

# NASA Contractor Report 172286

NASA-CR-172286  
19840017298

## DEVELOPMENT AND EVALUATION OF A FAULT-TOLERANT MULTIPROCESSOR (FTMP) COMPUTER Volume IV FTMP Executive Summary

T. Basil Smith, III and Jaynarayan H. Lala

THE CHARLES STARK DRAPER LABORATORY, INC.  
555 Technology Square  
Cambridge, Massachusetts 02139

CONTRACT NAS1-15336  
FEBRUARY 1984

FOR REFERENCE

NOT TO BE TAKEN FROM THIS ROOM

LIBRARY COPY

CON 14 1984

LANGLEY RESEARCH CENTER  
LIBRARY, NASA  
HAMPTON, VIRGINIA

**NASA**

National Aeronautics and  
Space Administration

Langley Research Center  
Hampton, Virginia 23665



R-1603

DEVELOPMENT AND EVALUATION  
OF A  
FAULT-TOLERANT MULTIPROCESSOR (FTMP) COMPUTER


VOLUME IV  
FTMP EXECUTIVE SUMMARY

by

T. Basil Smith, III  
Jaynarayan H. Lala

February, 1984

Contract NAS1-15336  
with  
The NASA Langley Research Center  
Hampton, Virginia 23365

Approved: 

N. E. Sears  
Department Head

The Charles Stark Draper Laboratory, Inc.  
Cambridge, Massachusetts 02139

1084-25366#



**CSDL-R-1603**

**DEVELOPMENT AND EVALUATION OF A  
FAULT-TOLERANT MULTIPROCESSOR (FTMP)  
COMPUTER  
VOLUME IV FTMP EXECUTIVE SUMMARY**

**by**

**T. Basil Smith, III and Jaynarayan H. Lala**

**February 1984**

## FOREWORD

This report was authored by Dr. T. Basil Smith and Dr. Jaynarayan H. Lala. Dr. Smith was the project engineer. Mr. Charles Meissner was the NASA technical monitor for the period January-December 1982, and Mr. Nicholas Murray was the technical monitor from August 1978 to December 1981. Following are some of the people who contributed to the success of this project.

### Draper Laboratory

Dr. Albert Hopkins  
Mr. Jack McKenna  
Ms. Linda Alger  
Mr. Kevin Koch  
Mr. Alan Wimmergren  
Mr. Robert Scott  
Mr. Joseph Marino  
Mr. David Hauger  
Mr. Mario Santarelli

### Collins Avionics

Mr. Ron Coffin  
Mr. Charles Schulz

## TABLE OF CONTENTS

<u>Chapter</u>		<u>Page</u>
1	INTRODUCTION .....	1
2	THE FAULT-TOLERANT MULTIPROCESSOR CONCEPT .....	5
3	SYSTEM ARCHITECTURE AND EXECUTIVE DETAIL .....	9
	3.1 Overall Architecture Organization.....	9
	3.2 Processor Module and System Bus Interaction .....	12
	3.3 Slave Region and System Bus Interaction .....	13
	3.4 Clock Generation Module .....	18
	3.5 Bus Guardian Units .....	18
	3.6 System Bus Interface Circuits .....	19
	3.7 Power System .....	20
	3.8 Executive Software .....	21
4	RELIABILITY AND AVAILABILITY MODELS .....	25
	4.1 Survival Probability Models .....	26
	4.2 Dispatch Reliability of the FTMP Computer .....	28
5	THROUGHPUT AND RESPONSE TIME PERFORMANCE .....	31
6	TEST AND EVALUATION EXPERIMENTS .....	37
7	CONCLUSIONS .....	43
8	REFERENCES.....	47

## LIST OF FIGURES

<u>Figure</u>		<u>Page</u>
1	FTMP Physical Organization and Packaging .....	6
2	Software Appearance of FTMP .....	10
3	Processor and Memory Redundancy .....	11
4	FTMP Probability of Failure .....	27
5	FTMP Dispatch Reliability .....	30
6	Frequency Distribution of Total Recovery Time (1 second scale) .....	40
7	Frequency Distribution of Total Recovery Time (100 msec scale) .....	41



## CHAPTER 1

### INTRODUCTION

This report is the final volume of a multi-volume report on the Fault-Tolerant Multiprocessor (FTMP) project sponsored by the Langley Research Center of the National Aeronautics and Space Administration under Contract NAS1-15336 covering the period from August 1978 to December 1982. This volume is an executive summary of the project. Previous reports produced under this contract are as follows:

Volume I - FTMP Principles of Operation

Volume II - FTMP Software

Volume III - FTMP Test and Evaluation

The FTMP architecture is an evolving concept which is directed toward ultra-high reliability aerospace applications. The engineering prototype was sized to serve as the information processing and control system for an advanced civil transport application. Its projected reliability, a probability of failure of  $10^{-9}$  over a ten hour flight, is suitable for full time fly-by-wire control in such an application. The ultimate reliability and throughput of the FTMP can be varied by addition or deletion of processing and memory modules and is also affected by such parameters as packaging and circuit technologies. The

FTMP thus has an applicability which is broader than this single design point, and it can be tailored to a wide range of demanding applications.

The flightworthy engineering model of the FTMP was constructed under the present contract by the Collins Avionics Division of Rockwell International Corporation per architectural specifications provided by the Charles Stark Draper Laboratory (CSDL). The FTMP engineering model was delivered by Collins to CSDL in June 1980. It was then integrated with CSDL-developed software, MIL-STD 1553 remote terminal emulators, special test hardware, and a real-time aircraft simulator.

About six months after delivery to Draper, hardware modifications, which were required to correct implementation and design flaws, were completed and an executive kernel capable of demonstrating fault tolerance and dynamic reconfigurability of the FTMP was operating. A display console attached to the FTMP via the 1553 I/O interface provided system status displays allowing convenient tracking of system reconfigurations in response to fault insertion. In the following year, the executive and flight control software was elaborated and the system was integrated into a jet transport aircraft simulator with cockpit. During this phase of the program, the fault-tolerance of the FTMP was demonstrated to hundreds of visitors from NASA, all branches of the armed forces, the Department of Defense, other government agencies, industry, and academia.

Once the software development was complete and the FTMP was fully integrated into the simulation laboratory, it was subjected to over 20,000 faults as part of a test, evaluation, and validation plan. FTMP response to each fault was recorded and were reduced to frequency histo-

grams. Fault insertion and data acquisition were done automatically under control of a PDP-11/60, using a fault injection device designed and built at Draper.

The FTMP engineering model and software along with the test facility hardware and software were delivered to NASA Langley in August 1982.

Overall, the FTMP program was a notable success, demonstrating and validating many of the underlying design concepts.

The first three volumes of this report dealt with various aspects of the FTMP in great detail. This final volume summarizes the results of this project. Chapter 2 highlights the FTMP architecture, hardware implementation, and executive software. Chapter 3 summarizes reliability modeling results. Chapter 4 is a brief discussion of performance bottlenecks and advantages of the FTMP architecture. Results of experimental test and evaluation are presented in Chapter 5, and Chapter 6 concludes this report.



## CHAPTER 2

### THE FAULT-TOLERANT MULTIPROCESSOR CONCEPT

The two underlying philosophic assumptions of the Fault-Tolerant Multiprocessor Concept are: 1) that crucial elements of the fault-tolerant computing system can be most economically implemented in hardware, and 2) that the chosen architecture for a fault-tolerant system should have performance, programmability, and useability attributes which are as good as those of equivalent simplex or non-redundant systems.

The architectural organization which was selected for the FTMP was that of a conventional homogeneous multiprocessor. It was felt that this organization was particularly well adapted to the programming and performance requirements of the projected commercial transport environment. Figure 1 illustrates such a system organization. The multiprocessor environment provides an easily expanded and general purpose computing system which is able to handle the multiple concurrent tasks projected for such an application. The use of a multiprocessor structure with its shared memory also provides for the required high speed communications and data buffering between the tasks. The multiprocessor organization also has secondary reliability advantages. Degraded versions of such a

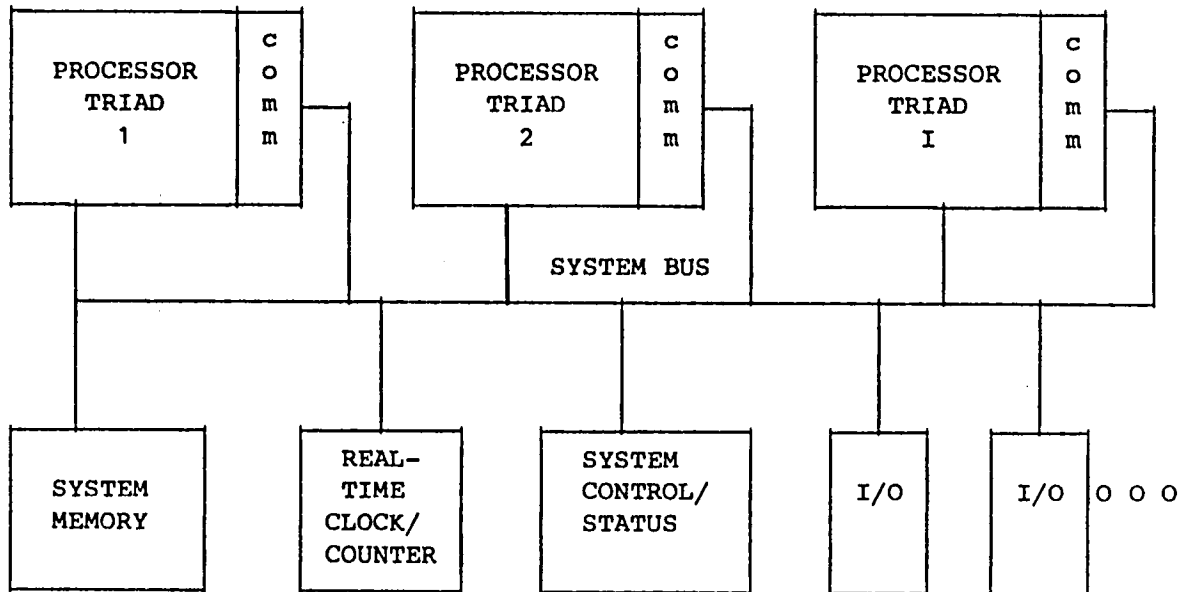


Figure 1. Software Appearance of FTMP

system can function with reduced numbers of processors, but these advantages would not have justified a multiprocessor organization had that organization not been a natural choice for its intended application.

The second design choice was to provide redundancy and fault-handling mechanisms which are transparent to the software organization, both as regards to its structure and performance. This was, in fact, critical. After matching the architecture to the application (redundancy and fault-tolerance aside) it is essential that the additions required for fault tolerance not compromise the rationale for the original architectural choices. The FTMP solution allows for the independent design of applications and operating system code design, apart from the design solutions to fault handling and fault recovery problems. The aggregate complexity of these two problems is thus much less than if they are handled jointly. A hardware implementation of the primary fault-detection and fault-masking mechanisms achieved this goal with minimal impact on system throughput. This compares to software-intensive solutions which generally must devote over 50% of available processing power to such functions.

Experience with the FTMP confirmed the soundness of these principal architectural decisions.

All of the software of the FTMP was developed, coded and debugged as if it were to run on a simplex equivalent of the multiprocessor. Code, debug and test was in fact actually conducted using a simplex development station. Code was then routinely moved from this development station to the redundant configuration without difficulty. The

programmability and software productivity advantages hoped for were thus largely confirmed. Executive design was conventional and the architecture allowed efficient control from a higher level language. Programmer productivity exceeded 140 assembler level language statements (or their equivalent) per programmer day, competitive with the best practice today.

The choice of hardware mechanisms to provide primary fault handling also proved to be sound. Performance degradation due to fault-handling hardware was minimal and the implementation was indeed largely transparent to the software. Despite implementation flaws (in areas unrelated to fault-handling hardware) which caused overall performance to fall somewhat short of projections, comparisons with the SIFT approach indicated the substantial advantage of this approach. Basic fault-handling functions in the FTMP were handled by hardware and were at least two orders of magnitude faster than the software equivalents in SIFT [1,2]. A basic paradox was that this specialized voting and fault-handling circuitry of the FTMP proved to be smaller and less complex and much less trouble-prone than the specialized data exchange circuitry of the SIFT machine. This can be attributed to the efficiencies of synchronous (FTMP) versus asynchronous (SIFT) exchange mechanizations and the trivial amount of circuitry which was in fact required to implement the basic fault-handling mechanisms.



## CHAPTER 3

### SYSTEM ARCHITECTURE AND EXECUTIVE DETAIL

This chapter outlines the functional and design concepts of the Fault-Tolerant Multiprocessor engineering model, FTMP.

#### 3.1 Overall Architecture Organization

Figure 1 of the preceding chapter illustrates the functioning of this system from a logical software or programmer's viewpoint. Three processor triads function as the logical equivalent of simple processors with shared access to a single shared system bus. Through this bus they share access to common memory, I/O ports, a real-time clock/counter, and control and status registers. Each processor triad can also directly write to the communications registers of the other processor triads using the system bus. This provides processor to processor interrupt facilities which are required for efficient implementation of the multiprocessor execution. The entire system is synchronized by the equivalent of a single system-wide clock.

The actual redundancy underlying the buses, system memory triads, processor triads, the clock quad, and the I/O system is invisible to the programmer. This redundancy is shown conceptually in Figure 2 for the processor and memory portion of a full up system configuration and in more detail for the entire system in Figure 3. Any three processor

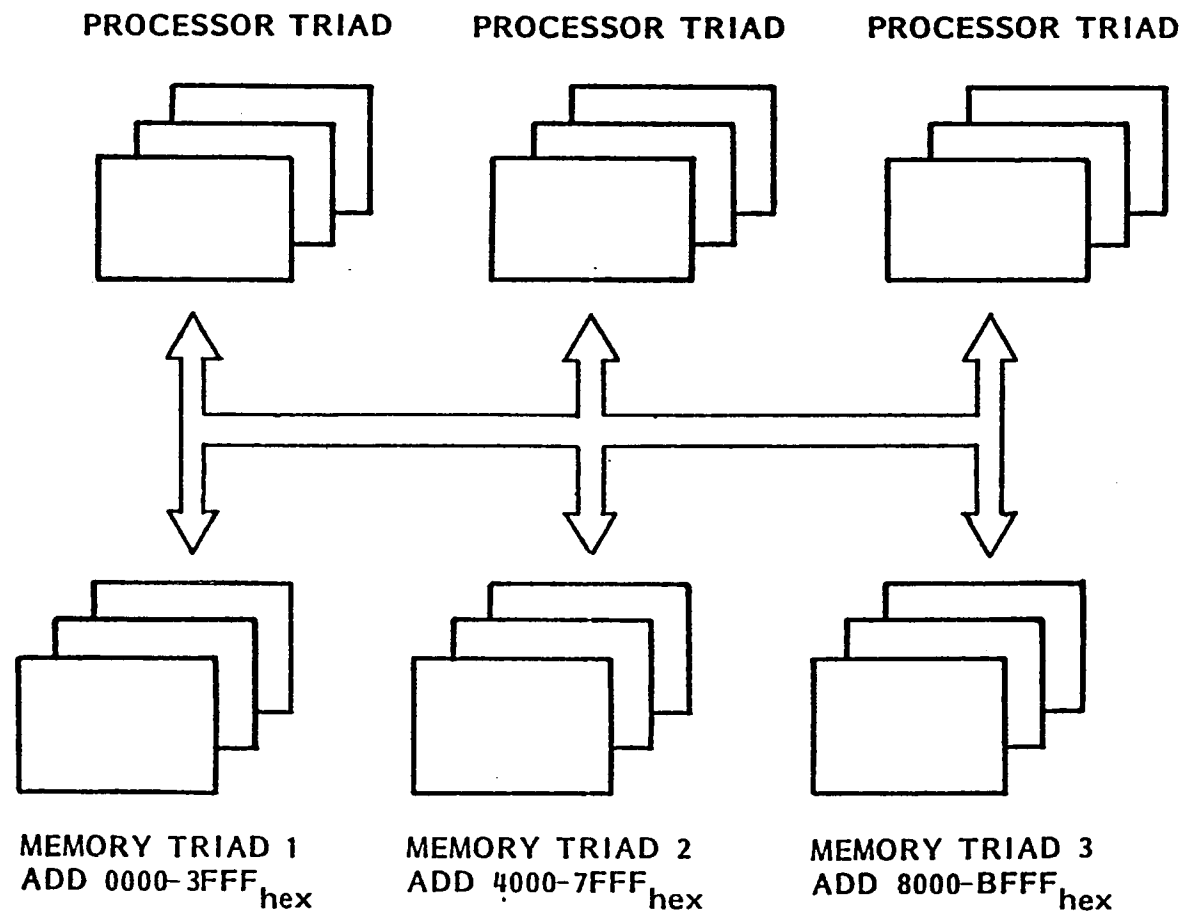


Figure 2. Processor and Memory Redundancy

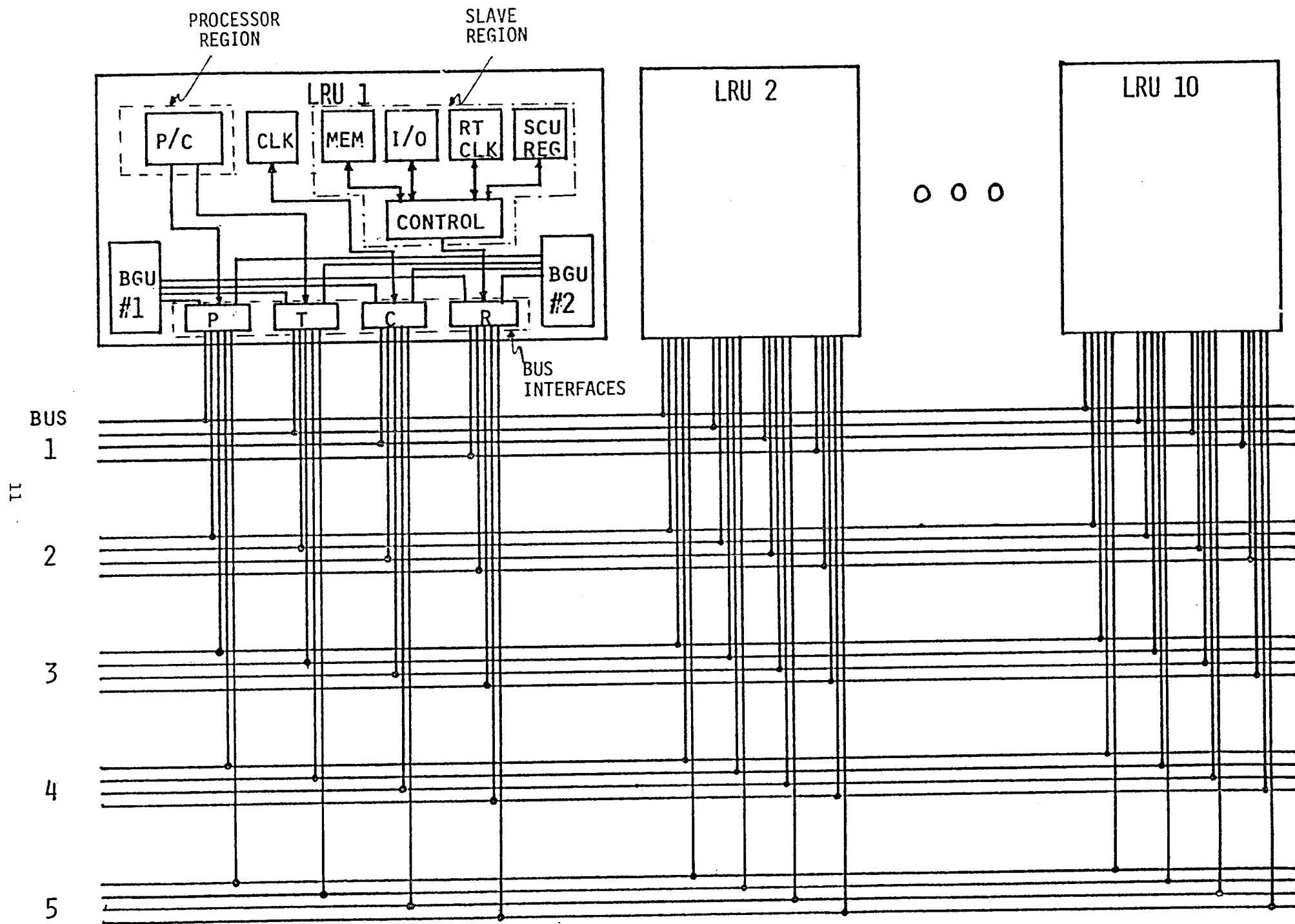


Figure 3. FTMP Physical Organization and Packaging

modules can be combined to form a processor triad, as can any three memory modules.

The system is constructed of ten identical line replaceable units (LRUs). Each LRU contains a Processor module, a Slave module (Common System Memory and I/O), a Clock Generation module, two Bus Guardian units, System Bus Interface circuits, and a Power subsystem. Figure 3 illustrates this physical organization and packaging.

### 3.2 Processor Module and System Bus Interaction

Processor modules operate in groups of three called processor triads. Processor triads are formed by assigning the processor modules of any three LRUs to work together in tight synchronism. In the FTMP it is possible for up to three processor triads to be in operation simultaneously, utilizing processors from nine of the ten available LRUs. The processor of the tenth LRU serves as a spare. A processor triad functions as if it were a single processor executing a single instruction stream. With three triads operating simultaneously, three instruction streams are in parallel execution. The system then functions as a conventional three-processor multiprocessor. The failure of a single processor of a triad does not impact the correct execution of that triad's instruction stream, because voting is used to mask the effects of the failure. Comparison techniques also enable the failed processor to be detected and identified. A spare processor region can then be used to replace the failed element of the triad. If no spares are available, the damaged triad is retired from service with the surviving functional

elements being used to replenish the spares pool, and thus available to repair processor failures in the other triads.

The processor triads write data to, or read data from, locations within the system bus address space by means of the System Bus. This bus is a quintuply redundant, fully duplex, eight megabit per second serial bus. During a block transfer, data can be written from a processor triad at a peak rate of one word every 5 microseconds. Data can be read by a processor triad at a peak rate of one word every 3 microseconds. The error correction and error detection relies upon voting and comparison of data and addresses appearing on redundant elements of the System Bus.

At any one time three of the five redundant bus lines are active. These active lines are called the bus triad. Each element of a processor triad transmits data and addresses on a different one of the bus triad's lines. Since the elements of the processor triad are all operating in tight synchronism, it is possible for any unit receiving a processor triad read or write request to compare the separate versions of that request by examining the separate copy of that request arriving on each bus of the bus triad. The receiving unit can correct any errors caused by a single processor region failure or single bus failure by using majority voting.

### 3.3 Slave Module and System Bus Interaction

The Slave Module contains a number of subsystems, all of which are addressed, read, and written as locations in the System Bus address space. These subsystems are the shared System Memory Module, the I/O

Port, LRU Control/Status and Communications Registers, and a Real-Time Clock/Counter. Certain of these modules are normally operated within triads of three; others are operated as single units.

Shared System Memory Modules are normally operated within a triad. The system memory modules of three LRUs are assigned to function together servicing a block of the system bus address space. Up to three memory triads can be formed from the system memory modules of nine of the ten LRUs. The tenth LRU's system memory module would then serve as a spare. In the FTMP as programmed only two memory triads were required and they were backed up by four spares. Each memory triad is assigned to serve a different block of the system memory address space. The failure of a memory module within a triad does not impact the integrity of data stored in that block, as voting is used to mask the effects of the failed module. Comparison techniques also enable the failed module to be detected and identified. A spare memory module can then be used to replace the failed element of the triad.

The real-time clock/counters of each LRU are also intended to operate together as a triad. The real-time clocks are used to maintain the absolute time base for the system. All real-time clocks are addressed by the same system bus address. A processor triad write to that location sets all real-time clocks to the same value. The real-time clocks of three LRUs can be armed to respond to read requests. Only those three LRUs respond to any processor triad real-time clock read requests and as such they function as the real-time clock triad. A failure of any element of that triad is masked by voting. Comparison

techniques enable the faulty unit to be identified. Any one of the unarmed real-time clocks can be used to replace the failed element of the real-time clock triad. Note that even the unarmed real-time clocks respond to write commands from a processor triad. Thus they will always agree with the elements of the real-time clock triad and can therefore be used to replace an element of the triad without reinitialization.

Unlike the processor modules, the memory modules, and the real-time clock/counters, the I/O ports operate independently of one another. Each I/O port responds to its own unique set of system bus addresses. Data and command words are transferred from a processor triad to an I/O port over the System Bus, appearing to the processor triad as routine system bus writes. As with any processor triad writes, voting at the receiving end serves to mask the failure of any one of the processor triad elements. The I/O port buffers any I/O transmissions, assembling an entire message before initiating an I/O bus transaction. The I/O port also buffers any incoming I/O transaction, assembling an entire remote terminal message. The entire transaction is then transferred as a block to a processor triad in response to a read request from that processor triad. The I/O port utilizes MIL-STD-1553A data bus protocols and signalling standards in its communications with the exterior. Two twisted shielded pairs can be used, one for transmitting and one for reception, creating a fully duplex data link. If these two pairs are tied together to a single twisted pair, then they conform to all specifications of MIL-STD-1553A. A MIL-STD-1553A avionics data bus is a 1 MHz serial data bus employing Manchester encoding to send both clocking and

data information on a single shielded twisted pair bus line. Maximum I/O transaction length can be 32 data words, one command word, and a status word requiring up to 700 microseconds for the transaction to be made. During this period the I/O port can act independently, and the processor triad may release the System Bus for regular bus traffic. Since each I/O port can operate independently, it is possible for the FTMP to be engaged in up to ten I/O bus transactions simultaneously, one on each of the I/O buses dedicated to each of the ten I/O ports.

The remaining elements of the Slave Region are System Control/Status and Communications Registers. These elements are used to control various parts of an LRU, to read the status of the error detection circuitry of an LRU, and to provide direct processor triad to processor triad communications.

The control registers are all write only. They are assigned fixed locations within the system bus address space depending upon LRU identification number. These LRU control registers control which bus lines the LRU uses for voting, triad assignment for the processor region, memory relocation factor for the system memory, whether the real-time clock is armed or not, and other LRU assignments or functions.

The status registers, or error latches, can only be read by a processor triad. They report any bus errors observed by the error detection circuits of the LRU. Like the LRU control registers, the status registers are assigned fixed system bus addresses dependent upon their LRU identification number.



The communications registers are used to implement direct processor triad to processor triad communications. Each communications register can only be written using the system bus. The communications register can be read by the processor region of the LRU directly, appearing as a local memory location on its internal processor module data bus. The system bus address assignment of each communication register within the LRU is keyed to the processor region triad assignment of that LRU. This assignment is contained in one of the control registers of the LRU. The local processor region transfer bus address of each communications register is fixed and is the same for all LRUs.

Only one LRU responds to control register writes or status register reads, that LRU being determined by the system bus address of the register being accessed. All LRUs with the appropriate processor region triad assignment will respond to communication register writes. When multiple LRUs are responding to a communication register write, they act in tight synchronism with one another.

Each LRU's Slave region is assigned to transmit on only one element of the redundant system bus. These assignments are made so that each element of a system memory triad or real-time clock/counter triad is assigned to a different bus. Each element of a processor triad can therefore mask a fault within a responding triad by appropriate majority voting circuitry. When reading from a simplex source, such as the I/O port or status register, the processor triad does not receive redundant information, but instead must accept the data from the single system bus line on which it appears and verify its accuracy by other means, which are described in Chapter 5.

### 3.4 Clock Generation Module

All elements of the multiprocessor operate using a common time reference. This time base is provided by the Clock Generation Modules of four LRUs which are phase locked to one another. The Clock Generation Regions of the remaining LRUs are then phase locked to any three of elements of the clock quad. Each clock generator thus provides a timing source for its LRU which is in synchronism with all other correctly functioning generators [3]. Such a system can tolerate the failure of any one of the clock generators. All correctly functioning clock generators remain synchronized despite such a failure. A failure within the quad is detected and identified and another clock generator can be assigned to replace the failed unit. Of the ten clock generators, four are assigned to the quad clock and the remaining six are either spare and in standby mode, or failed.

### 3.5 Bus Guardian Units

The overall integrity of the system relies upon the ability to reliably control the access that any element of an LRU has to the system bus. Each LRU of the system has two Bus Guardian Units, BGUs, which function to protect the system bus from a failed or malfunctioning LRU or other element within the LRU. Each BGU has bus enabling registers which control the LRU access to each individual line of the system bus. In order that the LRU be enabled to transmit on any line of the system bus, it is necessary that the enabling bit from both BGUs within that LRU be set. Either BGU can block the LRU's ability to transmit on a line.

When the configuration control program creates a processor triad, for example, it must first assign the processor regions of three LRUs to the same processor triad. It does this by writing into the system control registers of the selected LRUs. It must then assign each processor region to transmit on separate lines of the system bus triad. It does this by writing to the enabling registers of both BGUs of each selected LRU, assigning each LRU to the appropriate processor transmit lines of the system bus.

The register loading mechanism of each BGU responds to a unique system bus address keyed to that BGU's LRU and location within the LRU. This register loading mechanism allows each BGU register to be written by a normal system bus write transaction. Because it is important that a BGU act independently of the LRU in which it is located, each BGU receives all the redundant copies of all processor triad transmissions appearing on the system bus. All registers within the BGU are nonvolatile so that the bus assignments and line select codes are remembered through a power failure.

### 3.6 System Bus Interface Circuits

The actual connections to the system bus are made by the System Bus Interface Circuits within an LRU. These circuits perform several functions.

First, and most obviously, they provide the necessary drivers and receivers for each individual bus line. Selection of these drivers and

receivers is dictated by electrical constraints such as adequate power and noise immunity.

Secondly, the interface circuits provide the means for controlling the fault environment of the system bus. The driver circuit for each line performs the necessary gating of LRU signals onto system bus lines depending upon the values obtained from the BGU enabling registers. Thus it is this driver circuit which actually functions to cut the connection between an LRU and the system bus. The receiver circuits distribute independent copies of the system bus data, as required to each BGU within the LRU and to the other LRU circuitry. Each BGU and the input processing circuitry of the slave region must receive independent copies of the processor transmission lines so that they can each perform an independent vote and act independently of one another in response to processor triad commands. Partitioning within the interface circuitry is designed so that a single fault can bring down no more than one element of the system bus or pollute no more than one copy of the received data from the system bus. The partitioning and fault containment aspects of these interface circuits are critical to the overall fault tolerance of the FTMP.

### 3.7 Power System

Each LRU has its own power subsystem. This subsystem consists of a power supply, which provides all the required voltages used within the LRU, and a battery backup circuit, which provides low power battery power for maintaining the CMOS system memory and the nonvolatile registers

during primary power loss. This local power subsystem is overvoltage and overcurrent protected.

The local power subsystem of each LRU draws power from four 28 VDC power buses. Each LRU is fused at its connection point to this quad redundant power bus so that internal shorts within an LRU can at most only momentarily disrupt power on the primary power buses. The local power subsystems have adequate energy storage to tolerate these interruptions while the fuses blow. The local power supplies draw power evenly from all of the power buses. Thus under most circumstances, each of the power buses is fairly equally loaded. Any one of the power buses is capable of fully supplying all power to operate the entire FTMP.

The four primary power buses are driven from four independent primary power supplies. In this particular implementation, each of these power supplies is identical and converts three phase 208 VAC, 400 Hz input power to 28 VDC.

### 3.8 Executive Software

The Executive software is responsible for managing the hardware and software resources, for maintaining system integrity and for timely and orderly execution of applications tasks. It forms a link between the FTMP hardware and the user programs. The FTMP hardware, along with the Executive, project an image of a virtual machine to the user such that the hardware redundancy and redundancy management become transparent to the user. The user is only aware of a multiprocessing environment in

which several processors execute tasks in parallel and are linked together by a single shared memory.

The part of the Executive software that is responsible for the execution of the applications tasks is called the Dispatcher. The Dispatcher is at the heart of the Executive. The remaining executive or systems tasks, that is, all systems tasks other than the Dispatcher, are treated by the Dispatcher like applications or user tasks. These other systems tasks include a system configuration controller, system status displays and self-test programs. They are scheduled to run by the Dispatcher as user tasks.

All the user and systems tasks are repetitive in nature. They are executed at one of three different iteration rates. The three rate groups, called R4, R3, and R1, presently execute at 25, 12.5, and 3.125 Hertz, respectively. The highest frequency tasks, that is, those in the R4 rate group, are given the highest priority, while the lowest frequency tasks, that is, those in the R1 rate group, are given the lowest priority for execution. The configuration controller and display tasks are dispatched at the R1 rate.

The Executive is a timer-interrupt driven "floating" executive. Each iteration of the R4 rate group tasks is initiated by a hardware timer interrupt. The Executive can run in any processor triad, no processor triad is a master or slave triad and all triads have equal authority. Only one triad is allowed to alter the Executive data bases at any given time. Access to the shared data bases, such as task queues etc., by different processor triads, is controlled by semaphores resident

in the shared memory. The timer interrupt initiates R4 task iteration or frame in just one triad. The other triads follow the lead triad when prompted to do so through an IPC (Inter-Processor Communication) interrupt. At the completion of an R4 frame, the triad completing execution of the last R4 task becomes responsible for initiating the next R4 frame using its timer-interrupt. All I/O is performed in the R4 frame.

The second element of the Executive is the system configuration controller. The configuration controller detects hardware faults by analyzing information from the error latches associated with the bus voting circuitry of each LRU. It is also responsible for identifying faulty units and reconfiguring the system to replace them with spares, or gracefully degrading the system if no spares are available. Faults are isolated to the LRU sub-unit level such as processor, memory, clock, I/O port, and system bus line. Weak intersections of LRU's and buses, that is, an LRU unable to transmit or receive on a particular bus line, are also identified. Faults that do not persist long enough to be isolated are handled by a transient fault analysis algorithm. Demerits are assigned to all possible sources of a transient fault. The accumulated demerits are then analyzed for statistical significance to locate the source(s) of transient faults. In addition to on-demand FDIR (Fault Detection, Isolation and Recovery) the configuration controller periodically checks critical hardware elements using self-test programs. Examples of such items are voters and error latches. Spare sub-units are also constantly cycled into active state to ascertain their integrity.

These include processors, system memory units, fault-tolerant clock elements, I/O ports, and system bus lines.

The third element of the Executive software is the bootstrap/restart program. This program is responsible for bootstrapping the FTMP when the system is "cold", that is the system memory has not been initialized. In the cold start case, the system memory is initialized using an external device such as a cassette tape on a 1553 remote terminal. The bootstrap program itself is resident in the cache PROM of each processor unit. The restart program is also responsible for restoring the system to a correct operational state after a power interruption. In this case the system memory is already loaded with all the programs, data, and the pre-power interrupt system configuration. The FTMP is brought up to this configuration and all the relevant data bases such as task queues for the dispatcher are re-initialized.

The fourth element of the Executive is the cache memory management. The cache RAM in each processor unit is only 8k words, while the total amount of data and programs in the system memory can be much larger. Therefore some method is necessary to page programs/data in and out of the cache. This is accomplished by an on-demand paging algorithm that reads the required page of data/program from the system memory into a cache page. The cache page is chosen on a round-robin basis.

The last element of the Executive is really a hardware enhancement function. The CAPS-6 processor does not have vectored interrupts. This function is now provided in the software. That is, all interrupts are vectored to their respective interrupt handling routines through the Executive software.



## CHAPTER 4

### RELIABILITY AND AVAILABILITY MODELS

The reliability and availability requirements of the FTMP are extremely severe due to the intended life-critical application of the multiprocessor. The desired catastrophic failure probability is less than  $10^{-9}$ . The preferred maintenance interval is of the order of 200 to 300 flight hours and the likelihood of requiring unscheduled maintenance during this period is to be at most a few percent.

The FTMP architecture to meet these stringent goals has evolved over a period of time, and reliability modeling has played an important role in it. Modeling tools were used at various stages to evaluate alternate architectures and provide feedback in deciding upon degree of redundancy for various components and revealing any other weaknesses in chosen architectures from the viewpoint of meeting reliability and availability goals. An ideal arrangement of elements so arrived at was modified slightly by the reality of packaging constraints.

The modeling effort concentrated on random hardware failures. It did not address other classes of faults such as design faults, execution errors and generic faults, either hardware or software.

The following two sections summarize the results of that effort.

#### 4.1 Survival Probability Models

The computation of survival probability of the FTMP for random failures is divided into the following three submodels:

- 1) probability of failure due to the lack of perfect coverage using a Markov process model;
- 2) probability of failure due to exhaustion of spares using a combinatorial model;
- 3) probability of failure due to BGU failures in enable mode using a combinatorial model.

In the FTMP some time is required to detect, isolate, and recover from any failure. During this time a second failure may arrive in such a place as to be catastrophic. Therefore, the coverage is imperfect. This phenomenon is most conveniently modeled using a Markov model. The probability of failure due solely to exhaustion of equipment can be computed independently using combinatorial methods. In addition to these, there is a third failure mode peculiar to the FTMP architecture, which relates to bus guardian unit (BGU) failures.

The results of the three models are shown in Figure 4. Time taken by FTMP to recover from a fault is assumed to be exponentially distributed with a mean value of 250 milliseconds. Component failures are assumed to be of constant hazard type with the mean time between failures for an LRU equal to 2,610 hours. This LRU MTBF is derived from failure rates estimated for 18 LRU subunits.

The system failure probability due to near simultaneous occurrence of multiple faults is seen to dominate the mission time of interest, that

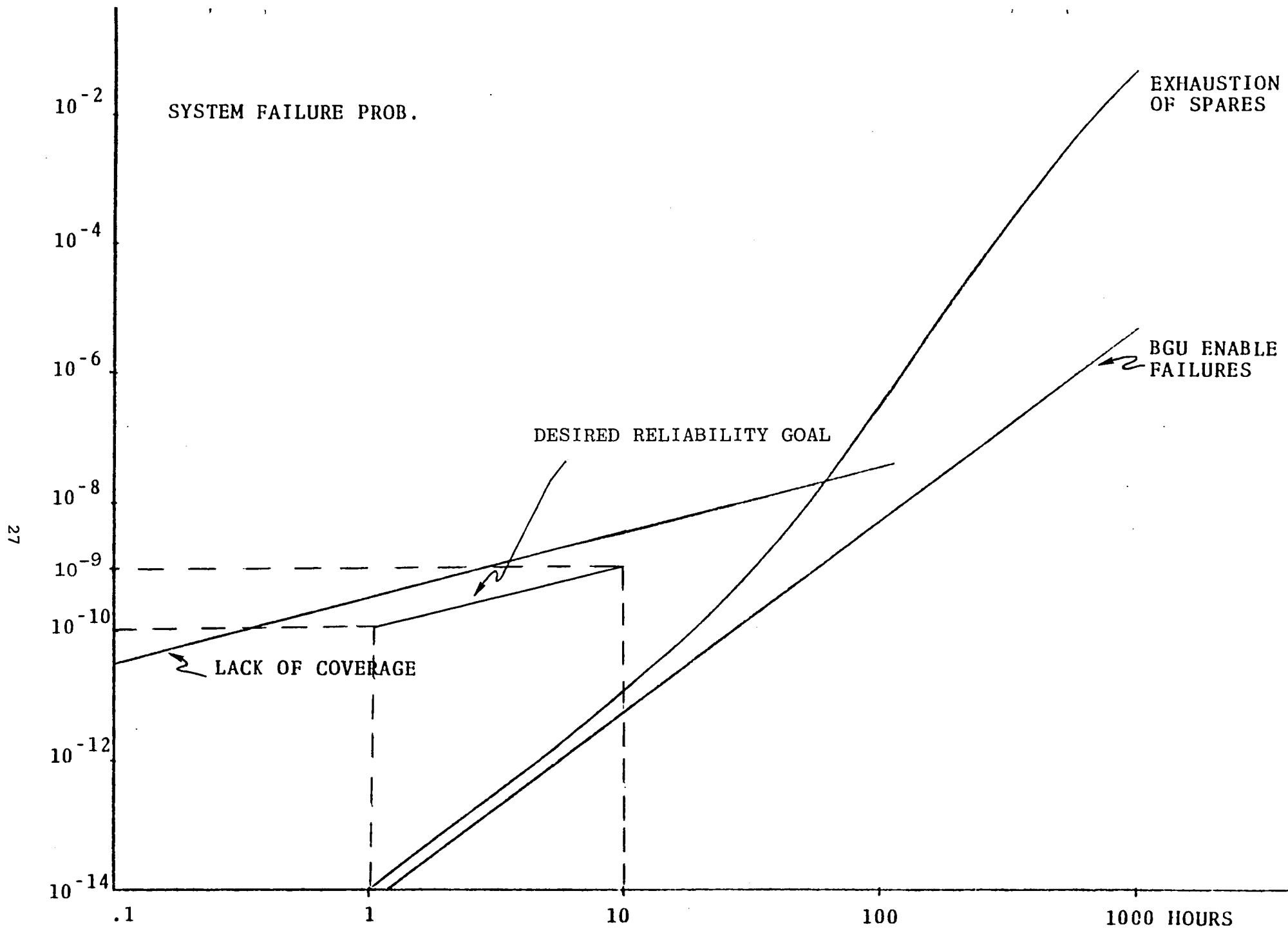


Figure 4. FTMP Probability of Failure

is, 0 to 10 hours. During this time period the likelihood that the system will fail catastrophically increases linearly with time. At one hour it is found to be  $2 \times 10^{-10}$  and at 10 hours it is  $2 \times 10^{-9}$ .

#### 4.2 Dispatch Reliability of the FTMP Computer

Availability of equipment, in general, is an important concern in the commercial air transport industry. Availability of avionics equipment, in particular, is more important for a central computer with digital "fly-by-wire" authority. It is imperative, therefore, that the dispatch reliability of the FTMP computer be commensurate with its high survival probability. A preliminary estimate of the dispatch reliability is summarized here.

Let the "dispatch minimum complement" (DMC) denote the amount of equipment (processors, memories, etc.) necessary to be operational before take-off so that flight reliability will still meet goals for a ten-hour flight. As long as the amount of operational equipment in-flight does not fall below the "critical minimum complement" (CMC), the system is said to be successful. The CMC is the amount of equipment necessary to perform all the flight-critical functions. The CMC was estimated to be two processor triads, a memory triad, a bus triad and one power supply. In the critical minimum configuration, a triad need have only two operating members to work correctly. The DMC was found to be as follows [4]:

	Dispatch Minimum Complement	Critical Minimum Complement
Processors	8	5
Memories	6	2
Buses	4	2
Power Supplies	3	1

The question to be answered at this point is, how long would it take an initially fully operational FTMP to degrade below the DMC and thereby fail the dispatch criteria? The probability of this event at time  $t$ , assuming no maintenance, is shown as a function of time in Figure 5. It is seen from this figure that there is a seven percent chance that the computer will be below the dispatch minimums if the maintenance is scheduled every 300 hours. The probability of requiring unscheduled maintenance can be reduced to just over two percent by carrying an extra LRU or by shortening the maintenance interval to 200 hours.

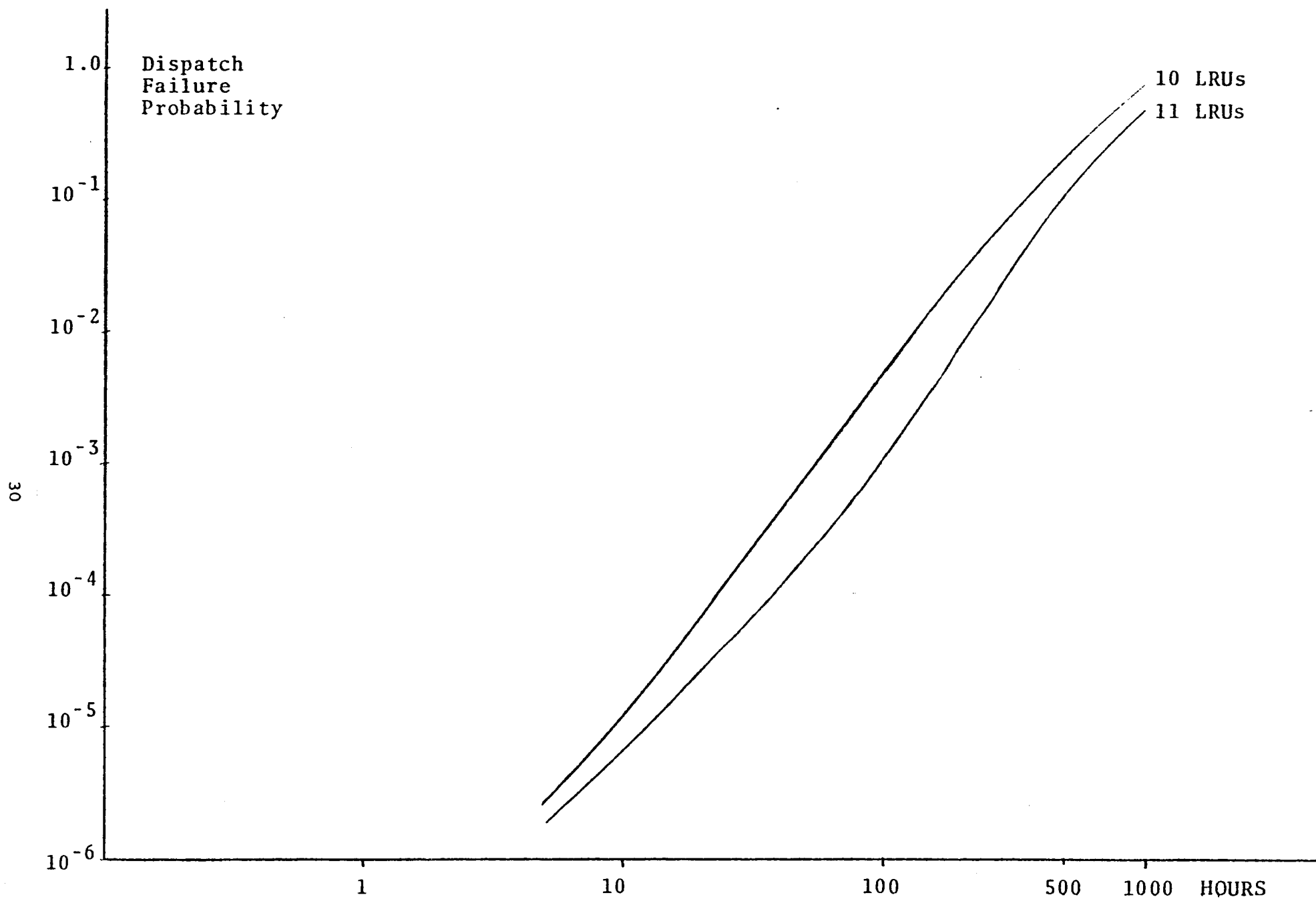


Figure 5. FTMP Dispatch Reliability

## CHAPTER 5

### THROUGHPUT AND RESPONSE TIME PERFORMANCE

The real-time, life-critical applications for which FTMP is designed not only place severe reliability requirements on the computer but also demand quick response and adequate throughput to handle all the critical tasks in a timely fashion. Performance of the multiprocessor was modeled using GPSS (a general purpose modeling language) before the computer was built. This model did not reveal any weaknesses or inadequacies in the expected performance of the machine. However, once the actual hardware was programmed to run the executive and typical flight control programs, the actual performance was somewhat short of the desired goal. As it turned out, a critical bottleneck, unforeseen of course, was left out from the model. The bottleneck became quite apparent with the hands-on experience of programming and using the system. Pleasantly, at the same time, a number of advantages of the FTMP architecture were also revealed. These two sides of the FTMP are briefly discussed next.

The initial performance goal was to have enough throughput to support execution of flight control, navigation and guidance, air data displays, and other on-board systems management tasks. It was hoped that

some of these tasks would be able to run at as high as 40Hz with other tasks executing at lower frequencies such as 20, 10, and 5 Hz. The present version of the executive supports three rate group tasks running at 25, 12.5, and 3.125 Hz. Additionally, the current application work load is less than the intended levels. This shortfall in the multiprocessor throughput performance can be traced to a single bottleneck relating to system bus access.

Presently, to access any device attached to the system such as shared memory, I/O port, error latches, configuration control registers and so on, it is necessary to load four registers. This is done by an assembly language level subroutine and takes up to 80 microseconds for each bus access. This is just the set-up time and does not include the data transaction time itself. Given the large number of bus transactions, this set-up time alone accounts for a third of all time spent by the highest frequency task dispatcher. This bottleneck was not adequately recognized at the outset.

However, there is a way to solve this problem. It is possible to microcode system bus access routines. Adequate storage is available in the CAPS-6 microcode memory to add these instructions. If this step were taken, the system bus access set-up time could be reduced from 80 to about 10 microseconds. Over 50 Hz frequencies for highest rate group tasks and proportionately higher frequencies for lower rate group tasks as well could be easily achieved.

On the plus side, the FTMP architecture allows one to achieve a relatively high level of 'useful' throughput for a given amount of raw



machine throughput. The useful throughput is defined here as the effective machine use available to the user or applications programmer. This is the same as the total raw throughput less that used by the executive software for managing and allocating system resources, contending with system malfunctions, and managing system redundancy.

The real-time malfunction management in the FTMP is done extremely efficiently in hardware. The microsecond to microsecond masking of erroneous data is performed by hardware-implemented majority voters. Hardware also provides immediate symptoms of faults by latching disagreements between voter inputs in error registers. Correlating error symptoms to determine the identity of the faulty module is done in software as are the system reconfiguration algorithms following fault identification. This division of labor between hardware and software gives FTMP a tremendous performance advantage over software-intensive fault-tolerant systems. Error detection and masking must be performed on all redundant data and this imposes a load that is proportional to the number of words as well as frequencies at which they are read. In the FTMP, a majority vote on three copies of a data word is taken in a few microseconds while the same task in software would at least take many tens of microseconds. So the load on the latter system is roughly ten to a hundred times higher. The fault identification and system reconfiguration tasks which are not time critical are performed in software. These tasks are too complex to be implemented in hardware and since they are seldom executed, they place a negligible load on the system. Note that hardware voting maintains system integrity in the face of a fault and that the config-

uration control can therefore proceed with some leisure to effect the repair.

The bit-by-bit error detection and masking done by hardware in the FTMP is possible due to the tight synchronization of parallel computational channels. The tightly synchronous architecture was originally selected and pursued for its elegance and simplicity in handling fault detection and identification. By having parallel computers perform identical operations on identical data, one can expect them to produce outputs which also match each other bit for bit. Any disagreements not only indicate a fault but also reveal the identity of the faulty unit immediately. This makes the fault detection and identification task and the error masking task much less complex. (The parallel-hybrid redundancy employed in FTMP makes fault identification slightly more complicated.) These advantages of the FTMP architecture were known at the outset. What was not appreciated fully at the time was the performance gain realized by hardware error correction and the transparency of these functions to the software. This is explained in the following.

Midway through the program it became apparent that the number of words passing through voter functions had been severely underestimated and would increase considerably due to 'simplex source congruency' requirements. This requirement may be defined as the identical distribution of data obtained from a simplex source (internal or external to the computer) to redundant processing or memory elements. In the absence of source congruency even redundant systems can suffer single point catastrophic failure. In the FTMP, source congruency is simply accom-

plished by having the processor triad write the word to shared memory (where each memory module receives three copies of the word, votes upon it, and then stores it) and reading it back from the memory. Had the voters been implemented in software and the processors only loosely synchronized, the burden of source congruency for a reasonable I/O loading would probably be unbearable.

Last, but not least, there are many features of the FTMP which make it very attractive and appealing from a user's viewpoint. The software appearance of the machine is such that the applications programmer is not aware of the hardware redundancy underlying each element. It is as if one had an extremely reliable simplex multiprocessor. From a user's viewpoint the FTMP architecture is that of a conventional multiprocessor. As a testament to this simplicity, the flight control software running on the FTMP was written by someone totally unfamiliar with the intricacies of the redundancy underlying the processor, memory, and bus structure.

Complexities of the FTMP architecture are not only transparent to the user under normal circumstances but also when there is a hardware malfunction. Fault detection, diagnosis, and recovery is such that applications programs do not need to be involved in this process. There are no program rollback requirements, no checkpoints or other constraints imposed upon the user for the sake of fault tolerance.

All of these comments apply to most of the Executive software as well. That is, with the exception of the configuration control software

the Executive is unaware of the hardware redundancy and is not involved in fault handling.

This implies that applications programs and most of the executive can be written and debugged quite early on a simplex version of the machine. In fact, all software for the FTMP was written and debugged using only a single simplex processor. No facility was even available to debug software in triplex operation.

Since hardware costs much less than software and the gap is likely to continue to become wider, any feature of the architecture that simplifies software requirements has a proportionately positive impact on the cost as well.

All of these advantages add up to make the FTMP a more 'usable' computer.

## CHAPTER 6

### TEST AND EVALUATION EXPERIMENTS

Reliability and availability models of the FTMP and their results were presented in Chapter 3 of this report. Assumptions underlying these models concerning failure and recovery rates of the FTMP were briefly mentioned there. To verify these and other assumed characteristics of the FTMP implicit in the modeling process, a series of test and evaluation experiments were undertaken. In these experiments the FTMP was subjected to numerous artificially created faults while operating routinely in a simulated aircraft environment, and its response in each case was observed and recorded. Other factors that motivated FTMP test and evaluation were expanding its validation envelope, building confidence in the system, revealing any weaknesses in the architectural concepts and/or their realization in hardware and software, and a general stressing and shaking out of the fault detection hardware and the fault identification and system configuration control software.

Towards this end a total of 21,055 pin level stuck-at class of faults were injected into one LRU of the FTMP, and in each case time to detect the fault, to isolate the fault to processor, memory, or bus, and to remove the faulty module by reconfiguring the system were recorded.

The FTMP executed the normal repertoire of system and applications software while being subjected to faults. This included the operating system, self-test programs, displays and flight control. The flight control program operated with dummy inputs obtained from the 1553 Remote Terminals since the actual aircraft simulation was not operational during fault injections. Faults were injected using a Draper-designed and built fault injection device which was controlled by a PDP-11/60 support computer. The process of fault injection was highly automated with the fault injection software maintaining closed loop control of the FTMP status through 1553 based communication with the configuration control program. The results are summarized next.

A total of 21,055 pin level faults were injected into the FTMP. Of these, 17418, or 83 percent, were detected. Of the 3,637 undetected faults at least 80 percent were estimated to be on unused gates and pins. A few of the remaining undetected faults were analyzed and found to belong to the 'don't care' class. Further analysis of undetected faults is required to arrive at a definitive detection coverage value. Identification and reconfiguration coverages, on the other hand, were found to be perfect. The system identified all detected faults correctly and successfully recovered in each case.

The total time to recover from a fault was dominated by time spent in the detection phase. Time to identify a fault and reconfigure the system was found to be deterministic and bounded, as expected. Average identification and reconfiguration times were found to be 88 and 82

milliseconds, respectively. Total recovery time averaged over all 17,418 (detected) faults was found to be 1.16 second although it would be only 549 milliseconds if BGU faults were excluded. In the absence of BGU self-test programs, which was the case here, BGU faults were solely uncovered by very low frequency routine system reconfigurations. BGU diagnostics are straightforward to code but were left out due to limited time and resources.

The FTMP reliability models described in Chapter 4 assumed an average recovery time of 250 msec. The experimentally determined average (assuming BGU diagnostics) is about twice as long. This would increase the FTMP failure probability by a factor by two over the analytically derived value.

The distribution of the total recovery time, though not exponential, was found to be favorably skewed. Over 95 percent of all faults are recovered from in a second or less. Figures 6 and 7 show this distribution on two different time scales. The 119 second maximum recovery time was for BGU faults which were detected by slow system reconfiguration. Assuming BGU diagnostic programs were coded for the FTMP, the maximum recovery times would be of the order of several seconds rather than a hundred seconds. This would make the tail of the distribution very short. An exponential distribution has an infinitely long tail. Hence, the experimentally determined fault recovery distribution for the FTMP is thought to be more favorably skewed than an exponential distribution.

The test and evaluation experiments did uncover a serious omission in the FTMP Engineering Model. The Bus Gaurdian Units were found to have no overvoltage protection. Although an overvoltage protection mechanism was specified for BGUs, it was not built into the hardware. This is discussed in more detail in Volume III. In any case, it is not difficult to add such a circuit on the BGU cards.

In addition to this hard data, a number of very important though intangible results were obtained as well. The hardware and software, in general, and the fault detection hardware and the fault identification and system configuration control software, in particular, performed extremely well under the stress of thousands of faults. In a sense the FTMP architecture, the hardware, and the software have been validated informally.

The test and evaluation experiments, their positive results, and the 100 percent availability of the FTMP during 13,000 hours of routine operation at the Draper Laboratory have all substantially bolstered confidence in the FTMP concept as well as its realization in hardware and software.



CARD: ALL

15:54:00 11/18/82

\* FAULTS: 17418

AVERAGE: 1160

MAXIMUM: 118843

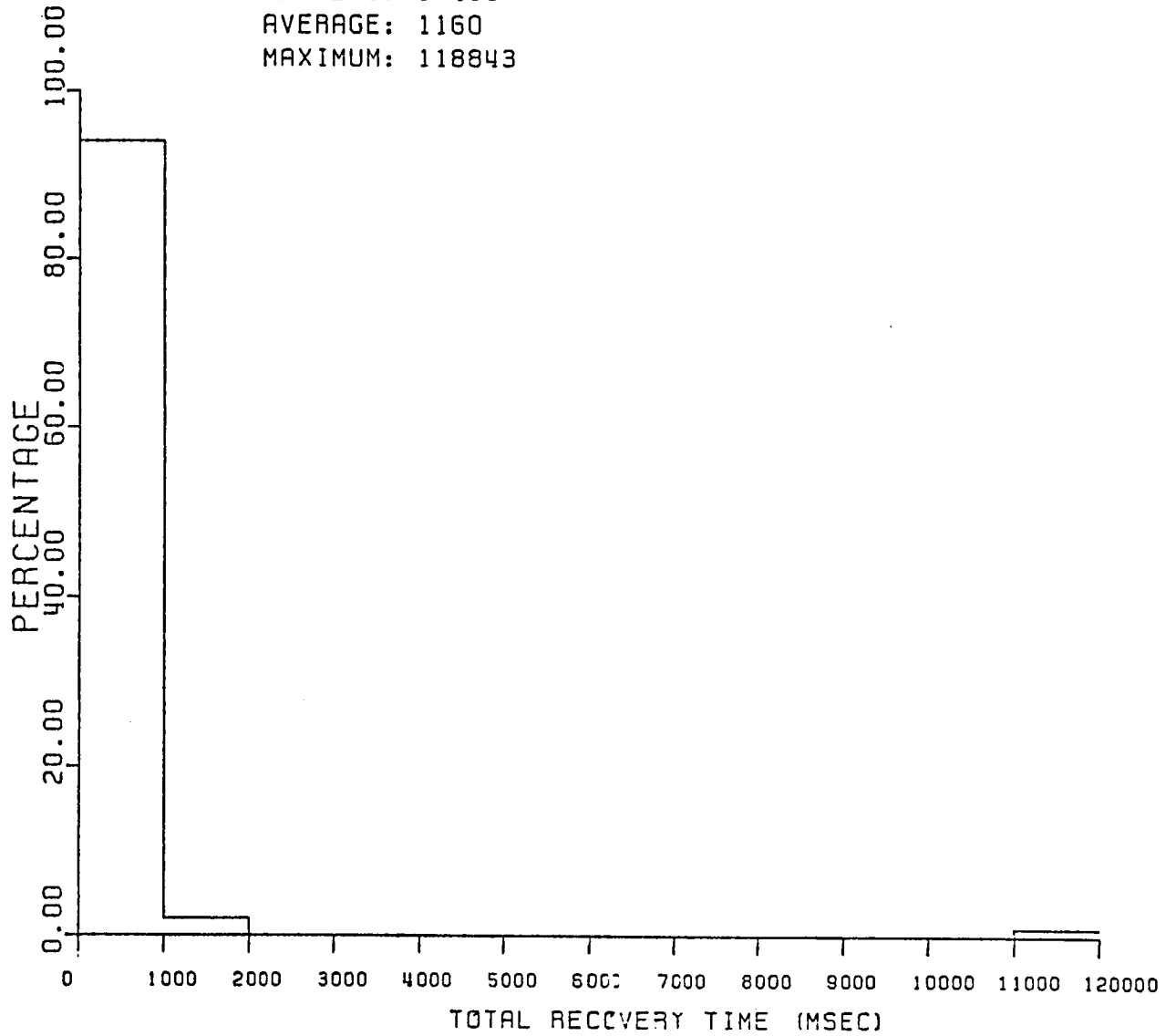


Figure 6. Frequency Distribution of Total Recovery Time  
(1 second scale)

CARD: ALL  
# FAULTS: 17418  
AVERAGE: 1160  
MAXIMUM: 118843

15:54:00 11/19/82

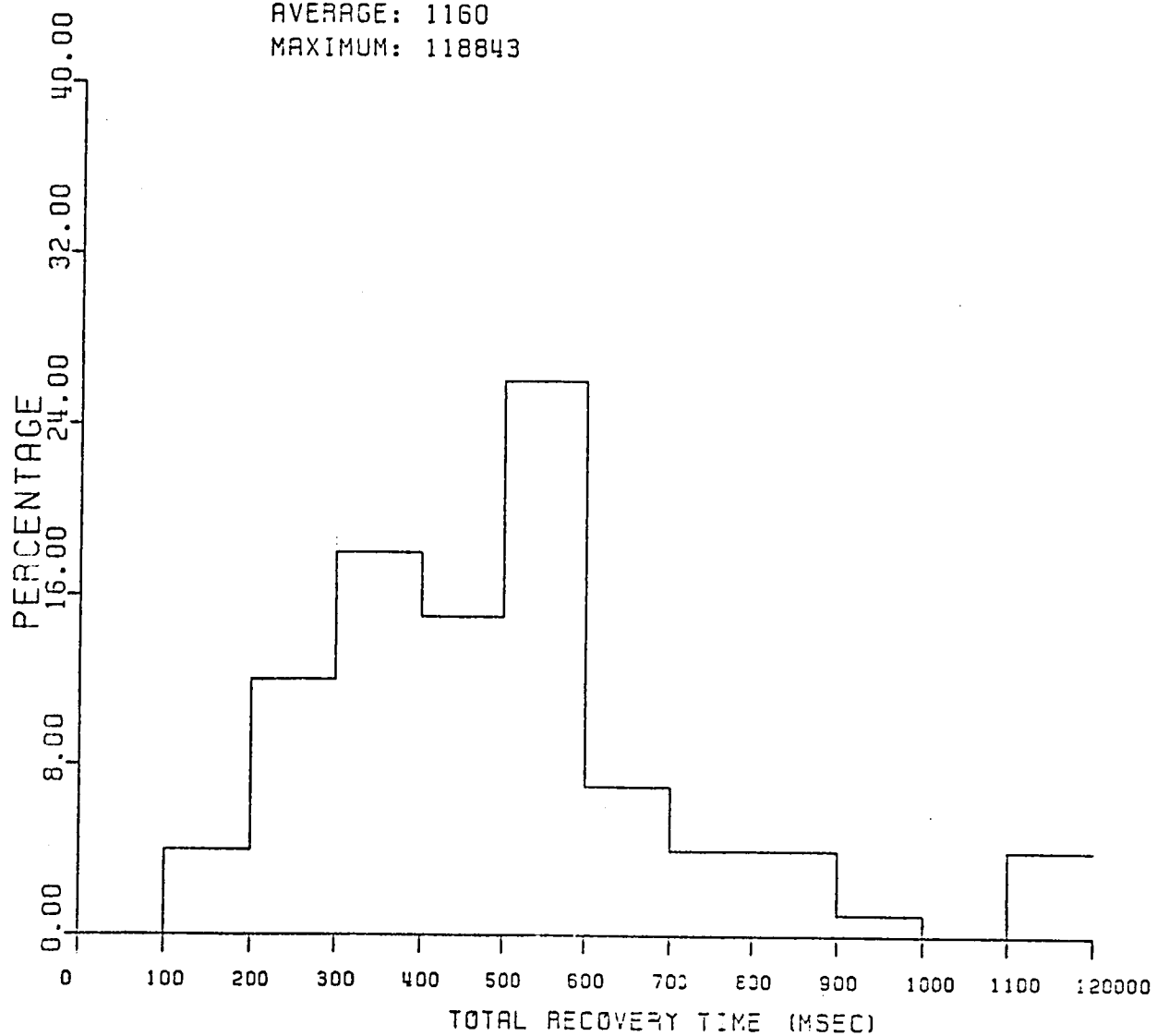


Figure 7. Frequency Distribution of Total Recovery Time  
(100 msec scale)

## CHAPTER 7

### CONCLUSIONS

The FTMP project has been a multi-year technology development. The immediate focus was the demonstration of a concept for providing generalized fault-tolerant computing for future transport aircraft. It became clear during the course of this project that such a concept is feasible. It also became clear that computing technology is a very broad and swiftly moving field and that if fault-tolerant computing is to mature it must mature in reasonable harmony with companion developments. Fault-tolerant computing must therefore provide the throughput, performance, operating systems, and software environment of mainstream computing. A fault-tolerant machine must be as usable as any state-of-the-art machine. Toward this end it must emulate or duplicate virtually all of the widely accepted and desirable features of state-of-the-art, non-fault-tolerant, general purpose machines. To suggest that fault-tolerant computing can grow without acceptance of the bulk of experience and practice from other computing technologies is arrogant, naive, and errant.

The FTMP architecture sought to provide a software environment which was equivalent to that of a general purpose homogeneous multipro-

cessor. The choice of a multiprocessor was dictated by throughput requirements. In this regard the machine architecture mimics the architecture of state-of-the-art multiprocessor configurations. The use of tightly synchronized parallel computation paths in all components of the multiprocessor created a software environment which was identical to that of a non-redundant multiprocessor of equivalent throughput. Hardware synchronization and voting techniques were provided to manage all of the moment to moment error masking and error correction independently of the software.

The operating system of the FTMP, as a result, is quite ordinary. The structure, coding, and organization of the operating system is almost a textbook example of a 'simple' multiprocessor executive, as might be covered in an undergraduate computer science course. The simplicity and efficiency of the executive were dictated not by requirements related to fault tolerance but by the time criticality and throughput requirements of the real-time nature of the FTMP applications set. That the FTMP executive is so ordinary was a significant achievement of this project.

The applications code of the FTMP similarly benefited from the software environment. All applications were coded and run as if they were being executed on a non-redundant multiprocessor. The organization of such code is unencumbered by the fault tolerance of the machine. The FTMP applications code organization is thought to be reasonably representative of that which might have been employed in real life, and was adequate to demonstrate closed loop control of a simulated aircraft in the CSDL Simulation Facility. This code probably represents a somewhat

idealized and sophomoric view of aircraft control and its complexity, but the significant factor is that it is structured solely to address the designer's perception of the real-time control problem and is not backward engineered to meet an artificial or unnatural environment created by a fault-tolerant architecture. An alternate structure could have been demonstrated with equal ease, the chosen structure being an example.

The unique piece of software, the code which is linked to this architecture and this implementation, is the configuration control program. The FTMP employs dynamic redundancy to achieve its high degree of reliability. Failed components are eventually replaced with spares and hardware voting maintains system integrity until the repair is effected. This process of reliability renewal or repair is directed by software, embodied in the configuration control program. The complexity of this configuration control was modest and well within the grasp of one programmer.

Cast in the light of a systems problem, the FTMP can be seen as problem oriented. The machine is a multiprocessor because it was felt the problem demanded a multiprocessor. A standard executive structure was chosen because it met the problem needs and was fairly well understood. Applications code was developed as an example. The total design hides fault tolerance at a lower implementation level, separate and partitioned from the system problem. The techniques and technology which enabled this partitioning proved to be tractable and successful. Over 20,000 faults injected during the test phase demonstrate this success. Newer versions of the FTMP would no doubt benefit from this experience

base and provide even greater performance and a richer I/O environment. Additionally, the technology is available to provide the same degree of transparency for alternate architectures, single processor systems, or attached processor systems, for example.

The success of the FTMP was that it shows that problem-oriented solutions can be made fault tolerant and that the fault tolerance can be hidden such that advances in operating systems, languages, instruction sets, and applications code structures can be capitalized on and used.

## CHAPTER 8

### REFERENCES

1. Goldberg, Jack, et al., "Development and Analysis of the Software Implemented Fault-Tolerance (SIFT) Computer," NASA Contractor Report CR-172146, 1983.
2. Palumbo, D.L. and Butler, R.W., "SIFT - A Preliminary Evaluation," Proceedings of the 5th Digital Avionics Systems Conference, Seattle, Washington, October, 1983.
3. Holt, A.W. and Meyers, J.M., "An Approach to the Analysis of Clock Networks," C. S. Draper Laboratory Contract Report CR-1289, NASA Contract Report CR-166028, August 1979.
4. Hopkins, A.L., Jr., Smith, T.B., III, and Lala, J.H., "FTMP - A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft," Proceedings of the IEEE, Vol. 66, No. 10, October 1978.

1. Report No. NASA CR-172286		2. Government Accession No.		3. Recipient's Catalog No.	
4. Title and Subtitle Development and Evaluation of a Fault-Tolerant Multiprocessor (FTMP) Computer, Volume IV, FTMP Executive Summary				5. Report Date February, 1984	
				6. Performing Organization Code	
7. Author(s) T. Basil Smith, III and Jaynarayan H. Lala				8. Performing Organization Report No. R-1603	
9. Performing Organization Name and Address The C. S. Draper Laboratory, Inc. Cambridge, Massachusetts 02139				10. Work Unit No.	
				11. Contract or Grant No. NAS1-15336	
				13. Type of Report and Period Covered Contractor Report	
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546				14. Sponsoring Agency Code	
15. Supplementary Notes Langley Technical Monitor: Charles Meissner, Jr.					
16. Abstract  This report is the final volume of a multi-volume report on the Fault-Tolerant Multiprocessor (FTMP) project. It is an executive and technical summary of the project.  The FTMP architecture is a high reliability computer concept modeled after a homogeneous multiprocessor architecture. Elements of the FTMP are operated in tight synchronism with one another and hardware fault-detection and fault-masking is provided which is transparent to the software. Operating system design and user software design is thus greatly simplified. Performance of the FTMP is also comparable to that of a simplex equivalent due to the efficiency of fault handling hardware.  The FTMP project constructed an engineering module of the FTMP, programmed the machine and extensively tested the architecture through fault injection and other stress testing. This testing confirmed the soundness of the FTMP concepts.					
17. Key Words Suggested by Author Fault-Tolerant Distributed Processing Synchronous Distributed Executive Reconfigurable			18. Distribution Statement  Unclassified - Unlimited  Subject Category 62		
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 52	22. Price A04		





