intellitech

The Intelligent Use of Technology

REVIEW OF MULTIPROCESSOR SYSTEMS AND THEIR SPACECRAFT APPLICATIONS

REVIEW OF MULTIPROCESSOR SYSTEMS AND THEIR SPACECRAFT APPLICATIONS

Report No. INT-82-14

March 1982

Authors: Mr. J.G. Ouimet
Dr. C. Laferriere
Dr. S.A. Mahmoud
Mr. T.F. Martin

Approved by: Dr. S.A. Mahmoud

Government of Canada / Gouvernement du Canada

Department of Communications

DOC CONTRACTOR REPORT DOC-CR-SP-82-044

DEPARTMENT OF COMMUNICATIONS - OTTAWA - CANADA

SPACE PROGRAM

TITLE: Review Of Multiprocessor Systems And Their Spacecraft Applications

AUTHOR(S): J. Ouimet
C. Laferriere
S.A. Mahmoud
T. Martin

ISSUED BY CONTRACTOR AS REPORT NO: INT-82-14

PREPARED BY: Intellitech Canada Ltd.
352 MacLaren St.
Ottawa, Ontario
K2P 0M6

DEPARTMENT OF SUPPLY AND SERVICES CONTRACT NO: 3ER.36100-1-0273
SN: OER81-03151

DOC SCIENTIFIC AUTHORITY: R.A. Millar

CLASSIFICATION: Unclassified

DATE: March 1982

Preface

This work was performed for the Department of Communications, Communication Research Centre, under DSS Contract No. OER81-03151, entitled "Computer-Aided Engineering Tools for Spacecraft Multi-Microprocessor Design", from September 15, 1981 to March 31, 1982. This report is one of the following four contract deliverables:

1. Executive Summary

2. Report #1 - Review of Multiprocessor Systems and their Spacecraft Applications.

3. Report #2 - A Survey of Computer-Aided Engineering (CAE) Tools for the Design and Simulation of Multiprocessor Systems.

4. Report #3 - The Definition and Specification of an Integrated Set of CAE Tools for Spacecraft Multiprocessor System Design.

Acknowledgement

The study team gratefully acknowledges the technical guidance of Mr. R.A. Millar of the Communications Research Centre. His knowledge and experience in the field of computer simulation of spacecraft systems have contributed to the quality of the work and provided a constant source of encouragement to the study team.

As well, the study team wishes to thank Mr. J.M. Savoie of C.R.C. for many fruitful discussions and critical reviews.

TABLE OF CONTENTS

1. INTRODUCTION

Interest in multiprocessor and distributed intelligence computer systems has increased dramatically in recent years. This interest has been fostered by the availability of microprocessors with even increasing performance/price ratios and the expected emergence of monolithic systems with still high capabilities in the near future.

A significant advantage of distributed systems is their potential capability for providing very high reliability through redundancies and dynamic reconfiguration. Since reliability is of prime concern in the design of spacecraft systems, distributed systems have received lot of attention from spacecraft computer systems designers. The aim of this report is to review the technology of distributed systems and to establish their applicability to the spacecraft environment.

Section 2 of the report presents different alternative architectures for multimicroprocessors. These achitectures differ in the degree and method of coupling processors and memories, their complexity and ability in isolating faulty components. The survey of the full spectrum of architectures was conducted irrespective of the number of processors involved in the architecture. Both architectures commonly used today, and those in theoretical or design development stages, are presented.

Section 3 surveys briefly the techniques of designing and implementing Executive Software for multimicroprocessors. The emphasis is placed on the general functions of the executive and the approaches followed for scheduling tasks and resources and for handling intertask communications and external interfaces.

Section 4 surveys five research activities reported in the literature which culminated in the development of five experimental prototype multimicroprocessors for avionics and spacecraft applications. These are: the Cm* system (Carnegie Mellon University), FTMP (Draper Laboratory), SIFT (Stanford Research Institute), UDS (Jet Propulsion Laboratory) and the french Matra system.

Section 5 is an introduction to the field of spacecraft computing technology. The aim of the section is to establish a baseline description of the environment with attempting to be an exhaustive discussion of the subject. The section first reviews the processing requirements of spacecrafts, outlines the processors used, then exemplifies the subject through the use of the example spacecrafts and finally presents some apparent trends in the development of these computers.

2. ARCHITECTURES FOR DISTRIBUTED SYSTEMS

2.1 Introduction

Significant advances in the development of powerful and reliable computer systems have been based largely on the use of multiple processors. A system which consists of several processors can be implemented in many variations. Normally these variations fall within one of the following broad classifications:

a. Multiprocessors: Defined as a computer employing two or more processing units under integrated control. The multiprocessor has the capability for the direct sharing of memory and I/O devices by all processors under control of a single operating system.

b. Multiple Computer-Network: These systems are composed of a number of heterogeneous computers loosely coupled, sometimes only by communications links. Each processor operates independently under control of its own operating system, sharing data with other systems as required.

c. Multiple Computer-Distributed: These systems fall somewhere between multiprocessors and networks. Jensen defines them as "... a multiplicity of processors that are physically and logically interconnected to form a single system in which overall executive control is exercised through the co-operation of decentralized system elements". [JENS78]

These definitions should not be viewed as the basis for absolute classification; it is possible to find examples of systems that do not fit exactly into a single definition and hence combine features from more than one definition. The real purpose of these definitions is to provide a framework for structuring the following discussions on architectures. Before proceeding further, it is necessary to point

out that a review of all the different architectures in the three categories would require an effort beyond the scope of this work. The review conducted here in concerned primarily with multiprocessor architectures that are used in spacecraft applications. Since many multimicroprocessors and most multiprocessor systems designed for spacecraft applications fit within the 'distributed system' group, we will limit our discussions to this category.

This section will first present a classification structure (taxonomy) used to define the environment of distributed systems. It will then examine some architectures that have been implemented. Because of its special nature, the issue of fault-tolerance in architectures will be examined more closely.

2.2 Classification of Distributed Systems

There have been a number of taxonomies proposed to classify distributed systems. An evaluation of these various classification schemes [JENS76] has concluded that the Andersen/Jensen taxonomy [ANDE75] is probably the most complete. It is based upon the decomposition of systems into three primary elements:

a) a Processing Element (PE) which is a hardware unit in which processes can execute;

b) a Path which is the medium on which messages are transferred without alteration; and

c) a Switch which intervenes between the sender and receiver of a message by modifying the message (eg. changing its destination

address) and/or by routing to one of a number of alternative paths.

To implement a distributed system, a designer must select various interconnection parameters. A tree structure can be used to represent these alternatives as shown in Figure 2.1. This tree is divided into four decisional levels. These levels are:

a) Link Level: A designer may select to use direct links between all PE's or may choose to use a switch, thereby implementing an Indirect link architecture. As discussed above, the switch will perform address translation (eg. from a logical address to a physical one) or perform a routing function.

b) Routing Level: For Indirect architectures, the designer may select to centralize the switching/routing function to one entity or decentralize to a number of entities.

c) Path Level: At this level, message transfer paths can be shared or dedicated. A shared path is one to which more than two PE's are connected.

d) Architecture Level: The final level of the tree are the nodes which represent specific architectures.

Figure 2.2 gives a number of examples of various architectures possible. The name of each example is preceeded by a three-character classification where the first letter indicates whether the link is direct (D) or indirect (I), whether routing is non-existent (.), centralized (C) or decentralized (D) and whether paths are dedicated (D) or shared (S).

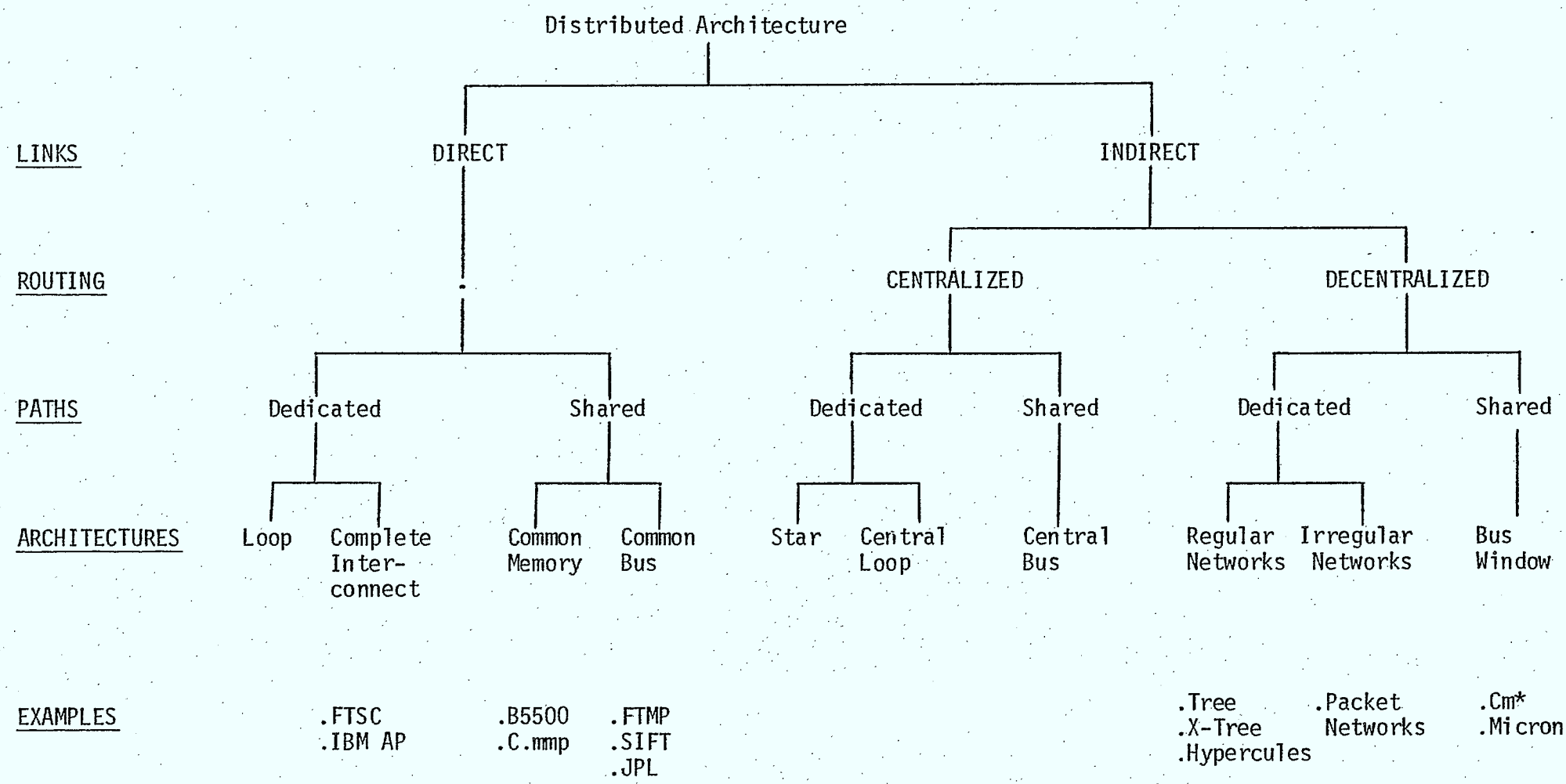
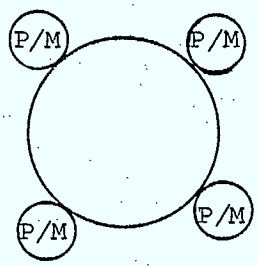
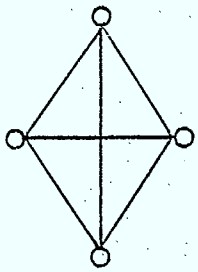


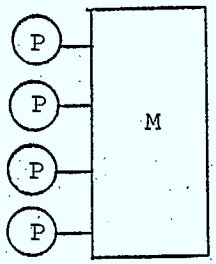
FIGURE 2.1 - DISTRIBUTED ARCHITECTURE TAXONOMY OF ANDERSEN/JENSEN



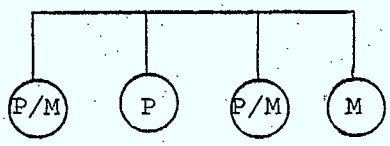
a) D.D - Loop



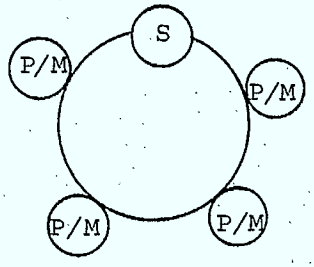
b) D.D - Complete Interconnect



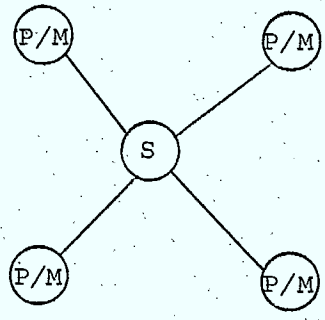
c) D.S - Common Memory



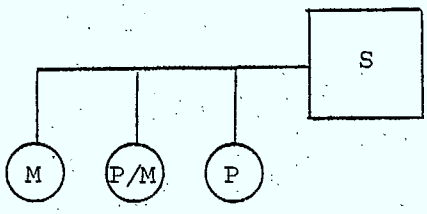
d) D.S - Global Bus



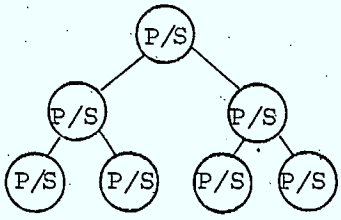
e) ICD - Loop



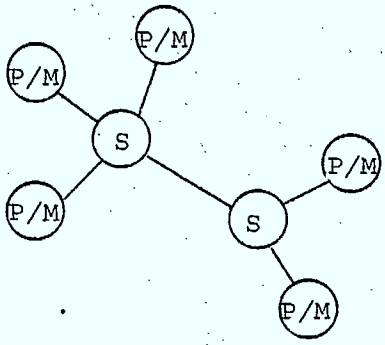
f) ICD - Star



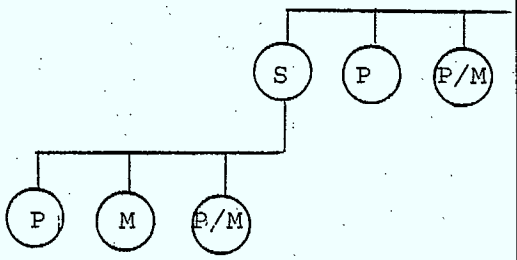
g) ICS - Centrally Controlled Bus



h) IDD - Regular Network



j) IDD - Irregular Network



k) IDS - Bus Windows

Legend: P - Processor
M - Memory
P/M - Processor & Memory
S - Switch
P/S - Processor with Switch Functions

FIGURE 2.2 - ARCHITECTURAL EXAMPLES

2.3 Implementations of Distributed Architectures

Most of the earlier implementations of multi-processor systems were done for large machines (IBM 360 and 370 families; Burroughs 5000 and 6000 families, CDC 6000, etc.) in order to improve their throughput. Most of these machines included a shared memory and used one of the following interconnection mechanisms between the processors, memory and I/O processors:

a) common bus (Figure 2.2d)

b) crossbar switch

c) multi-port memory (Figure 2.2c)

Enslow [ENSL77] surveys these architectures and the existing implementations (in 1977). He also analyzes the interconnection mechanisms and gives a list of advantages/disadvantages for each (see Table 2.1). Although these multi-processor systems do not meet the definition of Distributed Systems because of the extremely tight coupling between the components, much of the analytical work done for those systems still remains valid.

More recent work in distributed systems has been directed primarily towards three architectures: bus, circuit switches and indirect. The use of single and multiple buses has been extensive, particularly for fault-tolerant systems (see Sections 2.4 and 4). Some theoretical foundations for bus structures have also appeared [KINN78], [HAMA80]. A number of indirect networks (Indirect Decentralized Dedicated) have also been proposed; these include

binary-trees [HORO81], [HARR79]; X-trees [DESP79]; hypercubes [WITT81]; cluster structures [WU81], etc. Considerable research is also now taking place in the area of switching where efforts have been underway for some time to replace the crossbar switches with switches having all its advantages but none of its disadvantages. In particular, some of the switches which have been proposed could eventually be fully implemented in VLSI. Some of these types of switches are known as delta [DIAS81], omega, banyan [FRAN81], etc. [FENG79].

Common Bus	Crossbar Switch	Multi-port Memory
Lowest overall cost	Functional units remain single and cheap	Requires most expensive memory units
Least complex	Most complex interconnection	Complex interconnection
Easy to add or remove	Expansion simple to implement	Extremely difficult to expand since design is normally size-dependent
Limited bus throughput	Highest total transfer rate capability	Potential for high transfer rate
Failure at bus is a catastrophic system failure	Switch partitioning alters inherent redundancy and reconfigurability	
A single unit can degrade the performance of the whole system	Easy to remove malfunctioning units	
Lowest system efficiency of three types	Potential for highest system efficiency	

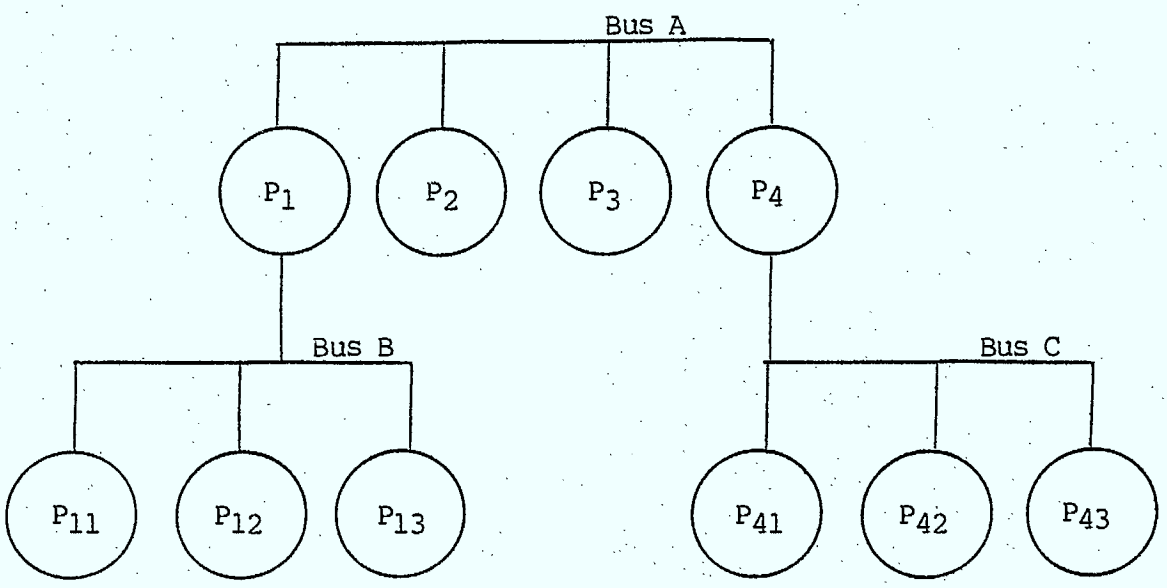
TABLE 2.1 - ADVANTAGES/DISADVANTAGES FOR THREE TYPES OF MULTI-PROCESSOR ORGANIZATION

2.4 Fault Tolerant Distributed Architectures

Some applications, real-time systems in particular, require a high degree of reliability. The computer system is expected to withstand individual component failures while continuing the processing function. These 'Fault Tolerant' (FT) computer systems have specially tailored architectures designed to provide this feature.

The Global (or common) Bus architecture has been widely used in the design of many FT systems. Some of the better known systems under development (FTMP, SIFT, UDS) use the bus concept. There is some speculation that the global bus will remain the basic architecture for aerospace computers [CARB77], though probably with the addition of multiple level of buses [CROS77]; Figure 2.3 is an example of a dual-level bus system.

As shown in Table 2.1, the common bus concept has a number of inherent disadvantages. In particular, fault tolerance is not an implicit quality of buses; in order to provide FT, designers must replicate the buses and design complex bus interface mechanisms to provide the many-CPU to many-buses connections required and to prevent any one CPU from disabling a bus. Additionally, bus architectures are very prone to bottlenecks. This has not been a problem so far since the amount of data passed between processors has always been relatively small. However, as processors are added and as data-intensive functions (eg. real-time image processing) come on board, bus architectures may reach limit points.



Notes: - If P_2 can address P_{13}, this is an IDS architecture.

- If P_2 does not know P_{11-13} exist but rather talks to P_1 as if it was the final processor, then this is the equivalent of three D.S architectures.

FIGURE 2.3 - DUAL-BUS LEVEL ARCHITECTURE

Because of these problems, many attempts have been made at implementing other types of architectures in FT systems. Architectures implemented are, with examples:

a) Bus(D.S) - FTMP, SIFT, UDS: (see Section 4)
- SARGOS: the computer system of the ground stations was designed for high reliability. SARGOS is a system to be used to identify and locate distress beacons by satellite. [DESW81]
- EPOS: the Experimental Polyprocessor System designed circa 1979 at Toshiba in Japan uses multiple independent common buses. [MAEK79]
- C.vmp:

b) Cluster Networks (IDS Bus window)

- Matra System, Cm*: (see Section 4)
- MuTEAM: a multimicroprocessor system now in development in Italy, designed for embedded real-time applications using loosely coupled clusters of bus connected microprocessors. [GIOF81]

c) Ring (D.DLoop)

- DDLCN: a fault-tolerant reconfigurable network using dual loops and tri-state control logic interface developed circa 1979 at the Ohio State University. [WOLF79]

d) Complete Interconnect (D.D)

e) Irregular Structure (IDD)

- RHEA: a reliable and survivable real-time system designed in France in 1976, it uses a two-level structure composed of an irregular network controlling groups of local star structures. [POWE78]

f) Circuit Switched (IDD Regular)

Other IDD Regular architectures such as Tree structures [LIU80], [KWAN81] and partially meshed Rings have also been studied. This last architecture (Figure 2.4), in particular, has been analyzed in detail by Pradhan [PRAD81a], [PRAD81b], [PRAD81c] and others [GRIN80] and found to present a very high degree of fault-tolerance and fault-diagnosability as well as having good distributed systems properties (low complexity, extensability, partitionability, etc.)

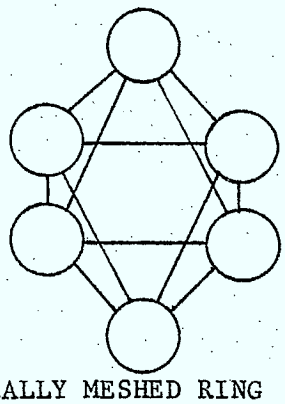


FIGURE 2.4 - A PARTIALLY MESHED RING

2.5 Conclusions

From this review of Distributed Architectures, a few concluding remarks can be drawn:

a. The 'common bus' is still the most widely used implementation architecture, especially for FT systems.

b. Multiple bus levels (bus window) and loop architectures are at the design and test stage.

c. Radically new architectures (meshed rings, tree, etc.) are now being theoretically studied and show promise for long-term implementations.

3. SURVEY OF DISTRIBUTED OPERATING SYSTEMS TECHNIQUES

3.1 Introduction

A distributed system is an amalgamation of individual computing entities with the aim of forming a combined single computer system. Activities take place in the individual entities concurrently and overall co-ordination is the responsibility of a decentralized executive. This definition of distributed computer system is that of [JENS78].

The decentralized executive is what binds together the collection of processing entities. This binding can be done in different ways, ranging from very tight coupling to very loose co-ordination. It should be emphasized that the executive itself is a control algorithm. As such, it can be implemented in hardware, in software, or as is more likely, in a mixture of both.

Hardwired executives (ie. executives whose algorithms are implemented in hardware) can be found in systems such as array processors or multiprocessor systems where the processors are tightly coupled to the executive through a common bus. At the opposite end of the spectrum, systems such as distributed databases can be found running on a loosely coupled network of computers. In a distributed database, the database management systems makes the distributed nature of the system totally transparent to the users. The database management system is a purely software entity and works in conjunction

with some communications facilities to provide the required co-ordination. So far, the executive has been characterized by:

1. An algorithm (or set of algorithms) implementing a policy (eg. loose or tight control scheme), and by

2. An implementation such as hardware, software or a combination of both.

The software part of the implementation of the executive is called the operating system (O/S). The purpose of this section is to highlight the software methods used in providing an executive for distributed systems. This will be done in the context of a general purpose distributed operating system, in order to cover most of the mechanisms involved.

From a user point of view, the purpose of an operating system is to manage efficiently the computer resources under its control. More specifically, the O/S is involved in the following:

1. Management of Computing Resources

Managing computing resources involves scheduling and synchronization of processes. A process in an instantiation of a program. The latter is a series of instructions residing in core or in a core image file on disk while the former is the action of executing the instructions. A process in characterized by its Process Control Block (PCB) which contains the program counter, the stack pointer, the program status word, some registers, etc. In a typical computing entity, many processes may be competing for access to the processor, that is, actively running as opposed to

waiting to run. The O/S will oversee the running of the processes and will ensure that each is given a fair share of the processor.

Processes which execute concurrently may want to co-operate among themselves. To this end, interprocess synchronization facilities should be provided by the O/S. It should also be realized that, because synchronization involves blocking and unblocking of processes, it shares the use of some conceptual mechanisms with the O/S scheduling facilities.

Also of importance is how the O/S will handle real time interrupts which are demands for service originating from devices. This area is relevant to control system such as on-board processors for spacecraft.

2. Management of Physical Resources

Management of physical resources is mostly concerned with the allocation (or granting) of hardware devices to processes. Several items are covered under the heading hardware devices; such items can be: more memory space for a process, access to fast floating point processor, control over a disk drive, or a printer, etc.

The result of the management activities of the O/S is a set of services. As explained before, those services constitute the sofware part of the executive of a distributed system. The remainder of this section will elaborate on each type of service with appropriate reference to relevant experimental systems.

3.2 Synchronization

In any system supporting concurrent execution (real or contrived) of processes, facilities have to exist to enable processes to co-operate among themselves in the execution of certain functions (eg. managing communication devices, storage devices, etc.). This is the purpose of synchronization. As a system service, synchronization strongly reflects the type of control structure embodied by the hardware. For example, synchronization through semaphores can be provided in a tightly coupled system with shared memory, whereas a message-based synchronization scheme is more suited to a loosely coupled environment.

Synchronization can be broadly classified into two types:

1. Implicit Synchronization

As the name implies, the O/S does not provide synchronization since it is either provided by the environment or by another O/S service. An example of synchronization being provided by the environment is a system [RENN78a] in which all activities are deemed to be synchronous. In fact, time provides synchronization. In some other sysems [KATS78] [ORNS75], synchronization is implicitly provided to a "strip" (a strip is a short, non-interruptible process), through scheduling of another strip.

2. Explicit Synchronization

In this case, synchronization is truly a system service implemented in software by various techniques. The performance of these techniques will depend upon factors such as: hardware configuration, granularity of the activities of processes using the synchronization service, frequency of utilization, etc. [JONE80].

When dealing with explicit synchronization techniques, it should be remembered that lower level techniques are always necessary at a processing entity; synchronization across processing entity boundaries may necessitate a different type of mechanism which will be implemented using the former techniques. Explicit synchronization techniques will then be listed by order of sophistication.

a) Locks

The implementation of locks relies on the availability of an indivisible operation (eg. test-and-set) at a given processing entity. Locks are typically used for co-resident processes having fine granularity of activities and those processes will usually implement a policy of "Busy Waiting", that is, will constantly check the value of the lock until they acquire it. Two observations can be made in connection with locks:

i) A "Busy Waiting" policy implies that a blocked process will keep the use of the processor. Other processes will not be permitted to run, even though no constructive work is done by the blocked process.

ii) While a process is blocked on a lock, the resources it is currently holding are not available to others. Releasing those resources, however, is not necessarily the best policy since the process would eventually have to re-acquire them at a later time.

b) Semaphores

Based upon locks, a more constructive approach can be taken in which the blocked processes are suspended and put in a special queue. A semaphore is the implementation of that policy and is represented by a mechanism such as a lock and an associated queue. Semaphores release the processing element from a blocked process and give other processes a chance to run. However, the policy involves context switching from one process to another. Obviously, if the cost of context switching is higher than that of blocking for a given process then blocking is preferable; in other situations the reverse may be true. Furthermore, the resources claimed by a suspended process are still being inaccesible to others.

c) Message Passing

Message passing is a synchronization technique in which the sending and receiving of messages (not necessarily their contents) define blocking and re-start for a process. Local to a given processing entity, this technique can be implemented by using semaphores which in turn are implemented partly with locks. The synchronization object is the mailbox which allows processes to wait (ie. suspend themselves) for the arrival of a message and to send a message to another mailbox (ie. signal another process). Clearly, synchronization through message passing is very well suited to a distributed environment since:

i) it goes across processor boundaries,

ii) it is not bound to any physical set-up, but

iii) it is understandably very much slower than semaphores.

The flexibility of message passing synchronization made it the choice as a basic synchronization technique for many experimental distributed operating systems such as [WULF74], [WULF75], [LEVI75], [COHE75], [JONE79], [OUST80], [SOL079], [WITT80]. It has to be mentioned, however, that in these systems, the O/S maintains a consistent interface for all the processes. In actual fact, two processes synchronizing each other through message passing may be using a single semaphore if they are resident in the same processor or may be exchanging messages (on a bus, through shared memory or through a parallel or serial interface) if the processes are at different processors. From the point of view of uniformity, offering one type of service and making its implementation transparent to the users seems a good policy, inasmuch as the O/S has the necessary flexibility.

A summary of synchronization techniques is shown in Figure 3.1, together with the systems using them. At this point, a brief description of some of those systems is in order.

1. The Unified Data System (UDS) [RENN78a], [RENN80] is not an operating system as such, but a multiprocessor organization in which all operations are synchronous. It is on that basis that it is included in this section. Further description is to be found in Section 4.

2. Pluribus (circa 1975) [KATS78], [ORNS75] is a computer system composed of three types of elements, called busses: processor-busses, memory-busses and I/O busses. Those busses can be con-

figured to form a multiple processor system. Some special features are important to the hardware structure:

a) all activities are on the form of short, non-interruptible processes called strips.

b) synchronization among strips is through the scheduling of other strip(s).

c) scheduling, for both real-time interrupts and software initiated requests, is through insertion of the identity of the desired strip(s) into a hardware queue called a Pseudo-Interrupt Device (PID).

Pluribus and UDS are examples of systems where a sizable portion of the distributed executive is implemented by hardware structures.

3. Hydra [WULF74], [WULF75], [LEVI75], [COHE75] is the operating system developed for the C.mmp [WULF72]. This multiple computer system was developed at Carnegie-Mellon University and is composed of a number of DEC PDP-11/40 with separate memory modules. Memories and processors are interconnected by means of a 16x16 crosspoint switch so that all of the available memory is accessible by any one of the processors. The C.mmp (circa 1970) is the earliest of such developments at Carnegie-Mellon and precedes the Cm* both in age and in concepts.

Hydra, the operating system of C.mmp was given the responsibility of managing the latter's resources. Consequently, Hydra provides facilities for managing the total memory space, for synchronizing

and scheduling processes, and for allocating hardware resources. Support of the concepts of capabilities and objects is provided by Hydra in an effort to increase security and data confinement.

4. StarOS [JONE79] and Medusa [OUST80] are operating systems designed for the Cm* (circa 1975) multiprocessor system. (Cm* is described in detail in Section 4.) StarOS was the first O/S developed for the Cm*. Medusa was developed later, based upon the experience gained with StarO/S. Both StarO/S and Medusa are close to Hydra in terms of general philosophy of operations. The services they provide are, in general, similar to Hydra's, except where hardware differences between C.mmp and Cm* either require extra facilities or make some services redundant.

5. Other multiple processor operating systems are in existence, such as Roscoe [SOL079] for the Rochester Intelligent Gateway and Micros [WITT80] for Micronet. The services they provide are very similar to those of StarOS or Medusa.

3.3 Scheduling and Resource Management

The problems of scheduling the execution of processes and of allocating the existing resources efficiently are both concerned with the performance of a multiprocessor system. On a more general level, the task of management takes on two aspects: static and dynamic.

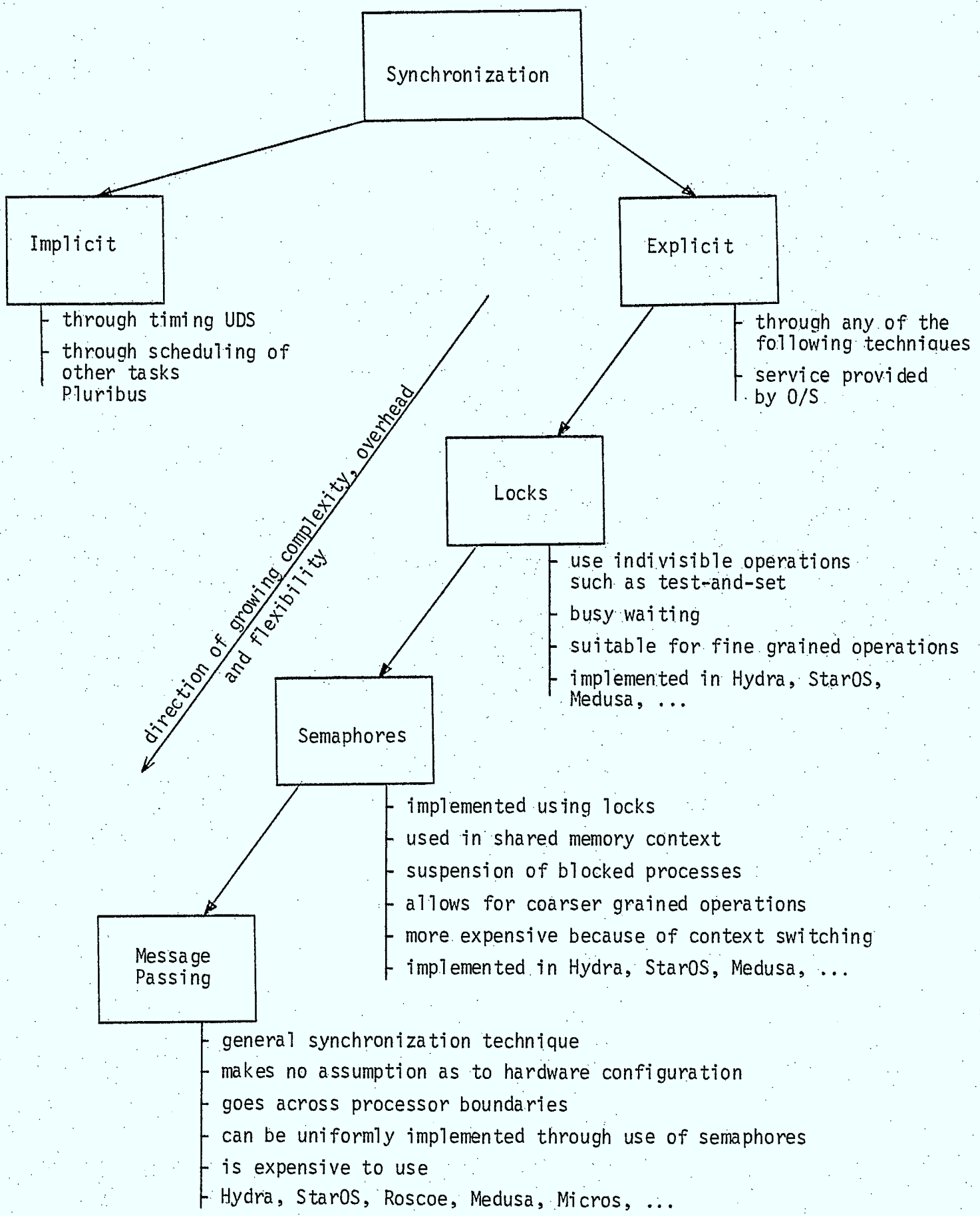


FIGURE 3.1 - OVERVIEW OF SYNCHRONIZATION TECHNIQUES

1. Static management is mostly concerned with how to distribute the code and data structures of O/S utilities and other processes among the various memory modules in the system. It can be seen readily that static management is extremely dependent upon the hardware configuration worked with. In highly specialized systems or in systems with only global memory (eg. [WULF72]), all processors (or group of processors dedicated to the same functions), have copies of all O/S utilities and other processes. In more general systems (eg. [FULL78]) memories are arranged in a mix of local and global partitions. The cost (speed and contention) of accessing local memory is much less than that of accessing global memory, indicating that the distribution policy and by extension the configuration itself, will play an important role in determining the performance (speed, throughput, response time) of the system.

   There also exist other systems (eg. [WITT80] in which all memories are local to some processors. In those systems, as opposed to special purpose systems, processes with non-resident code have to be executed elsewhere and are started through a remote procedure call (eg. using message passing constructs).

   Static management is therefore concerned with the allocation of O/S utilities and other processes into the various types of memories with the aims of: a) keeping as much free space as possible for each processor, and b) maximizing the speed of execution of O/S processes and others. These aims were, as mentioned previously, dependent upon the local/global memory arrangement and how much of each type was provided.

2. Dynamic management is concerned with the scheduling of processes and the management of resources.

a) Scheduling of processes can be accomplished implicitly in systems (eg. [RENN78a]) where time is used to schedule synchronous processes. This approach is conceptually simple but only applies to a narrow range of applications and is also inflexible. Explicit scheduling can be accomplished by queue(s) of ready to run processes (called ready to run queue). Scheduling can then be implemented as a single or multiple ready to run queue. One possible arrangement is to have independent ready to run queues at each processor with static priorities assigned to queues and dynamic priorities assigned to processes.

In tightly coupled systems, scheduling involves a context switch to the O/S and then a second context switch to the chosen process. In the case of a remote call in a loosely coupled system, messages will have to be exchanged and extra context switching at the recipient processor will have to be done. The resulting overhead indicates that scheduling should be tailored to the need of the application so as to minimize unnecessary context switching and to avoid inter-processor calls as much as possible.

b) Resource management is responsible for the dynamic allocation of resources (eg. devices, buffers, etc.) to processes. The overall performance of the system is very sensitive to bad decisions made by the resource allocation module. In most special purpose systems, however, the management policy is de-

termined statically at design time. This policy would likely accept excess capacity in terms of resources and computing power in order to minimize contention.

Another concern of the resource management module is that of deadlock. In special purpose very rigid systems, possibilities of deadlock can be totally eliminated whereas in more flexible ones distributed deadlock detection has to be used. Deadlock detection is an added overhead that can prove expensive in terms of actual processing time used, of memory space for the data structures and of lost capacity resulting from blocked or pre-empted processes.

3.4 Real-time Interrupts

The handling of real-time interrupts from devices requesting service is an important concern in an environment such as a spacecraft. Interrupts introduce a non-deterministic element in the execution of software. This makes the validation and debugging of the software an arduous task.

Interrupts have been masked out entirely in systems such as UDS [RENN78a] in which continuous polling of devices is used instead. This approach, albeit less responsive, makes it conceptually easy to assess the correctness of the software. Similarly in Pluribus, interrupts have limited effects on the system; all an interrupt service routine can do is to arrange for the scheduling of a "strip".

Other systems which are more general in nature accept demand interrupts but try to contain their effects and isolate their data structures by limiting the scope of the interrupt service routines.

3.5 Reliability

In the design of hardware components, physical distribution is the important criterion whereas in the design of the executive, logical distribution takes precedence [JONE80]. In the execution of a function, it is evident that if software processes are dependent upon one another to achieve success, failure of one of them will incapacitate the rest. Logical distribution requires that such dependency should not exist or at least should be kept to a minimum. Reliability can also be enhanced by having several instances of a particular process (at various processors) and by having them operate in a fashion analogous to back-up processors in hardware. It is also interesting to note that there is a trade-off between hardware and software reliability in the sense that software algorithms can be designed to enhance the overall system reliability.

3.6 Applicability to Special Purpose Processors

Special purpose processors, whether of the uni- or multiprocessor type, will have well defined functions to perform. In the case of multiple processor systems, the precise definition of their functions will allow for simpler designs and for some degree of optimization.

The software mechanisms presented hitherto addressed co-ordination problems of a general nature, hence their complexity. Special purpose systems will deal with a fixed population of processes whose needs will be known a priori. It should be pointed out that the basic requirements for synchronization and scheduling will still apply to these systems.

4. SURVEY OF CURRENT MULTIPROPROCESSOR PROJECTS FOR SPACECRAFT AND AIRCRAFT CONTROL

This section examines the design philosophy and general architectural concepts of five different multimicroprocessor systems that have been reported recently in various publications. Section 4.1 examines the general software and hardware features and the basic design methodology of these systems. A comparative study of their architecture is then presented in Section 4.2.

4.1 General Description

A survey of recent publications in the field of spacecraft and avionics multimicroprocessors revealed substantial research effort, concentrated mainly in building prototype systems and examining their use to realize certain properties. Such properties include software verifiability, fault tolerance, survivability under hostile conditions, self-checking capabilities, etc. Most prominent among these projects are the following four systems:

(1) Cm*

(2) FTMP

(3) SIFT

(4) UDS

(5) Matra System

A brief description of the above systems follows. The details of these systems can be found in the list of references at the end of this section.

(1) Cm* [SIEW78],[JONE77],[SWAN77]

The Cm* multimicroprocessor system was designed and implemented at Carnegie-Mellon University. The system is based on interconnecting a set of LSI11/23 processors in a configuration which permits extensibility, address mapping and software pruning of faulty components.

The Cm* structure consists of a set of computer modules grouped into clusters. The clusters are connected by a mapping controller which is a programmable high performance processor. Thus each controller is shared by several computer modules connected to it by a common bus via simple interfaces (referred to by S.local).

Figure 4.1.1 gives the details of each computer module. Figure 4.1.2 illustrates a simple three cluster network. The mapping controller is termed K.map, while each computer module is marked by M.

Cm* has a 2^{28} byte segmented virtual address space. The addressing structure provides considerable support for operating system primitives such as control switching and interprocess message transmission. Each processor in Cm* uses the top page in its address space (page 15) for direct program interaction with K.map. However, each of the 16 pages of the processor provides a

window into the system-wide 2^{28} byte virtual address space and can be independently bound to different segments in the virtual address space.

A significant feature of the Cm* system is that all communication is performed by packet switching, except at the local memory bus level, where conventional circuit switching is used. These buses are allocated only for the period required to transfer data. The data is latched at each interface, rather than establishing a continuous circuit from the source to the destination. This approach is taken in order to increase bus utilization and avoid bus allocation deadlocks.

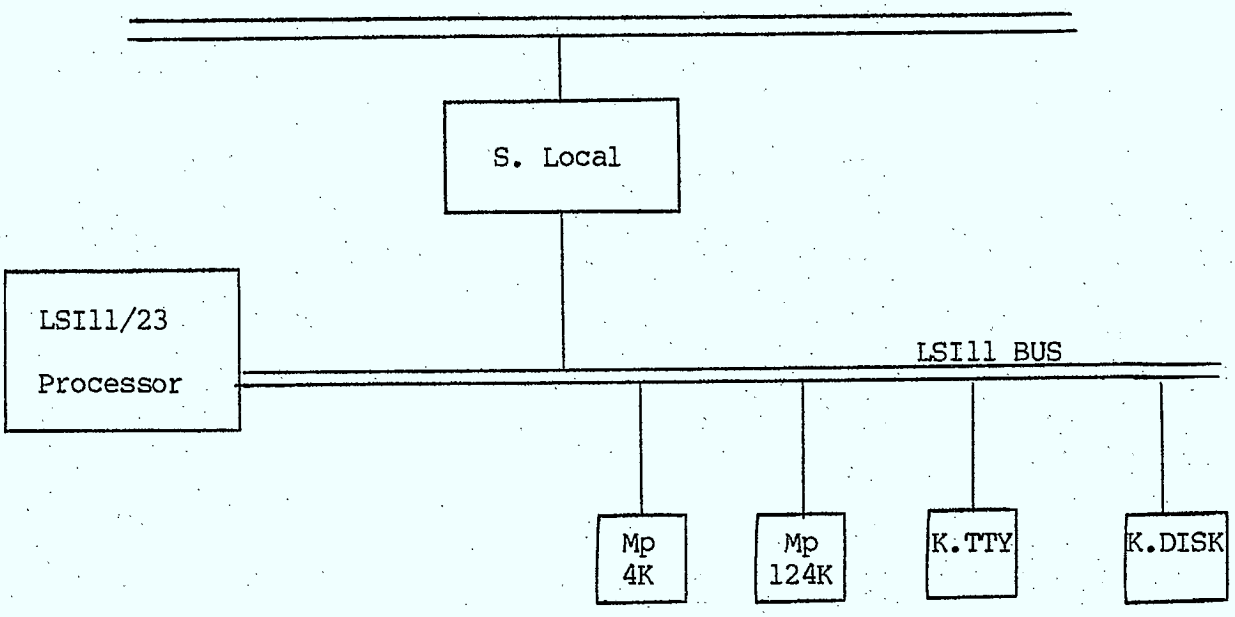


FIGURE 4.1.1 - DETAILS OF A COMPUTER MODULE IN Cm*

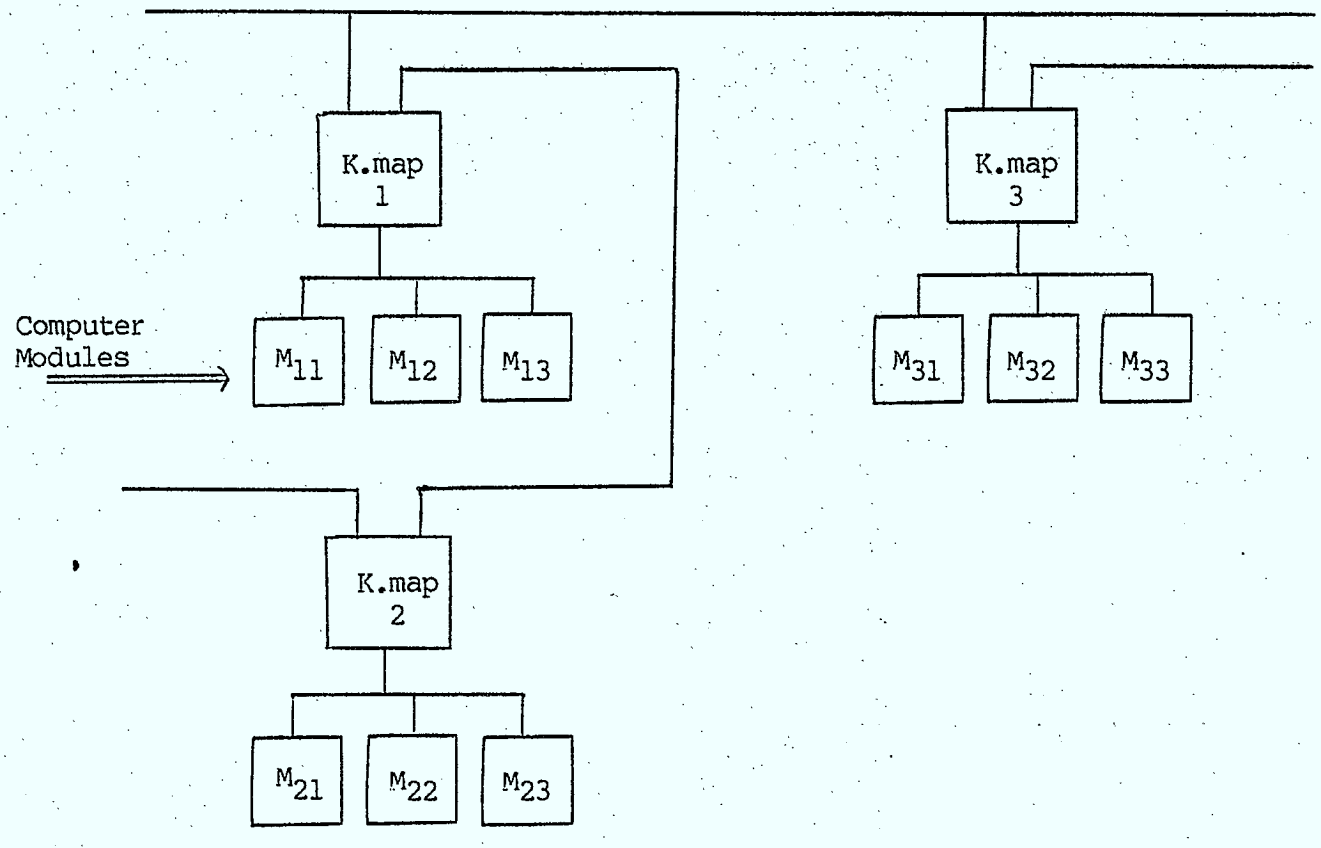


FIGURE 4.1.2 - A SIMPLE Cm* THREE-CLUSTER NETWORK

(2) FTMP [HOPK76b],[HOPK78]

The FTMP (Fault-Tolerant Multiprocessor) is a computer architecture that has been studied, simulated and emulated at the Draper Laboratory (Cambridge, MA). The target of the project is to achieve an overall failure rate less than 10^{-9} failures per hour, provided that maintenance is available within no more than ten hours per dispatch.

The FTMP structure is composed of an arbitrary number of processor modules with local memories and an arbitrary number of memory modules, interconnected by redundant serial buses. Modules are associated into groups of three to perform triply redundant functions. All data is distributed synchronously and in triplicate, and every module contains a voting element to mask bus disagreements. All modules contain special circuits to create logical and physical boundaries to halt the propagation of faults from one module to another. The essential features of the FTMP system are summarized in the following.

Redundant Organization

All activities in FTMP are conducted by triads of modules and triads of buses. A module triad is formed by associating any three like modules with one another. Thus any module can serve as a spare for any triad. A three member subset of N bus lines is chosen on a quasistatic basis to serve as a bus triad. These three lines are connected to a voter in each module, thus constituting a TMR element. The three active bus lines carry three independently generated versions of the data, each version

coming from a different member of the triad that is transmitting the data.

Functional Resource Allocation

The programmer of the FTMP system sees this multiprocessor as a machine for executing job steps. The procedure for each job step is written in a suitable language and resides in common memory. Each job step is typically scheduled to occur at a given time or following a given event. The relevant dispatch data for each scheduled job is kept in a queue, where it is frequently examined to see if the job step is eligible to be run or invoked. The frequent examination is conducted by processors that have completed their earlier assignments, and are available to undertake new ones. Thus job allocation is dynamic and adjustable according to memory load distribution and module failures.

Synchronization

FTMP employs tight synchronization using a common time reference that supports hardware voting, allows instantaneous validation of internal data, configuration control and in some cases interface data. The problem of maintaining a continuous timing reference is solved by a fault-tolerant redundant clocking arrangement using voltage-controlled crystal oscillators.

In addition to the above features, the FTMP system contains several fault detection and isolation mechanisms. The voting process which exists at the triad level is the primary tool for detecting any faulty behaviour during the operation of the system [HOPK75b], [HOPK78].

(3) SIFT [WENS78],[WENS72]

SIFT (Software Implemented Fault Tolerance) is an ultrareliable computer for critical aircraft control application. The prototype system was developed and tested at SRI. This includes error detection and correction, diagnosis, reconfiguration and the isolation of a faulty unit.

The structure of the SIFT hardware is shown in Figure 4.1.3. Computing is carried out by the main processors. Each processor's results are stored in the main memory associated with the processor. The I/O processors have much smaller computational and memory capacities than the main processors as they connect to the input and output units of the system (eg. actuators and sensors of the aircraft). Each processor and memory form a processing module. Each module is connected to a multiple bus system.

The SIFT system executes a set of tasks, each of which consists of a sequence of iterations. The input data to each iteration of a task is the output data produced by the previous iteration of some collection of tasks. The input and output of the entire system is accomplished by tasks executed in the I/O processors. Reliability is accomplished by having a number of processors executing redundantly the same iteration of a task; the output of this iteration is placed in the memory of the processor. A processor which uses the output of this iteration determines its value by examining the output of each processor executing it and taking a majority decision. Errors are discovered in cases of disagreement and are used by the executive to discover faulty

units and reconfigure the system. The processors in SIFT are loosely synchronized (eg. to within 50 microseconds) since we only need to ensure that the different processors allocated to a task are executing the same iteration.

Fault isolation in SIFT is accomplished by several mechanisms. Each unit is autonomous with its own control. This makes it possible to ignore improper control signals and to time-out signals that never arrive. Protection against corruption of data is provided by the way in which units can communicate. A processing module can read data from any processing module's memory, but it can write only into its own memory. Finally, each processor receives multiple copies of the data; each copy is obtained from a different memory over a different bus, and the processor uses majority voting to obtain a correct version of the data.

Formal models for scheduling strategies, software verification and reliability prediction were developed for SIFT in order to guarantee a correct operation within acceptable performance limits [WENS78],[WENS72].

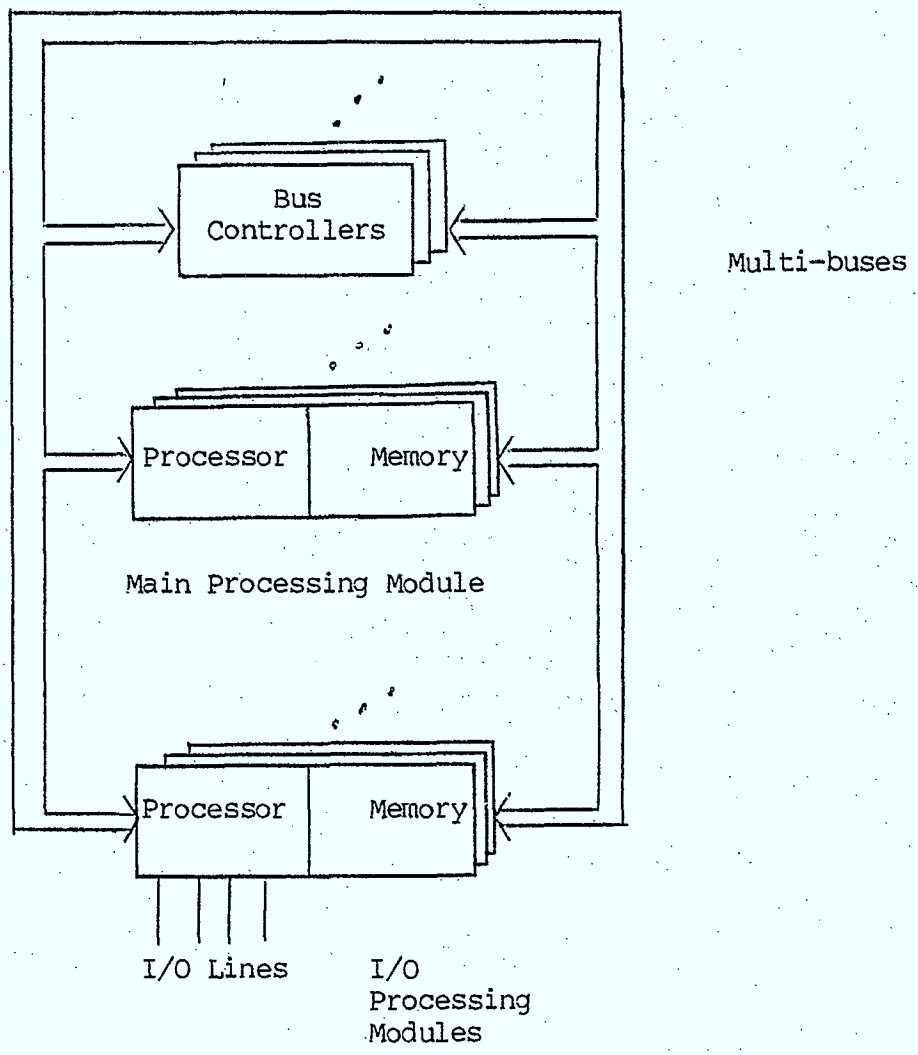


FIGURE 4.1.3 - STRUCTURE OF THE SIFT SYSTEM

(4) UDS [RENN78b],[RENN80]

The UDS architecture consists of a set of microcomputer modules connected by a redundant set of intercommunications buses. Two types of modules exist: terminal modules (TM) and high level modules (HLM). The TM interfaces with other modules in two ways: 1) it receives a single Real-Time Interrupt (RTI) which is common to all modules and which is used for timing and synchronization, and 2) each TM is interfaced to all intercommunications buses. DMA techniques are used to enter and extract data from the TM. A TM cannot initiate bus communications. An external HLM enters commands, data and timing information into the memory of the TM.

HLM's are responsible for coordinating the processing which is carried out in the remote TM's, for control of intercommunications over the bus systems, and for high-level processing such as data compression and decision making.

The UDS design is oriented towards moving "hard core" items whose failure can cause catastrophic system failure. For example, the buses are made independent to avoid any common failure mechanisms. Independent clocks are used within each module; the clocks are synchronized by the common RTI. As well, the UDS architecture prevents the TM's from propagating their errors to the HLM's since the TM's are not allowed to issue communication calls to the common bus system. Details of the UDS architecture and the software executive are available in [8] and [9].

The design of the UDS emphasizes testability and simplicity, even at the expense of the computer power and memory utilization.

These aspects are manifested in the following architectural features:

Computer Utilization: Low speed and memory utilization enables the designer to concentrate on the correctness of his code rather than its efficiency.

Error Confinement: Most TM's will be performing dedicated low level functions and are not permitted to modify the memory of other modules.

Minimization of Interrupts: Demand interrupts are avoided whenever possible. This may limit I/O response to milliseconds rather than microseconds. However, it leads to more predictable operation, is more easily modified, and allows for software self-defence.

Control Hierarchy: As explained previously, the HLM's control the TM's. The programs in the TM's are self-synchronized to process the data when they arrive and place results in their memories for subsequent extraction by the high-level computer.

Other simplification features in the UDS architecture include synchronous communications, timing hierarchy and I/O timing granularity to simplify software modifications.

(5) Matra System [SOUB77]

The Matra organization, in France, has developped a multi-microprocessor system designed to automate the signal and data processing functions at Attitude and Orbit Control Subsystems

(AOCS). The system consists of three microprocessors (INTERSIL 6100) using a common bus, some local memory for the processor supervisor programs and global memory for the common data areas and for the AOCS programs (Figure 4.1.4).

Each processor is controlled by its supervisor program. When idle, a processor will read the common supervisor table to determine which pending task has the highest priority. The processor will then that task. The task may generate the activator of other tasks, through the common supervisor table. External interrupts and Real Time Clock Interval pulses can also activate tasks. The Real Time Clocks are used for local processor monitoring and overall process synchronization.

Fault detection is implemented using:

a. local watch-dog with each processor (software)

b. processing function watch-dogs (sofware)

c. monitoring and reconfiguration capability of local Interval times

d. self-test programs.

The system is capable of surviving failures through:

a. halting and disengaging a failed processor

b. reallocating memory module tables in case of a failure in memory blocks 2 to 7

c. switching the redundant memory module implemented for block 1 (common data).

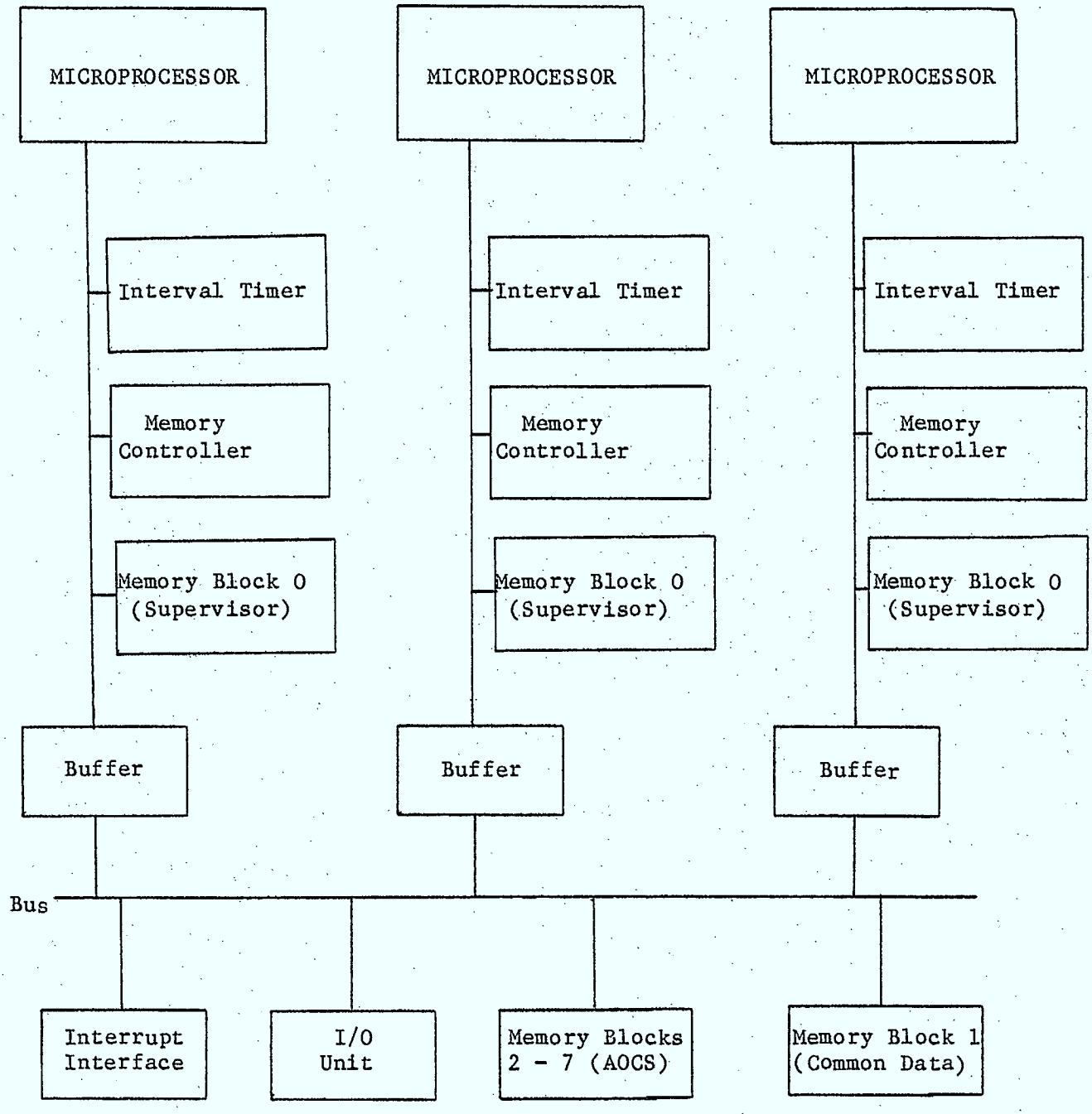


FIGURE 4.1.4 - MATRA MULTI-PROCESSOR BLOCK DIAGRAM

4.2 Architectural Features of Three Systems

Out of the systems discussed in the previous section, three were designed for a high reliability environment: FTMP, SIFT and the JPL system. Although similar in concept, these systems represent a wide ranging set of implementations. A study of these features will therefore allow a better understanding of the design of upcoming spacecraft systems.

Table 4.2.1 is a condensed comparison of these three systems:

a) Memory: The JPL and SIFT both have memory which is local to each processor while the FTMP uses mostly global memory (it has small local caches for improved performance). The way the memory is structured has two main influences: the mechanisms for passing information between tasks will be very different and the use of the buses by the FTMP will be much larger, leading to potentially earlier saturation.

b) Number of Bus Types: Both the JPL and SIFT use only one type of bus while the FTMP uses two buses in an attempt to minimize the potential bus loading problem.

c) Bus Replication: Buses in the FTMP and SIFT are quintuply replicated while the JPL triplicates its bus.

d) Bus Controllers: The SIFT uses dedicated devices as Bus Controllers while the JPL lets a task in one of the general purpose

processors carry out the bus arbitration functions. The FTMP appears to use a decentralized bus control mechanism which resides in each device.

e) Module Types: The FTMP uses three different sorts of module devices (memory, processor/cache and I/O access). The SIFT also uses three types of modules though they are somewhat different (bus control, processor/memory and I/O processor). The JPL attempts to limit its module types to one by designing an all-purpose building-block; although this is an interesting concept which simplifies implementation, it is also potentially restrictive.

f) Fault Detection: In FTMP, fault detection is done by having hardware devices compare the output of three (fixed number) processors which carry out the same computations simultaneously; if not all processors agree, voting is used to determine which result is correct and which processor is faulty; this comparison/voting is performed for each instruction. In SIFT, a similar comparison/voting mechanism is used, however, it is implemented in software; it allows a variable number of processors to participate and it is carried out at fixed regular intervals, rather than for every instruction. In the JPL system, a hardware implemented verification mechanism is imbedded within each bus and is therefore transparent to other modules.

g) Processor Synchronization: Although in all three systems cooperating tasks must be closely synchronized with respect to each other, the actual inter-processor synchronization is implemented differently in all three processors. In FTMP, since hardware

voting for each instruction is carried out, the processors within triads must be very tightly synchronized. In SIFT, synchronization is only required at the voting interval level while in JPL, since no voting takes place, processor synchronization is not required (of course, task synchronization still is).

System Feature	FTMP	SIFT	JPL
a) Memory	Global; processors have a local cache memory	Local	Local
b) Bus Types Types	Memory buses Interface buses	Global Bus	Global Bus
c) Bus Replication	5	5	3
d) Bus Controller	No	Yes	Imbedded
e) Module Types	Memory Processor/cache I/O Access	Bus Control Processor/Mem. I/O Proc.	Building Block
f) Fault Detection	By Voting, in Hardware, in triads	By Voting, in Software, variable no. of processors	Internal to each Block
g) Processor Synchronization	Tight	Loose, by interval	Not required

TABLE 4.2.1 - ARCHITECTURAL COMPARISON OF THREE SYSTEMS

5. THE SPACECRAFT APPLICATION ENVIRONMENT

5.1 Introduction

This section will introduce the computing environment of spacecraft applications. It is not intended as a comprehensive review of spacecraft systems but rather as a short introduction to the concepts, techniques and elements of spacecraft computers. The next four subsections will:

a. present the processing requirements in terms of functions and of resources needed,

b. describe some of the computers that have been or will be used in spacecrafts,

c. illustrate the use of these processors through two example systems: Voyageur and Galileo,

d. conclude by discussing the present trends of spacecraft processing including the NASA NEEDS program.

5.2 Processing Requirements

Typically, the on-board computing functions include [HOLC80]:

a. pyro/propulsion control,

b. mechanical actuator control,

c. radio control,

d. data handling,

e. command decoding,

f. instrument control,

g. attitude control,

h. data compression.

Some functions, such as actuator or instrument control, involve issuance of predetermined command sequences, while functions such as attitude control involve complex calculations based upon readings from instruments (eg. star tractors and gyros). The developers of the Fault Tolerant Spacecraft Computer determined that attitude control and navigation were the primary functions requiring automation and that it was their processing requiremens that most affected the FTSC design; Table 5.1 summarizes the FTSC designers' conclusions regarding the on-orbit computation requirements [FANE77]. The support of the science instruments is another main area of computer utilization; the computers must be capable of reading all the data provided by the instruments and perform the necessary calculations to prepare the data for transmission. Table 5.2 gives the maximum data rate and the amount of memory space required for each of the instruments of the Galileo orbiter spacecraft.

Task	Program and data memory, words	Computation rate, operations/sec
Attitude control and pointing	2,343	68,250
Telemetry	1,281	13,692
Commanding	912	19,500
Supervisor	2,909	33,042
Subsystem management and recovery	876	1,875
Subtotal	8,321	136,359
Payload processing	6,500	60,000
Subtotal	6,500	60,000
Memory size without payload processing	10K	...
Memory size with payload processing	16K	...
Maximum computation rate	...	200,000

TABLE 5.1 - ESTIMATED ON-BOARD PROCESSING REQUIREMENTS
(Reprinted from [FANE77])

Instrument	Maximum Data Rate, bits/s	Memory Size, kbytes ROM	RAM
Solid state imaging (SSI)	768.	3	3.5
Near infrared mapping spectrometer (NIMS)	11.52	3	1.75
Photopolarimeter radiometer (PPR)	0.18	4	0.25
Ultraviolet spectrometer (UVS)	1.0	-	0.75
Energetic particle detector (EPD)	0.92	6	3
Plasma subsystem (PLS)	0.60	8	8
Magnetometer (MAG)	0.24	4	4
Dust detector subsystem (DDS)	0.024	3	2
Plasma wave subsystems (PWS) (contains no microprocessor)	645.	-	0.25

TABLE 5.2 - GALILEO SCIENCE INSTRUMENT SUMMARY

5.3 Spacecraft Computers

The first general purpose programmable computers to be flown onboard spacecraft were centralized machines monitoring and controlling the various subsystems. More recent spacecraft designs have taken the approach of using more than one computer and of having computers dedicated to certain functions such as attitude control, while under the overall command of a central command and control computer.

The two extremes in spacecraft computer architecture are the completely centralized and the fully distributed. The completely centralized approach has one large central computer serving the needs of all the subsystems, while the fully distributed approach consists of several small computers serving these needs as peers. Neither of these approaches is optimal for the space environment. The highly centralized system calls for significant data communication overheads and complex resource allocations within the computer while a fully distributed system can result in system coordination problems [HOPK75a].

The approach being taken in current spacecraft computer system architectures is very much a combination of these two extremes which permits capitalizing on most of the advantages of both. The Unified Data System (UDS) which is the model for the command computer system aboard the most recent satellite, Galileo, is representative of that approach (see Section 4). Table 5.3 lists the characteristics of some current on-board computer system [STAK81] while Table 5.4 details the

characteristics of the processors available for space missions in the 1980s [HOLC80].

Computer	Mission	Number	Memory Size (in bits)	Memory Type	Type of Processor	Cycle Time
1. NSSC-1	MMS (Generic)	1 or 2	8-64K by 18	CORE	TTL	1.5
2. AOP	Landsat-B/C	1	4K by 18	PWM	TTL	--
3. NSSC-1	SMM	2	48K by 18	PWM	TTL	1.4
4. AOP	IUE	2	12K by 18	PWM	TTL	1.3
5. OBP	OAO-C	1	16K by 18	CORE	DTL	2.0
6. DOC	ATS-6	2	4K by 18	PWM	LPTTL	5.0
7. GCSC	Viking Lander	2	18K by 18	PWM	LPTTL	5.0
8. CCS	Viking Orbiter	2	8K by 18	PWM	--	--
9. FDS	MJS-77 (Voyageur)	2	8K by 18	CMOS	DMOS	2.48
10. CCS	MJS-77 (Voyageur)	2	8K by 18	PWM	--	1.37
11. AACS	MJS-77 (Voyageur)	2	8K by 18	PWM	--	1.37
12. SCP-234	TIROS-N	2	18K by 18	CMOS	CMOS	2.34
13. SCP-234 (USAF)	Block 5D	2	16K by 18	CMOS	CMOS	--
14. COSMAC	AMSAT Phase IIIB	1	16K by 18	CMOS	CMOS	1
15. NSSC-1*	Landsat-D	2	64K by 18	CORE	TTL	1.5
16. AAC-16ms*	Galileo	2	32K by 18	CMOS	LSI bit slice	0.25
17. CDC 469	HEAO	2	16K by 18	Plated Wire	PMOS/LSI	

Abbreviations

CMOS - Complementary Metal-Oxide Semiconductor
DTL - Diode-Transisor Logic
LPTTL - Low-Power Transisor-Transistor Logic
PWM - Plated-Wire Memory
TTL - Transistor-Transistor Logic
AACS - Attitude and Articulation Control Subsystem
AOP - Advanced On-board Processor = NSSC-1
CCS - Computer Command Subsystem
DOC - Digital Operations Controller
FDS - Flight Data Subsystem
OBP - On-board Processor
NSSC-1 - NASA Standard Spacecraft Computer
MMS - Multimission Modular Spacecraft
SMM - Solar Maximum Mission
IUE - International Ultraviolet Explorer
OAO - Orbiting Astronomical Observatory
ATS-6 - Applications Technology Satellite

TABLE 5.3 - CHARACTERISTICS OF SOME CURRENT ON-BOARD COMPUTER SYSTEMS (from [STAK81])

Identification	IBM NSSC-I	Martin M. Enhanced NSSC-I	IBM NSSC-II	Litton 4516E	App. Tech. ATAC-16MS
Configuration	2 CPU's 2 Stint's	2 CPU's 2 Stint's 2 APU's	2 Sing. String CPU's	2 Sing. String CPU's	2 Sing. String
Cost ($K) - single string	205	355	NA	392	115
Weight (lbs)	17.4	NA	29	12.5	18
Power (W)	31	28	240	25	34
Word Length (bits)	18	18/32	16/32	16/32	16/32
Maximum Memory Size (words)	64K	96K	40K	64K	64K
Memory Technology	Core	Core	NMOS	Semi-Cond.	CMOS
CPU Technology	TTL-LSI	TTL-LSI CMOS/SOS	Hyb. TTL	Shot. BIP TTL	Schot. TTL
Ave. Inst. Speed (Sec) Add	5	5	3.2	2.5	.3
Mult	38	33	35	21	5.4
Floating Pt.	No	Yes	Yes	Yes	Yes
Reliability (2 Yr)	.96	.96	.87	.99	.98
Qual. Status	Space	None	None	Space (1980)	Space
Space Flt. Exp.	SMM (1980) Space Telescope (1982)	None	None	None	Galileo (1984)

TABLE 5.4 - CHARACTERISTICS OF PROCESSORS AVAILABLE FOR SPACE MISSIONS IN THE 1980s [HOLC80]

Identification	NAR DF-224	Teledyne MECA-43	Raython FTSC	CDC 469	IBM Shuttle GPC
Configuration	3 CPU's	2 CPU's	4 CPU's	2 Sing. String CPU's	1 CPU 1 IOP
Cost ($K) - single string	2250	NA	1000	650	500
Weight (lbs)	102	15	50	10	59
Power (W)	85 (56)	44	35	20	350
Word Length (bits)	24	16	32	16/32	36
Maximum Memory Size (words)	64K	64K	96K	16K (32K)	16K 106K (OFT)
Memory Technology	Plat.Wire (CMOS)	CMOS	CMOS/SOS	Plat.Wire (CMOS)	Core
CPU Technology	PMOS	CMOS/SOS	CMOS/SOS	PMOS/LSI	TTL MSI/LSI
Ave. Inst. Speed (Sec) Add	1.6	5.4	5.4	4.0	1.9
Mult	8	11	11	10.4	5.7
Floating Pt.	No	NA	NA	No	Yes
Reliability (2 Yr)	.97	.98	.98	NA	NA
Qual. Status	Space	None	None	Space	Space
Space Flt. Exp.	Space Telescope (1982)	None	None	HEAO	Shuttle (1980)

TABLE 5.4 (Cont'd) - CHARACTERISTICS OF PROCESSORS AVAILABLE FOR SPACE MISSIONS IN THE 1980s [HOLC80]

5.4 Example Spacecraft Systems

5.4.1 Although there have been many spacecraft launched, two systems are remarkable from a computer utilization point of view. They are the Voyageur Spacecraft and the Galileo Orbiter Spacecraft, quickly described here.

5.4.2 The Voyager Spacecraft had three separate computers onboard - the Command Computer System (CCS), the Flight Data System (FDS) and the Attitude and Articulation Control System (AACS). [GILL80],[SCUL80]

The CCS (see Figure 5.1) is responsible for the decoding and distribution of commands and for the coordination of spacecraft functions. It has the following features:

- 4K memory
- 18 bit word length
- priority interrupt structure
- add/subtract cycle time of 90 microseconds. It has an average utilization of approximately 5%.

The FDS (see Figure 5.3) is the computer used for science instrument control and data processing. It is characterized by:

- 8K memory
- 16 bit word length
- 4 DMA channels
- an add/subtract cycle time of 15 microseconds

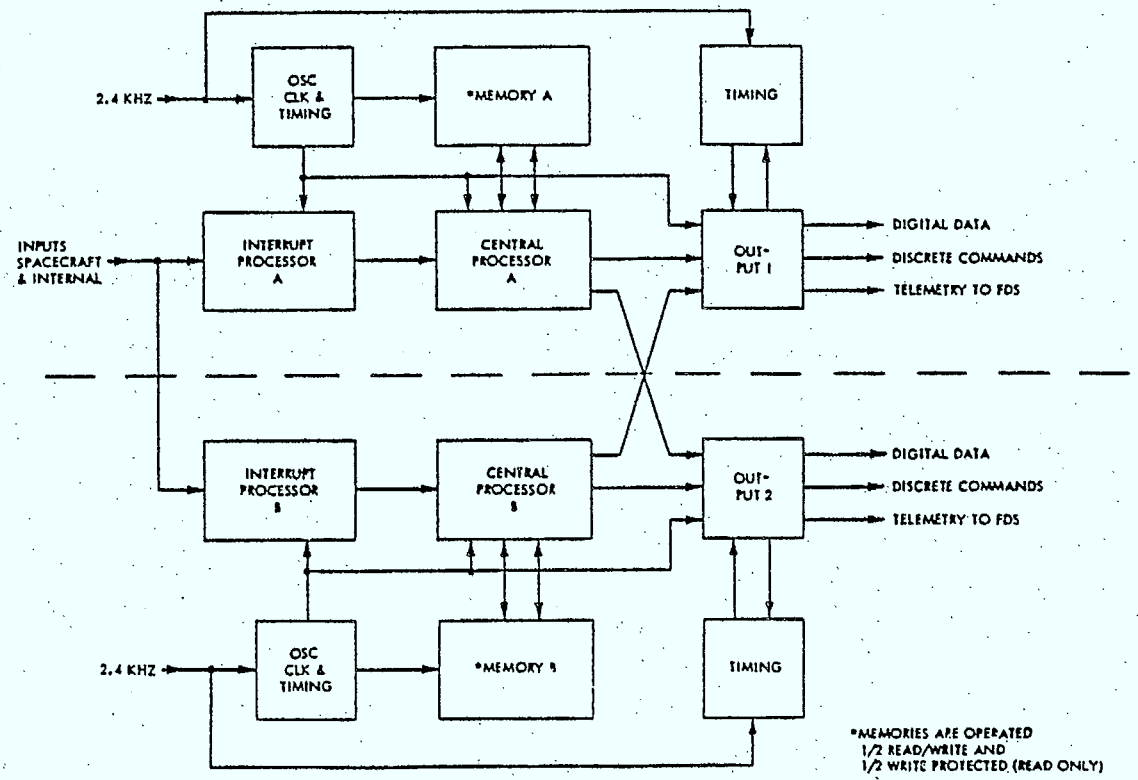


FIGURE 5.1 - VOYAGER COMPUTER COMMAND SYSTEM
(Reprinted from [SCUL80])

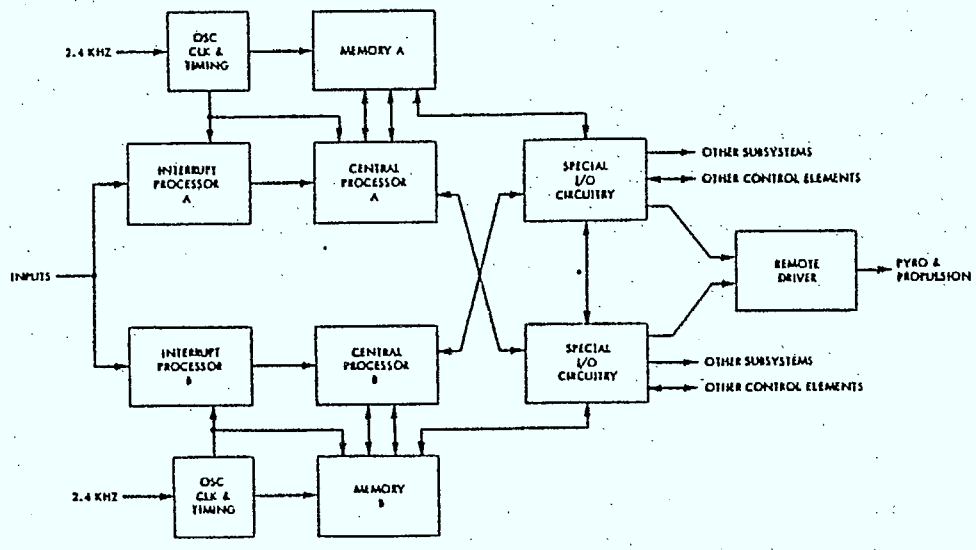


FIGURE 5.2 - VOYAGER ARTICULATION AND ATTITUDE CONTROL
SUBSYSTEM (Reprinted from [SCUL80])

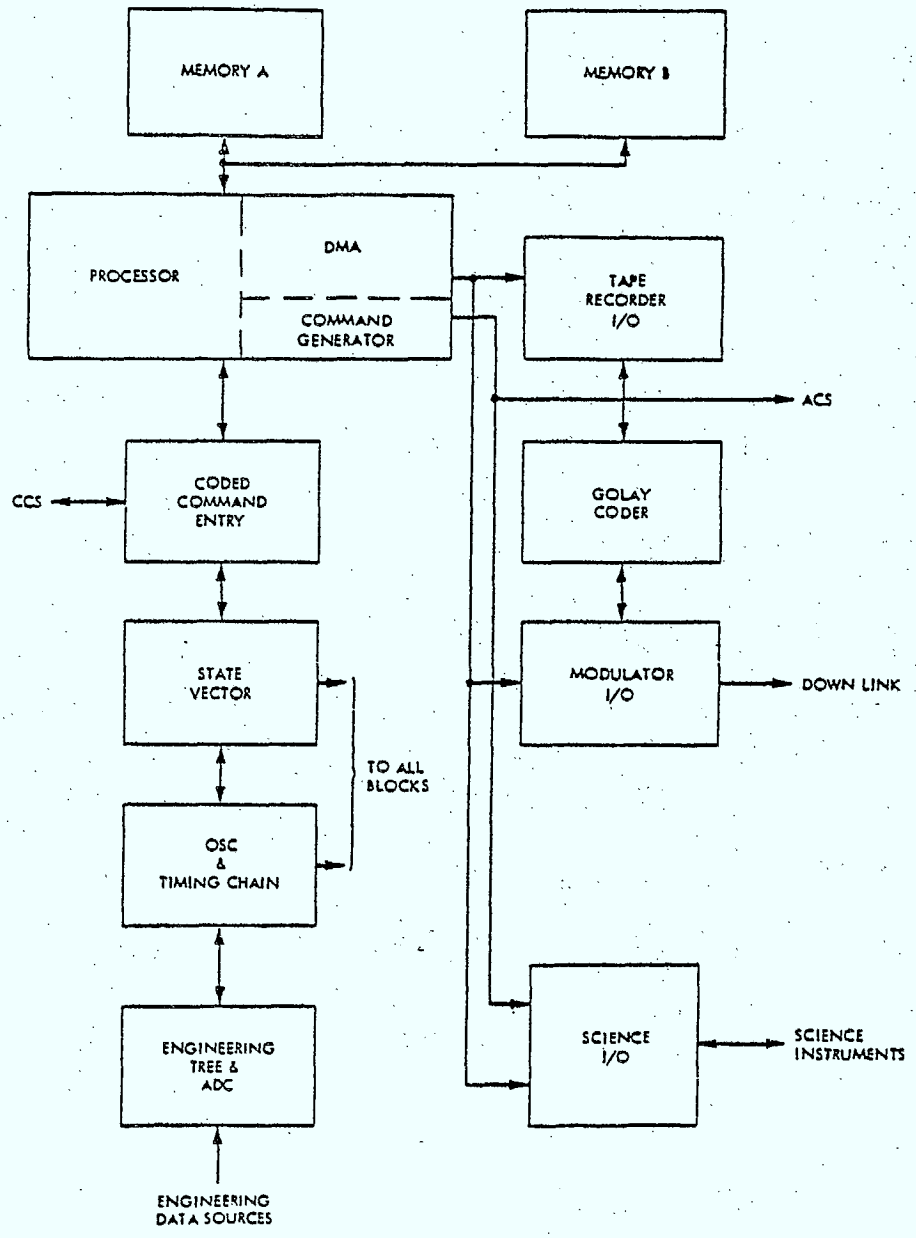


TABLE 5.3 - VOYAGEUR FLIGHT DATA SUBSYSTEM

The AACS (see Figure 5.2) controls the attitude of the spacecraft and is responsible for navigation. It is characterized by:

- 4K memory
- 18 bit word length
- a priority interrupt structure
- an add/subtract cycle time of 32.2 microseconds. It has an activity rate (utilization?) of about 95%.

5.4.3 The Galileo Spacecraft

The Galileo Orbiter spacecraft has two types of computer systems, the Command Data System (CDS) and the Attitude and Articulation Control System (AACS). The AACS is responsible for maintaining perturbations of the spinning section within acceptable levels to allow precise pointing of the scan platform. The CDS co-ordinates the activities of the spacecraft: co-ordination of fault protection, isolation and recovery; supplying the spacecraft clock and timing signals; collecting, formatting and encoding telemetry data; and analyzing and distributing commands received from the ground. In addition to these two subsystems, embedded processors are also part of most scientific instruments on board.

The CDS is a distributed computer system using six processors based on the Unified Data System described by Rennels in [RENN78a] and summarized in Section 4 of this report. It has overall responsibility for the control of the spacecraft and

for the operation of the science instruments as well. Both high level modules (32K RAM, 1K ROM) and low level modules (16K RAM, 1K ROM) are based on the RCA 1802 microprocessor with a 10 microsecond add/subtract cycle time. The bus data rate is 806.4 kbps.

The AACS uses two (one redundant) ATAC-16M microcomputers; these computers are built using four 4-bit slice 2900-series processing elements. The detailed characteristics of these processors are given in Table 5.5.

Weight	<18	
Power (Watts)	<34	
Word Size (Bits)	16	
No. Instructions	129	
No. Interrupts	8 levels vectored	
Programmed I/O	Up to 500K words/second	
DMA	Yes	
Floating point words	32 bits, 24/8 format	
Fixed point words	8, 16, 32 bits	
Microprogrammed	Yes	
Memory Size	30K words RAM 2K words ROM	
Memory Addressing	64K words directly addressed	
Execution Time	Fixed	Float
add/sub (s)	00.75	5.75
mult (s)	05.50	16.00
div (s)	11.25	28.50
Address Modes	8	
No. General Registers	16	

TABLE 5.5 - GALILEO AACS CHARACTERISTICS
(Reprinted from [GILL80])

5.5 Automation Trends

5.5.1 The Mariner Mars spacecraft (1969) was the first interplanetary spacecraft to use an in-flight programmable computer. Since then, the use of computers aboard spacecraft has been exhibiting a steadily increasing trend as exemplified by the Galileo spacecraft scheduled to go into orbit around Jupiter in 1985 with its twenty processors onboard. The growing use of computers aboard spacecraft has been driven by the increasing requirements for more autonomous operation. These requirements are discussed below [AREN77].

i) The need to protect against catastrophic failures. This requirement is spear-heading the development of fault-tolerant computer architectures.

ii) The need to perform very predictable and highly repetitive functions. Attitude control, for example, requires much automation to be effective.

iii) The need to prevent the loss of spacecraft communications. The spacecraft needs to be able to take care of itself during periods when it is out of contact with tracking stations.

iv) The need to accomplish mission objectives during very narrow time windows.

v) The need for real-time or near real-time control in such applications as the control of the descent of a vehicle to a planet's surface.

vi) The need to perform functions aboard spacecraft at great distances from the Earth. Many functions must be automated

since the round trip communication delay inhibits effective ground control.

vii) The need to implement and support increasingly more complex subsystems. Many subsystems require substantial processing (eg. some attitute control subsystems use statistical filtering techniques to increase precision) and some have processors incorporated into their design.

viii) The need to undertake multi-year missions. This requires more autonomy so that the spacecraft can take care of itself.

ix) The need to make better use of mission resources. Increasing emphasis is being placed upon autonomous spacecraft capabilities so that the quantity of Earth station equipment and the size of mission operations staff can be minimized. Figure 6.1.1 (reprinted from [GEVA79]) shows the present trend of ground operation costs per mission: illustrates how advanced automation will reduce overall mission costs.

5.5.2 As mentioned previously, the "intellitence" of spacecraft has been steadily increasing since a stored program computer was first used in 1969 on the Mariner Mars spacecraft. Three characteristics of a spacecraft which serve to define its "I.Q." are the number of processors onboard, the average speed of these processors, and the amount of memory available (see [AREN77]). Figures 5.5 through 5.7 show how spacecraft intelligence and autonomy have increased in recent years.

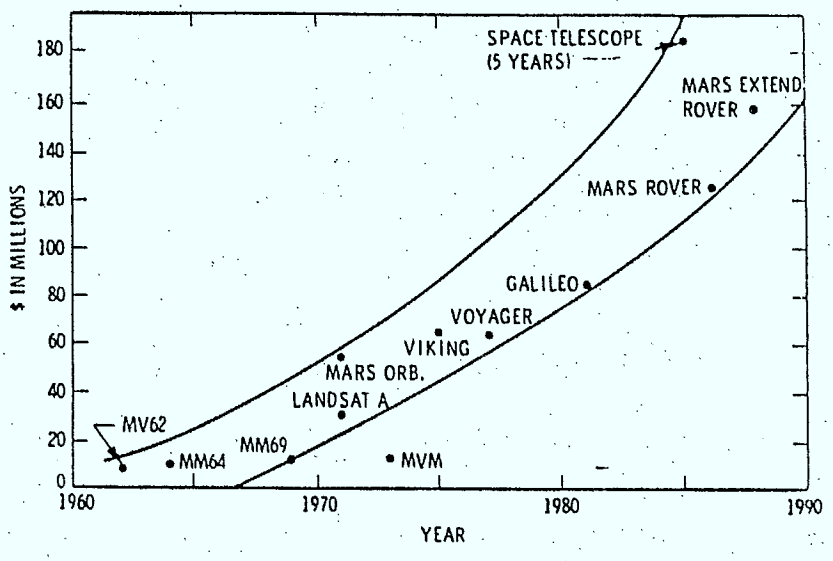


FIGURE 5.4 - TREND IN GROUND OPERATION COSTS PER MISSION
(Reprinted from [GEVA79])

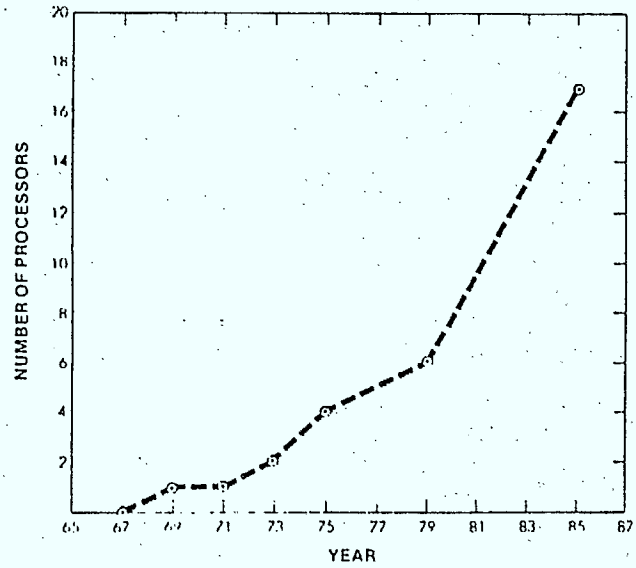


FIGURE 5.5 - NUMBER OF ON-BOARD PROCESSORS
(Reprinted from [BIRD79b])

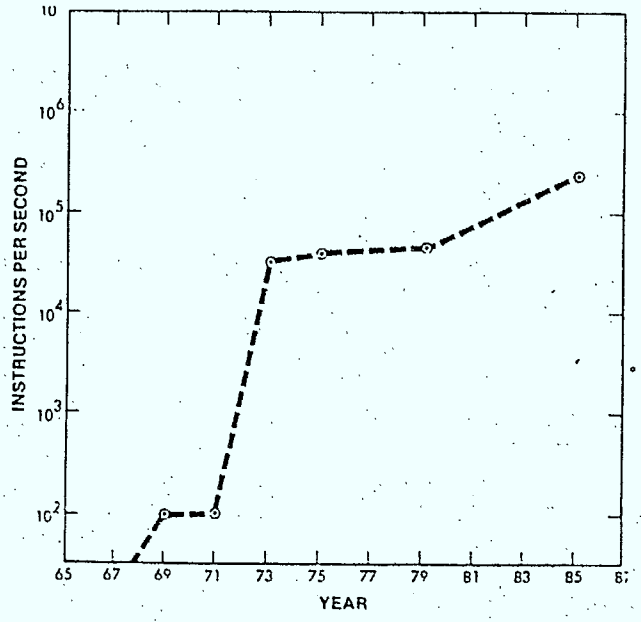


FIGURE 5.6 - AVERAGE PROCESSING SPEED
(Reprinted from [BIRD79b])

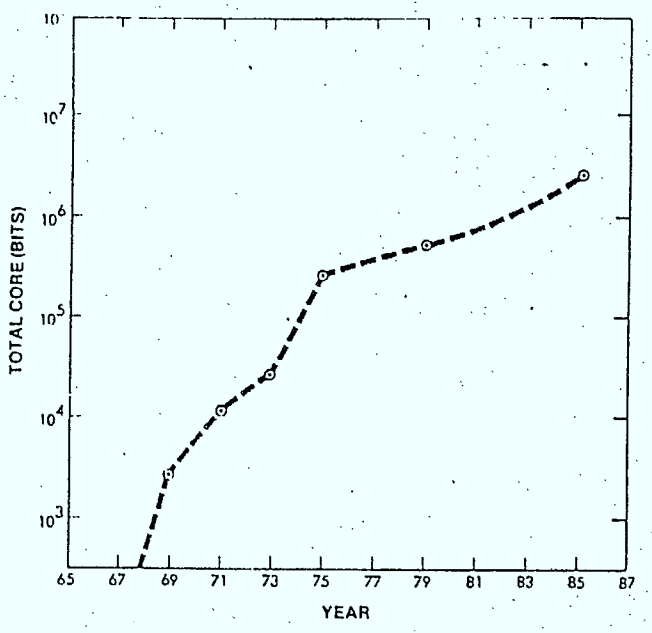


FIGURE 5.7 - TOTAL ON-BOARD MEMORY
(Reprinted from [BIRD79b])

Budget Category	Ground Operations	Orbital Operations	Data Analysis	Design
1. Mission Items	122/90	--	15/10	90/45
2. Multimission Operation Support	74/55	--	--	--
3. Post-Mission Data Analysis	--	--	103/70	--
4. Network Operations	44/30	--	--	57/30
5. Shuttle/Skylab Payload	--	14/10	20/10	80/40
6. Space Transportation	182/135	--	--	--
7. Space Industrialization	224/170	24/10	--	72/35
Totals	646/480	38/20	138/90	299/150

Budget Category	Test	Other	TOTAL
1. Mission Items	116/80	91	434/225
2. Multimission Operation Support	--	--	74/55
3. Post-Mission Data Analysis	--	--	103/70
4. Network Operations	8/5	108	217/65
5. Shuttle/Skylab Payload	100/65	--	214/125
6. Space Transportation	908/605	840	1930/740
7. Space Industrialization	120/80	160	600/295
Totals	1252/835	1199/0	3572/1575

Legend

x/y x: cost without advanced automation in the 1980s
y: potential savings by 2000 with advance automation in 1978 million $

TABLE 5.6 - ESTIMATED NASA YEARLY COSTS [GEVA79]

5.5.3 Committed to this trend, NASA has recently embarked upon a multi-year program which establishes a framework for computer systems research and development. The program is called NASA End-to-End Data System (NEEDS) and it aims to improve the effectiveness and efficiency of NASA's overall data/information management system by improving the performance and functional capability of on-board processors in the next ten to fifteen years [HOLC80]. The principal Advanced On-board Computing Facility components of NEEDS are:

1. Distributed Microprocessor Data Systems. The development of a microprocessor based distributed data system capable, in particular, of real-time on-board determination of accurate position, time and attitude.
2. Information Adaptive System. A set of hardware and software facilities interfacing directly with the sensors on board the spacecraft capable of direct preprocessing and editing to the maximum extent possible in order to minimize the amount of data transmitted and the need for ground control.
3. Massively Parallel Processor (MPP). High speed array processors capable of real-time image processing to be developed first for ground use and eventually for on-board use (1990s).
4. Synthetic Aperture Radar (SAR) Processor. Development concept similar to the MPP.
5. General Purpose On-board Computer Technology. In order to

limite proliferation, NASA has decided to standardize on the NSSC computers as much as possible. Since there is a large performance gap between the NSSC and the MPP, an intermediary solution will have to be developed.

6. SUMMARY

This report presented a basic review which serves as a background for further studies aimed at developing an integrated set of computer aided engineering tools for multiprocessor design. In particular, this report contains:

a. a review of architectures useful for the implementation of distributed systems: the common bus is the most widely used architecture, with multiple bus levels systems now appearing and with more advanced architectures now being studied.

b. an outline of the basic concepts of the software mechanisms required for the control and co-ordination of processes through synchronization and scheduling in distributed real-time systems.

c. a description of five current multiprocessor systems designed for spacecraft and avionics applications.

d. an introduction to current state and trends in the field of spacecraft computing through a review of processing requirements, processors used, systems implemented and development trends.

REFERENCES

[AIAA77] AIAA/NASA/IEEE/ACM Computers in Aerospace Conference, 1977, AIAA Paper # 77-1377.

[ANDE75] G.A. Anderson and E.D. Jensen, "Computer interconnection structures: Taxmomy, characteristics and examples:, ACM Computing Surveys, Vol. 7, pp. 197-213, December 1975.

[AREN77] W.E. Arens, "CCD Architecture for Spacecraft SAR Image Processing", Proceedings of the 1977 Computers in Aerospace Conference, Los Angeles, November, 1977, AIAA Paper # 77-1392.

[AUKS74] A.J. Aukstikalnis, "Spacecraft Computers", Astronautics and Aeronautics, July/August, 1974.

[BARR80] R.C. Barry and D.J. Reifer, "Galileo Flight Software Management - The Science Instruments", Proc. 4th Computer Software and Applications Conference, Chicago, 1980, pp. 684-690.

[BIRD79a] T.H. Bird and B.L. Sharpe, "Spacecraft Automated Operations", Proceedings of the Annual Rocky Mountain Guidance and Control Conference, 1979, Paper # AAS 79-016 (Advances in the Astronautical Sciences, Volume 39).

[BIRD79b] T.H. Bird and R.R. Sheahan, "Trends in the Automation of Planetary Spacecraft", Astronautics and Aeronautics, May, 1979.

[CARB77] R.L. Carberry, "Trends in Aerospace Computers", Proc. 1977 Computers in Aerospace Conference, 77, pp. 7-10.

[COHE75] E. Cohen and D. Jefferson, "Protection in the Hydra Operating System", Operating Systems Review, ACM SIGOPS, Vol. 9, No. 5, pp. 141-161, November 1975.

[COOP75] A.E. Cooper, W.T. Chow, "Shuttle Computer Complex", Proceedings of the IFAC 6th World Congress, 1975.

[CORL65] W.R. Corliss, "Space Probes and Planetary Exploration", D. van Nostrand Inc., Princeton, New Jersey, 1965.

[CROS77] W.A. Crossgrove, "Distributed systems: The next integration method", Proc. AIAA 2nd Digital Avionics Systems Conference, 77, pp. 45-53.

[DESP79] A.M. Despain and D.A. Patterson, "X-Tree: A Tree Structured Multi-Processor Computer Architecture", Proc. 5th Symposium on Computer Architecture, April 1979, pp. 90-101.

[DESW81] Y. Deswarte et al, "A Fault-Tolerant Multi-Microprocessor Architecture for SARGOS", 11th International Symposium on Fault-Tolerant Computing, June 1981.

[DIAS81] D.M. Dias and J.R. Jump, "Analysis and Simulation of Buffered Data Networks", IEEE Transactioon on Computers, Vol. C-30, April 1981, pp. 273-282.

[DYER77] M.C. Dyer, "Design and Development of Distributed Systems for Aerospace: A Hardware/Software Approach", Proceedings of the AIAA/NASA/IEEE/ACM Computers in Aerospace Conference, Los Angeles, November 1977, AIAA Paper # 77-1450.

[ENSL77] P.H. Enslow, "Multiprocessor Organization - A Survey", ACM Computing Surveys, Vol. 9, No. 1, March 1977, pp. 103-129.

[FANE79] E.V. Fanelli and H. Hecht, "The Fault Tolerant Spaceborne Computer", Proceedings of the AIAA 2nd Digital Avionics System Conference, 1977, AIAA Paper # 77-1490.

[FENG79] T. Feng, C. Wu and D.P. Agrawal, "A Microprocessor-Controlled Asynchronous Circuit Switching Network", Proc. 6th Symposium on Computer Architecture, April 1979, pp. 202-215.

[FRAN81] M.A. Franklin, "VLSI Performance Comparison of Banyan and Crossbar Communications Networks", IEEE Transactions on Computers, Vol. C-30, pp. 283-291, April 1981.

[FULL78] S.H. Fuller et al, "Multi-microprocessors: an overview and working example", Proceedings of the IEEE, Vol. 66, No. 2, pp. 216-228, February 1978.

[GEVA79] W.B. Gevarter, E. Heer, "Requirements and Opportunities for Autonomous Systems in Space", Advances in the Astonautical Sciences, Volume 39, 1979, AAS Paper #79-011.

[GILL79] G. Gilley, "The Fault Tolerant Spaceborne Computer", Advances in the Astronautical Sciences, Volume 39, 1979, AAS Paper # 79-015.

[GILL80] G. Gilley, "Digital Hardware for Use in Spacecraft Control Applications", Advances in the Astronautical Sciences, Volume 42, 1980, Paper # AAS 80-031.

[GIOF81] G. Gioffi et al, "MuTEAM: Architectural Insights of Distributed Multi/Microprocessor System", 11th International Symposium on Fault-Tolerant Computing, June 1981.

[GORD79] J.F. Gordon and A.J. Fucho, "Autonomy in Space Navigation", Astronautics and Aeronautics, May, 1979.

[GRIN80] A. Grinarov, L. Kleinrock and M. Gerla, "A Highly Reliable, Distributed Loop Network Architecture", 10th International Symposium on Fault-Tolerant Computing, October 1980.

[HAMA80] V.C. Hamacher and G.S. Shedler, "Performance of a Collision-free Local Bus Network having Asynchronous Distributed Control", Proc. 7th Symposium on Computer Architecture, May 1980, pp. 80-87.

[HARR79] J.A. Harris and D.R. Smith, "Simulation Experiments on a Tree Organized Multicomputer", Proc. 6th Symposium on Computer Architecture, April 1979, pp. 83-89.

[HECH77] H. Hecht, "Fault Tolerant Computers for Spacecraft", Journal of Spacecraft, Volume 14, Number 10, October 1977.

[HOLC80] L. Holcomb, "Overview of NASAs On-board Computing Technology Program", Proc. of COMPCOV 1980, IEEE cat. 80CH1491-0C, pp. 117-124.

[HOPK75a] A.L. Hopkins, "Hierarchical Autonomy in Spaceborne Information Processing", Proceedings of IFAC 6th World Congress, Cambridge, Massachussets, U.S.A.

[HOPK75b] Hopkins, A.L. and Smith, T.B., III, "The architectural elements of a symmetric fault-tolerant multiprocessor", IEEE Trans. Comput., Vol. C-24, No. 5, pp. 498-505, May 1975.

[HOPK78] Hopkins, A.L. et al, "FTMP - A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft", Proceedings of the IEEE, Vol. 66, No. 10, October 1978.

[HORO81] E. Horowitz ad A. Zorat, "The Binary Tree as an Interconnection Network: Applications to Multiprocessor Systems and VLSI", IEEE Transaction on Computers, Vol. C-30, April 1981, pp. 247-253.

[JENS76] E.D. Jensen, K.J. Thurber and G.M. Schneider, "A review of systematic methods in distributed processor interconnection", IEEE International Conference on Communications, June 1976, pp. 7-17 to 7-22.

[JENS78] E.D. Jensen, "The Honeywell experimental distributed processor - An overview", Computer, pp. 28-37, January 1978.

[JONE77] Jones, A.K. et al, "Software management of Cm*, a distributed multiprocessor", in AFIPS Conference Proceedings, Vol. 46, pp. 657-663, 1977.

[JONE79] A.K. Jones et al, "StarOS, a multiprocessor operating system for the support of task forces", Proceedings of the 7th Symposium on operating systems principles, pp. 117-128, December 1979.

[JONE80] A.K. Jones, P. Schwarz, "Experience using multiprocessor systems - A status report", ACM Computing Surveys, Vol. 12, No. 2, pp. 121-167, June 1980.

[KATS78] D. Katsuki et al, "Pluribus: An operational fault-tolerant microprocessor", Proceedings of the IEEE, Vol. 66, No. 10, October, 1978.

[KINN78] L.L. Kinney and R.G. Arnold, "Analysis of a Multiprocessor System with a Shared Bus", Proc. 5th Symposium on Computer Architecture", April 1978, pp. 89-95.

[KWAN81] C.L. Kwan and S. Toida, "Optimal Fault-Tolerant Realizations of Some Classes of Hierarchical Tree Systems", 11th International Symposium on Fault-Tolerant Computing, June 1981.

[LEVI75] R. Levin et al, "Policy/Mechanism Separation in Hydra", Operating Systems Review, ACM SIGOPS, Vol. 9, No. 5, pp. 132-141, November 1975.

[LIU80] T.S. Liu, "Availability Analysis of Tree-Structured Computer Communication Systems", 10th International Symposium on Fault-Tolerant Computing, 10th International Symposium on Fault-Tolerant Computing, October 1980.

[MAEK79] M. Maekawa, "Experimental Polyprocessor System (EPOS) - Architecture", 6th Symposium on Computer Architecture, April 1979.

[ORNS75] S. Ornstein et al, "Pluribus: A reliable multiprocessor", Proceedings of the AFIPS, National Computer Conference, AFIPS press, 1975.

[OUST80] J.K. Ousterhout et al, "Medusa: an experiment in distributed operating system structure", Communications of the ACM, Vol. 23, No. 2, pp. 92-105, February 1980.

[PATT79] D.A. Patterson, E.S. Fehr and C.H. Seguin, "Design Considerations for the VLSI Processor of X-Tree", Proc. 6th Symposium on Computer Architecture, April 1979, pp. 99-101.

[POWE78] D. Powell and J.G. Laprie, "RHEA: A System for Reliable and Survivable Interconnection of Real-Time Processing Elements", 8th International Symposium on Fault-Tolerant Computing, June 1978.

[PRAD81a] D. Pradhan and S. Reddy, "A Fault-Tolerant Communication Architecture for Distributed Systems", 11th International Symposium on Fault-Tolerant Computing, June 1981.

[PRAD81b] D. Pradhan, "Interconnection Topologies for Fault-Tolerant Parallel and Distributed Architectures", Proceedings 1981 International Conference on Parallel Processing, August 1981.

[PRAD81c] D. Pradhan, "Processor Interconnection Architectures for Fault-Tolerance and Diagnosability", Technical Report, Oakland University, Rochester, Michigan, September 1981.

[RENN78a] D.A. Rennels, "Reconfigurable modular computer networks for spacecraft on-board processing", IEEE Computer, pp. 49-60, July 1978.

[RENN78b] Rennels, D., "Architectures for Fault-Tolerant Spacecraft Computers", Proceedings of the IEEE, Vol. 66, No. 10, October 1978.

[RENN78c] Rennels, D., "Distributed Fault-Tolerant Computer Systems", Computer, Vol. 13, No. 3, pp. 55-66, March 1980.

[ROSS79] M.S. Ross, "NASA Standard Computers: A Description and Comparison", Advances in the Astronautical Sciences, Volume 39, 1979 AAS Paper # 79-024.

[SCUL80] J.R. Scull, "On-Board Computers for Control", Advances in the Astronautical Sciences, Annual Rocky Mountain Guidance and Control Conference, Volume 42, 1980, AAS 80-030.

[SIEW78] Siewiorek, D. et al, "A study of C.mmp, Cm*, and C.vmp: Part I - Experiences with Fault tolerance in Multiprocessor Systems", Proceedings of the IEEE, Vo. 66, No. 10, pp. 1178-1199, October 1978.

[SOL079] M.H. Solomon, R.A. Finkel, "The Roscoe distributed operating system", Proceedings of the 7th symposium on operating systems principles, pp. 108-118, December 1979.

[SOUB79] Soubirou, J., "Multiple-Microprocessor Systems in Attitude and Orbit Control Subsystems", Proceedings of AOCS Conference, Noordwijk, Oct 77, pp 233-239.

[STAK81] P. Staken, "One Step Forward - Three Steps Backup, Computing in the US Program", Byte, September 1981, pp. 112-144.

[SWAN77] Swan, R.J. et al, "A modular, multi/microprocessor", in AFIPS Conference Proceedings, Vol. 46, pp. 637-644, 1977.

[WENS72] Wensley, J.H., "SIFT software implemented for fault tolerance", in Proceedings of Fall Joint Computer Conference, AFIPS Press, Montvale, N.J., 1972, Vol. 41, pp. 243-253.

[WENS78] Wensley, J.H. et al, "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control", Proceedings of the IEEE, Vol. 66, No. 10, October 1978.

[WILS80] L.S. Wilson, "Needs of the '80s", Aeronautics and Astronautics, April 1980.

[WITT80] L.D. Wittie, A.M. van Tillborg, "MICROS, a distributed operating system for MICRONET, a reconfigurable network computer", IEEE transactions on Computers, Vol. C-29, No. 12, pp. 1133-1144, December 1980.

[WITT81] L.D. Wittie, "Communications Structures for Large Networks of Microcomputers", IEEE Transaction on Computers, Vol. C-30, pp. 264-273, April 1981

[WOLF79] J. Wolf et al, "Design of a Distributed Fault-Tolerant Loop Network", 9th International Symposium on Fault-Tolerant Computing, June 1979.

[WU81] S.W. Wu and M.T. Liu, "A Cluster Structure as an Interconnection Network for Large Multimicrocomputer Systems", IEEE Transactions on Computers, Vol. C-30, pp. 254-264, April 1981.

[WULF72] W.A. Wulf, C.G. Bell, "C.mmp; a multi-miniprocessor", AFIPS fall joint computer conference, AFIPS press, December 1972.

[WULF74] W. Wulf et al, "Hydra: The Kernel of a Multiprocessor Operating System", Communications of the ACM, Vol. 17, No. 6, pp. 337-345, June 1974.

[WULF75] W. Wulf et al, "Overview of the Hydra operating system development", Operating Systems Review, ACM SIGOPS, Vol. 9, No. 5, pp. 122-132, November 1975.

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