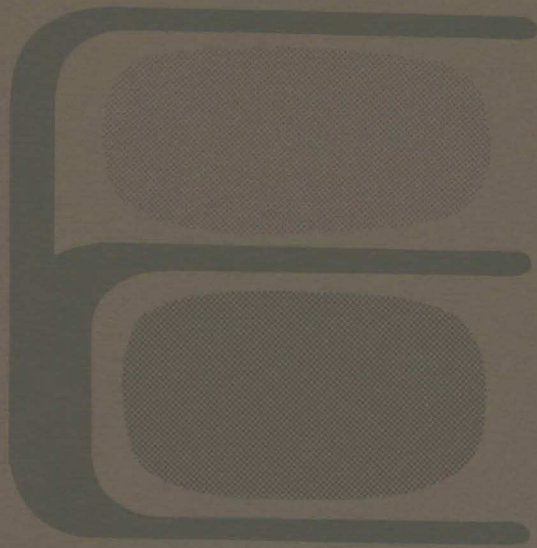
Eidetic Systems Corporation

A Fault-tolerant on-board computer system for spacecraft applications / T. Gomi, M. Inwood



Government of Canada / Gouvernement du Canada

Department of Communications

DOC CONTRACTOR REPORT DOC-CR-SP-82-50

DEPARTMENT OF COMMUNICATIONS - OTTAWA - CANADA

SPACE PROGRAM

TITLE: A Fault-Tolerant On-Board Computer System For Spacecraft Applications

AUTHOR(S): T. Gomi, M. Inwood

ISSUED BY CONTRACTOR AS REPORT NO: 82-002

PREPARED BY: Eidetic Systems Corporation
P.O. Box 13440
Kanata, Ontario
K2K 1X7

DEPARTMENT OF SUPPLY AND SERVICES CONTRACT NO: 3ER.36100-1-0274
SN: OER81-03138

DOC SCIENTIFIC AUTHORITY: R.A. Millar

CLASSIFICATION: Unclassified

DATE: March 31, 1982

A FAULT-TOLERANT ON-BOARD COMPUTER SYSTEM FOR SPACECRAFT APPLICATIONS

Technical Report No. 82-002

T. Gomi
M. Inwood
Eidetic Systems Corporation

March 31, 1982.

C O N T E N T S

ACKNOWLEDGEMENTS

This report is the second in a two part study of the future application of microcomputers in a fault-tolerant spacecraft environment. It was undertaken under DSS Contract OER 81-03138, at the Communications Research Centre, Shirley Bay, Ottawa.

Assistance was sought from Dr. George C. Gilley of The Aerospace Corporation, Los Angeles, California, on Fault-Tolerant Design and Autonomous Spacecraft; Drs. J.S. Albus and E.W. Kent, National Bureau of Standards, Washington, D.C., on the theory of hierarchical control systems; and also from Tony Anderson and John Beaston, of Special Systems Operations, Intel Corporation, Aloha, Oregon on the fault-tolerant aspects of the iAPX 432 series computer. Appreciation is expressed for this assistance. The authors also wish to acknowledge the guidance and support of Robert Millar, of Communications Research Centre, Ottawa.

ACRONYMS

AASC	Advanced Autonomous Spacecraft Computer (AASC)
Ada	DoD defined Ada programming language
AI	Artificial Intelligence
ALU	Arithmetic Logic Unit
AP	Attached Processor (AASC/NIU/AP)
ASM	Autonomous Spacecraft Maintenance
BERL	Bus Error Report Line (AASC/PCU/BERL)
BIU	Bus Interface Unit (AASC/PCU/BIU)
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
ECC	Error Correcting Code
FRC	Functional Redundancy Checking (AASC/FRC)
FTBBC	Fault-Tolerant Building Block Computer
FTC	Fault-Tolerant Computing
GDP	General Data Processor (AASC/PCU/GDP)
IIU	Subsystem I/O Interface Unit (AASC/IIU)
IP	Interface Processor (AASC/NIU/IIU/IP)
IPL	Interface Processor Link (AASC/PCU/IPL)
LAN	Local Area Network
LSI	Large Scale Integration
MCU	Memory Control Unit (AASC/PCU/MCU)
MERL	Memory Error Report Line (AASC/PCU/MERL)
MIPS	Mega Instructions Per Second
MTBF	Mean-Time-Between-Failures
NBS	National Bureau of Standards
NC	Network Controller (AASC/NIU/NC)
NIU	Subsystem Network Interface Unit (AASC/NIU)
PCU	Subsystem Processor Complex Unit (AASC/PCU)
PPB	Processor Packet Bus (AASC/PCU/PPB)
QMR	Quadruple Modular Redundancy
RAM	Random Access Memory
ROM	Read Only Memory
SPB	System Packet Bus (AASC/PCU/SPB)
SRAM	Static Random Access Memory
VLSI	Very Large Scale Integration

1. SUMMARY

A typical set of requirements for future unmanned spacecraft is given. This has been supplemented by an additional set of requirements which arose from the general trend towards automation based on computer technology and sophisticated system control studies. The introduction of Artificial Intelligence (AI) concepts in the context of space applications is of particular importance at this time, as the products of this technology are expected to affect almost all aspects of computer-related technologies in the relatively near future. Also discussed are a realistic introduction of the concept of software fault-tolerance, and examples of approaches taken by studies in this field. A set of design criteria to assure a high degree of fault-tolerance has also been determined and is described.

An on-board computer system that has a strong potential for meeting these functional requirements and design criteria is then introduced. Its global structure, probable hardware implementation and software peculiarities are discussed. The proposed system is designed around a local networking model and possesses a strong fault-tolerance that depends on the systematic management of non-dedicated, as well as some dedicated, redundancies distributed in the system. In particular, the Processor Cluster Unit, which achieves its extreme stability, high throughput, and tight protection through entirely new architectural and hardware component concepts, is described in detail.

Requirements for a control software that regulate active processes within a cluster are depicted. This is followed by a description of a method to properly inter-connect these clusters, each of which represents a spacecraft subsystem in a local networking scheme, and then establishes communication links between the on-board facility and the external world.

Finally, an explanation is given of the interface between the proposed computer system and its users, these being the application software and hardware which occupy a cluster and constitute a spacecraft subsystem.

2. INTRODUCTION

This report outlines a conceptual design for an Advanced Autonomous Spacecraft Computer (AASC). It is the second part of a study begun in 1981 in a search for a fault-tolerant computer suitable for future on-board spacecraft processing requirements. In the report on the first stage, "A Review of Spacecraft Fault-Tolerant Computer Design Concepts" (Ref.1), the Jet Propulsion Lab's Fault-Tolerant Building Block Computer (FTBBC) was studied as a possible candidate for meeting future needs. The results of the study showed, however, that innovative as it was in many respects, advantage had not been taken in the FTBBC design of important developments in several highly relevant fields. The report concluded that consideration of these developments would be necessary in designing a system capable of meeting future on-board processing requirements.

On-board processing is being required to further the concept of spacecraft autonomy to a greater degree than has so far been achieved in conventional spacecraft. Reliance on ground support is placing an increasingly heavy work load on ground support staff in addition to being time consuming, vulnerable and costly. The aim is, therefore, towards bringing control and maintenance on board spacecraft. A United States Air Force (USAF) study (Ref.2) points to the need for the development of technology in the field of highly reliant fault-tolerant computing systems if spacecraft autonomy is to be achieved. They have published a list of Design Requirements (Appendix A) to be applied to future developments and have recommended that it is both feasible and necessary for ASM to be an integral part of all U.S. spacecraft becoming operational from March 1989. These Design Requirements have been taken as terms of reference for this present study, which is concerned with the definition of an Advanced Autonomous Spacecraft Computer (AASC).

The ability of a spacecraft to survive for lengthy periods of time and under adverse conditions implies that control must be reliable and immediate. Reference to a ground-based authority may be too time consuming for survival under some contingencies and the implications are that this control should be implemented in an on-board, intelligent fashion. These implications have led to the inclusion of additional ASM requirements further to those already proposed by the ASM study group.

Another set of references used has been the FTC Design Rules, drawn up during the first stage of this study, and are considered essential principles in the achievement of system reliability. These are contained in Appendix B.

The concept of the AASC has been designed to take advantage of state-of-the-art developments in hardware and software philosophy, system architecture, fault-tolerance, and networking. It also reflects the increase in understanding of future spacecraft design needs which has been acquired during the current study. Indications are that the scope of such projected needs will go even further than the current work of the ASM Study Group.

3. THE ASM REQUIREMENTS

In 1980, the USAF initiated a study (hereafter referred to as the ASM study) of the means needed to achieve a greater degree of satellite autonomy, i.e. to increase space system survivability while reducing the ground station work load, vulnerability and related expenses. (Ref.3). Maintenance of a spacecraft can be divided into two categories:

a) welfare, which is concerned with satellite upkeep such as thermal control, battery charging, solar array orientation and sensor calibration, and

b) health, which is responsible for the detection, correction of, and recovery from malfunctions and other unplanned events.

Although health maintenance is generally far less understood than welfare, a study of the state-of-the-art technology substantiates its feasibility. An effective approach, Autonomous Spacecraft Maintenance (ASM) is seen as the fully fault-tolerant design of the entire spacecraft with a highly reliable fault-tolerant data processing subsystem at its core. It is intended that this computer should extend its fault-tolerance to all spacecraft subsystems and act as an "automated repairman" to them.

In order to consolidate and increase the knowledge of fault-tolerant technology, which is seen as a relatively immature field, a study group (the ASM study group) was formed of experts from industry, academia and NASA. After studying current spacecraft capabilities and the requirements for ASM, this group produced a Final Report (Ref.2) including an Implementation Plan and a list of ASM Design Requirements (Appendix A). They recommended that the USAF proceed with plans to implement ASM in operational spacecraft by March 1989. Under these requirements, the spacecraft is initially responsible for its own maintenance. Ground support would act in a supervisory role with the ultimate ability to override ASM functions. The increased complexity needed in the space segment to handle onboard navigation, fault detection, isolation and recovery will require the development of an autonomous navigation subsystem and a fault-tolerant data processing subsystem. It is the latter, considered in the context of autonomous spacecraft maintenance, which is the subject of this report.

4. ADDITIONAL ASM REQUIREMENTS

4.0 General

The ASM study group highlighted several areas of research which it deemed essential to future ASM development. Of these it is felt that three, in particular, should be considered within the scope of this study and should be considered as additional requirements in view of their importance.

The complexity inherent in an autonomous system should be viewed from a hierarchical viewpoint and would be best dealt with by the application of an integrated advanced control theory.

The concept of an on-board system acting as an "automated repairman" (Ref.3) leads to the issue of unanticipated faults. "Recovery by problem solving" implies some degree of intelligence and, therefore, the application of Artificial Intelligence (AI) is felt to be a necessary requirement in this context.

The third area to be considered is that of software fault-tolerance, in which progress has not been as rapid as in the hardware field but which is beginning to attract more study and attention.

4.1 Advanced Control Theory

The ASM study is among the first few to recognize that faults in a system form a hierarchical phenomenum. In addition, it successfully pointed out the need for a layered fault-protection or fault-handling scheme (Ref.2, p.2). The study also mentioned, as research agenda to be carried out in current and future phases of the ASM study, the importance of architectural revision of spacecraft control systems, software fault-tolerance, system modelling, and system verification methods, among others. The fault-tolerance mechanism, viewed as a hierarchical system, calls for a new unified theory that deals with the issues related to the design, implementation and optimized performance of hierarchical control systems.

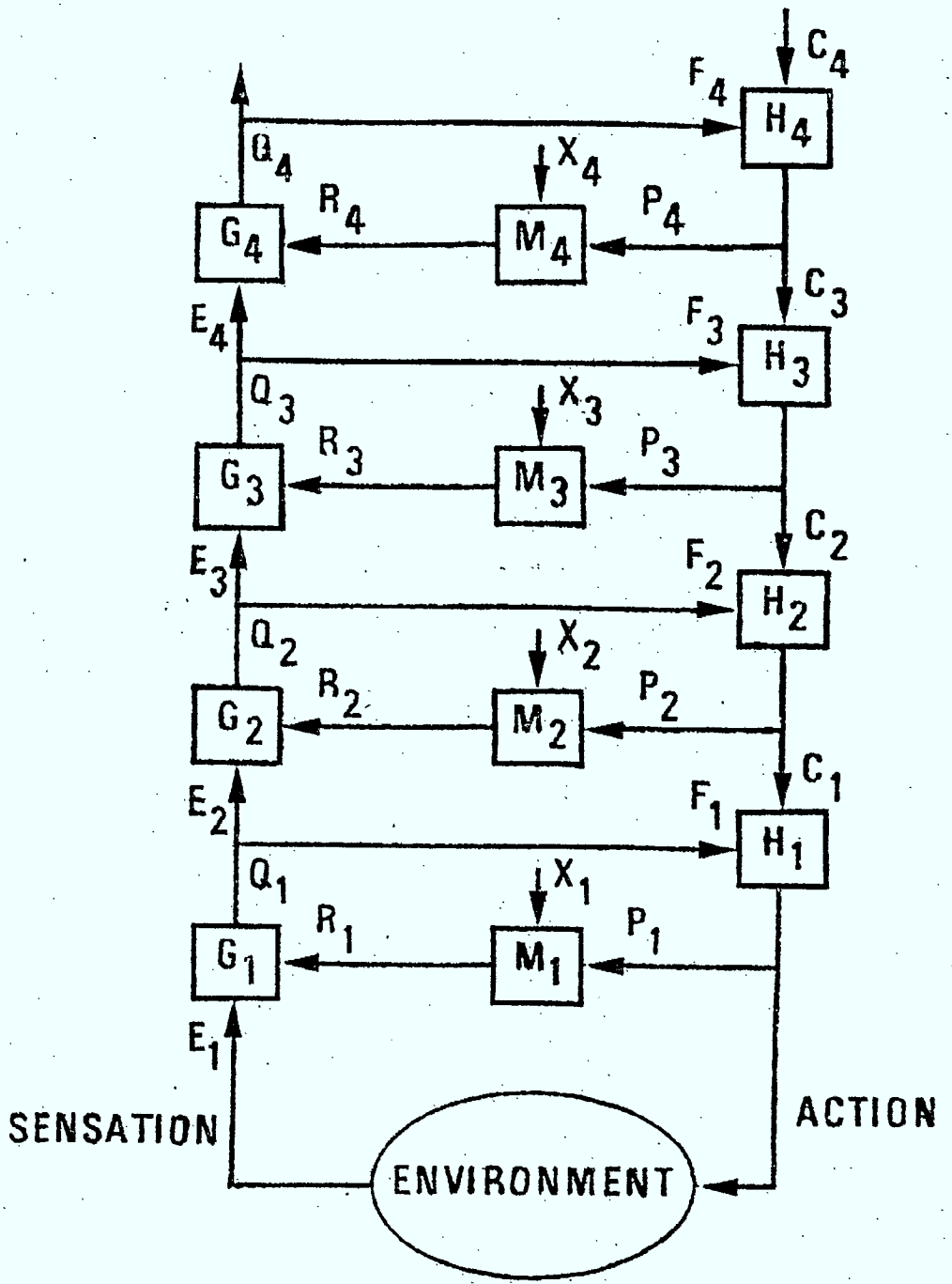
One successful undertaking in this area is a model developed by Dr. J. Albus and his group at the National Bureau of Standards (NBS) (Ref.4,5). A theoretically clearcut model re-

presents a layer in a multi-layered "intelligence hierarchy". Layers can be created using the model as a building block and the depth of the intelligence is, at least in principle, arbitrarily definable. The building block takes the form of a symbolized servomechanism which constitutes a closed loop feed-back system with a mechanism to store learned experience about its operating environment as detailed in Figure 4.1. Dr. Albus has produced an impressive demonstration of the application of this theory in the area of robotics. It consists of several layers and exists as a lab model. The theory is broad enough to be applied to the solution of far more abstract problems such as the establishment of a hierarchical fault-tolerant capability onboard a spacecraft.

Compared to this "organic" approach, a conventional adaptive control system lacks flexibility. System responses are calculated mechanically and they tend not to be a highly or intelligently optimized answer to a given situation. On the contrary, a learning control system, of which the NBS model is an example, develops a type of knowledge-base during its operation and learns to improve its response even to an identical set of environmental and control inputs. It is clearly recognized here that such an advanced control theory is essential to formulate a truly useful autonomy onboard a spacecraft. A system equipped with theory of a lesser, deterministic nature would constantly require undesirable human intervention.

4.2 Artificial Intelligence

In early 1980, the NASA Langley Research Center conducted a study to address topics in Artificial Intelligence (AI). These included operations research, and advanced control theory in the context of automated decision making; problem solving in relation to space mission-oriented machine intelligence; and robotics technology (Refs.4,5). A diagram of these relationships is given in Figure 4.2. Of these topics, advanced control theory was addressed in the study described in the previous subsection. Operations research deals mainly with the modelling of various control algorithms and spacecraft design, and hence will not directly affect on-board computing. NASA's aim is to provide a general direction for technological development activities in the space community for the period 1990 to 1999. The study addressed three major application areas for such new technologies: global services such as geostationary service satellites; deep space exploration; and space industrialization. The AI technology developed will be applied to both the space and ground segments of the operation with an emphasis to-



A cross-coupled processing-generating hierarchy. The M_i modules remember sensory experiences which occur in association with specific activity in the generating hierarchy (P_i) and other sensory modalities ($\underline{X}_i$). The M_i modules thus learn a set of internal expectations (i.e. a predictive model) of the external world as seen through the sensory input channels.

Figure 4.1

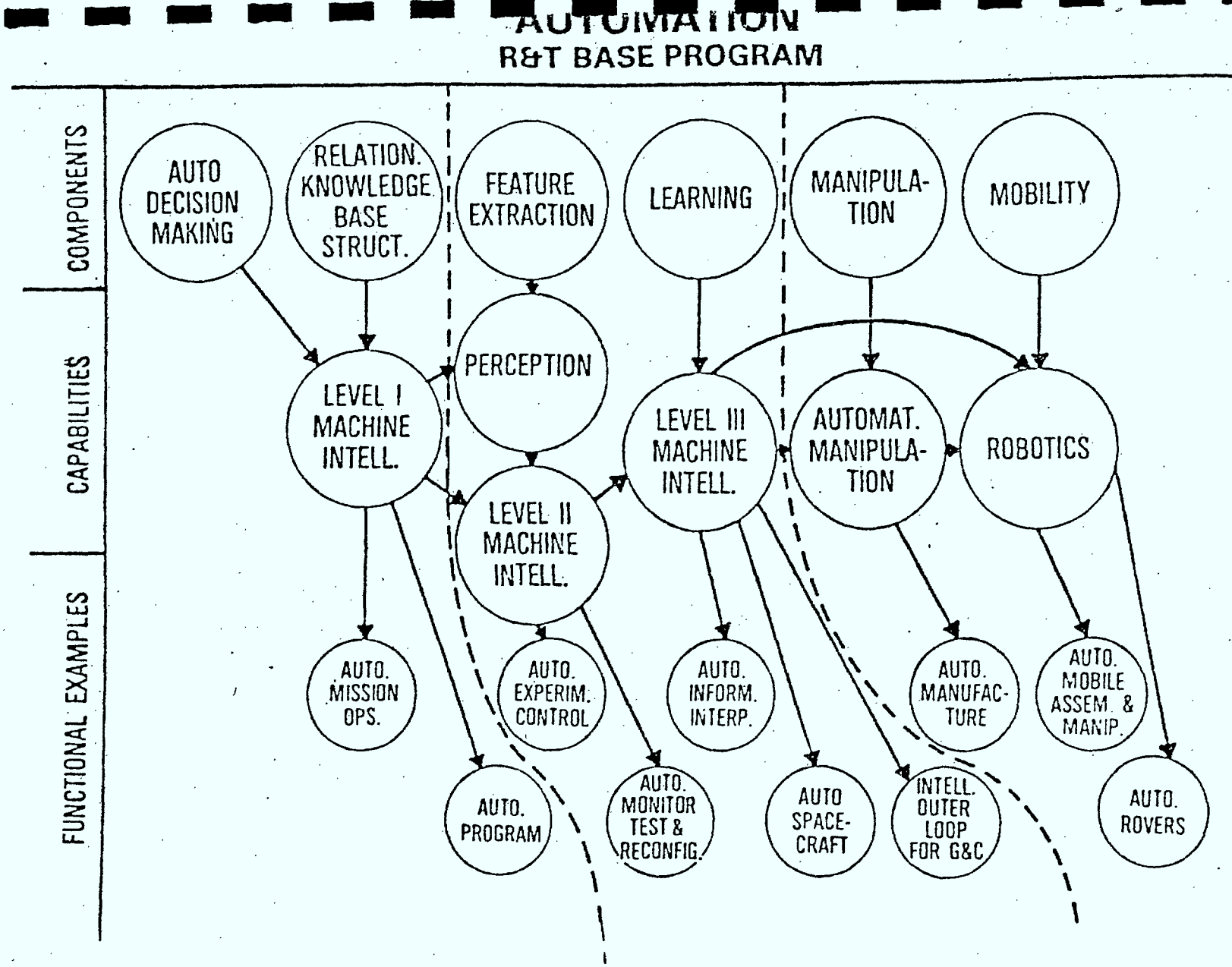


Figure 4.2 Automated decision making and problem solving in relation to space mission-oriented machine intelligence and robotics technology.

wards on-board use. Through this study NASA has clearly committed itself to the development of AI capability.

The NASA study group singled out the difficulty of developing software to realize algorithms that support machine intelligence as an expected bottleneck in the pursuit of the above technological objectives. The concentration of program objectives, first on activities leading to an "increased level of machine intelligence and perception, and then on those which enable automated manipulation and robotics" was thus justified. In practice, automated decision making, relational knowledge-base structuring, feature extraction, and learning, are the targets of immediate study. This explains NASA's strong interest in the Fifth Generation Computer project by the Japanese. The NASA study, it is hoped, will eventually lead to the establishment of in-house capabilities focussing on areas of advanced AI technology with a high expected payoff.

According to the NASA report (Ref.2), it is expected to see several versions of highly autonomous robots in space that would communicate with earth only when contacted or when a significant event occurs requiring immediate attention on earth. The similarity of this to the thinking of the ASM study group is obvious, although the NASA study addresses issues beyond those of ASM. A degree of continuity, accidental or not, is seen between the ASM study and NASA's AI approach: ASM addresses problems to be solved by 1990. NASA aims at solving the problems foreseeable during the period 1990-1999. One can conclude from this that the ASM capability is a prerequisite to the introduction of sophisticated on-board intelligence.

The trend towards implementation of space-borne intelligence seems logical from the standpoint of increased system level reliability and improved economy in space operation. The cost of operating any space program has long been exceedingly high. It also makes sense to acquire the capability to cope with ever increasing system complexities in monitoring and controlling activities in space. Interpreted in the context of the present study, this amounts to an additional set of requirements to be placed on the structure and capability of future on-board processing facilities. Among those already recognized are the following:

- high CPU throughput: AI computations are highly CPU-intensive

- large and hierarchical memory space: memory accesses are extensive in the execution of AI programs. They call for

a large memory space, most of which will be formulated into a hierarchical structure in terms of their size and access time, from the fastest but smallest (e.g., register sets, cache memories) to the slowest but largest (e.g., mass storage for an on-board knowledge base, ground-based archiving system).

- effective internal (on-board) and external communication links

- operating system software that supports efficient multi-tasking and multi- processing

- AI-oriented operating system extensions that coordinate various on-board AI activities and an AI protocol to be used among AI subsystems.

4.3 Software Fault-Tolerance

As demonstrated in Section 8 on the fault-tolerance capability of the PCU, a methodology to cope with basic computer hardware has been achieved at the manufacturers' level. One can now safely say that, within the foreseeable future, commonly used computer hardware components can be made to survive a series of incidents normally to be expected in space applications that could cause faults of considerable magnitude. The FTC technology exists in the form of a collection of theories, ideas, implementation techniques, and knowledge concerning materials, space environment, and ground operational needs. The stage is clearly set to enhance this technology by addressing fault-tolerance at higher levels of abstraction, while solving and improving the implementational aspects of the basic hardware fault-tolerance.

The higher level fault-tolerance issues that are now visible include stable operation, general networking welfare, and other communication methods, distributed operating systems, and application software. In these areas, signs of system-level fault-tolerance studies have just started to appear. The following are examples of topics that need to be entertained during the course of the detailed design of the AASC:

- the study of a diagnostic model that will be used to establish system-level fault profile, such as that suggested by J. Kuhl and S. Reddy of the University of Iowa (Ref.6). The implications of the third FTC Rule call

for distributed or non-fixed capability to perform fault diagnosis. Such a task often involves solving complicated logical issues and highly theoretical treatment.

- fault-tolerant data structures. It is now known that some data structures are more robust than others, thanks to efforts by J. Black et al of the University of Waterloo (Ref.7). Error detection and recovery algorithms often depend on the type of data structures used by the process. A formal study is necessary to axiomatize these relationships and apply them to on-board software.

- system level fault-tolerance in distributed processing also needs to be viewed in relation to networking. A large scale project that involves this area of study is in progress in France (Ref.8) and others are in the planning stage at other centres in Europe.

- theoretical study of recovery. The study of the formation of algorithms to handle system level recovery using advanced hardware components supported by non-dedicated redundancy concepts makes the issues related to a safe recovery of interprocess relationships become increasingly obvious. The majority of the FTC community is still not aware of the problem, as these are highly theoretical issues. Study topics will include the definition of requirements for fault-tolerant interprocess protocol, for example the studies by W.G. Wood of University of Newcastle-upon-Tyne, England (Ref.9); and ways to establish proper check- pointing in processes as presented by F. Schneider et al. of Cornell University (Ref.10).

5. SYSTEM STRUCTURE

5.0 General

The definition of the structure of the AASC consists of functional descriptions of a set of components, the size and number of which depends largely on the individual application. Those components that are expected to be implemented in hardware are described in this section. Discussions of issues related mainly to software components are contained in a part of this section and the following sections. Detailed descriptions of the Funtional Redundancy Checking (FRC) version of the processor complex are described in detail in Section 8.

5.1 The Network Based Architecture

The proposed AASC hardware consists of clusters (subsystems) of highly fault-tolerant computer complexes, each supporting a satellite subsystem, distributed along inter-cluster buses or Local Area Networks (LANs), which are shown in Figure 5.1. Local area networking is introduced to ensure the readiness of computer systems for the advent of large scale space structures as evidenced in recent post-shuttle space activities (Ref.11). A cluster interfaces the LAN(s) at one end and, normally, i/o subsystem(s) at the other, as shown in Figure 5.2.

5.2 Hardware Redundancy

Hardware redundancy is provided at several levels within the system hierarchy: every element of the system may exist in multiple copies to provide dedicated or non-dedicated redundancy at the hardware level. These include the following: clusters; LANs; LAN to cluster interfaces; intra-cluster processors; intra-cluster buses and bus controllers; intra-cluster memory control units; and local and global memory modules.

5.3 The Inter-Cluster Bus

The inter-cluster bus or LAN is not specified in detail at the present time as its design is, to a greater degree, intended to be independent of currently available technology. However, a fully distributed architecture is assumed, i.e. no dedicated controllers on the bus and no polling scheme enforced

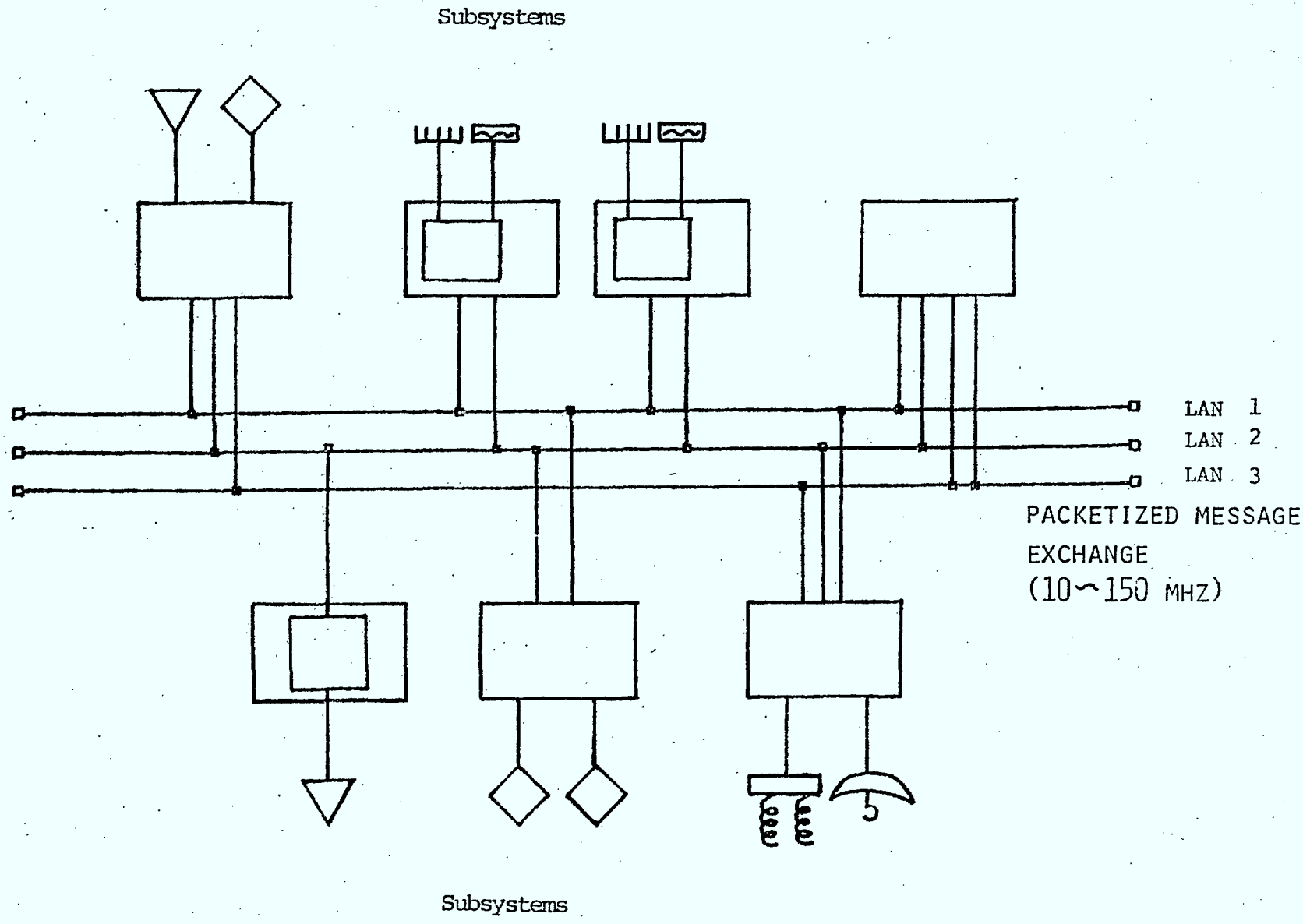


Fig. 5.1 Advanced Autonomous Spacecraft Computer

Various onboard subsystems are hosted in clusters, each of which has access to one or more onboard local area networks.

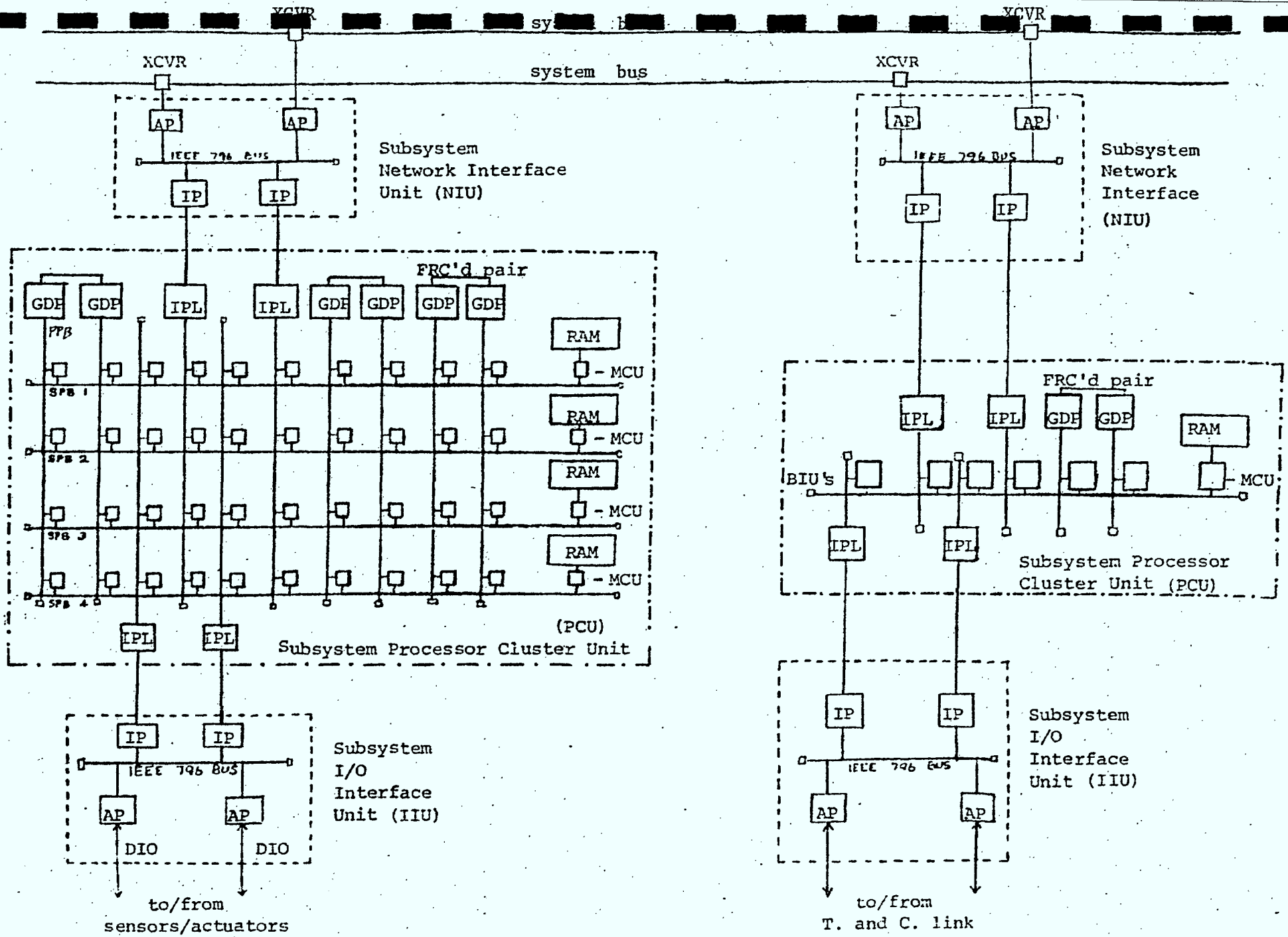


Figure 5.2 AASC/Breadboard - hardware configuration
A cluster consists of an NIU, PCU, and IIU. The number and size of the NIU and IIU can change according to the need as well as the complexity of the PCU. All units are built using several building blocks.

over the bus, but a multi-drop scheme on plural LANs tapped-on by fully independent clusters or subsystems which are operating asynchronously.

5.4 The Cluster

A cluster consists of three units; the Subsystem Network Interface Unit (NIU), the Subsystem Processor Complex Unit (PCU), and the Subsystem I/O Interface Unit (IIU). An example of a typical cluster is shown in Figure 5.3.

5.4.1 The NIU

The Subsystem Network Interface Unit (NIU) establishes contacts between a subsystem and the rest of the system. Each of the multi-drop connections to the inter-cluster bus is achieved by a Network Controller (NC). The Network Controller handles at least the first two layers of a multi-layer interconnect protocol. The NIU contains an Attached Processor (AP) which processes and deals with the issues associated with the interconnect protocol in Layers 3 and above. The other end of the AP is connected to an Interface Processor Link (IPL) which is a part of the PCU. The IPL is responsible for mapping address windows between the operating environments of the PCU and NIU.

5.4.2 The PCU

The Processor Complex Unit (PCU) is based on the Functional Redundancy Checking (FRC) version of Intel's iAPX 432 computer. The selection was made after a series of comparative studies including most of the modern microprocessors. The fault-tolerant features of the iAPX 432 out-scored any competitors even before other advanced features of the basic design of the computer were considered. The indication that the entire iAPX 432 itself will be space-qualified by the end of 1986 enhanced confidence in the decision.

The PCU, shown in detail in Figure 5.4, will be used to house the upper layers of local intelligence required for a subsystem and provides most of the computing power needed by the subsystem. It consists of a mesh of packet oriented buses which connect processors and controllers. The System Packet Bus (SPB) handles the inter-processor communication, while the Processor Packet Bus (PPB) controls distribution of information

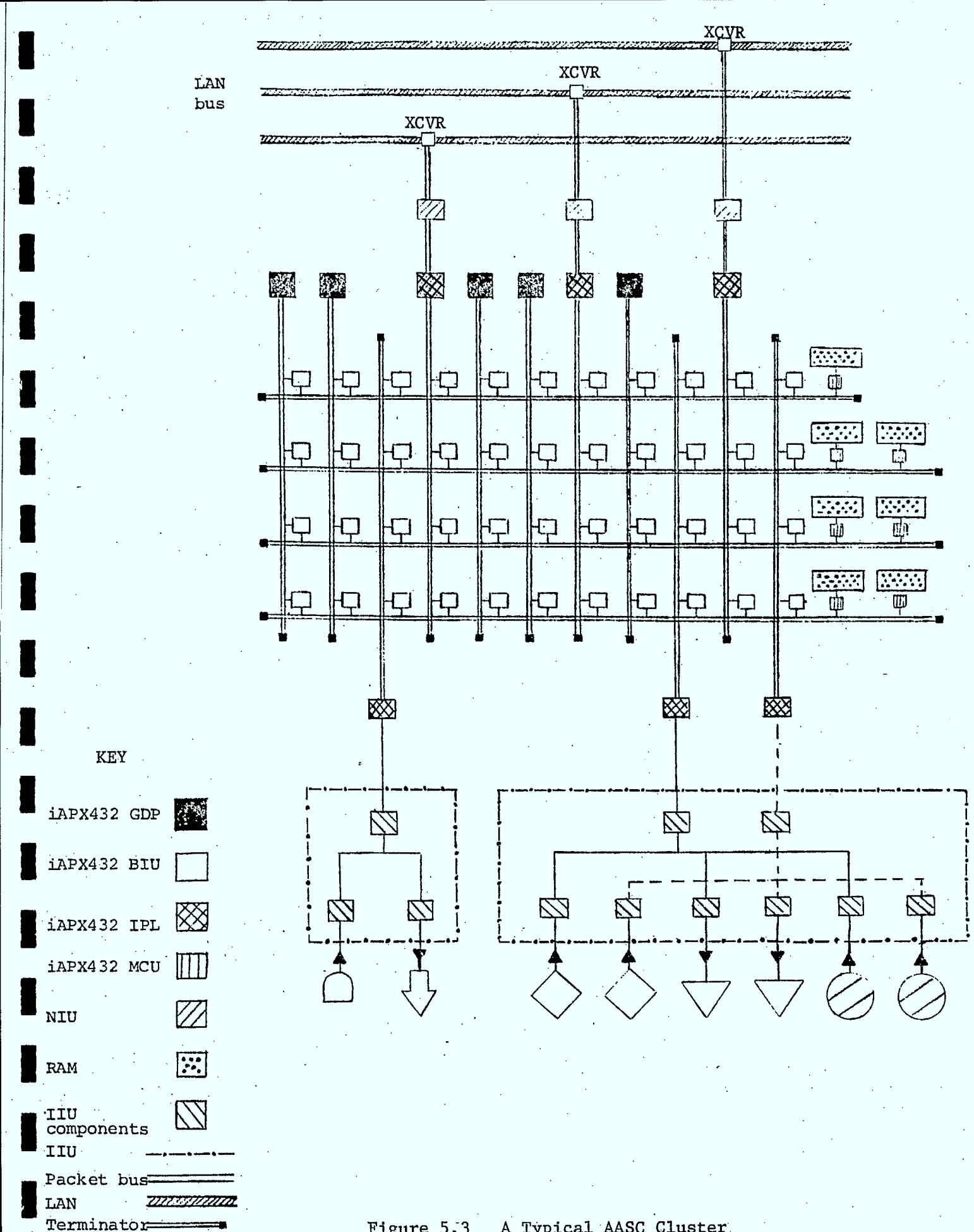


Figure 5.3 A Typical AASC Cluster

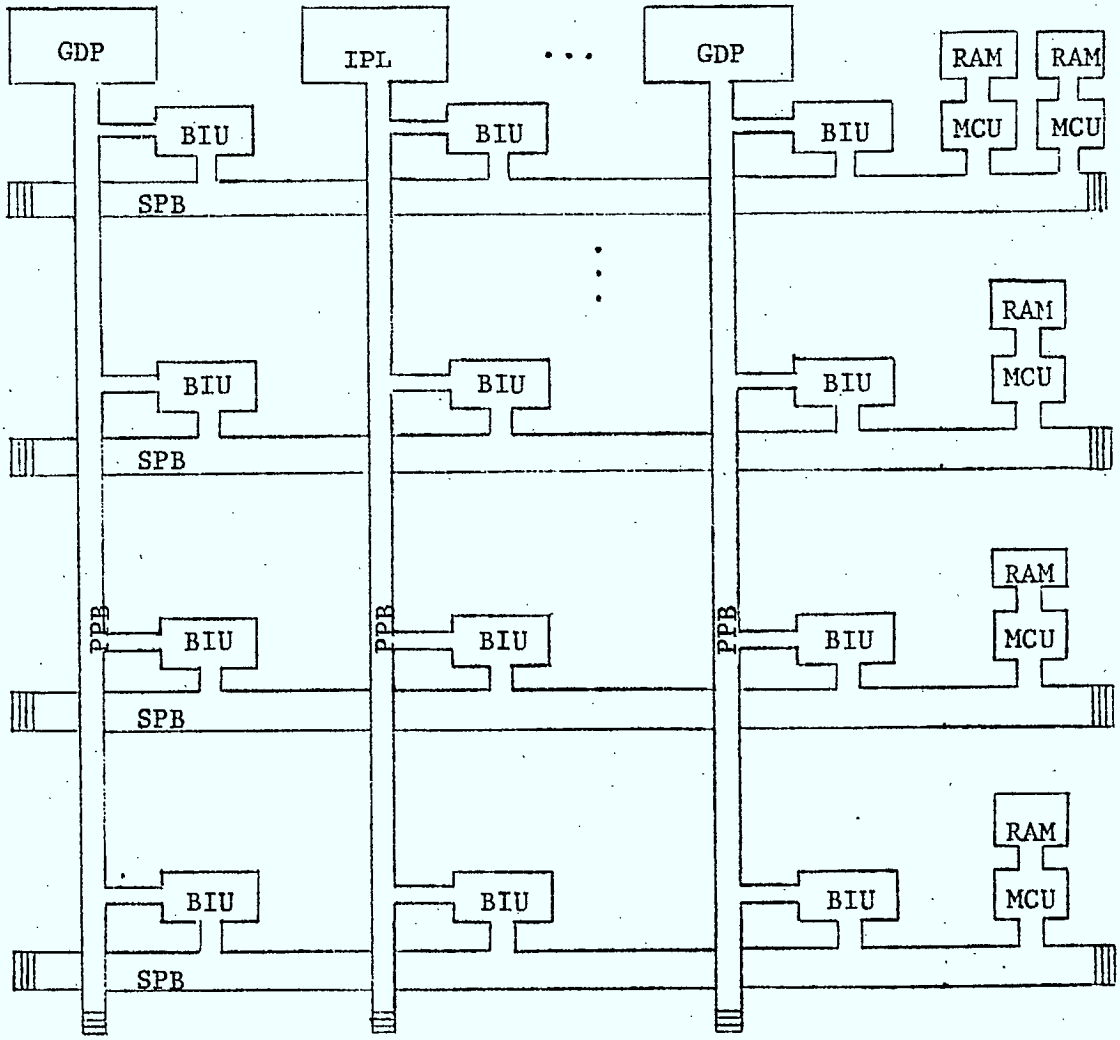


Figure 5.4 Processor Complex Unit

The PCU consists of processors (either GDP or IPL), two types of packet oriented buses (system packet bus and processor packet bus), Bus Interface Units (BIU), Memory Control Units (MCU), and Memory Arrays (RAM).

between a processor and a number of System Packet Buses. The former is supported jointly by all processors in the complex, while the latter is sponsored by a single processor - either a General Data Processor (GDP) or an Interface Processor Link (IPL). At the intersection of the two packet buses is a Bus Interface Unit (BIU), which controls information transfer between the two packet buses. The Memory Control Unit (MCU) attaches a memory module (RAM or ROM) to the System Packet Bus. The intelligent memory controller converts logical address spaces implied on the packet bus into physical addresses in the memory module.

The capability of the fault-tolerance mechanism embedded in the PCU hardware far surpasses that of any known computer system. The currently available version of the GDP provides a throughput of approximately 0.2 MIPS (Mega Instructions Per Second), or the capability of a typical minicomputer. The multiprocessing is performed in a manner completely transparent to the overlaying software structure and, hence, the user. The over-all computing power of the cluster is determined as a function mainly by the number of GDPs and System Packet Buses.

5.4.3 The IIU

The I/O Interface Unit (IIU) handles all non-network i/o to and from the PCU, as shown in Figure 5.3. It employs the same IPL-AP structure as in the NIU. In place of network controllers, however, the AP is connected to a structure that controls subsystem i/o. The subsystem i/o in this application would be an i/o section of a satellite subsystem such as an AOCS, a temperature control subsystem, an uplink/downlink channel, an on-board power management subsystem, etc. The fault-tolerance within the subsystem IIU will be provided using known FTC techniques, including the use of redundant copies of such a unit.

5.5 Software

The software will consist of three major module groups: the operating system and its support, the fault-tolerance management software, and the application software. They will be developed as Ada packages to take advantage of the excellent software engineering features of the language, and will remain highly configurable to permit intense customization by the application system designer. Together, modules from these three groups will establish and support a hierarchical process struc-

ture which may dynamically change its profile and infrastructure during its life, according to changes in operation environment and needs of the system.

6. OPERATING SYSTEM

6.0 General Characteristics

The operating system and, to a large extent, other (application) software to be used in the AASC should possess those characteristics normally attributed to modern real-time operating software for a sophisticated embedded application. These include the following:

- multi-tasking
- transparent multiprocessing
- queued inter-processor message exchange and control
- queued resource request handling
- generic device
- device independence
- layered communication access method
- capability-based access control.

The operating system itself shall be layered, or have a concentric "pie structure". The relationship between the computer hardware and the operating system must be "friendly": the operating system should be an extension of the basic computer hardware and maintain a partnership with it. Together they support the bottom layers of a hierarchically-defined system function.

Since a hardware/software distinction is not considered essential, portions of the operating system may be implemented in hardware as occurs, for example, in the iMAX 432 operating system (Ref.12). The reasons for the delegation of the operating system functions to hardware may vary: for example, cost-effectiveness, structural security, extreme complexity, and a tight response requirement. As many operating system functions become standardized, the implementation of higher level functions will be placed in hardware. In this respect, it is important that an understanding of the continuous evolutionary process exists in the operating system-hardware partnership. The basic hardware, even now, must include appropriate

support for some of the operating system primitives such as message exchange, process dispatch, interprocess control, i/o channel control, memory management, etc. In other words, the operating system software should not be directly involved in instructing processor hardware at a basic instruction level; it should issue instructions in a far more abstract fashion such as, "create process", "send message to process", "terminate process", "obtain buffer", etc. This trend will continue to include, eventually, even more abstract "instructions" in hardware.

The creation of CPUs with an elevated level of functionality may appear costly. However, a venture of this nature is justified by the ability of the operating system to cope with the greatly increased level of sophistication and complexity demanded by application software. One would constantly be encouraged to put as many routine functions of an operating system as possible into the hardware, since this would result in stabilization of those functions and an increased level of abstraction of the hardware machine which forms the basis of the system structure. The level of reliability demanded in the operation of the AASC and the expected complexity of the application software could certainly justify such effort. The operating system working in the AASC environment would then be able to devote its code to improved monitoring of the overall system, or cope with such higher level issues as the upper level fault handling and requests related to the continuous monitoring of spacecraft subsystems.

6.1 Separation of Policy from Mechanism

Another important characteristic demanded of the operating system used in the AASC environment will be the separation of control policies from the mechanisms that support such policies. For example, transparent multiprocessing support is a mechanism, while assignment of processors 1 to 5 as subsystem monitors and processors 6 to 12 as "blank spares" is a policy. Provision of structured filing is a mechanism, while a structured file tree depicting the relationship between on-board telemetry data belongs to the policy domain. Yet another example is process scheduling. A process dispatch port and its surrounding facilities are mechanisms, while the implementation of a specific scheduling scheme is a policy. In addition to standard multi-priority round-robin scheduling, the operating system should be capable of accepting such process dispatching policies as "last in, first out" or "random dispatch", since such scheduling schemes do exist in nature and real life.

Traditionally, policy and mechanisms have been confused and implemented as a tightly interwoven entity within an operating system. Since it is expected that the AASC will be used repeatedly in various versions and configurations, a clear distinction between the two will be of particular importance as the system will be expected to provide application (spacecraft) designers with tools with which to fulfill his/her functional requirements. An operating system should merely provide a mechanism on which policies can be implemented and enforced by the application designers, whose chief concern is the realization of an efficient and reliable operational environment in the spacecraft.

Again, the underlying rule is that the system structure is hierarchical. The operating system would only address issues that belong close to the bottom of the hierarchy. However, as a unified hierarchical structure, the completed system should not see any clear boundary between the operating system and application software. They should only represent different layers of the hierarchy, each layer being served by "tools" or mechanisms made available by the layer below it. The decision to use these tools always belongs to the current layer. This relationship must be universally true throughout a hierarchy.

6.2 Ada Packages

Further to the issue of optimization of operating systems, functional modules within an operating system should be supported and represented in the form of such clear-cut encapsulation and labelling mechanisms as the package concept in the Ada programming language. This is not to be confused with the often used "software package" concept, since an Ada "package" addresses more precise and unique concepts. Combined with the library concept also defined in Ada, the use of "packages" would allow the application designer a wider and finer range of configurability.

In fact, as the library and package concepts are such a rational means of building software structures, it is anticipated that many software products, operating systems or otherwise, will be implemented in these terms. This will contribute greatly towards the eventual widespread standardization of software products for various applications. Manufacturers will be forced to conform to such standards. The tremendous improvements in quality and in software productivity which are implied in the Ada package concept, should not be overlooked. An example of this is the ability to compose an application software

structure using readily available software components),

6.3 Operating System Dynamism

Another strong reason for the use of Ada packages and other related concepts in the design and implementation of an operating system is their ability to contribute to the current trend towards more dynamic operating systems. An operating system should not only be configurable at "sysgen" time but also remain configurable in a major way throughout its deployment in an application's operational environment. This means an operating system which can reconfigure itself in operation. Such reconfigurability within an operating system is essential when there is a need for a highly modifiable system. Logical shape and functionality must conform to changes in the mission profile and changes in the operational environment, many of which could be caused by faults of various origins and significance. The hierarchy of encapsulation methods provided in the Ada language (package, task and procedure) will permit the operating system designer to meet the demand for a higher level of dynamism or flexibility. Thus, while a conventional operating system would have offered the user an indifferent choice of operating modes, a modern operating system built around new concepts would provide almost countless modes of operation due to the greatly reduced size of its building blocks and the flexibility in configuring them which the structure affords. Thus if the application's execution environment is supported by an operating system, it will be able to alter its appearance from time to time by small increments or on a large scale, if needs be.

The concept of dynamic range as an attribute of an operating system should prevail. This is the ability of an operating system to offer a range of management functions in a smooth and continuous fashion. If an operating system can maintain a wide dynamic range without always having to carry a large burden around and can offer services without hitches, as and when they are needed, it is said to have a good regulation within its dynamic range.

6.4 Packaging Criteria

The packaging of operating system functions should be achieved in such a way that only one clearly defined function would be encapsulated in a module (package, task or procedure). This requirement comes from the advanced module management con-

cept studied and proposed by G. Myers (Refs.13,14), and listed as FTC Rules 4 and 5 (Appendix B). Myers' study demonstrates various types of modules and their relative quality. In short, Rule 5 contends that if a module represents only one function, its reason for being a module would be greater, thus increasing its "strength". The module would also offer simple and clean interfaces to other modules. Combined with the use of a well-structured package specification technique such as that introduced by the Ada language, modules will have fewer "couplings" among themselves, resulting in enhanced modular independence within the operating system. All in all, the operating system for the AASC should look more like "bags of tools" rather than a monolithic body. The "subsetting" of an operating system should be carried out smoothly by selecting appropriate tools from several bags and placing them in another set of bags.

6.5 Operating System Extensions

The collection of tools thus created for a specific application (a spacecraft computer design) may, whenever necessary, be supplemented by additional tools selected or developed by the designer of the application. Such an extension should handle application-specific functions attributed to the operating system. An operating system extension carried out in this way will provide further optimization of its original functionality.

To make this process simpler, access to functions should not depend on conventions which are normally particular to operating systems (such as supervisory calls), but follow a set of standard intermodule protocols. These may be universally applied across the system and include application and system software.

Such protocols include subprogram calls and process creation or invocation. Thus, operating system modules will be formed, linked together and accessed using the same conventions and formalities as those used for application modules. The ambiguity of the boundary between the operating system and application software is thus increased further.

6.6 Process and Data Protection

In the same way that modern structured computer languages address issues of finer and tighter intra- and inter-module

protection schemes, operating systems that create and support runtime environments for software modules written in such a language must also provide a high level of protection. This must cover both processes (software entities that perform functions) and data (that which is subjected to functional manipulation). This trend is in line with the increased reliability requirements for the AASC software. Unlike the philosophy behind many conventional operating systems, protection against accidental or malicious destruction of these software entities should not be left to the mercy of application software designers, or that of eventual users of the system. Such protection mechanisms must be clearly imbedded in the design of an operating system. In reality, such advanced features of an operating system can only be implemented efficiently enough with the aid of the underlying hardware facilities which support it. The above mentioned partnership between the operating system and computer hardware becomes essential. The hardware mechanism that offers such support to software is often called a capability-based machine. In capability-based systems the operating system checks the requestor's access rights and the requestee's protection specifications before granting an access to any of the system objects under its jurisdiction. It has been conventional for a segment of memory or a whole process to be protected in this manner. The trend now is towards a finer level of protection. In the AASC environment, an operating system should be able to enforce the protection of the smallest degree of granularity which the language of implementation, (such as Ada) can define.

Thus one must be able to define a protection scheme around a data structure, and distinguish access to it from its neighbouring data structures by giving it an independent protection mode. Similarly, a process may be granted, for example, a "message receive right" from a specific mailbox, but not a "message send right". If a faulty access was attempted by an application module - or an operating system module, for that matter - the hardware protection mechanism, operating system, or both, should be able to intercept such an access before any damage can be done to the rest of the system.

7. NETWORKING

7.1 Subsystem Network Interface

As described in Section 5, a type of networking is used to support on-board inter-subsystem communication in a highly fault-tolerant manner. The networking consists of redundant network pathways and subsystems fully distributed along them.

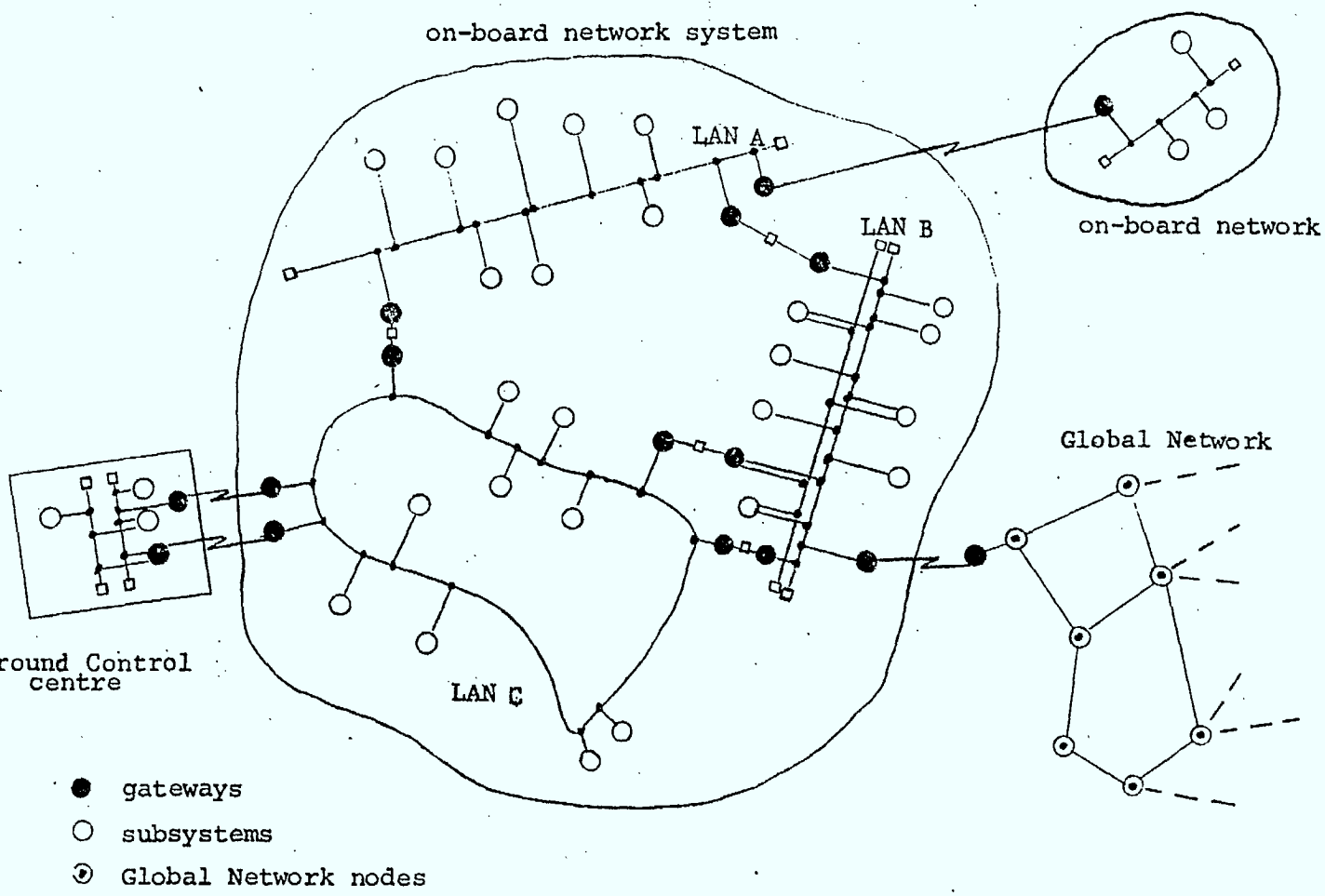
The Subsystem Network Interface Unit (NIU) shall manage the inter-cluster communication channel(s), in conjunction with its resident software, and other software in the configuration, such as the components of the distributed operating system in the PCU that are related to layered communication access.

Such a channel shall exist in the form of a fully distributed Local Area Network (LAN) similar or identical to Ethernet (Ref.15) to comply with the FTC design rules (Appendix B).

Having a fully distributed LAN architecture implies that the network is devoid of fixed bus arbiters or controllers of any sort which govern the flow of traffic on the network. Such control normally would occur in a system with a centralized controller which synchronizes signals, acknowledges access requests from subsystems, grants or refuses accesses, polls, provides timeslots, or otherwise regulates the activities of subsystems on the LAN. This ban on fixed controllers in the network does not preclude the use of a bus controller which is fully software implemented, or which achieves a greater degree of flexibility by some other means. Such a diversified controller may reside in portions dispersed across the network among several subsystems, or be distributed to a lesser extent but change its dispersal from time to time among the subsystems.

In order to avoid dependence on a single transmission channel, the LAN pathways must exist in duplicate, triplicate, or in whatever numbers are needed to satisfy the redundancy level that is demanded by a specific spacecraft bus design.

The number and type of LAN pathways to be incorporated into the design is also dependent on the quantity and patterns of traffic expected among subsystems. Not all subsystems have to be incorporated into a LAN - a network pathway that provides access points to certain subsystems but bypasses others is quite conceivable, as shown in Figure 7.1.



Onboard networking will be designed to meet individual application and logistics requirements of a spacecraft. The extent of redundancy provided to the onboard and ground communication systems must also be determined in terms of the fault-tolerance and throughput needs of the respective networks. All network interconnects shall be achieved through gateways which adopt a standard interconnect protocol.

Figure 7.1 Onboard Networks and gateweying

As implied above, the designer of AASC applications will be free to designate redundant LANs either for added throughput capacity or for increased fault-tolerance. The designation shall be made by providing a set of properly chosen parameters that defines the network's operational environment. The network monitor/control software, which exists in each subsystem, will respond to the parameter sets and attempt to establish the operational profile.

Furthermore, the network's operational profile shall be dynamically definable so that changes to the operational requirements of a spacecraft may be possible even after it is launched. For example, an added level of payload activities decided by ground control may require the opening up of a subsystem's accesses to a LAN that was previously installed during the fabrication of the spacecraft but kept dormant until its use became justified. On the other hand, an on-board equipment failure might cause a portion of an inter-cluster linkage to become a critical path requiring backup support. In order to provide a backup route, traffic on a LAN would need to be suspended and then emptied, or rerouted. In either example, the on-board network monitor/control software shall oversee the implementation of the reorganization, as well as the initial network setup.

7.2 Network Monitor/Control Software

As discussed above, the network monitor/control software is responsible for first establishing, then maintaining and updating upon request, the appropriate operational environment for spacecraft on-board communication networks. The software shall exist independently of the network access control software and have the following functionality:

- to accept parameter sets that define the profile of an on-board network structure and operation
- to verify such configuration requests and report any inconsistencies or predictable weaknesses in the configuration
- to plan and execute a smooth transition from one configuration to the other without seriously disrupting network activities
- to maintain a network operation log by collecting data each time an access or other significant event occurs

and filing them in a log file.

7.3 Flexibility of Networking

The reconfigurability of on-board communication networks shall exist in the form of network monitor/control software and shall be supported by it. For each application, the hardware to support on-board networks and their access mechanisms shall have sufficient redundancy in terms of spare components and communication pathways. Furthermore, such redundant components and pathways shall be strategically placed so that the software can make best use of them in the event of reconfiguration. In short, software shall maintain the flexibility of the networks and manage trade-offs between the maximum over-all throughput and maximum reliability. For example, there shall be a sufficient number of physical connections between each subsystem and a LAN so that the software can respond to a sudden need for alternate routing by allowing a subsystem to tap onto a new LAN. (If a key subsystem is not already provided with access to a LAN which is then chosen as a candidate for an alternate path, the usefulness of the rerouting will be greatly reduced.)

7.4 Cluster Isolation

In addition to providing a signal contact between a LAN and a subsystem, there shall be a mechanism to isolate the subsystem effectively from the rest of the network. Depending on the type of communication media used, such isolation must be effective mechanically, electrically, electromagnetically, optically, or otherwise. Needless to say, proper isolation is necessary to avoid contamination or drainage of signal traffic on the network(s) by accidental transmission to, or grounding of the pathway by the subsystem while the subsystem is logically disconnected or inoperative but physically still engaged to the network.

As an example, if the nature of the LAN traffic is electrical, optical isolation or transformers shall be used to isolate the subsystem electrically from the pathway. Some other technique shall be devised if the traffic is to be optical in nature.

7.5 Standard Methodology

The internal organization of the NIU must adhere to widely

recognized computer network interconnect standards applicable to similar communication needs. This is not to prevent designers from providing the means to absorb structural differences arising from added fault-tolerance requirements or other application-specific needs which are particular to the spacecraft use of such interconnect standards.

The standards shall affect the design of both software and hardware. In designing modules, both vertical and horizontal module boundaries must be recognized and retained. For example, a "layer" within an interconnect model, such as a horizontal module boundary, shall remain visible throughout the design and implementation so that its independence from other layers is enforced.

The rationale for adopting such standardized network access methods comes from the recognition of the greater benefits to be obtained by establishing and maintaining a relationship between the on-board usage of networking technology and its usage elsewhere, both at conceptual and practical levels. Computer communication technology is highly complex and is advancing in a reasonably logical fashion. Because of this complexity, and the astronomical development costs involved in achieving high quality networking, there will be a far greater chance of success if a standard methodology is adopted. This means that one may use readily available components or equipment which are then optimized to a specific application and allowed to evolve as existing standards develop. Although embarkation on a completely new, one-of-a-kind development has been a tradition widely followed in aerospace applications, from the standpoint of system (global) level project optimization it shall be avoided now that superior quality networks are available.

Another benefit to be derived from "going standard" is the possibility of ready connection of on-board communication facilities to other ground or space-based networks. Thus the ground and on-board communicatio networks. Th established using recently developed gateway technology. Another use of "gateWaying" would permit a clear-cut interconnection of networks on board a spacecraft. In time, computer communication technology is likely to evolve to accommodate such diverse communication modes as ground-based office or factory systems, and multi-satellite global networks in a unified fashion.

Any deviation from standard methodologies which may become necessary when they are applied to a specific on-board use, shall be handled in terms of customization of the smallest pos-

sible, and well isolated portion(s) of such methodologies or models. If customization alone will not satisfy the requirement, distinctive and well modularized components, either software, hardware or both, shall be added to contain the deviation in an identifiable manner.

7.6 Layered Protocol Structure

Implementation of the above mentioned standard network interconnect schemes, including any application specific extensions to it, shall be achieved in such an open-ended fashion that the process may easily benefit from evolving technology. A layer must be designed and implemented with maximum independence from neighbouring layers. If a particular hardware implementation would force the use of any specific type of hardware or software in the neighbouring layers, the choice of that hardware must be avoided unless there is an over-riding rationalization. To give another example, if the subsequent upgrading of a layer, either in software or hardware, results in a modification to the software, hardware or both, of neighbouring layers, the original design for the layer would be termed faulty.

The emphasis on layered design does not imply an interconnect model with fixed, rigid layers that would never change its infrastructure. The objective of the argument here is to incorporate in the conceptual design as much flexibility as possible in order to maximize the benefits from future technological developments. If future changes in technology demanded the subdivision of one or more protocol layers, a re-evaluation should be made at that point and a decision made as to the accommodation or rejection of such major modifications. Also, if at some point in the future, the amalgamation of two or more layers made better networking structure, it should then be given serious thought.

As an example, the use of an Ethernet-like LAN is given. If the present 10MHz serial access coaxial cable pathway were to be replaced by a 100MHz fibre optic cable in the near future, the only impact of this drastic change in transmission speed, and hence capacity of the channel, would be in terms of greatly increased throughput. To implement this technological change, only the protocol used and the bottom layer handler would be altered. (Layer 1 being the physical layer, both handler and protocol are likely to be hardware implemented.) In other words, what happens to the physical layer shall be fully transparent to the layers above it. A similar transparency should be in effect at all layers.

A layer in such a multi-layered interconnect model shall represent a "level of abstraction". The level of abstraction shall be manifested in the form of the functional complexity attributed to that layer. Functional complexities of various degrees will be ordered into a "hierarchical functional tree" and inter-relationships established between them. The higher structural layers shall contain the highest levels of abstraction.

Levels of abstractions in a hierarchical structure must exist in an orderly fashion: the granularity of abstraction from one layer to another shall remain reasonably uniform, and there shall never be an inversion of abstraction, i.e., the existence of a functional layer with a lower-level neighbour of a higher degree of abstraction than itself.

7.7 Gateways

Each on-board LAN shall have at least one gateway station. The gateway shall be the sole method of linking two or more LANs or one LAN to some other form of communication network. There shall be no designed-in restrictions as to the maximum number of gateway stations on a LAN except for those due to the traffic capacity of the LAN and its access facilities. For instance, an on-board LAN may require two gateway stations, one for the ground link (e.g., T & C subsystem) and the other to communicate with another LAN.

Thus, a gateway shall serve as a standard means of exchange between a LAN and others, and its functional and implementational profiles shall resemble those of the general description of a spacecraft subsystem: it shall connect itself to a LAN in a standard multi-drop fashion; it shall be fully isolatable; it shall be powered in the same way as other subsystems; access to it shall be governed by the same LAN protocol as is applicable to other subsystems on the same LAN. Figure 7.1 depicts the relationship between LANs, gateways and other forms of communication network, and the manner in which a gateway could be used to interconnect various networks.

In principle, if there is more than one LAN on board a spacecraft, there shall be linkage among them via gateways. The reason being to minimize the chance of accidentally causing a total isolation of a LAN and its associated subsystem cluster from the rest of the on-board system. Ideally, a LAN should have duplicated gateways to other LANs or another communication

media in order to further reduce such possibilities.

A gateway station will talk and listen to a gateway station that belongs to the target network. The possible destinations of a gateway shall include the following: ground control station; ground or satellite-based global network; another LAN on board the same spacecraft; a LAN on board another space or aircraft, or other forms of mobile network (such as cars or ships). In spite of the variety of destinations, a gateway must maintain consistent functional and interface characteristics.

8. PROCESSOR COMPLEX UNIT

8.0 General

The PCU complex, as shown in Figure 5.2, will consist of redundant System Packet Bus(es), Subsystem Processor module(s) and Memory module(s).

A subsystem processor module will contain a processor (either a GDP or an IPL), a processor packet bus and bus interface unit(s).

A memory module will consist of a Random Access Memory (RAM) array and a Memory Control Unit (MCU).

These modules are to be connected by one or more system packet buses which also connect the Interface Processor Link (IPL) to the Processor Complex Unit (PCU).

8.1 System Packet Bus.

The System Packet Bus (SPB) connects the various modules within the system. A system may have up to 8 SPBs, depending on the throughput and redundancy levels required. The SPB has 16 address/data/specification lines, 3 arbitration, 3 control, 2 parity and 1 error reporting line. The principal features of the system packet bus are message-based protocol, synchronous operation, and independent distributed arbitration.

The message-based protocol will be optimized for the minimum number of bus cycles and will ensure that there is no 'bus dead-time'. The packet bus will support multiple requests through a very tight protocol. Messages will be sent back-to-back with a very strict ordering. The end of a message will be indicated in advance to facilitate a smooth transition from message to reply.

The system is designed to be synchronous, with all units running off the same clock.

Arbitration and control will be fully distributed among BIU/MCUs, complying with Rules 2 and 3 of the FTC Rules. The control lines specify message type and end-of-message in an encoded form. Any device on the bus will have the ability to look at the control lines and observe what is happening in the system.

Bus arbitration will take place on two levels. At the primary level each BIU will maintain a time-ordered FIFO queue. This operates when there is no contention for the bus. When multiple requests occur within a clock cycle then arbitration goes to the secondary, priority-ordered level, again conforming to the second of the FTC Rules. Arbitration is hidden from the system so that a high bus utilization is maintained with a minimum arbitration overhead. The processor priority is software controlled and is not related to the hardware ID. Replies will not require arbitration since strict message/reply ordering is maintained.

Parity lines will be interlaced and cover address, specification, data and control lines. Any time a device is driving the bus it creates parity, which is checked by every BIU on the bus regardless of the message destination. The extensively distributed checking emphasizes the third FTC rule.

The bus will have an error report line driven by any module detecting an error, which will specify the type and location of errors.

A system with a single bus system can only support one memory module. On a multiple bus system accesses can be distributed across all buses. An address is split into sections which are sent consecutively, one per bus. On long accesses this will allow more than one MCU to fetch data, with the proviso that memory on each bus must be of identical size. BIUs have a separate communication line which enables them to coordinate this 'memory interleaving'.

8.2 MCU Features

The MCU is a form of logical memory controller. It will support up to four megabytes of memory per chip. The current implementation of the memory array is 32 bits wide with 7 ECC bits and 1 spare. This spare can be switched by software control into any of the other 39 bits. Currently, an MCU will support 16K, 64K, 256K, or 1M bytes of dynamic and static Random Access Memory.

In order to prevent aliasing and ensure that the correct address has been accessed, the Error Correcting Code will be generated across address and data lines. The ECC will only correct data as it goes to the bus. Errors in memory itself will be corrected by 'scrubbing' and refreshing by the MCU. Special commands will allow access to the ECC bits to ascertain the lo-

cation of errors and to clear location and memory.

The MCU will support read modify write locking for indivisible operations, to maintain data integrity in a multiple access environment.

8.3 Processor Module

The processor may be a GDP or IPL. The Processor Packet Bus (PPB) belongs to a processor. It is similar in design to the SPB but does not require the same message ordering. A processor will merely issue a request and wait for a reply. A PPB intersects with SPB(s). At the intersection a BIU controls the data flow. Thus the number of BIUs is dependent on the number of SPBs and PPBs.

8.4 Fault-Tolerant Aspects

The ASM Final Report points to the need for development of a highly reliable fault-tolerant computing capability if autonomous spacecraft maintenance is to be achieved. The philosophy involved in the development of the AASC Processor Complex Unit has been to provide such fault-tolerant coverage in as complete a way as possible combined with error-free computation. It is intended that the AASC will meet the ASM Design Requirements and conform to the FTC Rules.

In this context, the following are points which the designers have been particularly concerned with:

- the capability for recovery should occur at the same level as the fault
- hardware should have the ability to handle hardware fault recovery, utilizing the four fault-handling stages:
    - detection
    - containment
    - diagnosis or analysis
    - recovery
- the achievement of a range of fault-tolerant capabilities through hardware replication which can be directed to support increased performance or fault-tolerance as required.

- the creation of an architecture which will never die.

In support of these considerations, a fault-handling cycle (detection, confinement, recovery) is designed into the hardware level.

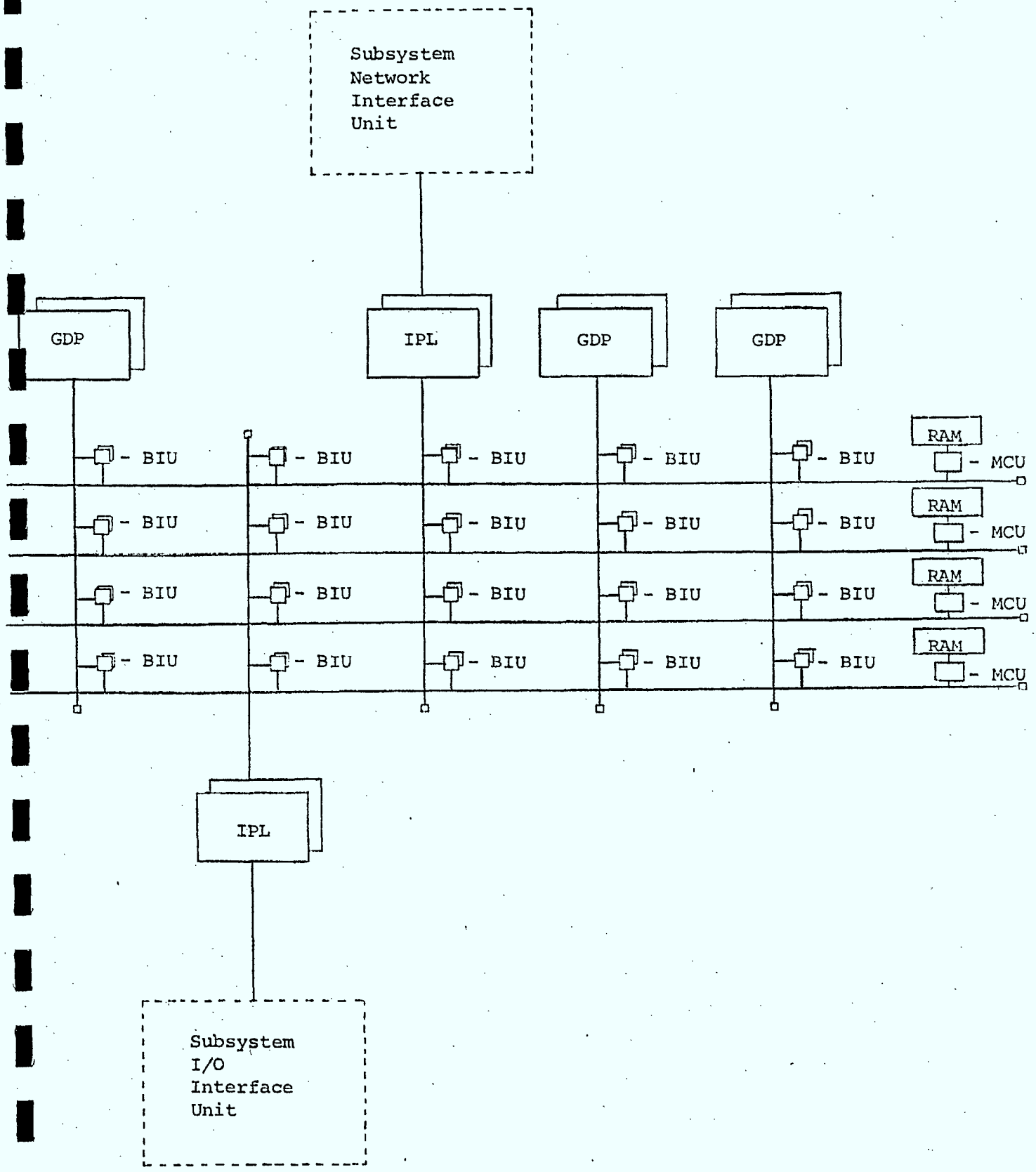
8.4.1 Detection

Methods of detection used will include ECC for checking memory errors, parity to check bus errors, duplication and Functional Redundancy Checking (FRC) to verify processor integrity.

FRC mode (Ref.16) is a specific fault-tolerant feature designed for checking components. It allows for the replication of components as Master and Checker, as in Figure 8.1. For example, in the case of the GDP, this is to be achieved simply by pulling a pin on one chip and mounting the two piggy-back style. In this mode, the Master drives the bus, while the Checker only receives signals. Any disagreements between Master and Checker will be indicated on a hardware error line. The alliance of Master and Checker does not consititute a Master/Slave relationship, which would be a violation of the second FTC Rule. As will be explained later in Section 8.4.6 on Module Shadowing, this is not a fixed state.

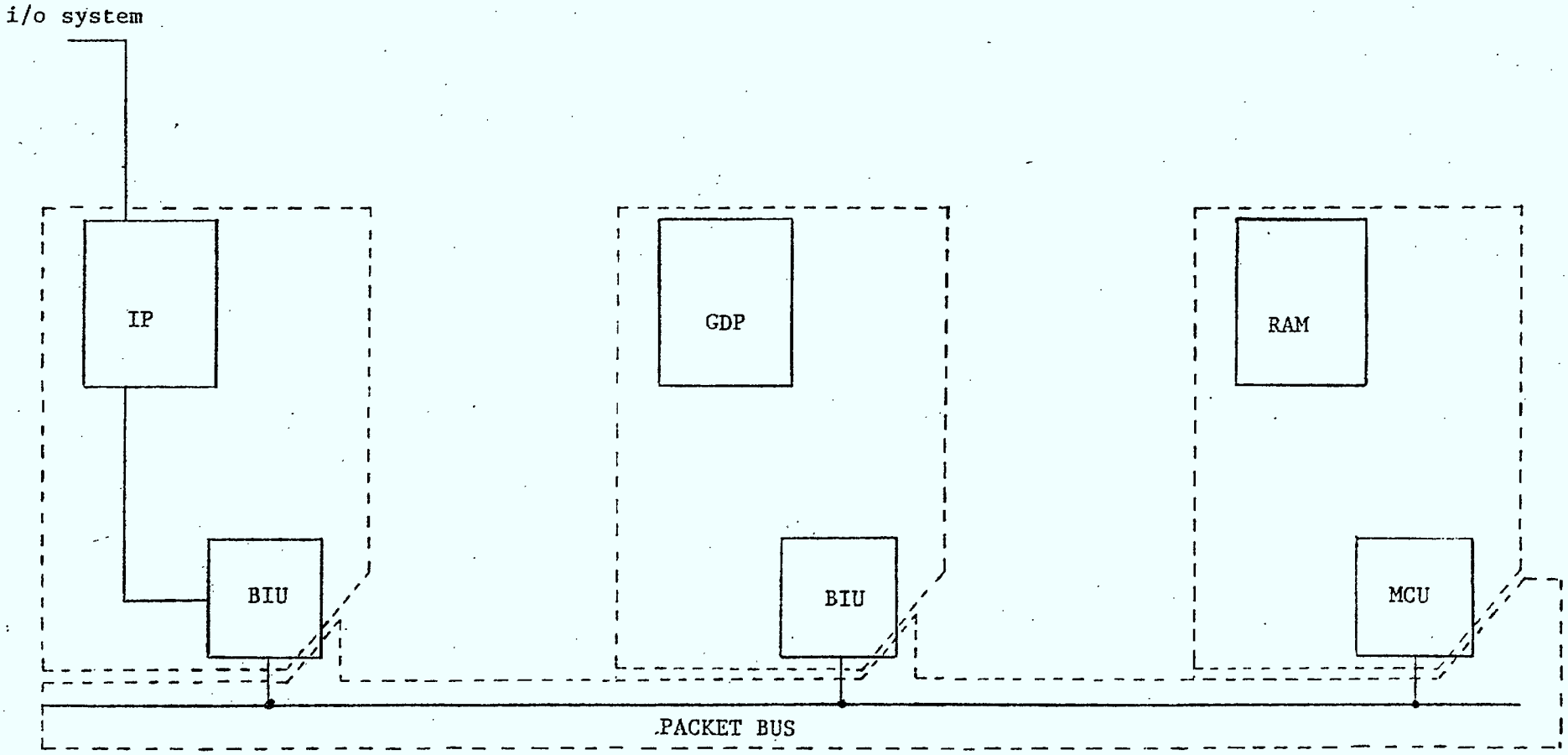
8.4.2 Error Confinement

The complex is divided into four confinement areas as indicated in Figure 8.2, in order to limit error propagation and localize faulty and damaged areas. There will be a limited number of tightly controlled interfaces between them to prevent bad data entering or leaving the area. These confinement areas consist of GDP, IP, RAM and SPB areas. Where the SPB is contiguous with other areas, there will of necessity, be a slight overlap within the BIU/MCU. The introduction of distinctive hardware-supported fault barriers in the design is another feature of the processor complex. An appropriate combination of detection methods will be used in the different confinement areas.



FRC provides for units to be physically superimposed in a checking mode. On processors it is a simple hardware operation, on BIU's it is a software configuration.

Figure 8.1 Functional Redundancy Checking in the PCU



The four confinement areas; GDP, IP, RAM and SPB. The inclusion of the arbitration network within the BIU FRC coverage causes a slight overlap between the BIU/MCU and packet bus confinement areas.

Figure 8.2 PCU Confinement Areas

8.4.2.1 GDP Confinement Area

The aim is to check the interface rather than the individual GDPs in order to contain faulty data. Therefore, in this instance, both GDPs will be run as Masters. FRC on GDPs themselves has been described earlier. FRC on BIUs will be a software configuration. In a GDP confinement area, all modules would be duplicated. One set of GDP/BIUs would be designated as Masters, the other as Checkers.

The coverage of this area, as shown in Figure 8.3, includes the GDP, its processor packet bus, packet bus hardware and its BIU(s).

8.4.2.2 IP Confinement Area

FRC in this area will only cover outgoing data, as the verification of incoming data must be the responsibility of the Application Processors (APs).

8.4.2.3 Memory Confinement Area

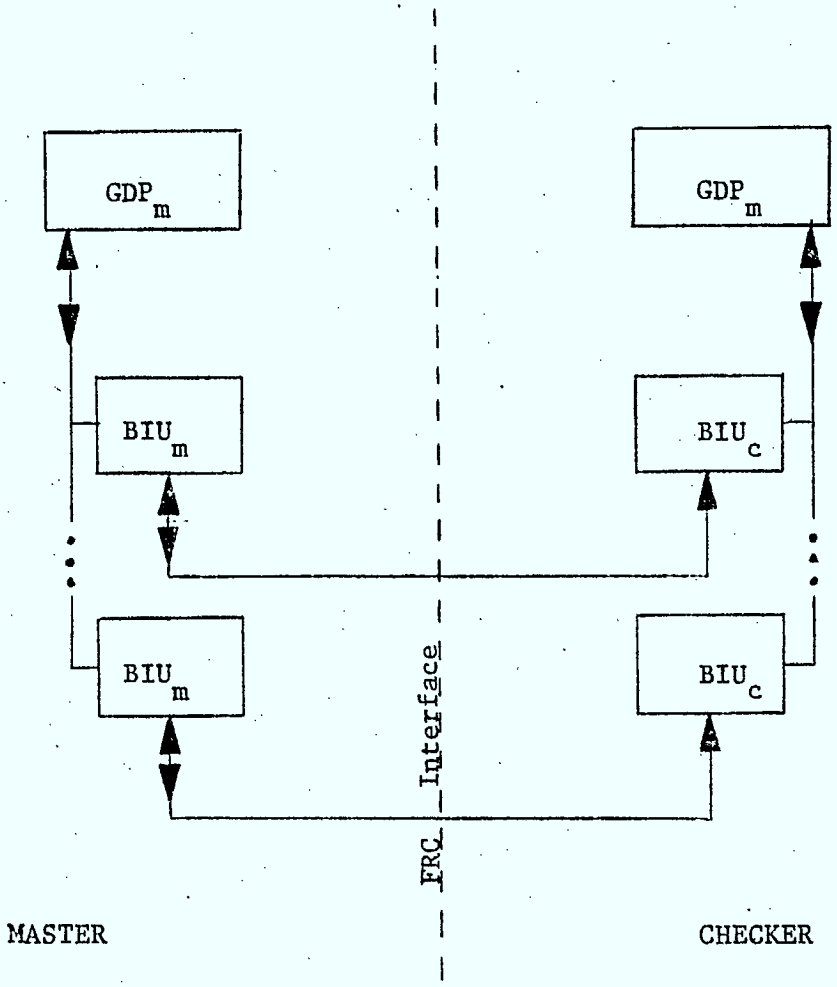
This area covers the RAM elements, address lines, data lines, array support logic, and MCU. It will employ ECC with scrubbing and FRC, as previously described. Figure 8.4 shows the MCUs as Master and Checker in FRC mode and sharing the interface to the RAM array. This mode covers the support logic.

8.4.2.4 System Packet Bus Confinement Area

The Address, Data and Control lines are within this area, as described in Figure 8.5, and are covered by parity. The arbitration lines are duplicated. One set is driven and received by the Master BIU and checked by the Checker. The Checker drives and checks the other set. A disagreement in the arbitration network will cause the Master/Checker to enter different states which will, in turn, be reflected on the control/data lines and eventually will result in the rejection of the pair.

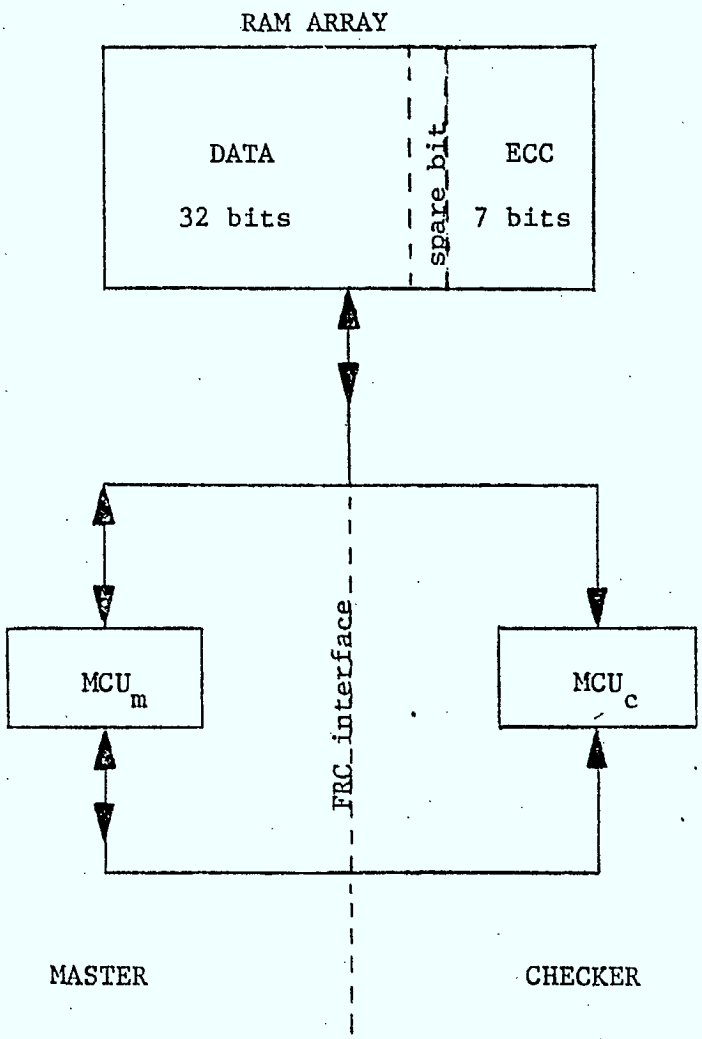
8.4.3 Reporting and Logging Network

This consists of a report line for each system packet bus (BERL) and one for each processor packet bus (MERL), with BIUs and MCUs forming nodes on the network. If a module detects an error on the bus it will broadcast an error report, which will



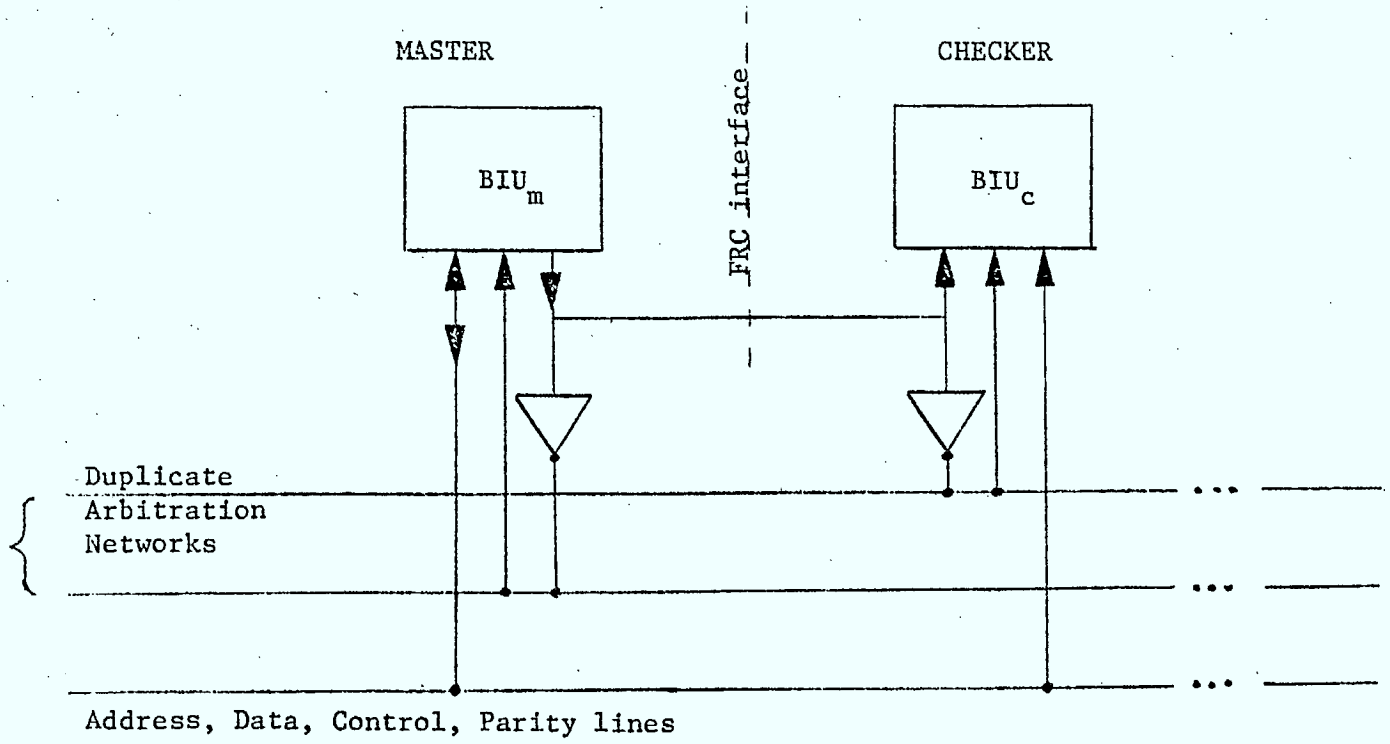
FRC provides a duplicated set of modules. As the concern is to prevent faulty data leaving the area only the BIU's are configured in a Master/ Checker relationship to check the output.

Figure 8.3 GDP Confinement Area



FRC covers the interface to the RAM array and Packet Bus. An ECC log is kept in the MCU.

Figure 8.4 Memory Module Confinement Area



This figure shows the detail of the overlap between packet bus and BIU confinement areas involving the arbitration lines. The address, data and control lines are checked by parity.

Figure 8.5 Packet Bus Confinement Area

be received by all nodes on the reporting network, as shown in Figure 8.6. Everyone on the system will know where and what kind of an error has occurred and will be capable of acting appropriately. BERLs are duplicated and MERLs are included in FRC in the GDP confinement area. There will be no single point dependencies in this arrangement in accordance with FTC Rule 1.

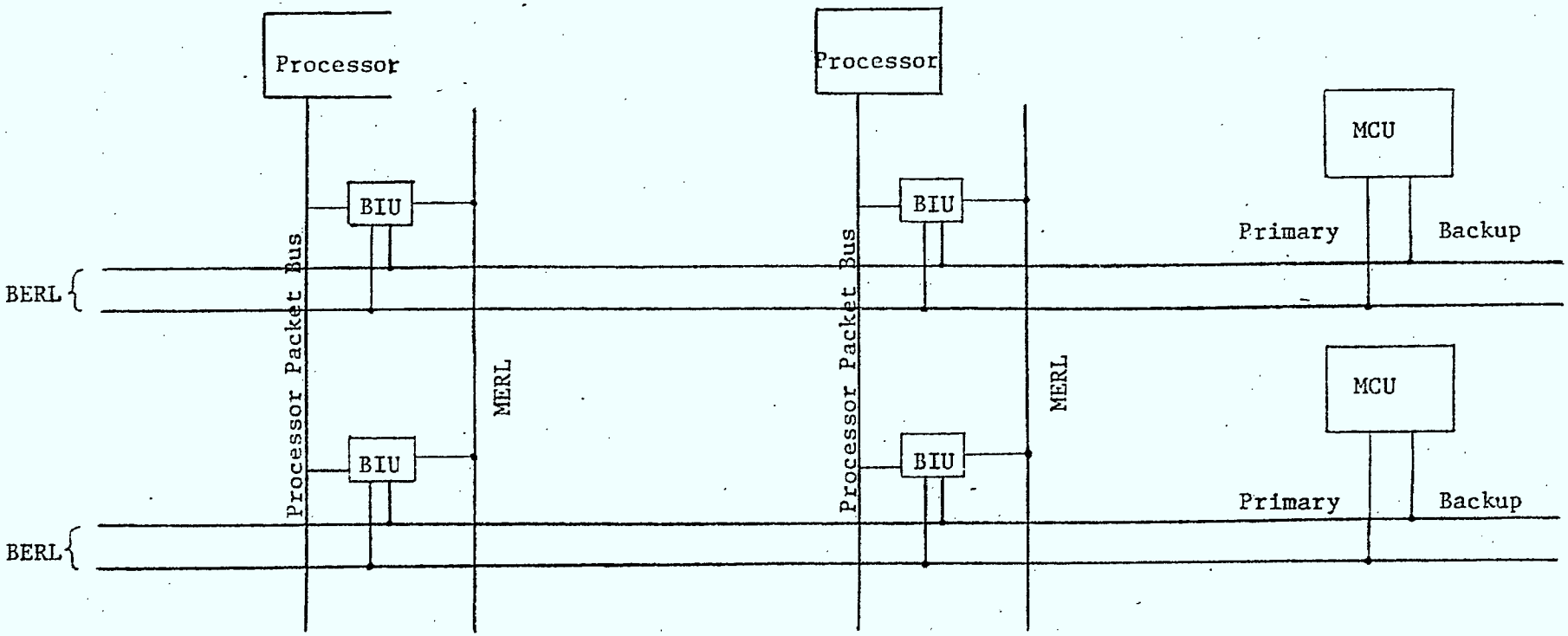
When the error message has been broadcast, all modules will stop. A pause occurs to allow transient errors to subside, following which all accesses will be retried. If the error reoccurs during retry, it is considered permanent. If the retry was successful, a second pause follows. Recurrence of the error at this point indicates its permanence and no further retries are attempted. Both pauses are programmable from 10 microseconds to 2 seconds.

The first error to be notified is logged. Succeeding errors are only counted as they may be a direct result of the initial error and, if permanent, will reoccur later. Each BIU and MCU has an identical copy of the log which indicates the nature of the error, location and count.

It should, therefore, always be possible to read the error log at some point for ground reporting purposes and maintain a fault management activity record in accordance with ASM Requirements #9 and 10.

8.4.4 Error Response

If retry fails, i.e. the error is permanent, then reconfiguration takes place either on a bus basis or a module basis. The BIUs are capable of recognizing the location of the error from the report and will reroute the message accordingly. If that fails then the processors try to lay down their processes. If they encounter an error during this procedure then the fatal line is pulled and the processor is disconnected automatically. The only impact at the application level will be a slightly longer access to memory or the early setting down of a process. However, it is felt that a clear indication to the upper system layers must be made at this point to allow them to take collective action at the respective levels. An example would be the firing of orbit control thrusters. If a reconfiguration occurs due to an internal error right in the neighbourhood of this critical action, the thruster control software in the AOCS subsystem must know the fact in order to determine whether the refiring of the thruster is necessary or not.



An error causes a message to be propagated throughout the system on both MERL's and BERL's to all nodes. Duplicate logs are kept in all nodes.

Figure 8.6 Error Reporting Network

8.4.5 Redundancy

As stressed elsewhere, redundancy can be utilized either for increased throughput or fault-tolerancy. A memory module can be physically connected to two buses, one primary and one backup. Both buses can be used normally but if one fails, the MCU is switched to the backup bus, as shown in Figure 8.7. The BIUs are informed via the reporting network and all the address space is given to the backup. This reconfiguration is transparent to the processor.

8.4.6 Module Shadowing

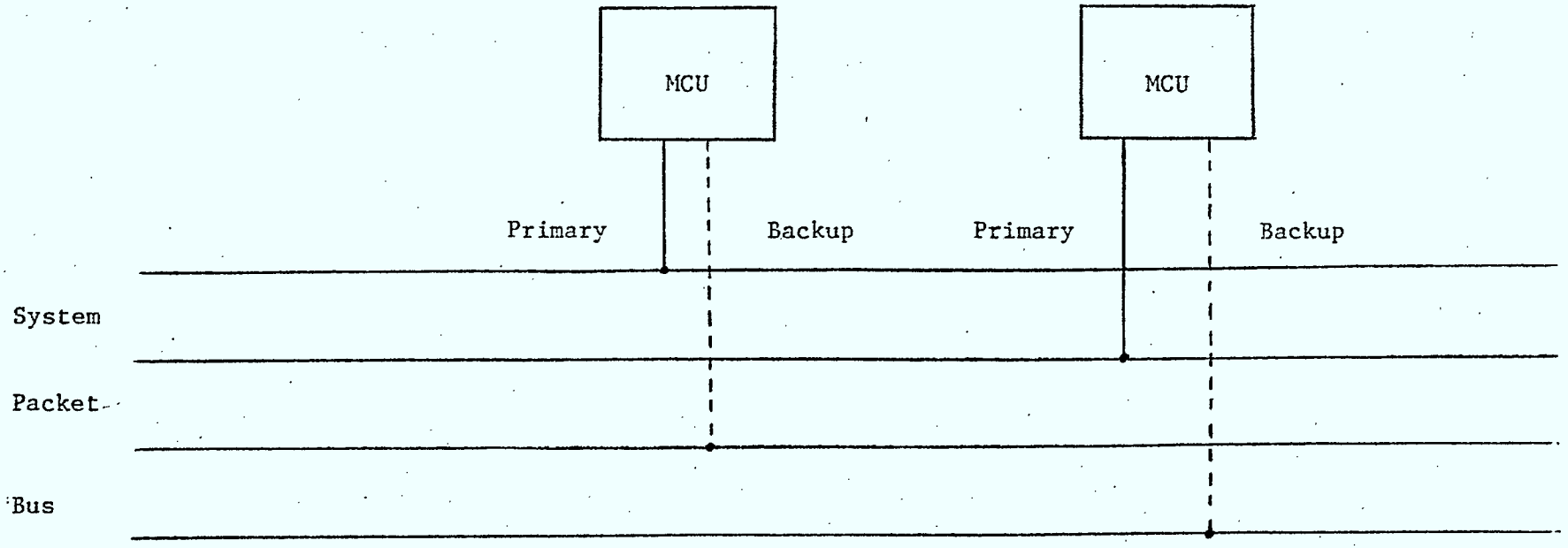
This is a higher level of fault-tolerance which can be used when requirements are more stringent, such as in an ASM application. A shadow is a redundant resource totally dedicated to maintaining a complete and current backup of the module state information for a period of time designated by software control. In the case of processors, two identical FRCd modules, primary and secondary, are married using software. They need not be physically contiguous. Figure 8.8 shows an example of processors configured as a married pair. When two cards are married they will alternate addresses, i.e. one will drive the bus, whilst the other receives. After a fixed time period, their roles will be reversed. This provides a hot standby, each pair having a complete set of information. If an error occurs, it is reported to the whole system. The BIUs recognise the faulty unit, which is then switched off and the access is retried on the remaining unit. System software can retry the pair as married or divorce them if the fault is permanent. This mode is completely controllable by software.

Memory modules can also be shadowed, in which case four buses will be required. Each module will be linked to two different buses, so that failure in one of a pair of buses will not impair RAM shadowing.

8.4.7 Latent Faults

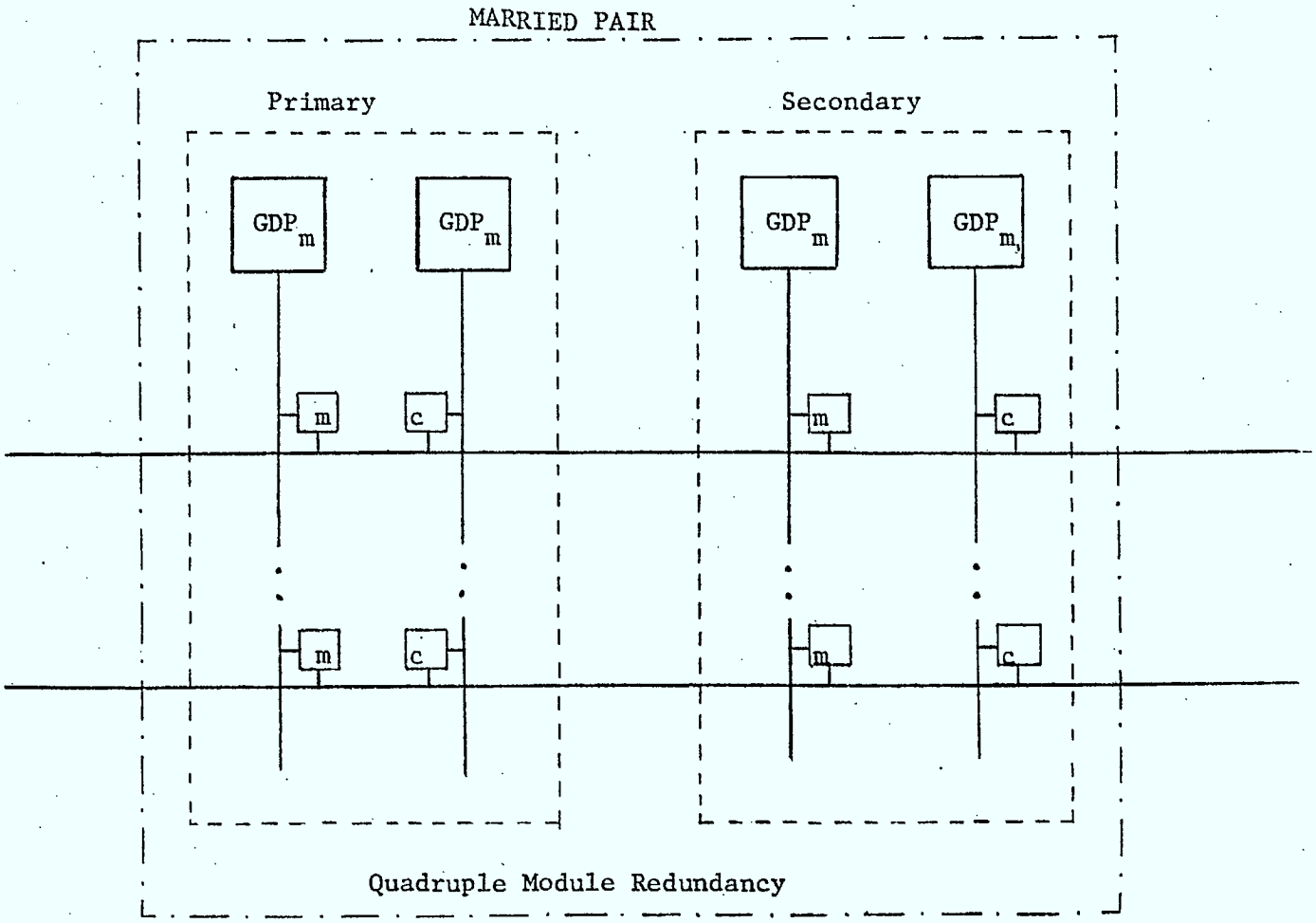
This is a fault existing in some part of the system which would normally be dormant. The strategy here involves exercising these parts at times when they would usually be inactive.

The MCU refreshes memory locations regularly in every array, and soft errors are erased.



Bus reconfiguration requires four system packet buses in order to ensure that there will be only one MCU per bus even after reconfiguration.

Figure 8.7 Bus Reconfiguration



A married pair is joined by software. Accesses are alternated between them so that each maintains identical information. FRC ensures checking of individual partners. Software ensures disconnection of faulty units and correct continuation of the healthy partner.

Figure 8.8 Module Shadowing

Detection mechanisms periodically have error conditions forced into them by software. In accordance with FTC Rule 3, the roles of master/checker are reversed by software to ensure there are no permanent arbiters or judges and to allow for checking.

Software control initiates and observes error reports and also periodically invokes recovery operations.

The BIU/MCU have special commands to facilitate the exercising of these dormant areas.

8.4.8 System Configuration

A high degree of fault-tolerance can be built into the subsystem. Figure 8.8 shows a typical highly fault-tolerant configuration. The GDPs are arranged in FRCd pairs, i.e. physically superimposed as Master/Checker. Through software control two pairs can then be married as primary and secondary processors in a Quadruple Modular Redundancy (QMR) arrangement. If one pair fails and will not respond to a retry, FRC will indicate which is the faulty pair. If necessary, a third pair, as shown in Figure 5.2, can be brought in as a replacement partner. If either of these partners fail, the remaining pair can continue to operate but at a greatly increased risk, in which case, the configuration would resemble Figure 8.3. When the fault-tolerant requirements are not so stringent the three pairs will operate separately to provide a greater processing capability, but always checked by FRC mode.

System packet buses can also be quadrupled to enable reconfiguration in the event of bus failure.

The level of fault-tolerancy designed into the configuration need carry no penalties in the way of performance, unused options or changes to the architecture during recovery.

9. APPLICATION INTERFACE

9.0 General

There are five aspects to be considered in terms of establishing linkage when a spacecraft subsystem is to be loaded to the AASC. They are the following:

- hierarchical control structure interface
- operating system interface
- network protocol
- subsystem i/o interface
- physical connections.

The first four of these will be achieved mainly by arranging linkages in software terms. The last one will require hardware considerations in achieving the linkage and installation. The following subsections will clarify the nature of the linkages and the general method of achieving them.

9.1 Hierarchical Control Structure Interface

Each AASC application will have its system objective(s) and functional requirements to achieve such objective(s). Some of the functional requirements will be broken down into subfunctions, each of which may be further decomposed. The collection of functions and subfunctions will form a hierarchical control structure.

The functionality of a given subsystem will occupy a region in this abstract structure. Within the hierarchical structure, in order to achieve over-all functional objectives a subsystem shall communicate with its neighbouring subfunctions by exchanging messages that follow a defined protocol, as shown in Figure 9.1. This control system protocol shall be designed and defined for all of the hierarchical structures by the spacecraft applications engineer even before the detailed design of any subfunctions begins. Note that the hierarchical control structure will exist in an abstract functional domain. In this respect, some subsystems will represent functions of a higher abstraction than others. However, the hierarchy does not imply

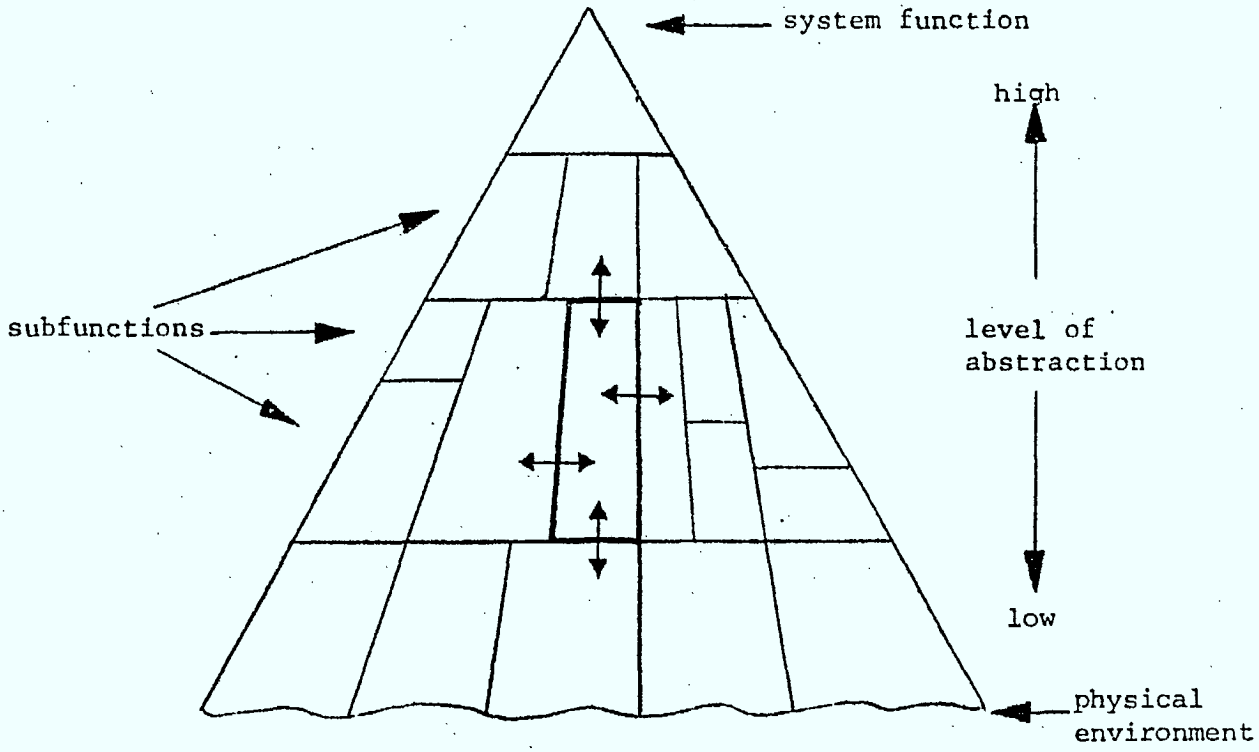


Figure 9.1 Hierarchical control structure and inter-function protocol

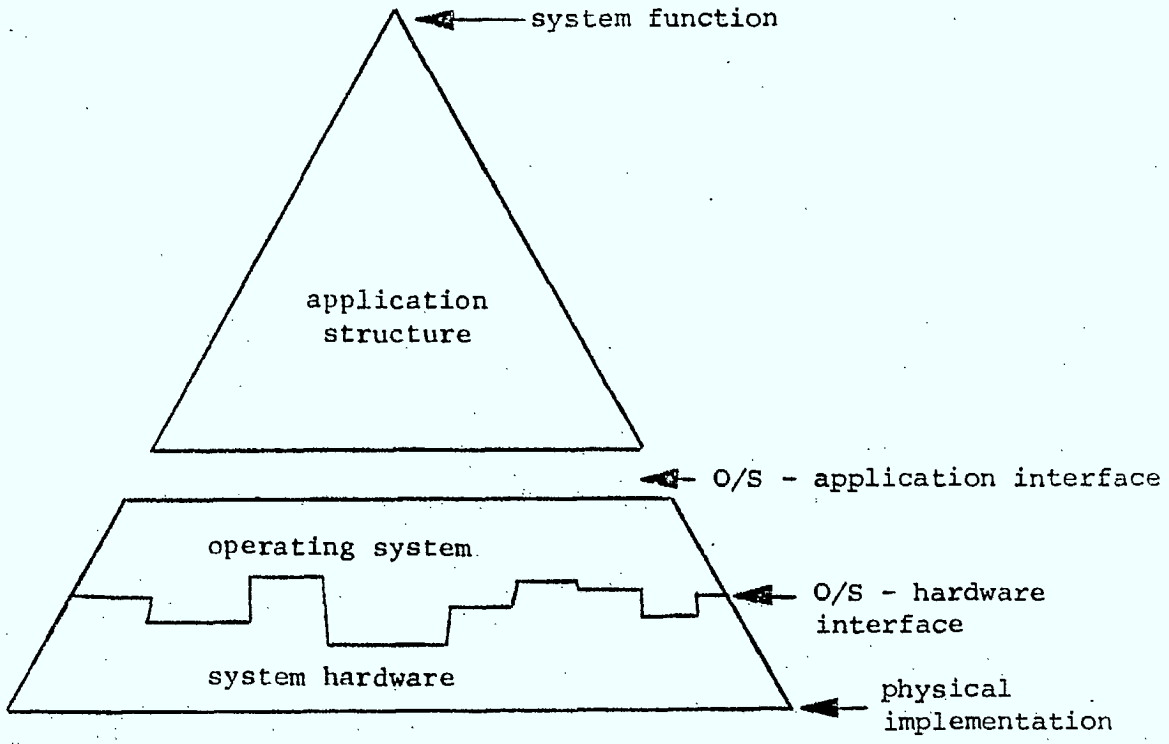


Figure 9.2 Standardized application-operating system interface

that a fixed master-slave relationship be implemented in hardware. A fully distributed hardware structure is still used for flexibility in configuration and to support non-dedicated redundancy.

9.2 Operating System Interface

As discussed in Section 6, the operating system used in the AASC shall be well integrated with the computer hardware to provide application software with an elevated access interface to the processor and other hardware capabilities. The protocol exchanged between application modules and the operating system constitutes the interface.

There is a trend in the software industry to standardize this interface. If that materializes in a reasonable fashion, it is strongly recommended that it be adopted even at the cost of an increased overhead. This implies that access to operating system services will be restricted to those adopted in the standard set. By doing so, system to system portability of application software is assured and thus a way is opened to exchange application software between projects, or incorporate externally available software without difficulty. It is a well established fact in Software Engineering that any small benefit one obtains by optimizing access methods to a specific need is too often wiped out by the greater losses suffered in losing compatibility with a common scheme. The isolation of software from the main growth path shall be avoided to maintain the overall quality of the software.

In spite of the fact that the AASC is an on-board system, use of a file system is not precluded. The implied higher level of data representation made possible by processes operating at higher levels of abstraction will naturally demand the creation of files and a management facility for these files.

9.3 Network Protocol

A subsystem will be implemented on a cluster that constitutes a node on the AASC networking. When two or more subsystems are to exchange messages via network(s), the inter-subsystem, or the inter-cluster communication will rely on a network communication protocol as shown in Figure 9.2. While the operating system and the NIU will look after the lower layers of the multi-layer protocol scheme, application

software on the subsystem will still have to provide support for the upper layers of the protocol.

The nature of exchanges over the network will be substantially more abstract than that seen on those conventional spacecraft system bus. The results of analyzed or processed data will be sent or received via the LAN(s) instead of raw data such as telemetry information. However, to comply with the overriding ground requirements specified in the ASM requirements, there must be a provision in the protocol to allow mass transfer of data. A burst mode of transfer defined as part of the protocol will permit, for example, bit by bit reporting from telemetry subsystems to the ground control via LAN(s) and the ground gateway subsystem. Similar mass transfer shall be anticipated for the occasional remote-loading of software or data files to selected subsystems by ground control.

The basic (lower layer) network protocol supported by the cluster shall have a hierarchical broadcasting facility to allow selective broadcasting.

9.4 Sybsystem I/O Protocol

At the lower end of the control hierarchy most of the subsystems will interface i/o devices of some kind, as depicted in Figure 9.3. These i/o devices are often integral parts of a spacecraft design. An AOCS subsystem, for example, will employ several different types of sensors and a few actuators. For a Grand Gateway (Telemetry and Control) subsystem, transmitters and receivers with their antennae assemblies will serve as i/o devices. An on-board knowledge base will probably require access to a few layers of mass storage devices of some kind.

Since the nature and characteristics of subsystem i/o differs drastically from one subsystem to the other, the protocol at this level will be highly customized. However, at a few layers above the actual physical i/o, a great deal of effort must be made to standardize device access methods. For instance, there shall be as few device handlers as possible (say, one each for stream and byte i/o). Access protocol between application software and these device handlers shall also be standardized (standardized calling sequence, parameter set format, system response format, logical device naming, request queueing conventions, error handling schemes, etc.).

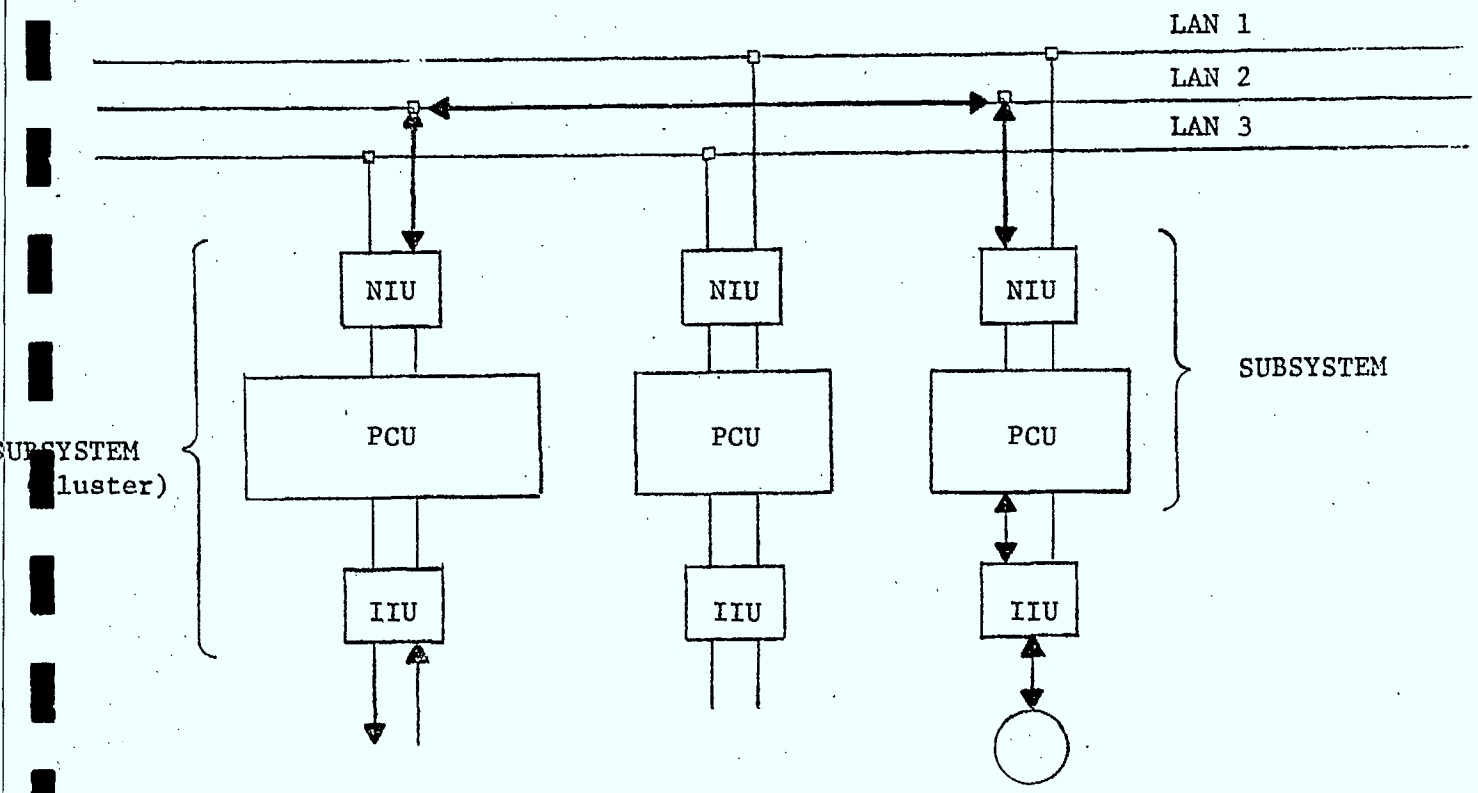


Figure 9.3 Inter-cluster Network Protocol

Subsystems exchange information via onboard networks. The communication channel is supported by a standard hierarchical protocol of which the lower layers are supported by the LAN and the NIU hardware. The subsystem software must provide whatever upper layers the protocol needs to achieve the requirements of its host subsystem, again following the standard protocol scheme.

9.5 Physical Connections

Physical connections between a subsystem and the AASC occur in two areas: subsystem linkage to LAN and subsystem i/o interface to i/o devices. In the former, network interface cables from the NIU will be connected to multidrop transceiver units on selected LAN(s). In the case of multiple LANs, the application design engineer should take the decision during system design as to which LAN will receive the connection.

The Subsystem I/O Interface Unit (IIU) is basically a computer system in itself, apart from the PCU, and of a more conventional architecture with a localized system bus. There will be several versions of IIU based on the selection of CPUs, controllers, the amount and type of local memory support, and the type and volume of sensory inputs to and control outputs from the unit.

In some cases IIUs will be embedded in subsystem i/o devices in the form of dedicated redundant units to achieve a customized control function in a conventional fault-tolerant arrangement. In other situations where the subsystem does not require encapsulation of its controlling computer elements, the IIU will be physically mounted outside the i/o device, next to the PCU for that subsystem.

10. CONCLUSION

A conceptual design has been introduced for an on-board computer to meet the stringent requirements and design criteria outlined in this report. This work is one stage in the development of such a computer system. In furtherance of this process, it is recommended that the study of the advanced concepts already begun should be pursued so that a sound technological base for on-going space utilization may be laid down. It is felt that such a knowledge is a necessity for achieving preeminence in future space activities.

In addition to the continuing theoretical study needed, some recommendations for further development and study are made. These are concerned with a detailed breadboarding of the AASC, and would lead to the development of a system which would make use of state-of-the-art concepts and techniques and would open the way to the future adoption of advanced control and fault-tolerant features which are continually being explored.

11. RECOMMENDATIONS FOR FOLLOW-ON WORK

In order to verify the AASC concept and obtain hands-on experience with the new computer architecture proposed in it, the following phased actions that include theoretical and practical activities are recommended. Such actions are expected to be carried out step-by-step, each one feeding results for the next stage in the design cycle.

Step 1 Concept Review.

An elaboration should be made of the system concept, building blocks, hardware/software trade-offs, fault-tolerance features, and related technologies of the AASC in respect of the anticipated requirements for future spacecraft.

Step 2 Conceptual Breadboard Development and Tests.

An experiment should be planned using a breadboard system to test concepts refined in Step 1. A flexible breadboard should be designed and built to test the basic concepts of the AASC. Figure 11.1 shows the breadboard system consisting of at least two (2) computer stations (micro or minicomputer) to be linked by a local area network.

The computers should be used to test the algorithms defined or implied in the AASC but not the AASC hardware. The local area network would simulate actions of the AASC system bus. The objective of the breadboard experiment should be verification of system level control concepts, inter-cluster communication, and fault-tolerance algorithms. The similarity between the breadboard and the actual AASC should be consistent with low experimental costs and maximum benefits from the hardware. The computer stations should be loaded with software modules that simulate the algorithms and control methodologies. Tests would be planned and carried out.

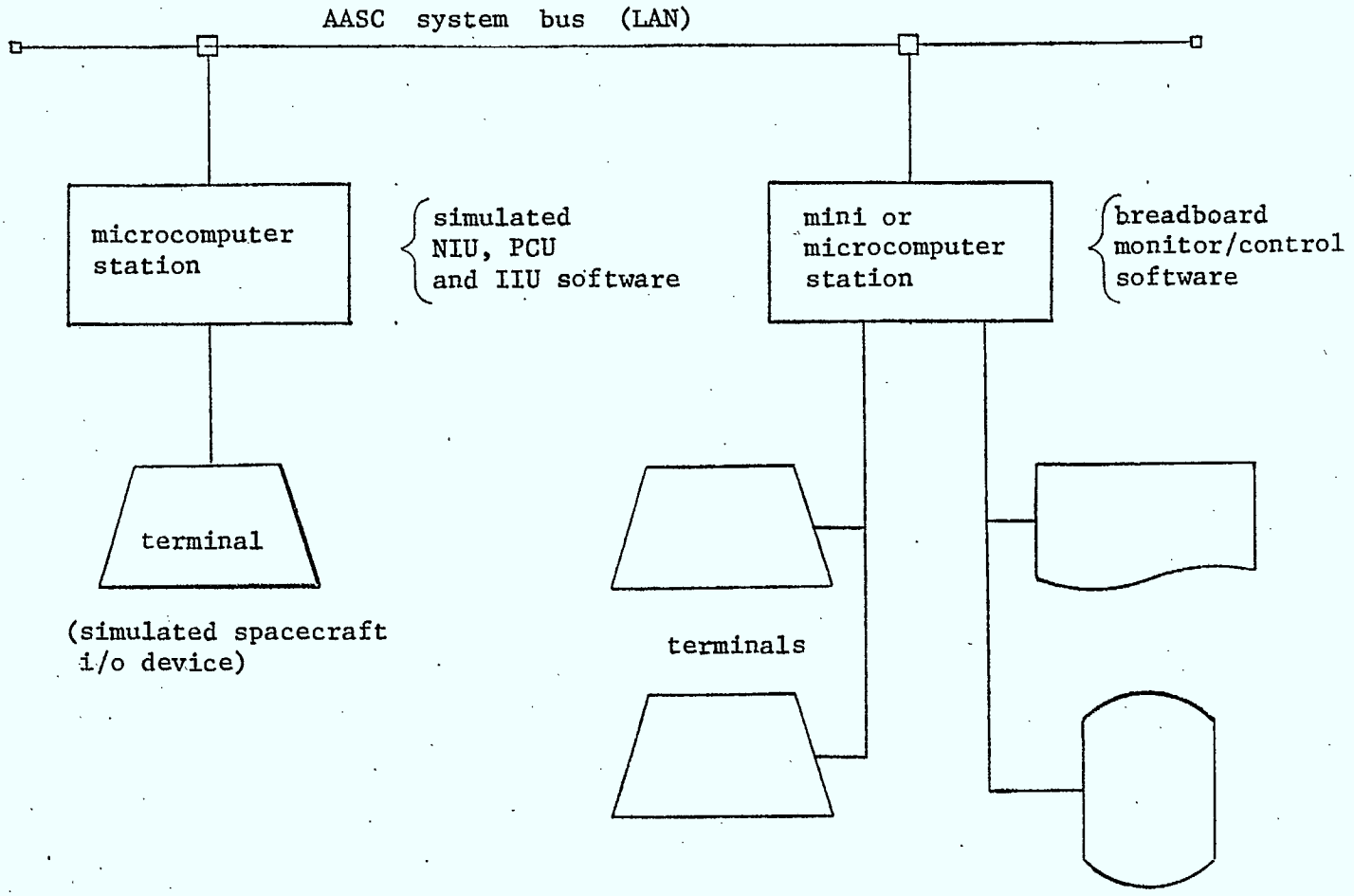


Figure 11.1 Basic Breadboard

Step 3 Refined Breadboard Development and Tests.

Two smaller hardware elements of the AASC should be developed, the NIU and IIU. They would be integrated into the basic breadboard system of Step 2, as shown in Figure 11.2, and the experiment repeated. The software should reflect the necessary changes discovered during Step 2. It should also include the new software elements which control the real NIU and IIU. A real or simulated spacecraft i/o system would be employed in testing the IIU.

Step 4 Full Breadboard Development.

The PCU hardware should be developed and added to the breadboard. PCUs of various sizes and combinations of components should be created. FRC'd processor pairs and married couples would be included. A breadboard system consisting of more than two stations should be configured, as shown in Figure 11.3. It would be loaded with the latest versions of the software resulting from the tests conducted in Step 3. New software to control the real PCU would also be necessary. A real or simulated spacecraft i/o would be connected to the IIUs, as appropriate. A monitor/control system should be set up for systematic execution of the experiment.

Step 5 Full Breadboard Experiment.

A detailed test plan should be generated for testing the full AASC breadboard developed in Stage 4. The plan should take into account the results of earlier tests conducted in Steps 2 and 3, but should be more comprehensive than the earlier tests. The plan should also include tests on the characteristics of the PCU, as identified during its development in Step 4. Tests should be conducted using the full breadboard system.

Step 6 Recommendations for Prototype Design.

Revision of the system, software and hardware requirements should take place, if necessitated by the results of the full experiment performed in Step 5. Recommendations would then be listed for developing a prototype AASC.

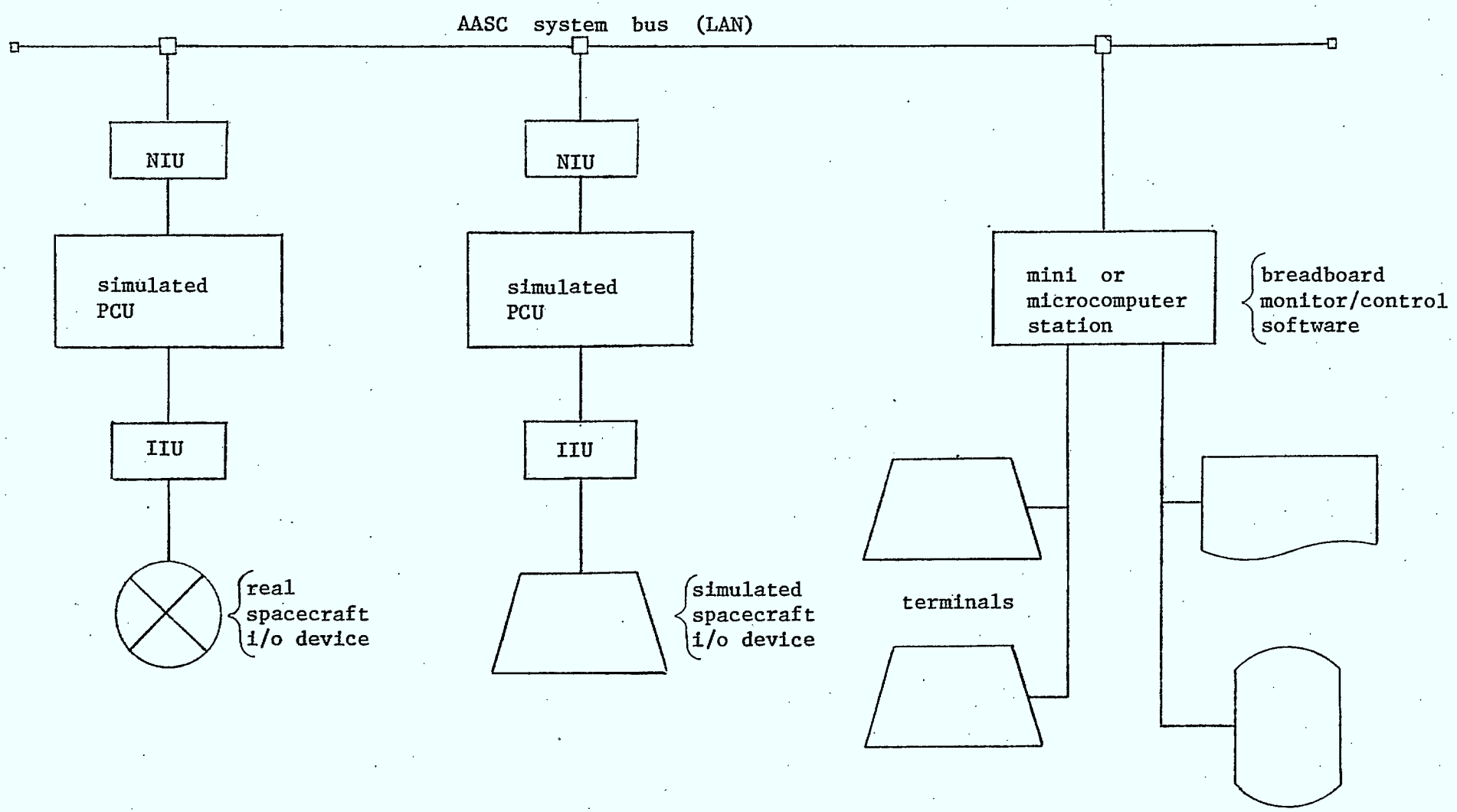


Figure 11.2 Revised Breadboard System

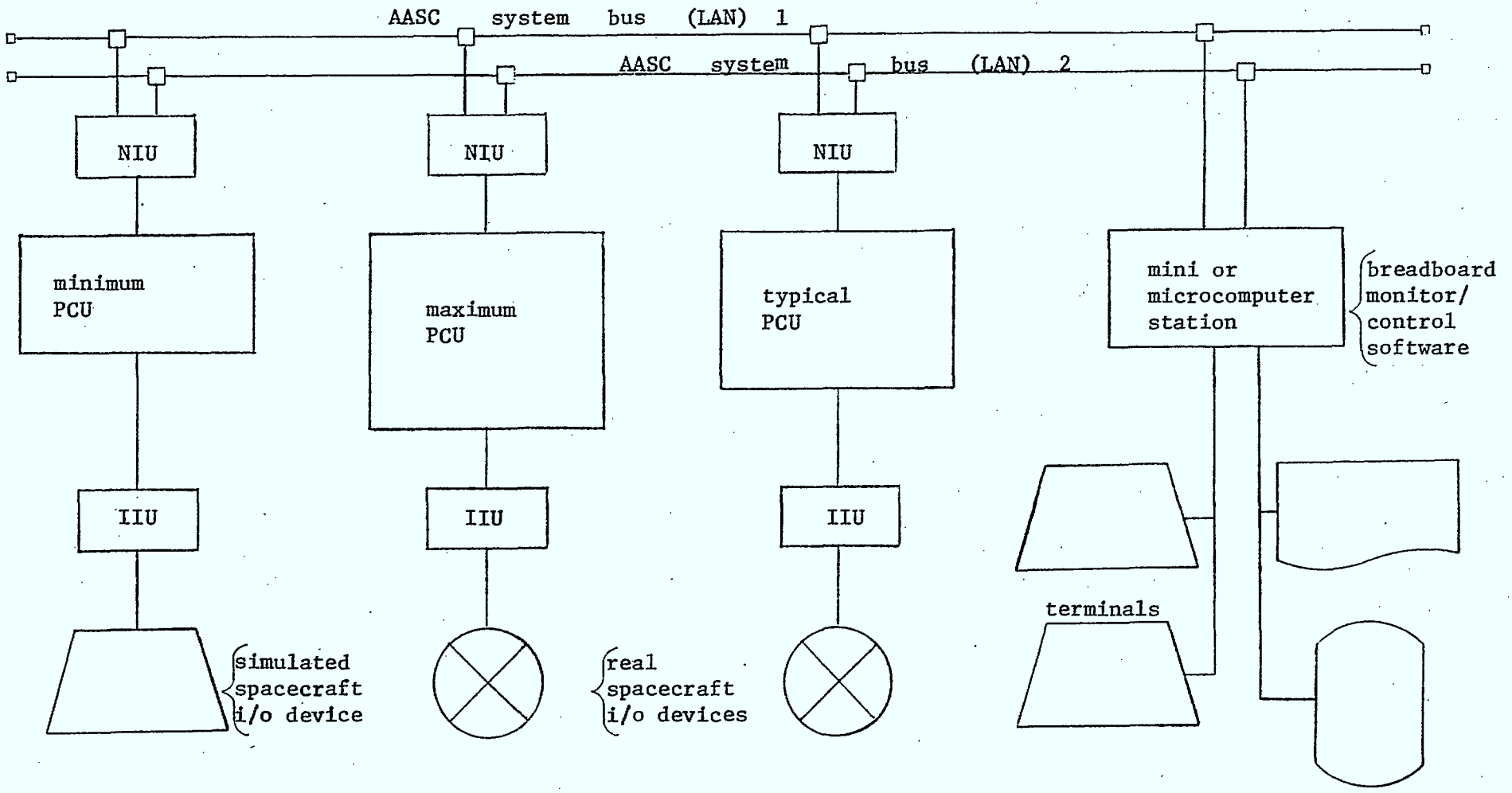


Figure 11.3 Full Breadboard System

REFERENCES

1. T. Gomi, M. Inwood, "FTBBC - The Fault-Tolerant Building Block Computer", Eidetic Systems Corp., 1981.

2. Michael H. Marshall, G.David Low, "Final Report of the Autonomous Spacecraft Maintenance Study Group", February 1, 1981. JPL Publication 80-88

3. George Gilley, "Fault Tolerant Design and Autonomous Spacecraft." Aerospace Corp., Los Angeles, California.

4. James S. Albus, "Brains, Behaviour and Robotics", McGraw Hill, New York, 1981.

5. E.W.Kent, "The Brains of Men and Machines", McGraw Hill, New York, 1981.

6. J.G. Kuhl and S.M. Reddy, Division of Information Engineering, Univ. of Iowa, "Fault-Diagnosis in Fully Distributed Systems", proceedings of The 11th Annual Symposium on Fault-Tolerant Computing.

7. J.P. Black and D.J.Taylor, Dept. of Computer Science and Computer Communications Network Group, Univ. of Waterloo, Ont. "A Compendium of Robust Data Structures", procs. of The 11th Annual Symposium on Fault-Tolerant Computing.

8. Pierre Azema, et al, Labratoire d'Automatique at d"Analyse des Systemes du C.N.R.S., Toulouse, France, "Virtual Ring Protection in Distributed Systems", procs. of The 11th Annual Symposium on Fault-Tolerant Computing.

9. W. Graham Wood, Computing Laboratory, Univ. of Newcastle upon Tyne, England, "A Decentralised Recovery Control Protocol", procs. of The 11th Annual Symposium on Fault-Tolerant Computing.

10. Fred B. Schneider & Richard D. Schlichting, "Towards Fault-Tolerant Process Control Software", Cornell University, Ithaca, N.Y.

11. Ewald Heer, "Automated Decision Making and Problem Solving." Vol.1 - Executive Summary. NASA Conference Publication 2180. Procs. of the 11th Annual Symposium of F.T. Computing.

12. iMAX432 Operating System, External Product Specification,

Revision 2.1, September 30, 1981.

13. Glenford Myers, "Software Development by Composite Design", 1975.

14. Glenford Myers, "Software Development", 1978.

15. Ethernet Specification, Version 1.0, Digital Equipment Corp., Intel Corp., Xerox Corp., Sept. 30, 1980.

16. iAPX432 FRC document, Intel 1981.

APPENDIX A

THE ASM-ENHANCED SYSTEM

Design Requirements

1. All Air Force spacecraft launched after March 1989 shall meet the ASM requirements listed below.

   On this date, the Department of Defense would require all subsequent spacecraft purchased to include the fully operational ASM capability.

   (Prior to this date, it is desirable to add incremental ASM capabilities, consistent with system performance, as they are developed.)

2. The ASM spacecraft shall operate without a ground support control link for up to 60 days without degradation of performance.

   This is the essence of autonomous operations. The spacecraft will function until ground support is available or desirable from the viewpoint of the ground support team.

3. The ASM spacecraft shall operate with not more than 10% degradation of key functions over a 6-month period of autonomy.

   This requirement will set some sizing constraints, such as data storage, and require some definition of loss of performance. It stresses the need for continuous function of the spacecraft on an "ad hoc" basis if scheduled ground support is not provided. The 10% figure is somewhat arbitrary; however, at the end of 6 months, the performance of the entire system shall be at a useful level.

4. The ASM spacecraft shall interact with the ground support segment for not more than 90 minutes to perform all required support functions without performance degradation.

   After a period of autonomy, it is required that the spacecraft and ground support perform all the required support functions in this window. The functions include (a) downlink of all stored maintenance history, (b) uplink of all data load (such as star tables and ephemeris), (c) redundancy management, and (d) testing. Specification of the du-

ration of the support window is mission dependent. The intent would be an uplink support period approximately the same as that required for non-ASM spacecraft.

5. ASM shall not change the design lifetime of the spacecraft.

   The imposition of the requirement for ASM on a spacecraft development is in addition to mission-imposed requirements, particularly the design lifetime. ASM will impact the design methodologies. Such design issues as depth of redundancy must take into account the rate at which resources are used up with the ASM design so that the total lifetime or mean mission duration shall not be reduced.

6. ASM shall not change the performance of the spacecraft or its payload.

   All requirements placed upon the spacecraft development for performance of either spacecraft or payloads shall not be affected by the presence of autonomous spacecraft maintenance. The spacecraft must be designed to provide these performance levels in the absence of frequent ground control interaction. Specific additional spacecraft functions, such as navigation, may be required to meet the autonomy requirement. If so, the performance of these functions (e.g., navigation accuracy) must support non-ASM system performance requirements.

7. The ASM spacecraft shall be able to recover from failures that have been defined a priori, and the probability that any particular failure was defined a priori shall be >= 0.98. The ASM functions include monitoring the spacecraft performance for faults and problem symptoms, and, in the presence of a fault, identifying, isolating, and implementing the recovery mode at both subsystem and system levels. The a priori analysis shall be sufficiently complete that, during the lifetime of the spacecraft, at least 98% of the failures (e.g., where some component has failed) will be identified in this manner (the coverage is >= 98%). Compound failures wherein multiple symptoms occur simultaneously or near simultaneously during the detection and recovery period can be exempted from this requirement.

8. Following launch, the ASM spacecraft shall go through a period of on-orbit checkout and initialization of the same duration as that of a comparable non-ASM spacecraft.

   The autonomy requirements discussed here are applied to the

operational period of the spacecraft, which is deemed to begin following the on-orbit checkout period. In the checkout period, maintenance will be under ground control, with autonomous capabilities turned on or off as appropriate. Since the addition of ASM does add certain functions, operation modes, and complexities to the spacecraft, these must also be checked out during the same period. Following checkout, all autonomy requirements will apply.

9. The spacecraft shall process and store all onboard management data required for ground support, and shall telemeter the data during the ground support periods upon ground command. The capability shall handle all necessary data for 6 months.

   No matter how confident designers may be of the maintenance capability of the spacecraft, it will be necessary to leave a record for ground support (an audit trail). Without this information, the ground support function cannot evaluate the state of the spacecraft and use the record of performance to extend the lifetime of the spacecraft, develop or implement alternative operating modes, or improve future designs.

10. The ASM spacecraft shall transmit a message to the ground at the first opportunity following any on-board fault-management activity.

    Whenever an incident occurs that requires maintenance activity in response to failure symptoms, it is important that the ground be given the opportunity to review the action and to verify the status and mode of the spacecraft. Thus, a telemetry message indicating that some activity had taken place would be sent to the ground at the first pass over an appropriate ground station. This type of signal may be coded into the user data to trigger an alarm at the ground support station. Sending of the message does not abolish the obligation of the spacecraft to retain the data for the maximum period, and to continue to operate in an autonomous manner for the established periods.

11. The ground support shall be able to override ASM management activities for the system and the subsystems.

    While the ASM spacecraft shall have the ability to perform redundancy management in the presence of an apparent fault or problem, it is necessary that the ultimate control over these functions be maintained at the ground, and that the

spacecraft shall allow for ground communication that overrides and can reverse the prior decisions of the ASM functions. The capability is necessary so that the system will be able to recover from such learning curve uncertainties as misdiagnosed problems or design flaws. In this way, nonfailed components may be recycled back into the configuration inventory, or the spacecraft alternate modes of functioning may be utilized to make use of partial capabilities of components. In terms of a hierarchical decision tree, the ground support personnel shall occupy the top level to maximize system performance.

12. The source of last resort for fault isolation and recovery shall be the ground support.

    The ASM spacecraft shall be designed to recognize when it has been unable to isolate, remove, and recover performance following a fault. When this occurs, the spacecraft shall take action to protect itself from self-injury or dissipation of resources (such as an engine firing limit cycle that would consume propellant), and await ground intervention.

APPENDIX B.

The Fault-Tolerant Computing Rules (FTC Rules)

Eidetic has compiled a tentative list of rules which a good fault-tolerant computer system should comply with. Such rules were proposed, from time to time, by several groups and individuals in the fault-tolerant and space computing communities, mainly on an empirical basis. Added to this existing set of findings are three further constraints (Rules [4], [5] & [6]) which are brought up anew by Eidetic, and which have been accepted among researchers and practitioners in the field of software sciences as system design principles considered essential in order to increase reliability of complicated software systems. Here, the distinction between the software rules and system or hardware rules is considered insignificant as trends towards acceptance of functional decomposition as the fundamental methodology of system design are increasing among planners and designers.

The following are the proposed fault-tolerant computing design rules (FTC design rules):

[1] There shall be no, or as few as possible, single points of failure in the system (the hardware rule).

(A chain is only as strong as its weakest link. If the system is dependent on a single item, that is a measure of its strength.)

[2] There shall be no fixed master-slave relationships among processing units (the democracy rule).

(The use of dedicated redundancy is an inefficient use of resources. In a fixed master/slave processor relationship a faulty master can propogate errors throughout the system before the damage is discovered.)

[3] There shall be no permanent fault arbiters or judges in the system (the modesty rule).

(If fault judging is software implemented then flexibility and distribution of the responsibility ensures that even the judges are judged.)

[4] Whenever a function is supported by processors, processes, tasks, subprograms, or other form of subfunctional mo-

dules, the method of inter-connecting them shall obey t module decoupling rules proposed by Glenford Myers (t module decoupling rule).

(In order to achieve the addition or removal of a module with the minimum of disruption, modular interfaces should be as simple and clean as possible.)

[5] Similarly, every subfunctional module must follow Myers' module strength rules (the module strength rule).

(The strength of a module lies in its raison d'etre. A functionally cohesive module will be easier to recognize and manipulate, and will not disintegrate in a dynamic environment.)

[6] As well as the horizontal breakdown, a function must be broken down vertically into layers. Levels of abstraction must be defined for each layer and independence between the layers must be observed (the layer rule).

(Since hierarchical thinking is natural to humans, complex structures arranged in orderly layers are more readily understood. Such a structured representation of concepts will be more accurate, revisions will be fewer and more readily implemented.)

GOMI, T.
--A fault-tolerant on-board computer ...