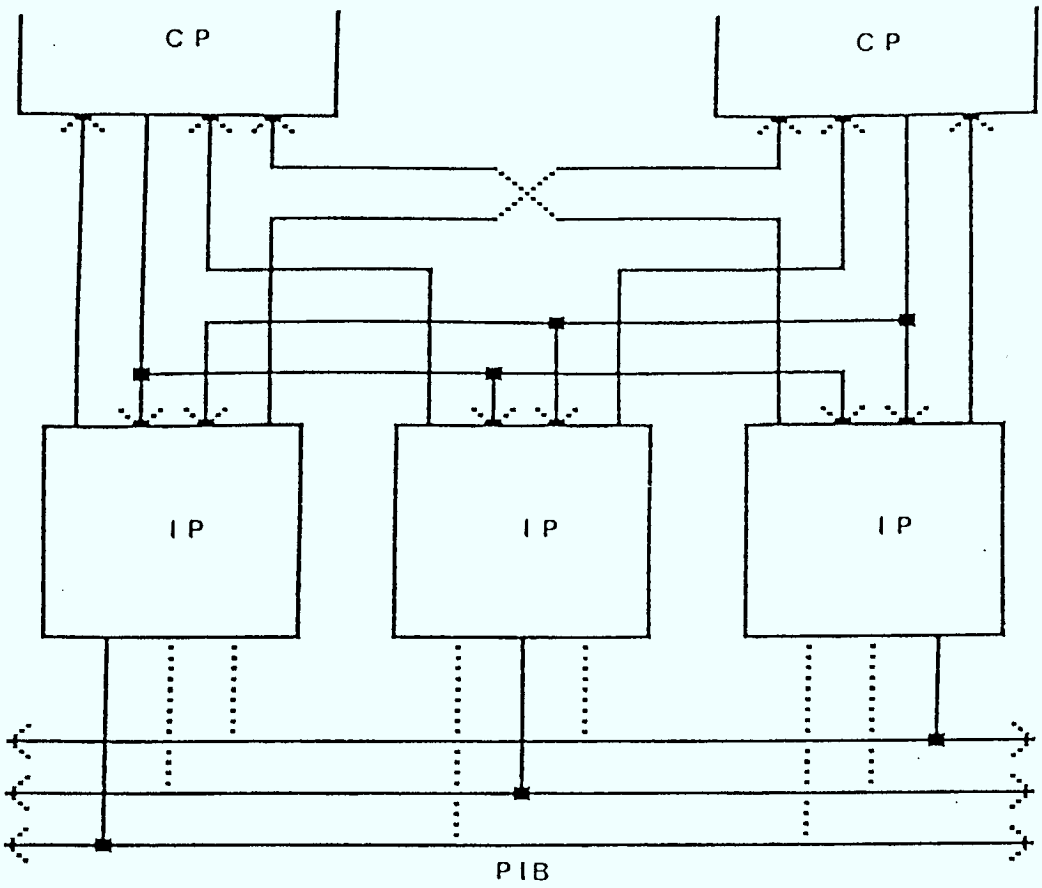


FIGURE 3-3: Peripheral Interface System

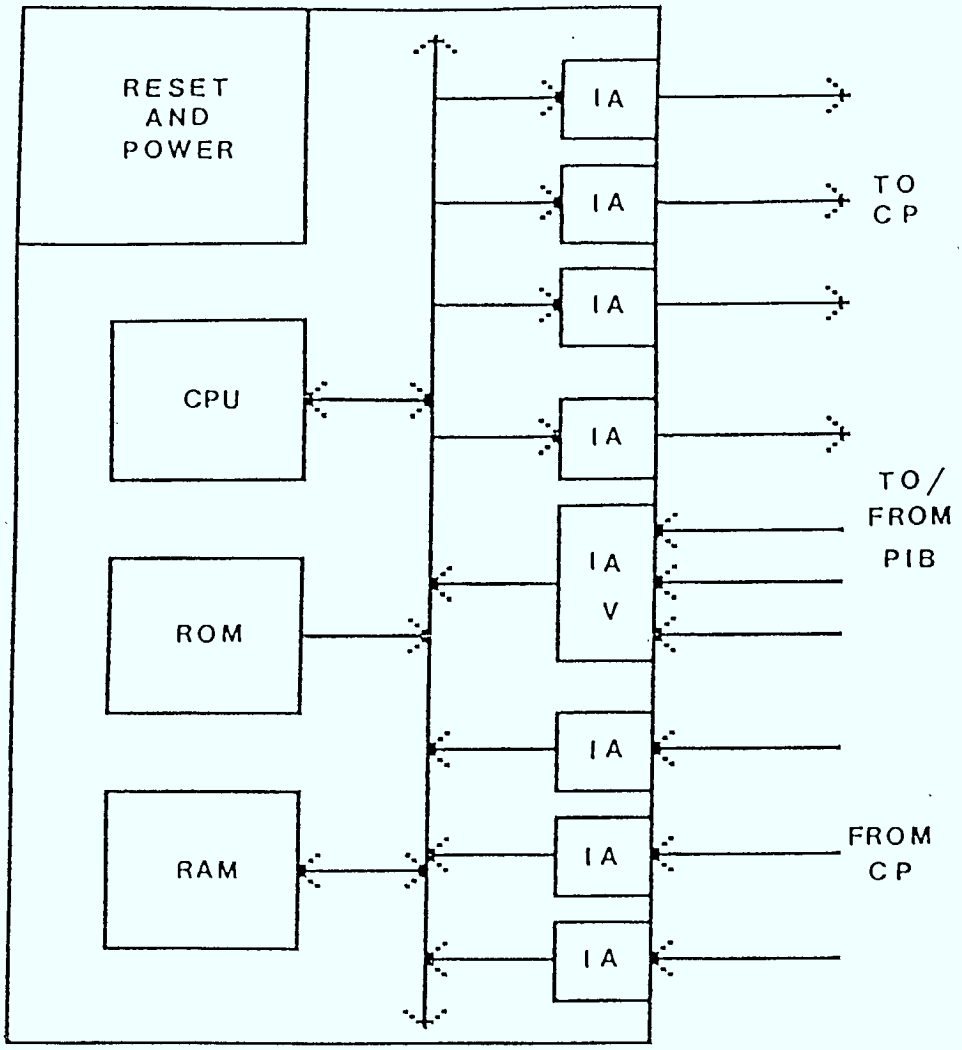


FIGURE 3-4: Interface Processor

be presented in a later chapter. Figure 3.5 shows an interface with two guardians controlling the gate to one of the busses.

Each guardian is directly addressable from the bus and therefore it can be accessed like any other peripheral. For a device to access a bus all the corresponding guardians must be enabled. The guardians will determine when a device will access the bus, which of the redundant busses will be accessed and for how long the device may have control of the bus. This interface is the same for both the interface processors and the peripheral processors and devices. The only difference is that an interface processor is only allowed access to one of the redundant busses at any given time whereas a device is permitted access to all active busses. At power up some of the guardians controlling the access of the interface processors will be automatically activated so that the system can start functioning.

Of particular importance at this point is the design of the bus guardians. The guardian's connections to the redundant bus only allow it to read from the bus. Therefore a faulty guardian cannot directly affect the bus. The guardian is directly addressable from the bus like any other device and is capable of receiving messages from the interface processors via the active bus triad. A message to the guardian contains commands which are latched by the guardian and applied to its outputs until superseded by another command.

A guardian attached to a peripheral processor is programmed so that it only enables the gate it controls for prespecified lengths of time. This ensures that a single faulty processor cannot monopolize the bus. All inactive guardians are programmed to ignore the bus during the time that data may be transferred to/from a peripheral. Thus it is not possible that data will be recognized as commands and cause undesirable

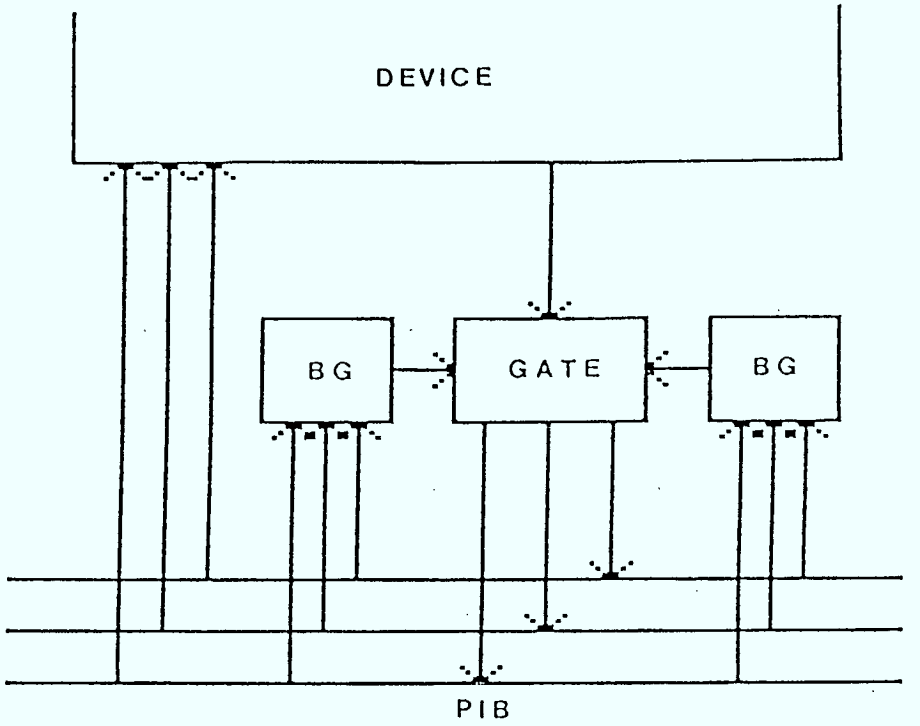


FIGURE 3-5: Bus Interface

state changes to occur.

3.4. Batch Extension

If there is a requirement for on-board high throughput processing then the batch extension subsystem shown in Figure 3.6 can be connected to the fault tolerant multiprocessor system. The main system acts as the controller (CNT) of the extension through an interface on the peripheral bus. The extension is a common bus, loosely coupled subsystem with both private and common memories. The batch extension subsystem does not incorporate any fault tolerant features as such but instead it is controlled and observed by the main system. The operation of this subsystem is as follows :

All tasks and all data are kept in a global data base in common memory. A free processor is assigned the first task in the Ready-to-Run queue. This task and its data is then copied into the processor's private memory and the processor starts executing the task. At predefined intervals the processor communicates with the controller and if all is determined to be well it is allowed to update the global data base and proceed with the task. Otherwise it is disconnected from the subsystem and the task it was executing is again placed in the Ready-to-Run queue. With this technique, only a part of the processing done by the processor before it crashed is lost (the part between the two checkpoints).

This subsystem requires much less hardware than a complete fault tolerant subsystem and can provide a good degree of reliability. Its main weakness is its controller which must be fully fault tolerant. However, by using the main system as the controller this subsystem can provide high processing power at a much lower cost.

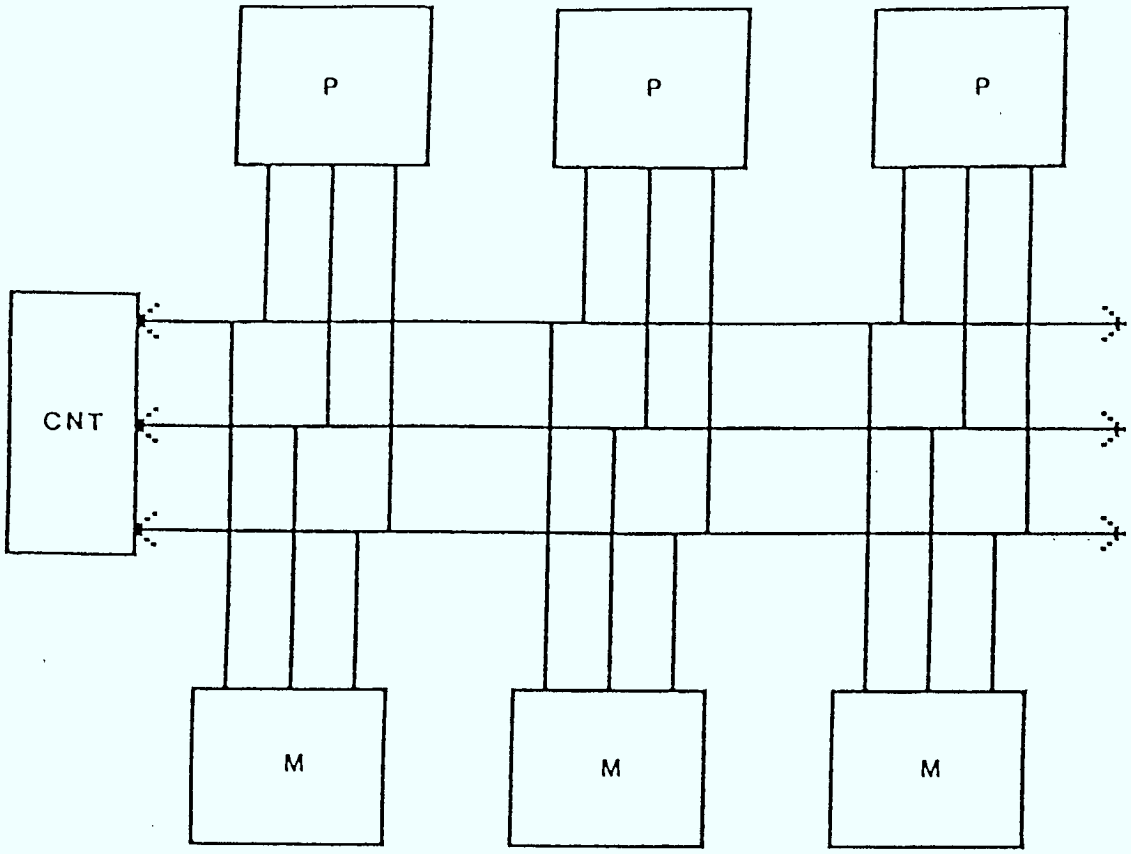


FIGURE 3-6: Batch Extension Subsystem

3.5. System Reset Mechanism

As shown in Figure 3.2 a processor module includes a power control and reset block. This block serves two functions: It provides some means of resetting a suspected processor, thereby forcing it to run its self test routines, and it allows for a faulty processor that has been determined to be unusable to be powered down. A block diagram of the block is shown in Figure 3.7. The block is connected to the peripheral bus like any other device and only consists of two switches that can be controlled from the bus. Since the block is connected to the bus in such a way that it can only read from the bus, the connection of these blocks does not require any special care. Effectively the switches can be controlled by a set of guardians in very much the same way a gate is controlled.

The switches must be designed so that their failure modes will be asymmetrical and in case of a failure they will fail safely. It is also desirable that the failure of any single switch will not incapacitate the processor it controls. This can be accomplished by employing a number of switches in suitable configurations so that single or even multiple switch failures will be masked without any effect on the operation of the block.

Some extensions to this block may allow the use of redundant power supplies with software controlled load switching.

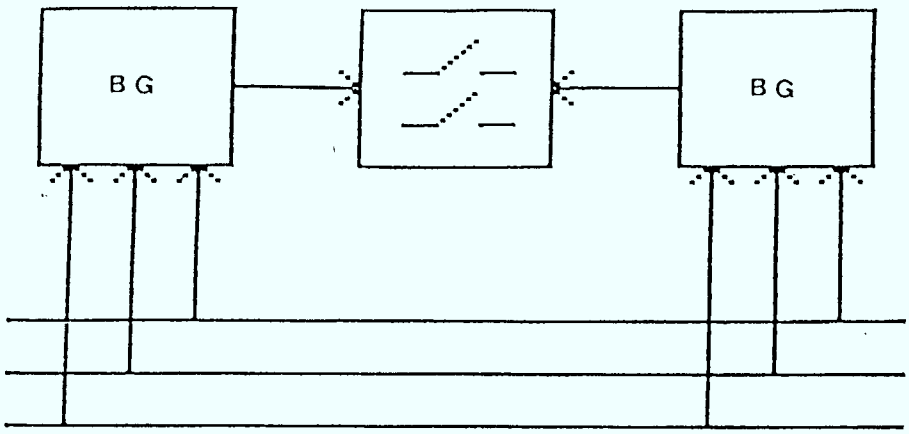


FIGURE 3-7: Reset Mechanism

4. RELIABILITY ANALYSIS OF THE FAULT TOLERANT MULTIPROCESSOR ARCHITECTURE

4.1. Introduction

In this chapter we will describe the analytical model used to determine various design parameters of the architecture. The model predicts the behaviour of the fault tolerant system described in the previous chapter as a function of time. The analysis here is concentrated on the behaviour of the peripheral subsystem (network). The behaviour of the central system is similar to the behaviour of the SIFT hardware [2,3,4]. Since the central system depends on software for fault tolerance it will not be considered here.

A stochastic model has been developed for the behaviour of a gate guarded by a number of guardians. This configuration constitutes a gate complex and is shown in Figure 4.1. This model is then used as the basis for the development of a model that covers the entire peripheral network.

The peripheral network is taken to consist of a redundant bus, and a number of interface processors. The bus can also be accessed by a number of peripheral processors and devices. For the purpose of this analysis we do not have to differentiate between peripheral processors and dumb peripheral devices.

The expected life span of the peripheral network is calculated as a linear function of the expected life span of a single guardian. In this analysis we take the expected life span of a guardian to be 20 years. This is a rather pessimistic assumption considering the present state of the art and the relative simplicity of a guardian (less than 200 logic gates).

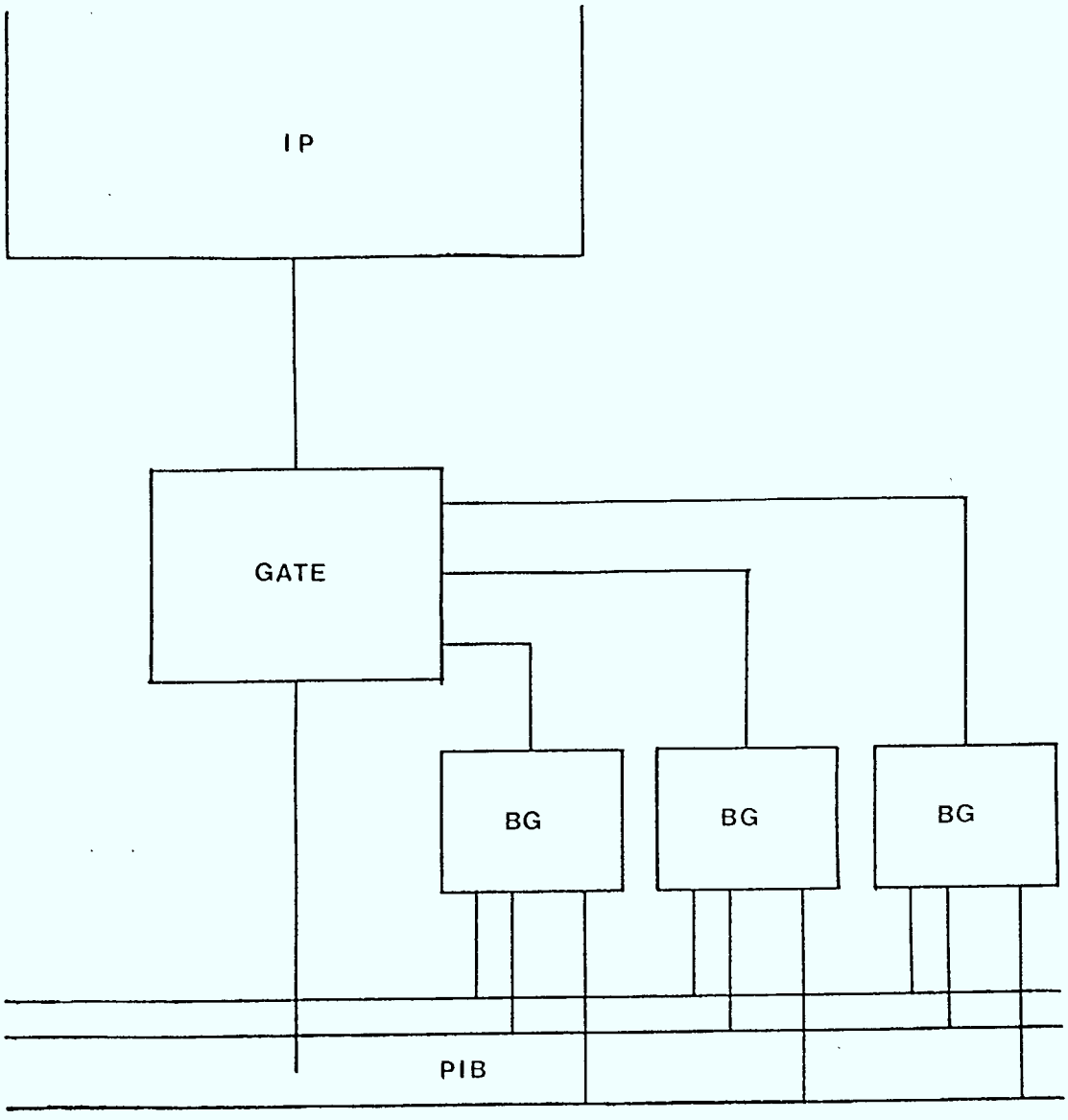


FIGURE 4-1: Gate Complex

4.2. Stochastic Model for Guardian Behaviour

In the proposed system each device and/or processor is connected to the peripheral network through a gate. The operation of the gate is controlled by a number of guardians as depicted in Figure 4.1. The guardians may permit the processor to write to the bus or may prevent it from doing so depending on the commands they receive from the interface processors. What we are interested in here is the probability that a device is still usable (i.e. an active and useful part of the computer system) along with the various modes of failure and their impact on the operation of the remaining system.

Starting the analysis we must determine the behaviour of the gate complex shown in Figure 4.1 when certain of its parts fail. For this analysis we will omit the gate itself and concentrate on the guardians. The impact of a gate failure can be incorporated in the analysis by an appropriate modification of the probabilities of failure of the guardians. Considering the fact that the hardware of the gate is much simpler than that of a guardian (less than 5 logic gates) the gate can be ignored without any noticeable impact on the results.

The following assumptions are made :

- At time zero all guardians function properly.
- Guardians fail independently of each other according to an exponential distribution and with a failure rate lambda.
- A guardian fails in the ON state with probability alpha and in the OFF state with probability (1-alpha).
- A guardian that has failed will not change state at a later time. For example a guardian that failed in the ON state will remain in that state forever. Also all failures are permanent.

The state in which a guardian fails is independent of the guardian's state prior to its failure. The state of the guardian before the failure is of no consequence to this analysis and is therefore ignored. Failure in the ON state means that the output of the guardian is such as to try to enable the gate. Similarly failure in the OFF state means that the output is such as to try to disable the gate.

Under the above assumptions the failures of the guardians in a gate complex like the one shown in Figure 4.1 can be considered as a pure death Markovian process [8]. If there are k guardians in the gate complex the process has k+1 states (numbered from 0 to k) and the transition diagram is shown in Figure 4.2. The physical meaning of state L is that L guardians have failed. The probability of a state L as a function of time can be calculated and is :

$$P(L) = C(k,L) * \sum_{i=k-L}^{k} (C(L,k-i) * (-1)^{(i-k+L)} * e^{(-i*lambda*t)}) \qquad (1)$$

where L = 0, 1, 2, ... k

Besides the number of guardians that have failed it is of great importance to know the number of guardians that have failed in the ON state. To incorporate that into the analysis we break each state of the Markov chain into substates. Thus state L is separated into L+1 substates, namely L0, L1, L2, ... LL. The physical meaning of substate Lm is that L guardians have failed m of which have failed in the ON state. Thus the state diagram of the process becomes that shown in Figure 4.3. The probability of each substate can be denoted as P(L,m).

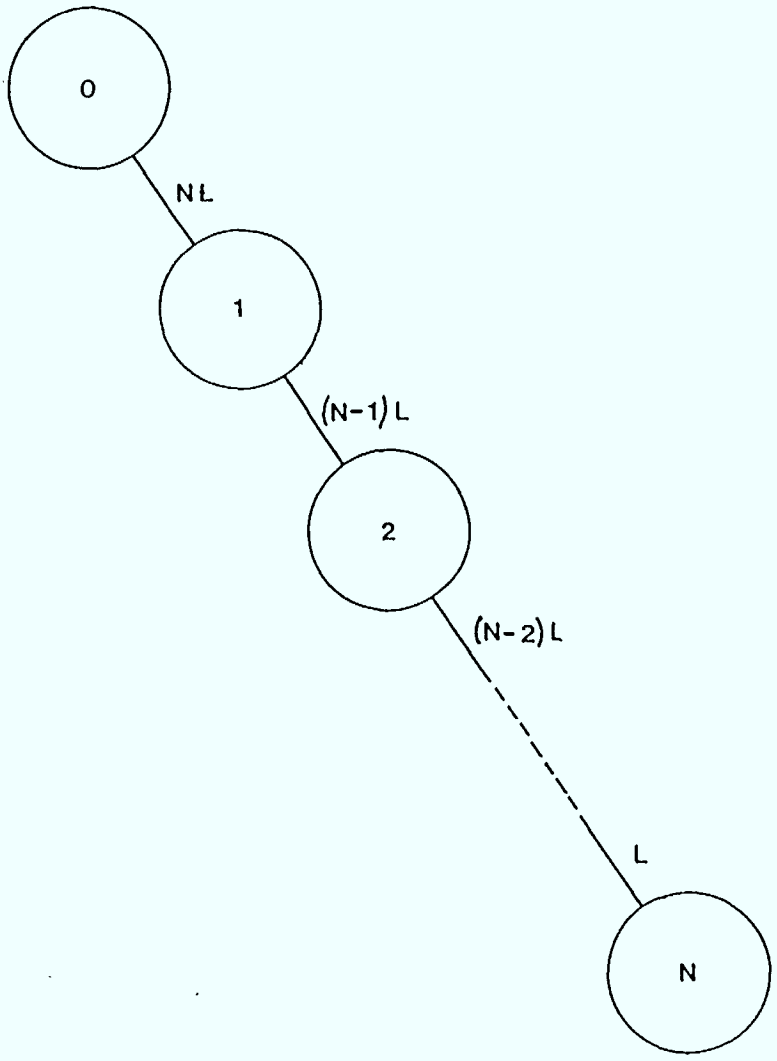


FIGURE 4-2: Transition Diagram of Markov Process for a Gate Complex

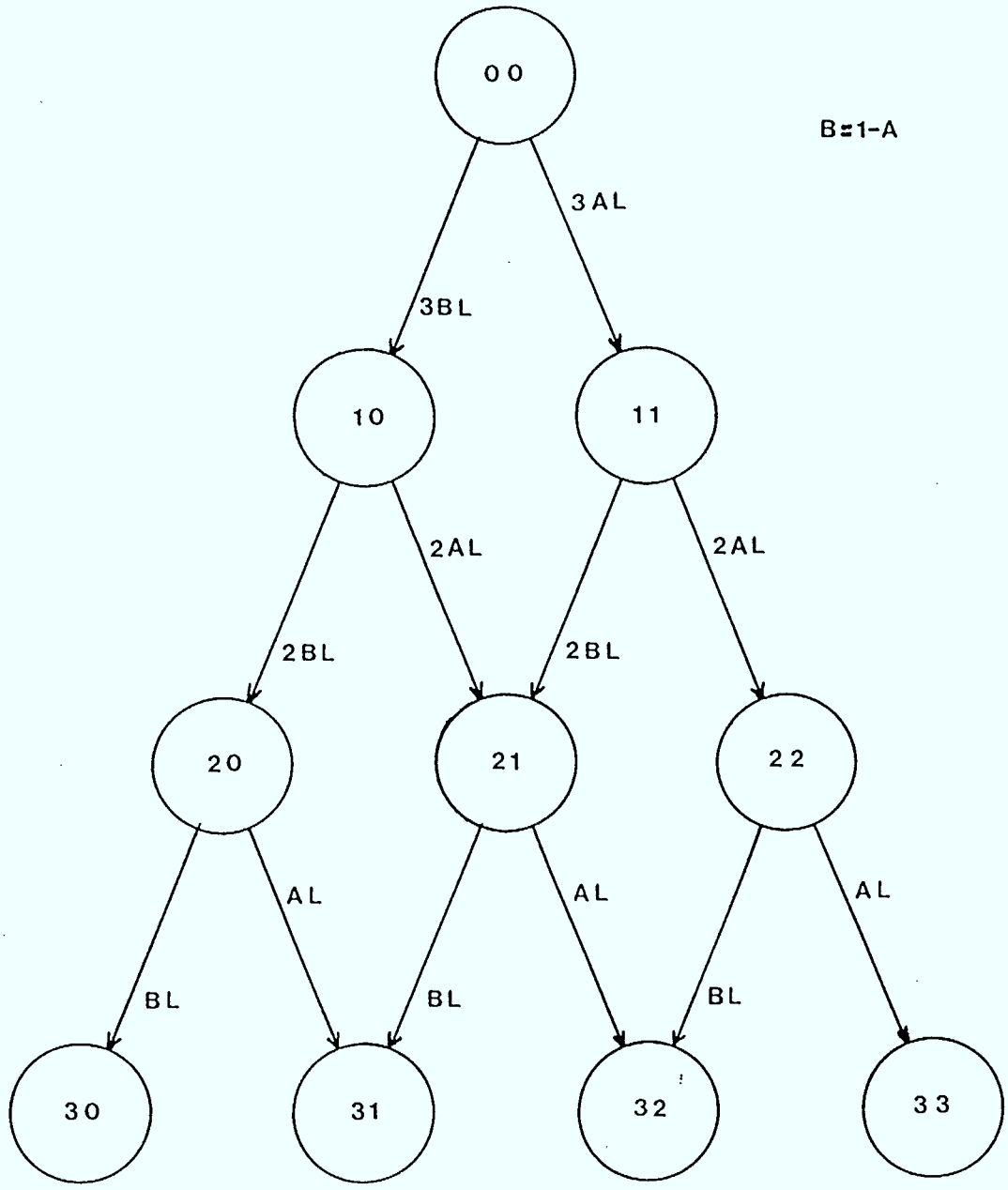


FIGURE 4-3: Transition Diagram of Complete Random Process

P(L,m) can be calculated from the formula :

$$P(L,m) = P(L) * P(m \mid L) \qquad (2)$$

and the conditional probability P(m | L) is :

$$P(m \mid L) = C(L,m) * \text{alpha}^{m} * (1-\text{alpha})^{(L-m)} \qquad (3)$$

By combining (1) and (3) we obtain the general formula for the probability of each substate :

$$P(L,m) = C(k,L) * C(L,m) * \text{alpha}^{m} * (1-\text{alpha})^{(L-m)} * \sum_{i=k-L}^{k} (C(L,k-i) * (-1)^{(i-k+L)} * e^{(-i*\text{lambda}*t)}) \qquad (4)$$

where $L = 0, 1, 2, \ldots k$
$m = 0, 1, 2, \ldots L$

From (4) we can calculate the probability of any state for any given size of a gate complex. However, in order to determine the overall behaviour of the gate complex we must also know how the guardians enable the gate. The two possible ways are by agreement and by majority voting. By agreement means that all guardians must agree for the gate to be enabled. By majority voting means that the gate will do whatever the majority of the guardians dictates.

Depending on the decision mechanism (agreement or majority) different states of the complex have different effects. The gate may be in any one of three states. These are :

- Functioning Normally
- Permanently OFF
- Permanently ON

If the gate is permanently OFF the corresponding processor is incapable of reaching the bus. Therefore it is effectivelly removed from the system. On the other hand if a gate is permanently ON the corresponding processor has unrestricted access to the bus. This may not be a problem unless the processor also malfunctions and starts an endless transmission. In order to accomodate the worst possible case we assume that when a gate is permanently ON the corresponding processor is always malfunctioning. Note that a gate may be functioning normally although some of its guardians have failed. For example in a gate complex with three or more guardians and decision by majority the failure of a single guardian will go unnoticed.

The effect of the decision mechanism to the behaviour of the gate can be understood by considering the relation of the state of the guardians to the state of the gate as a function of the decision mechanism. Figure 4.4 shows this relation for a gate complex with three guardians operating by agreement, while Figure 4.5 shows the same relation if the gate complex operates by majority.

Finally, the effect of the value of alpha to the failure rate lambda must be considered. If no care is taken the value of alpha will be about 0.5 (i.e. if a guardian fails there is a 50% probability that it will fail in the ON state). This value can be changed by altering the design of the guardian. However, the alternate design will require a larger number of components for each guardian and this in turn will increase the failure rate lambda. Since the component count of a device increases logarithmically with the number of states of the device, and alpha is inversely proportional to the number of states it follows that the failure rate lambda must be altered according to the formula :

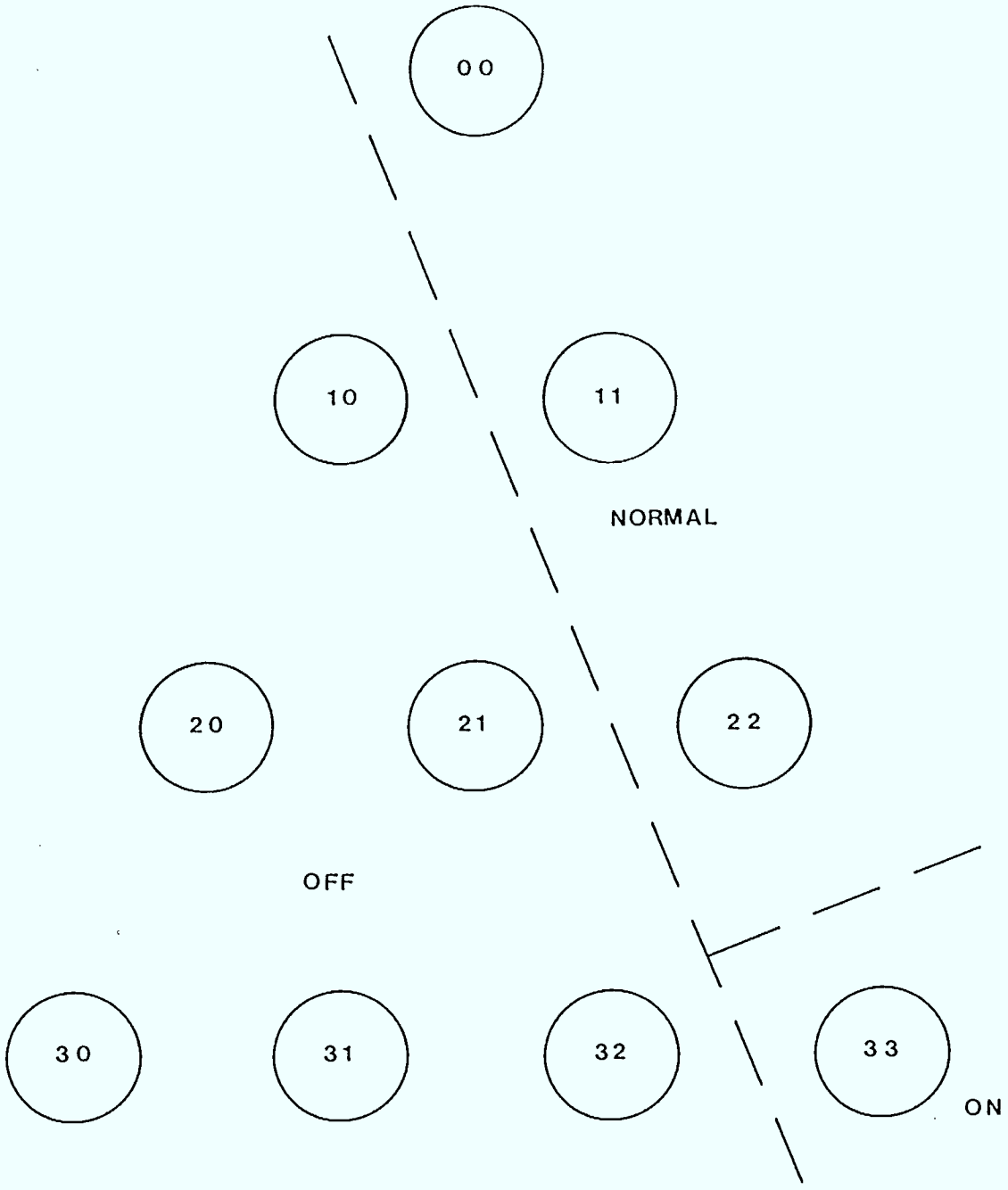


FIGURE 4-4: State Meaning in Operation with Agreement

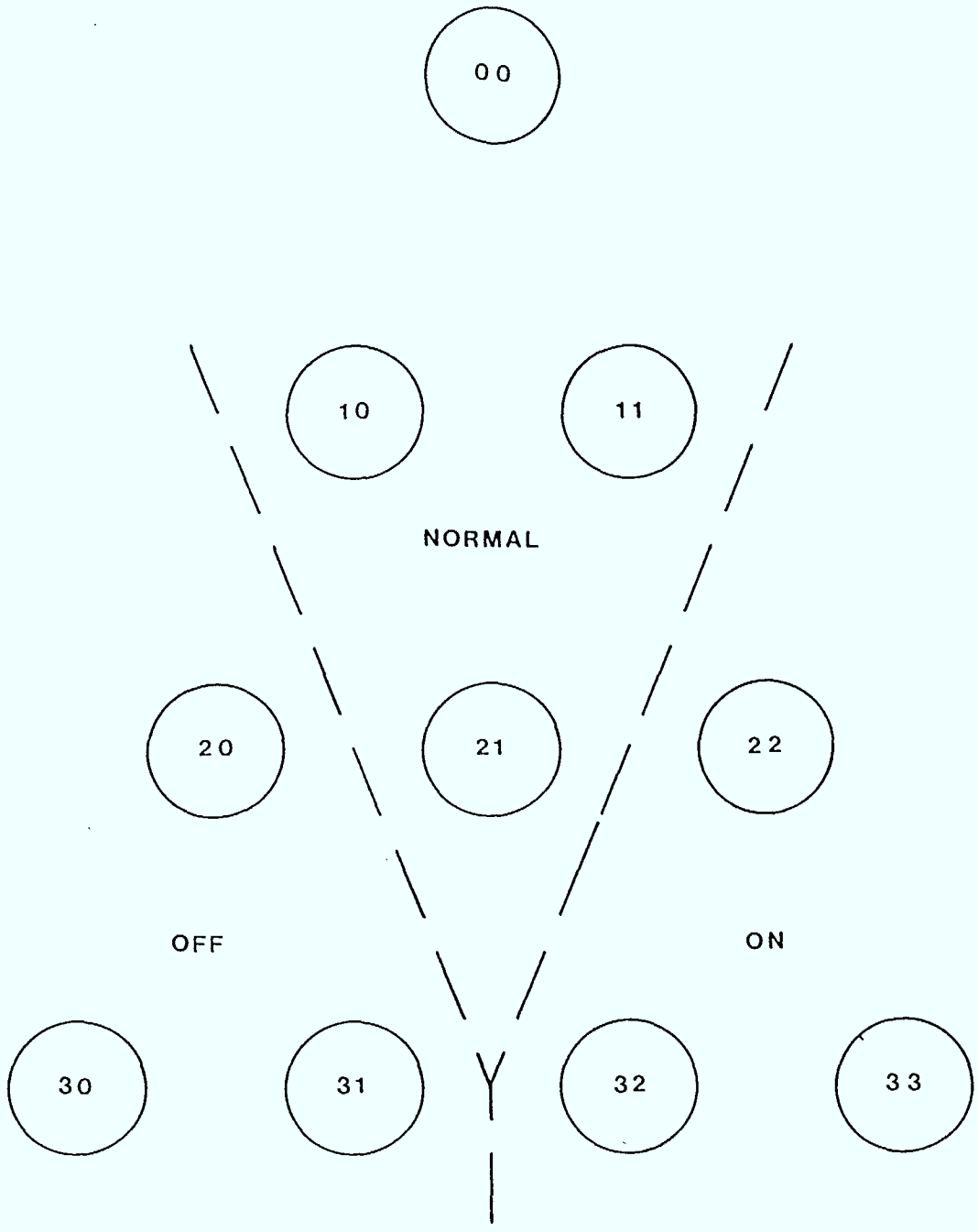


FIGURE 4-5: State Meaning in Operation with Majority

$$\text{lambda effective} = \text{lambda} * (s + \log_{2(0.5/\text{alpha})}) / s \qquad (5)$$

where s is the logarithm base 2 of the number of states of an unmodified guardian. In order to cover the worst case we assume s = 1 (i.e 2 states). Therefore the failure rate lambda is altered according to the formula :

$$\text{lambda effective} = \text{lambda} * (1 + \log_{2(0.5/\text{alpha})}) \qquad (6)$$

The following configurations of the gate complex were considered :

- 2 guardians with agreement
- 3 guardians with agreement
- 4 guardians with agreement
- 5 guardians with agreement
- 3 guardians with majority
- 5 guardians with majority
- 7 guardians with majority

For each of these configurations the probabilities of each of the gate states defined above was calculated as a function of time. For each configuration the value of alpha was varied from 0.1 to 0.5 in steps of 0.1. Representative results of this analysis are included in Appendix A. Appendix B shows plots of the probabilities of a device functioning normally as a function of time for various values of alpha. We can see that as the value of alpha increases this probability decreases. On the other hand the probability of getting the system in an undesirable state also decreases.

Therefore in making the decision for a certain configuration of guardians in the gate complexes we must not only consider the

probability of the bus remaining usable for the longest possible time but also the probability that a minimum required number of devices will remain usable. To do that we must develop a model that shows the behaviour of the entire system rather than a single gate complex. Clearly this model will be based on the model of the gate complex developed in this section.

4.3. Stochastic Model for the Behaviour of the Peripheral Network

The heart of the peripheral network is a redundant bus as shown in Figure 4.6. These busses interconnect the interface processors to the peripheral processors and the peripheral devices. For the purpose of this analysis the terms peripheral processors and peripheral devices can be used interchangeably. The peripheral network can be considered as consisting of the following parts :

- Redundant bus
- Interface Processors
- Essential Devices (ED)
- Non-essential Devices (NED)

An essential device is a device that must be functional for the system to be functional. It is assumed that such devices are always in pairs and the device is considered functional if at least one of the two devices is functional. A non-essential device is a device that is not necessary for the system to function. In other words a non-essential device performs some secondary function that can be discontinued without seriously affecting the operation of the system. Non-essential devices may be single units or in pairs. For the purpose of this analysis a pair of non-essential devices can be considered as two independent devices.

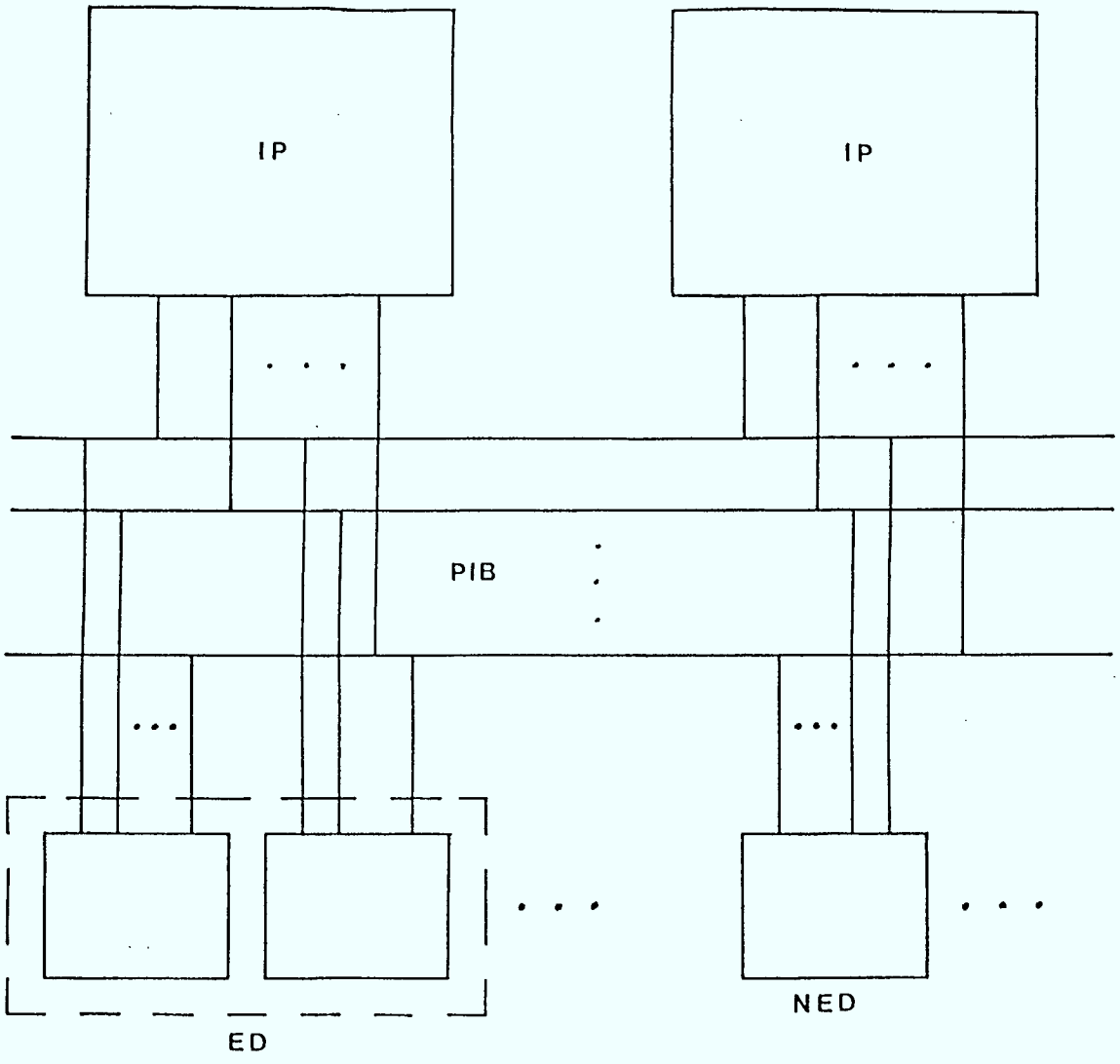


FIGURE 4-6: Peripheral Network

The peripheral network, and thereby the system, becomes inoperational if one or more of the first three parts fails. That is if the redundant bus becomes unusable, if the interface processors become unoperational or if one or more pairs of essential devices fail. The probability of each of these to occur will be calculated separately and then combined to give the probability of failure of the entire peripheral network.

4.3.1. Stochastic Model For The Behaviour Of The Redundant Bus

The redundant bus consists of a number of busses used in parallel. Under normal operating conditions three of these busses are active and are used for the transfer of information with error detection and masking. In a marginal case the operation of the redundant bus can be maintained even if only two of the busses are usable. This is because the third bus (any one of the remaining unusable busses) will be voted out of the result. However this configuration will fail to mask a soft error as was the original intent.

Therefore for the redundant bus to be usable at least two of the busses that comprise it must be usable. The probability of a single bus being unusable is the probability that one or more gates connected to this bus have failed in the permanently ON state. This probability can be calculated as :

$$P_{1b} = \sum_{i=1}^{nd+nip} (C(nd+nip,i) * P_{ON}{}^{i} * (1-P_{ON})^{(nd+nip-i)}) \qquad (7)$$

where nd is the number of devices
nip is the number of interface processors
P_{ON} is the probability of a gate being permanently ON

From this probability we can calculate the probability of the redundant bus being unusable. This is the probability that of the busses that comprise it at least nb-1 are unusable and can be calculated as :

$$P_b = \sum_{i=nb-1}^{nb} (C(nb,i) * P_{1b}{}^{i} * (1-P_{1b})^{(nb-i)}) \qquad (8)$$

where nb is the number of busses in the redundant bus

4.3.2. Stochastic Model For The Behaviour Of The Interface Processors

The function of the interface processors is to handle the communication between the asynchronous central system and the synchronous peripheral network. At any given time three of the interface processors are active and each of them drives one of the active busses. For the system to be functional at least two of the interface processors must be operational by a reasoning similar to the one used for the redundant bus. For this analysis an interface processor is considered to be inoperational if it has been cut off from the bus. This probability can be calculated as :

$$P_{lip} = P_{OFF}{}^{nb} \qquad (9)$$

where P_{OFF} is the probability of a gate being permanently OFF

The probability of the interface processors being inoperational can then be calculated as :

$$P_{ip} = \sum_{i=nip-1}^{nip} (C(nip,i) * P_{lip}{}^{i} * (1-P_{lip})^{(nip-i)}) \qquad (10)$$

where nip is the number of interface processors

4.3.3. Stochastic Model For The Behaviour Of The Essential Devices

Since the hardware connection of a device to the bus is identical to that of an interface processor, the calculation of the probability of a device being cut-off is similar to the calculation of that probability for an interface processor . This probability can be calculated for a single device as :

$$P_{1dc} = \sum_{i=nb-1}^{nb} (C(nb,i) * P_{OFF}{}^{i} * (1-P_{OFF})^{(nb-i)}) \qquad (11)$$

From this probability we can calculate the probability of both devices in a pair failing as :

$$P_{1dpc} = P_{1dc}{}^{2} \qquad (12)$$

Now the probability of the system failing due to the failure of a pair of essential devices can be calculated as :

$$P_{ed} = \sum_{i=1}^{ned} (C(ned,i) * P_{1dpc}{}^{i} * (1-P_{1dpc})^{(ned-i)}) \qquad (13)$$

where ned is the number of essential device pairs

4.3.4. Stochastic Model For The Behaviour Of The Peripheral Network

The peripheral network will become unusable if any one of the three cases listed above occurs. Since we know the probability of each case to arise and the various cases are independent since they depend on

different equipment, we can calculate the probability of failure of the peripheral network as :

$$P_{net} = P_b + P_{ip} + P_{ed} - (P_b * P_{ip}) - (P_b * P_{ed}) - (P_{ip} * P_{ed}) + (P_b * P_{ip} * P_{ed}) \quad (14)$$

Appendix C shows the values of these probabilities as a function of time for some typical cases. The peripheral network considered consisted of :

5 Interface processors

4 Essential Device Pairs

7 Non-essential Devices

Appendix D shows the values of these probabilities after ten years of operation of the system for all combinations of number of guardians per gate, decision mechanism and alpha. From these values we concluded that the optimum configuration is 3 guardians per gate with agreement. The optimum value for alpha was found to be 0.5. This configuration is optimum not only in terms of reliability but also in terms of cost.

5. DETAILED HARDWARE DESCRIPTION

5.1. Introduction

In this chapter we will describe in detail the hardware specified in the previous chapters. The proposed system consists of a relatively small number of distinct parts that are then combined to form a full fledged fault tolerant system. First each of these parts will be described separately. Then the interconnection of these parts into a fault tolerant system will be described.

First, the guardians and gates will be presented in detail. Section 5.2 will present a detailed explanation of the operation of a guardian. Section 5.3 will present the gate and will deal with the combination of guardians and gates to form a gate complex.

Section 5.4 will deal with the intermodule communications. These include the communication between the processor modules that constitute the central control system along with the communication of these processor modules with the interface processors. The communication ports used will be described at the block level. Section 5.5 will present the details of the communication ports employed.

Section 5.6 will deal with the implementation of the processor modules and the interface processors. The hardware configuration of both will be presented. Section 5.7 will deal with the connection of the interface processors and the peripheral processors to the redundant bus which is the heart of the peripheral network. The use of gate complexes will be discussed in detail. Finally section 5.8 will deal with the interconnection of all these parts into a full fault tolerant system.

5.2. Guardians

Basically a guardian is a simple finite state machine the state diagram of which is given in Figure 5.1. This FSM has a number of inputs and a single output. Normally a guardian is idle with its output set so as to disable the gate. At this state the guardian monitors the bus until it recognizes a command on it. The list of commands a guardian can recognize is given in Table 5.1. Except for the special case of a change in the triad of active busses, a guardian will remain idle until it recognizes either a Select or an Enable command. When a Select command is detected the guardian sets an internal flag and awaits an Enable command. As soon as the Enable command is received the guardian sets its output so as to enable the gate and leaves it enabled for a fixed period of time. At the end of this time period the guardian disables the gate and returns to its idle state.

If an Enable command is detected but the guardian has not received a Select command then the guardian does not enable the gate. However it still waits for the same fixed period of time. While waiting the guardian ignores the bus and any information on it. Thus it is not possible for some data on the bus to be interpreted as commands and cause erroneous operation of the system.

The operation described above refers to a guardian attached to a peripheral processor or a peripheral device. Guardians attached to the interface processors have a slightly different state diagram as shown in Figure 5.2. The commands they understand are also slightly different and are given in Table 5.2. Such a guardian will still ignore the bus for a fixed period of time after it detects a Turn On command. However once activated it remains on until explicitly turned off. Thus an interface processor that has been given control of a bus maintains this

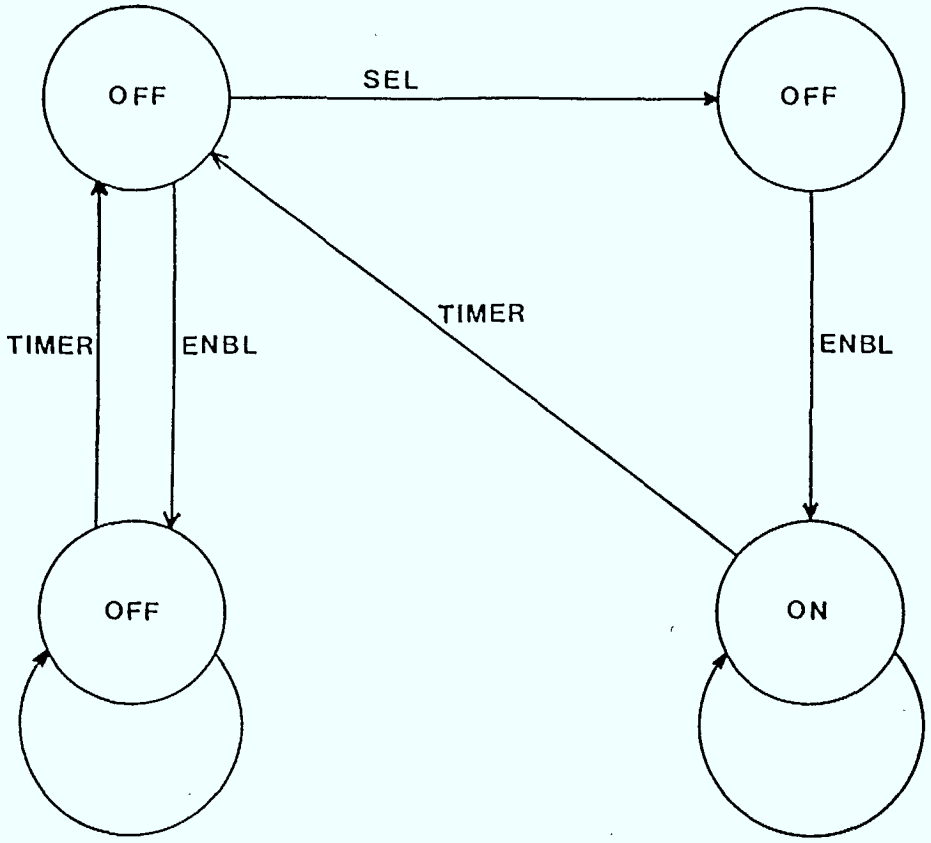
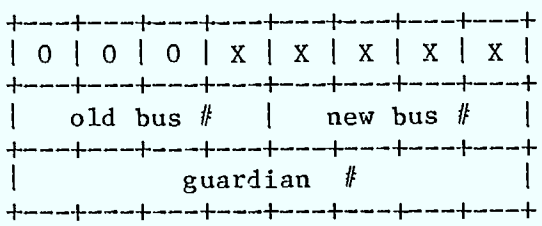


FIGURE 5-1: State Diagram of Peripheral Processor Guardian

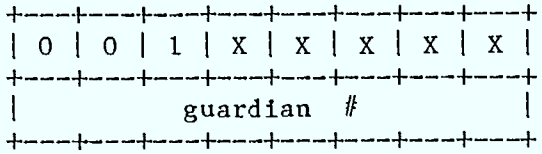
control except for the short periods of time that it grants the bus to a peripheral for some data transfer.

Figure 5.3 shows the outputs of the two types of guardians as a function of time and of the commands received. We can observe how the guardian of a peripheral processor is normally OFF whereas the guardian of an interface processor is normally ON. When the guardian of the peripheral processor is selected its output does not change. However when the Enable command is sent its output is activated and remains so for a fixed period of time. During this time all other guardians in the system disable their outputs after they recognize the Enable command.

Switch Bus



Select



Temporary Enable

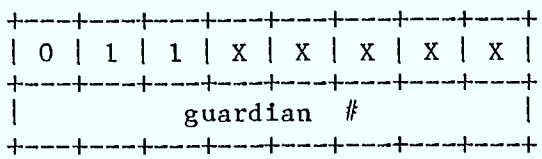


Table 5-1: Commands for Peripheral Processor Guardians

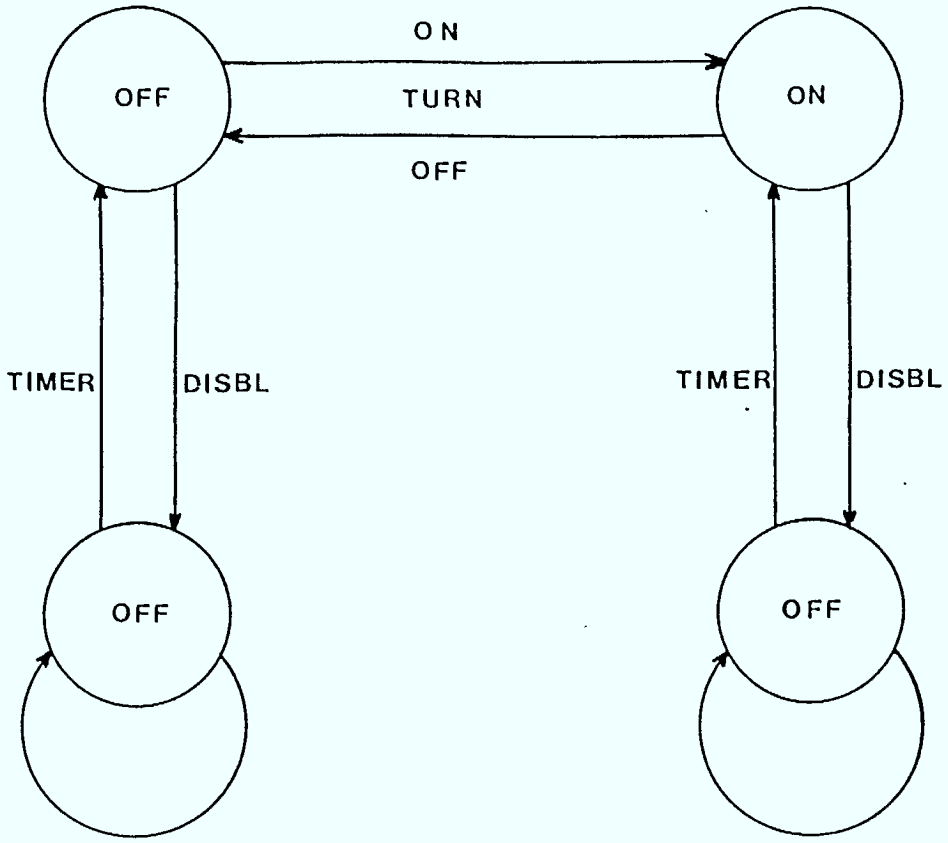
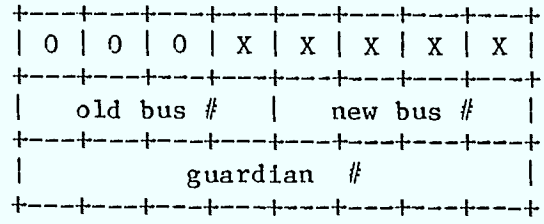
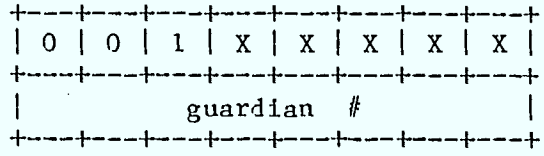


FIGURE 5-2: State Diagram of Interface Processor Guardian

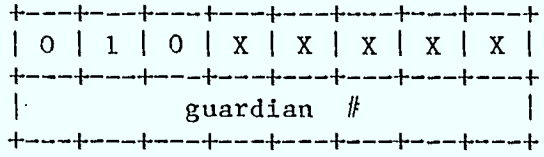
Switch Bus



Turn ON



Turn OFF



Temporary Disable

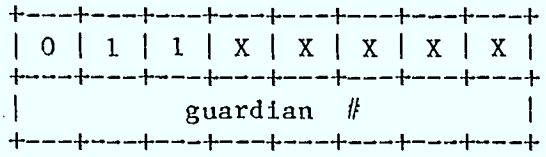


Table 5-2: Commands for Interface Processor Guardians

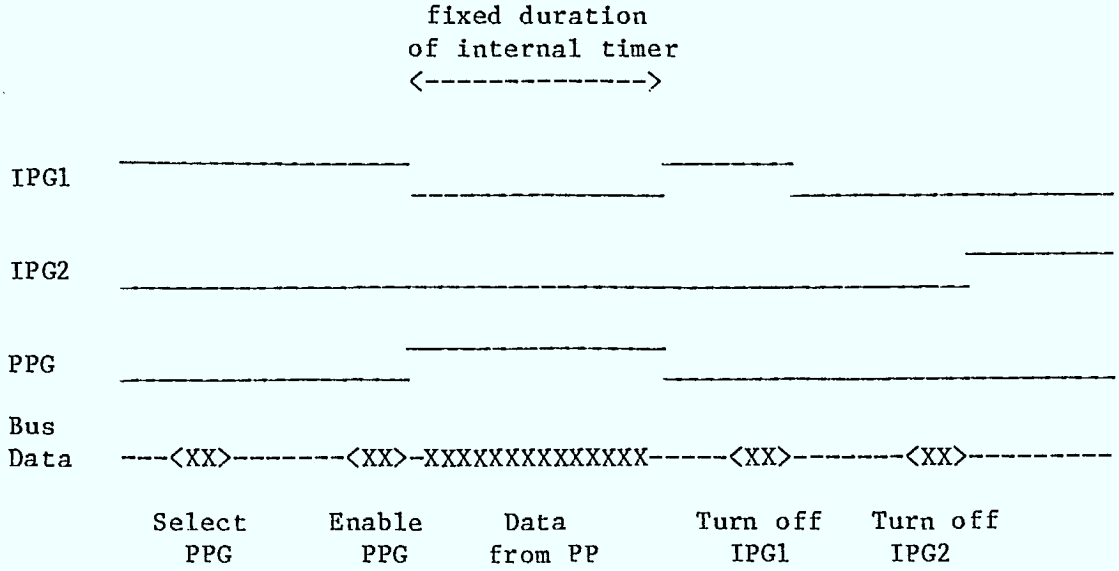


FIGURE 5-3: Example of Guardian Operation

Note that a guardian attached to a peripheral always starts in the OFF state whereas a guardian attached to an interface processor may start either in the OFF or in the ON state. This is controlled by a jumper or a switch. Some guardians always start in the ON state. Namely the guardians that start in the ON state are selected so that each of the three initially active interface processors will have access to one of the initially active busses. This is essential for the system to be operational after initialization.

5.3. Gates and Gate Complex

A gate is a simple tri-state buffer with multiple enable lines. Since the peripheral busses are serial the gate only needs to control one line. A gate can be implemented simply as a combination of two standard TTL gates as shown in Figure 5.4.

A gate along with a number of guardians form a gate complex. The function of a gate complex is to allow a processor or a device to access a bus in a controlled and fault tolerant fashion. The block diagram of a gate complex is shown in Figure 5.5.

The complex consists of a gate and three guardians. Each processor connected to the peripheral network is connected to each of the busses that form the redundant bus via a distinct gate complex. It is essential that no two gate complexes have any hardware in common so that hardware failures are as local as possible. Thus it is not possible to utilize a quad buffer chip to implement four gates. Each gate must be physically separate from any other gate. Note that the guardians of a gate complex can be made so that they all have the same address. This is possible because they always have to receive the same commands. Subsequently this reduces the control overhead.

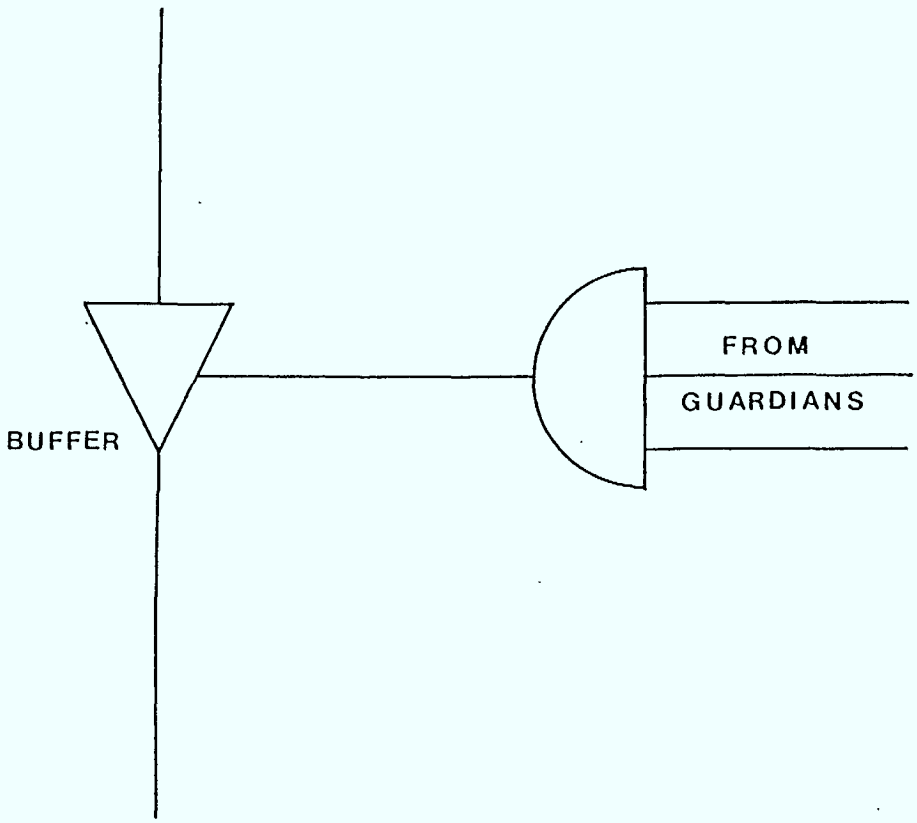


FIGURE 5-4: Implementation of a Gate with Agreement

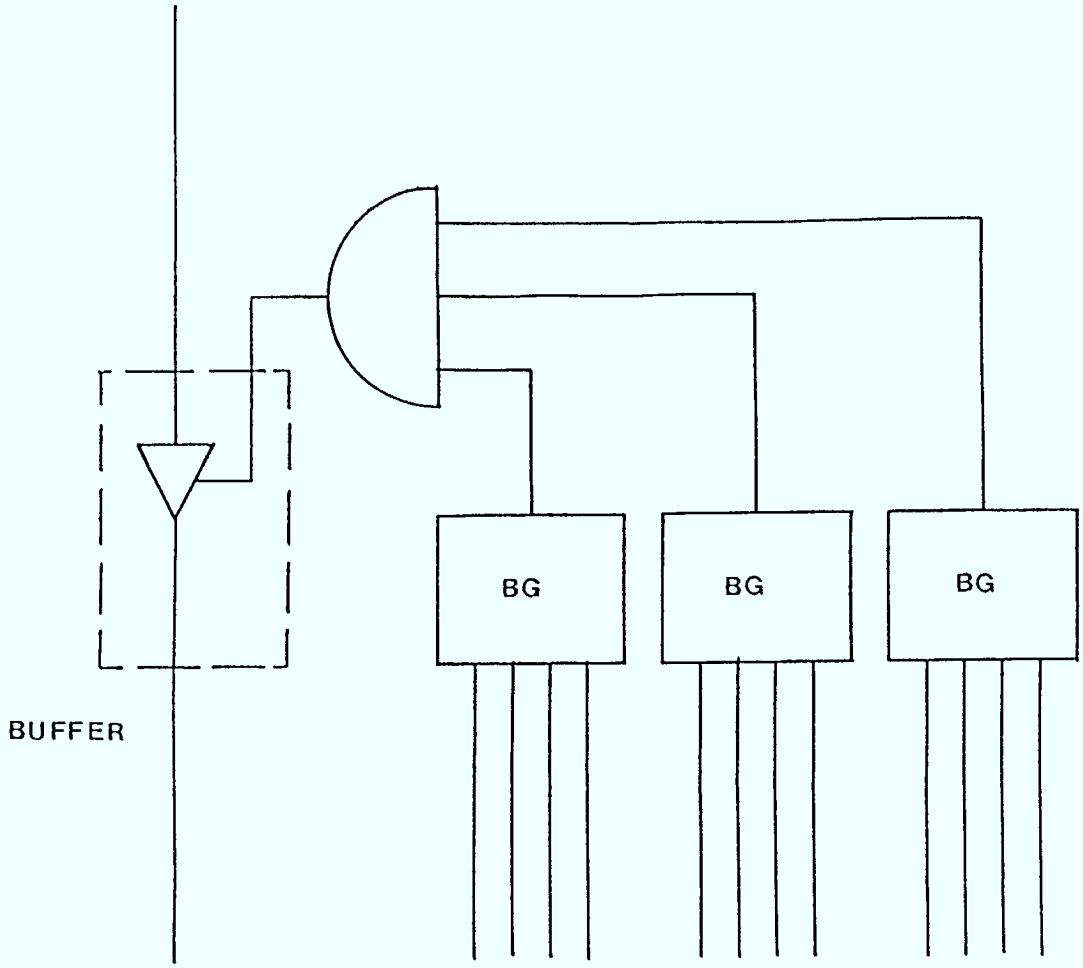


FIGURE 5-5: Block Diagram of a Gate Complex

5.4. Intermodule Communications

In this section we will present the ways in which the various modules of the fault tolerant system communicate. Communication is effected through a number of communication ports. Depending on their use communication ports may differ but they all consist of the same basic units, namely a transmitter and a receiver.

Communication ports intended to connect processor modules to other processor modules consist of both a transmitter and a receiver. These operate independently of each other. Each processor module has a dedicated port connected to every other processor module so that a fully interconnected system is formed. The detailed description of a transmitter and a receiver will be given in the next section.

The communication ports on the interface processors that connect them to the processor modules of the central control system are identical to the ports that interconnect the processor modules to each other. However the ports that connect the processor modules to the interface processors and the ports that connect the interface processors and the peripheral processors to the redundant bus are different. These ports consist of a single transmitter and a number of receivers. In both cases the transmitter broadcasts to all concerned units (either interface processors or busses). However there is a dedicated receiver for each distinct unit that is connected.

In the case of the processor modules the single transmitter ensures proper communication between the loosely coupled central control system and the tightly coupled interface processors. In other words it ensures that all the interface processors will receive their copies of the message simultaneously. This is essential for the interface processor to remain in perfect synchronization. Even if the central processors

were synchronous the time difference between sending the same message to different interface processors would be enough to throw them out of synchronization. In the case of the interface processors the processors need not know to which of the busses in the redundant bus they are connected. So they simply transmit through a single transmitter and the gate complexes ensure that only the proper bus is driven. This also applies to peripheral processors and is shown in Figure 5.6.

5.5. Transmitter and Receiver

The block diagram of a transmitter is shown in Figure 5.7. The transmitter consists of a FIFO queue and a serializer that converts the parallel data into serial data ready for transmission. The serializer can be thought of as the transmitter part of a UART. The FIFO serves to offload the processor from continuously checking the transmitter and to give the transmitter the capability of autonomous operation for a certain length of time. The size of the FIFO is chosen so that the transmitter can operate without any attention from the processor for the duration of the time slice of a task. In other words the transmitter requires the attention of the processor only during a context switch. This greatly simplifies the design and implementation of the operating system.

For example a processor module that wishes to send a message to another module can write the message into the transmitter FIFO and then proceed with its other tasks while the transmitter is sending the message. The processor sees the transmitter as a pair of I/O ports. One is a data port and the other is a command port. Data written into the data port are placed in the FIFO and transmitted. The command port enables the processor to control the transmitter and to check its

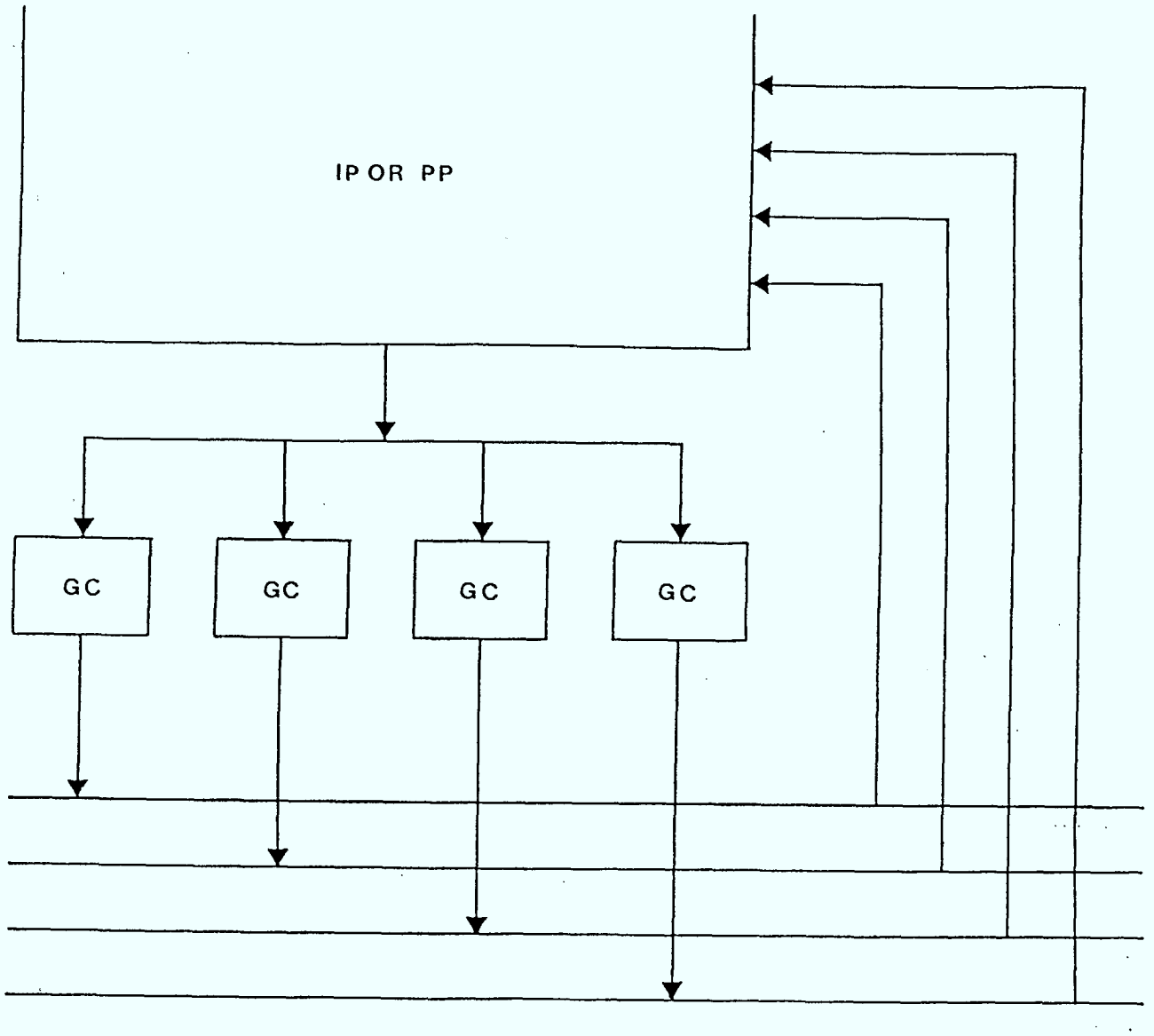


FIGURE 5-6: Connection of Processors to the Redundant Bus

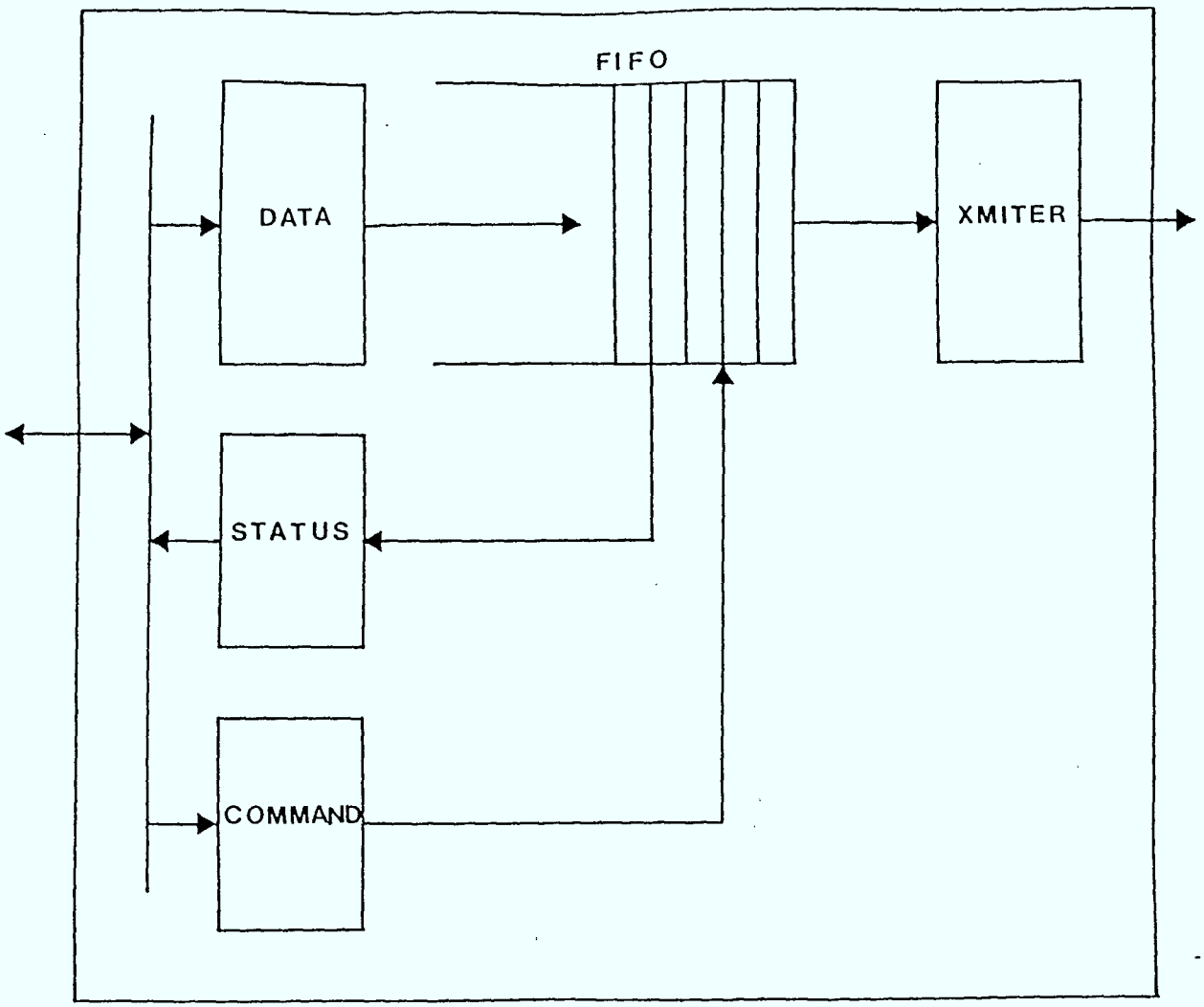


FIGURE 5-7: Block Diagram of a Transmitter

internal status. For the time being we are only interested in being able to reset the transmitter and to check if the queue is full.

The structure of the receiver is complementary to that of the transmitter and is illustrated in Figure 5.8. The receiver consists of a deserializer and a FIFO. The deserializer is actually the receiver part of a UART. When a byte of data has been collected it is placed in the FIFO for pickup by the processor. Like the transmitter, the receiver appears to the processor as a pair of ports. One port is a data port and the other a command port. A read from the data port will return the first byte in the FIFO. If the FIFO is empty the result is undefined. The command port enables the processor to control the receiver and check its status. Presently we need to be able to reset the receiver and check if the queue is empty or if it has overflowed.

5.6. Description of Processor Modules and Interface Processors

The block diagrams of a processor module and an interface processor are given in Figures 3.2 and 3.4. Besides the CPU each module includes some RAM memory and some ROM memory which will contain the kernel of the operating system. Each module also contains a set of programmable interval timers which are essential for some of the operating system functions, an interrupt controller, and a number of communication ports as described above. These communication ports will interconnect the modules into a fully interconnected central control system. A block diagram of the interconnection between two processor modules is given in Figure 5.9.

The modules include as well a single transmitter and a number of receivers that handle the communication with the interface processors. This special interface is necessary because of the synchronous nature of

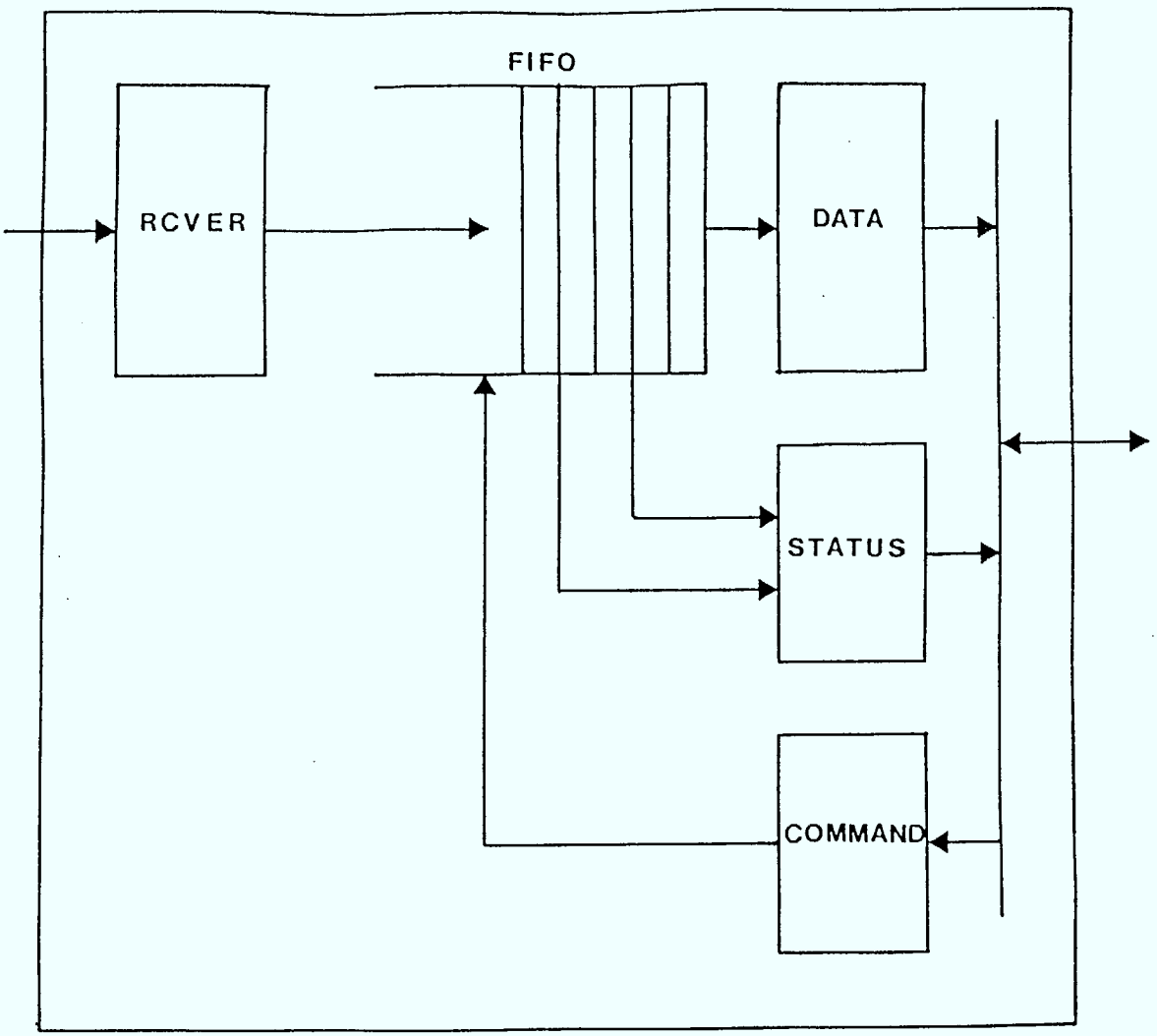


FIGURE 5-8: Block Diagram of a Receiver

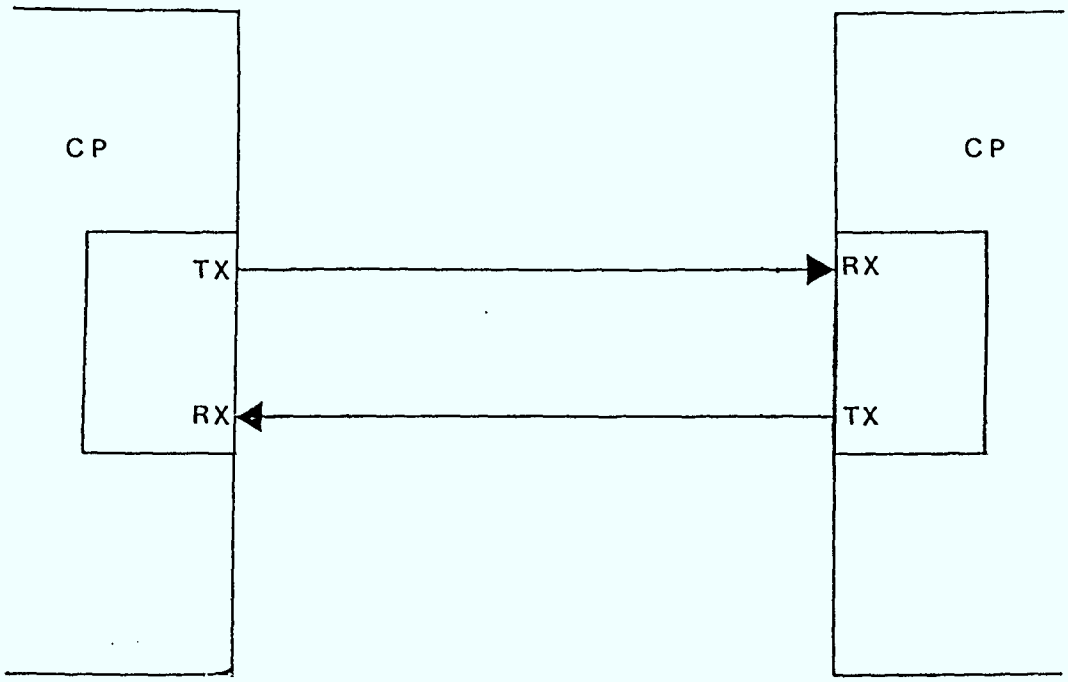


FIGURE 5-9: Block Diagram of Processor Interconnection

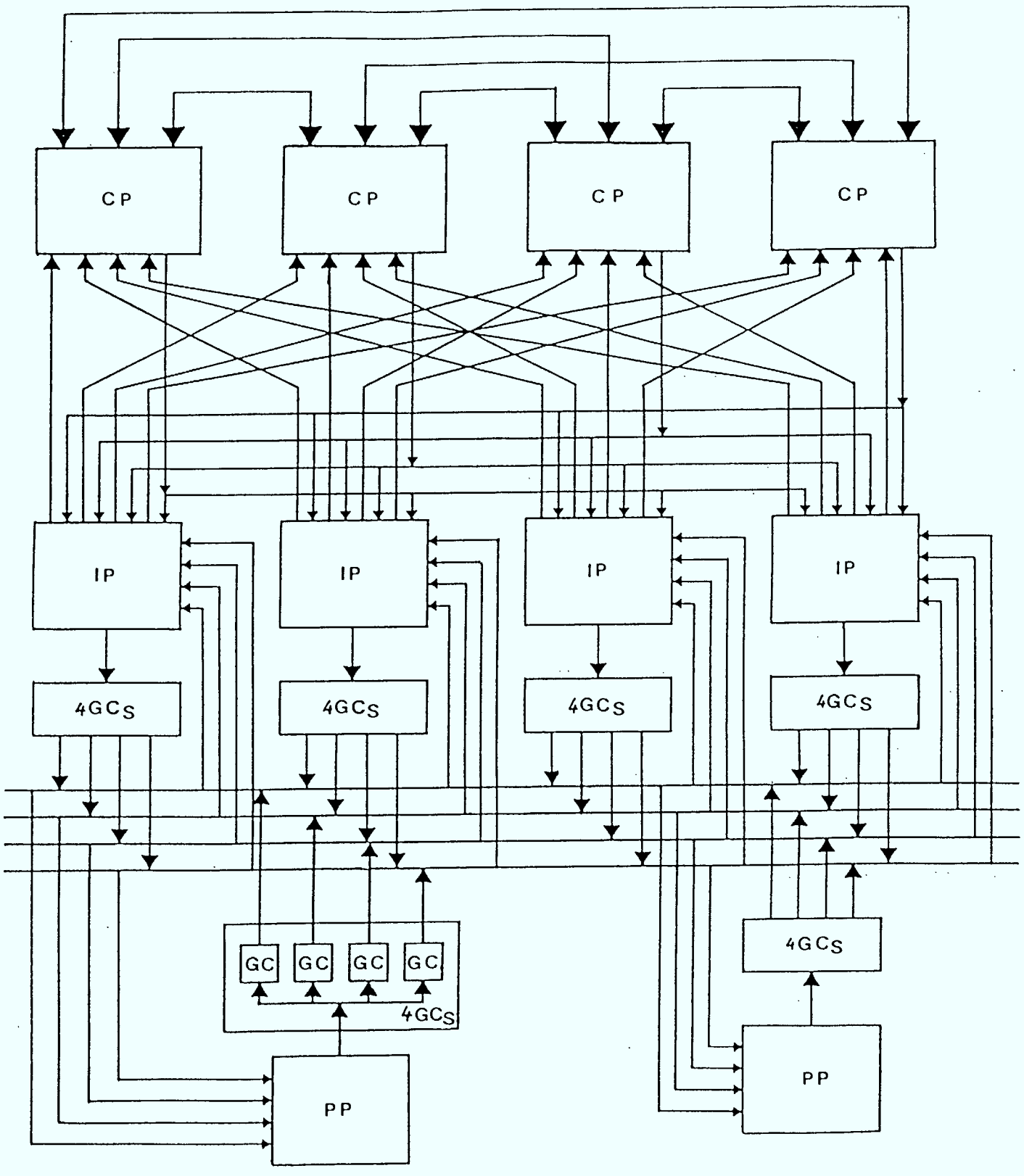


FIGURE 5-10: Typical System Configuration

the interface processors.

On the other hand the interface processors also have a number of communication ports. Namely an interface processor has one communication port for each processor in the central control system. No special care need to be taken for these ports since the central control system is asynchronous. Actually these ports are identical to the ports the processor modules use to communicate with each other. Aside from these ports the interface processor also has a single transmitter. The function of this transmitter is to drive the peripheral bus.

Actually any interface processor only drives one of the busses that comprise the peripheral (redundant) bus. However this is taken care by the gate complexes and is of no concern to the interface processor. Also an interface processor has a number of receivers. The function of these receivers is to receive data from the peripheral bus. Each receiver only receives data from one of the busses. Thus by comparing the values received by different receivers the interface processor can detect bus failures.

5.7. Typical System Configuration

A typical system configuration is illustrated in Figure 5.10. The system depicted consists of :

4 Central Processors

4 Interface Processors

4 busses (forming the peripheral bus)

2 Peripheral Processors

24 gates (1 gate/bus/processor)

72 guardians (3 guardians/gate)

This system normally operates with only three of the interface processors and the busses active at any time. The fourth interface processor and bus are spares to be used as replacements to faulty units. However all four of the central processors are initially active and share the processing load. In case one of them fails its load is shared among the remaining processors.

6. SUMMARY AND CONCLUSIONS

In this work we developed an architecture for a fault tolerant multiprocessor system for satellite applications. Special care has been taken to make the architecture both expandable and hardware independent. This report presents the development of this architecture.

Chapter 2 describes the various possible configurations of a fault tolerant computer. Some of these configurations have been used in the past in various fault tolerant systems. Each configuration is described briefly and its main advantages and disadvantages are presented. Based on this qualitative analysis and the study of previous implementations reported in the literature, a candidate architecture was selected and analysed further in later study phases.

Chapter 3 describes a specific system designed according to the selected architecture. This architecture is basically a combination of two seemingly opposite philosophies, namely the tightly coupled FTMP system and the loosely coupled SIFT system. The proposed architecture is based on a more global view of the system than the previously developed architectures. Thus it became possible to identify explicitly the strengths and weaknesses of each implementation. The architecture presented makes use of each approach for the part of the system where it is most suitable. This results in a system where the implementation of each part is matched to the function it has to serve and not to an abstract architecture. In addition to fault tolerance, the system presented provides graceful degradation, reconfiguration, expandability and technology transparency (hardware independent implementation).

In chapter 4 we developed a model for a part of the system (the peripheral network) and based on the model we derived the optimum values of some design parameters of the processing system. Some of the basic

parts of the peripheral network (guardians and gates) have many similarities to the FTMP system. However the FTMP system was designed for short missions and many of its features were based on intuitive rather than formal approaches. Results of the analysis showed that, for long missions, the optimum solution is quite contrary to what would have been intuitively chosen. The most notable example is the effort made in the FTMP system to ensure passive failures. Results of our analysis show that this approach leads to a much shorter life span for the system than if no precautions to combat passive failures were taken. This is because the accummulation of passive failures may result in system configurations which are separated into non-communicating parts incapable of fulfilling their overall functions.

Finally chapter 5 provided a detailed hardware description of the proposed fault tolerant architecture.

A simulation of the proposed fault tolerant architecture can now be implemented under N.mPc/N.2 based on the detailed hardware description given in this report. This step will be undertaken along with the integration of a fault tolerant multi-microprocessor operating system for the testing and evaluation of a complete fault tolerant multi-microprocessor system and will be described in a separate report.

REFFERENCES

REFERENCES

[1] A.L. Hopkins, T.B. Smith, J.H. Lala, "FTMP - A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft", IEEE Proceedings, October 1978.

[2] J.H. Wensley, L. Lamport, J. Goldberg, M.W. Green, K.N. Levitt, P.M. Melliar-Smith, R.E. Shotstak, C.B. Weinstock, "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control", IEEE Proceedings, October 1978.

[3] C.B. Weinstock, "SIFT: System Design and Implementation", International Symposium on Fault-Tolerance, 1980.

[4] P.M. Melliar-Smith, R.L. Schwartz, "Formal Specification and Mechanical Verification of SIFT: A Fault-Tolerant Flight Control System", IEEE Transactions on Computers, July 1982.

[5] D. Katsuki, E.S. Elsam, W.F. Mann, E.S. Roberts, J.G. Robinson, F.S. Skowronski, E.W. Wolf, "Pluribus - An Operational Fault-Tolerant Multiprocessor", IEEE Proceedings, October 1978.

[6] S. Boucouris, "Conceptual Design of a Fault Tolerant Multiprocessor Operating System and the Implementation of a Prototype Kernel", Intellitech Technical Report, INT-84-44, October 1984.

intellitech
Intellitech Canada Ltd
352 MacLaren Street
Ottawa, Ontario
K2P 0M6
(613)235-5126