UNIVERSITY OF SOUTHERN COLORADO
Pueblo, Colorado 81001
2200 North Bonforte Boulevard
SCHOOL OF APPLIED SCIENCE AND ENGINEERING TECHNOLOGY

FINAL TECHNICAL REPORT

NASA Grant NAG 2-2

"Investigation of Distributed Microcomputer Methods for High Authority Auto Flight Systems"

(NASA-CR-163615) RAMP: A FAULT TOLERANT DISTRIBUTED MICROCOMPUTER STRUCTURE FOR AIRCRAFT NAVIGATION AND CONTROL Final Technical Report, 1 Oct. 1979 - 1 Oct. 1980 (University of Southern Colorado, Pueblo.) G3/08

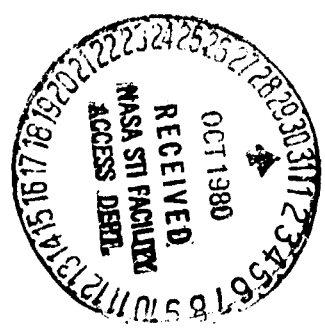
N80-33415

Unclas 28951

Principal Investigator: William R. Dunn

Period Covered by Report: October 1, 1979 - October 1, 1980

Institution: University of Southern Colorado
2200 N. Bonforte Boulevard
Pueblo, Colorado 81001



RAMP - A FAULT TOLERANT DISTRIBUTED MICROCOMPUTER STRUCTURE FOR AIRCRAFT NAVIGATION AND CONTROL

W. R. Dunn

September, 1980

RAMP - A FAULT TOLERANT DISTRIBUTED MICROCOMPUTER STRUCTURE FOR AIRCRAFT NAVIGATION AND CONTROL

W. R. Dunn

ABSTRACT

Based on recent, continuing advances in semiconductor technology, classical N-Modular Redundancy appears to be a viable approach for designing ultra-safe flight control systems in the near future.

RAMP consists of distributed sets of parallel computers partitioned on the basis of software and packaging constraints.

To minimize hardware and software complexity, the processors operate asynchronously. It is shown that through the design of asymptotically stable control laws, data errors due to

---

W. R. Dunn is with the University of Southern Colorado, in Pueblo, Colorado.

the asynchronism can be minimized. It is further shown that by designing control laws with this property and making minor hardware modifications to the RAMP modules, the system becomes inherently tolerant to intermittent faults.

A laboratory version of RAMP has been constructed and is described in the paper along with the experimental results obtained to date.

I. INTRODUCTION

The authors are currently engaged in an avionics systems research program seeking design methodologies for realizing future high authority autoflight control systems. This effort is broad in scope and includes control law development, experimentation with sensors and actuators, and the investigation of distributed microcomputer architectures. The latter, in particular the redundant asynchronous microprocessor (RAMP) structure is currently being investigated and is the subject of this paper. Before describing RAMP in detail, it is useful to first describe the general structure and the key elements that collectively make it a new and different approach in implementing digital avionics systems. This is done in the following section..

II. RAMP - AN OVERVIEW

Given the requirement for increased aircraft operational complexities by the 1990's (real time air traffic control, fuel minimization, autoland, etc.) it is clear that the digital computer will become the dominant component in avionics systems implementations. This need for these more complex operations is at first sight an invitation for more complex avionics systems.

It has been the authors' core philosophy to research concepts and methodologies that maximize digital avionics system simplicity without compromise of future operational requirements.

One result of such work is the compact Total Automatic Flight Control System (TAFCOs) algorithm (described in References 1 and 2) applicable to highly non-linear systems.

This same thinking has led the authors to depart from the conventional (e.g. data processing industry) approach of employing digital systems in an avionics environment and to instead investigate RAMP. Before discussing the extent of this departure, it is useful to describe the overall approach being taken as follows.

As seen in Figure 1, RAMP comprises a connected network of microcomputers which has as input command and sensor information and which generates servo information to drive actuators, thrust linkages, etc. Each microcomputer in the network moreover performs a specific, well-defined input/output function such as sensor data conversion and preprocessing, execution of flight control algorithms, servo-positioning, etc. Although consisting of digital system elements, the network is intentionally structured and operated as a conventional analog control system: sensor data are input and servo data are output as a function of command mode inputs.

Correspondingly, the individual microcomputers "appear" as analog-like elements each having a predetermined set of (command) selectable input/output characteristics. The microcomputers at each node are moreover autonomous:

1) Each has its own clock, i.e., the network has no master or central timing reference with the result that the microcomputers in the network are not synchronized.

2) The computers are electrically isolated to the extent that hardware failures do not propogate from a given, failed microcomputer.

3) Interconnect between microcomputers is confined to data traffic such that a given microcomputer simply "broadcasts" messages to one or more of the other microcomputers in the network, the received messages being authomatically buffered and used by the receiving microcomputers.

Finally, to insure flight safety in the event of hardware failures, redundant microcomputers are employed at each node as depicted in Figure 2 such that failures in a transmitting microcomputer (or the associated data transmission path) will be recognized by a receiving microcomputer in that former's data will be disparate with that of the remaining "good" microcomputers. Hence the receiving computer (which normally will also be replicated) can select the correct data.

Now this approach differs from the conventional, data processing approach (e.g. References 3 and 4) to digital systems implementation in several respects:

1) The RAMP network and modules are structured to perform a limited range of specific, analog-like functions; this is to be contrasted with the use of a network of general purpose computers.

2) The RAMP microcomputer modules are autonomous; the system does not employ an operating system, global executive software or complex intermodule communications software.

3) The modules are not synchronized; i.e., central timing hardware and software are not employed.

4) Finally, tolerance to hardware failures is achieved by static redundancy (Reference 5), i.e. results of a failed microcomputer are simply rejected; this is done in lieu of dynamic redundancy (Reference 5) wherein the distributed computer system performs real time fault detection and reconfiguration of the system.

These foregoing differences are summarized in the table which follows.

TABLE 2-1: RAMP VERSUS CONVENTIONAL DIGITAL SYSTEMS IMPLEMENTATION

RAMP	CONVENTIONAL
• Fixed function, Analog-like Implementation	• General purpose Data processing Implementation
• No operating system or Global Executive. Limited module-to-module communication software	• Centralized Operating System, Global Executive Software, Intermodule Communication Software
• Asynchronous	• Synchronized
• Static Redundancy	• Dynamic Redundancy- Requires Real Time Fault Identification and Reconfiguration
• Low Complexity	• High Complexity

III. THE RAMP NETWORK STRUCTURE AND OPERATION

The previous section has described the general architecture employed in RAMP.

Figure 3 shows a specific network structure based on the RAMP concept which is being investigated by the authors in the current flight research program. As shown, computers are distributed into five sets of triplets (e.g. N = 3).

There are three reasons for partitioning the computation given in their order of importance as follows:

1) Figure 3 shows that the system computation process divides into three groups: sensor/command processing, the control algorithm and servo processing. In the airborne application, each of these groups would be expected to physically reside in different parts of the aircraft system. (I.e. the partitioning into distinct computational sites is actually governed by packaging and cabling constraints.)

2) As will be discussed in more detail in Section V, the reliance on static redundancy for flight safety presumes that each computer be fully self-tested before flight. The distributing of the computation

load into several, smaller microcomputers facilitates reasonably short yet comprehensive preflight tests.

3) The final motivating factor in distributing the computation is that hardware boundaries can be set on the system software. E.g., as depicted in Figure 3 software is partitioned into corresponding hardware modules with the result that software can be concurrently developed and, more important, modified without effecting the remainder of the software in the system.

The network operates in the following manner:

1) Microcomputers in the set $\{M_{1j}\}$ input data from a triply redundant set of sensors* and redundant (crew generated) commands, the latter consisting of trajectory waypoints and flight modes. The microcomputers process the sensor data (correct values being selected from the redundant inputs) and derive estimates of the aircraft states. Correspondingly, correct waypoints and mode commands are selected. (The exact mechanics of this

* It is also possible in this type of application to employ an analytically redundant sensor mix as described in References 6 and 7.

selection process are discussed in the next section.)
Each microcomputer in the set $\{M_{1j}\}$ transmits results to all microcomputers in the set $\{M_{2j}\}$.

2) As noted, each microcomputer in the set $\{M_{2j}\}$ has the results of computations made by all the microcomputers in the set $\{M_{1j}\}$. From this set of inputs, each microcomputer in $\{M_{2j}\}$ selects by voting, the correct state estimates, command modes and commanded waypoints and computes the trajectory to be flown by the aircraft. These results are then transmitted to microcomputers $\{M_{3j}\}$.

3) This process continues in a similar manner, microcomputers $\{M_{3j}\}$ selecting results, computing the control (using the commanded and actual trajectory states) and transmitting these to $\{M_{4j}\}$ and so on.

4) The final set of microcomputers $\{M_{5j}\}$ drive redundant actuators. (A current trend in ultra-safe actuator research is not to use redundant actuators but instead single actuators with redundant hydraulic valving systems. These devices operate on redundant electrical inputs. See for example Reference 8.)

It is seen in this description that information flows from left to right in the figure. Each microcomputer moreover

operates recursively in what will be referred to subsequently as computation "frames". In the current experiments, the period T of these frames is constant (T = 50 ms. in the present laboratory version of RAMP.) (Section VIII discusses this aspect of timing in more detail.) As a result, the computation process of the system as a whole is a combination of parallel processing (e.g. the microcomputer set $\{M_{2j}\}$) and pipeline processing (Figure 4).

A key feature of the RAMP structure as depicted in Figure 3 is that each microcomputer in the network is designed to be autonomous. This autonomy is achieved by:

1) Letting each microcomputer have its own clock such that as a whole, the system is not dependent upon a central timing reference.

2) Having each microcomputer employ a set of buffer memories that are independently written by other microcomputers in the network. The configuration of these memories and the interconnecting communication paths are depicted in Figure 5. The essential features of this detailed structure are:

   a) Each microcomputer in the set $\{M_{1j}\}$ writes data into all microcomputers in the set $\{M_{i+1,j}\}$. Data are simply "forced" into these memories during the former's computation

frame. (I.e. there is no "handshaking" or other coordinating of the data transfer between microcomputers.)

b) Each microcomputer in the set $\{M_{i+1,j}\}$ fetches data from the buffer memories when needed during its computation frame. (The means of avoiding read/write conflicts is discussed in Section VIII.)

c) Electrical isolation (e.g. high impedance and/or optoisolation) is employed as shown between computers.

The purpose of providing this autonomy is to prevent propagation of hardware failures in any given microcomputer or in any given transmission path to the other hardware elements in the system. As a result, for the system of Figure 3, up to one microcomputer in each triplet can undergo a hardware failure without affecting the hardware integrity of the remaining pair of microcomputers in the triplet.

Now the handling of such failures has been somewhat vaguely referred to in the foregoing (e.g. selection of "correct" data, "voting", etc.). This topic is however central to the RAMP

concept and discussed at length in the remainder of the paper. Before embarking upon this however, it is important to note that the employment of asynchronous microcomputer modules places an important constraint on the design of the control laws implemented in RAMP. This is discussed in the next Section.

IV. FLIGHT CONTROL WITH PARALLEL, ASYNCHRONOUS COMPUTERS

A characterising feature of RAMP is that each microcomputer in the network has its own clock. Hence as a result of variations in (oscillator) components from microcomputer to microcomputer, operation of the system as a whole is asynchronous.

It has already been stated that this asynchronism places an important constraint on the design of the control laws hosted by the network. This is discussed in the following.

With little sacrifice of generality, consider a set of parallel computers employed in the control of a plant as illustrated in Figure 6. In the current control law work, aircraft control u is obtained from the plant states y using the following recurrence equation:

$$u(i + 1)T = A\,u\,(iT) + By \quad (i + 1)T \qquad (4\text{-}1)$$

where T is the period of a computation frame (See Section III and Figure 4). Referring back to Figure 6, it will be assumed that only computer C_I is selected for control of the plant and that its computation frame has period T. Correspondingly, it will be assumed that the remaining computers have different and

unequal computation frame periods such that only computer C_I is selected. (Some generality is lost here since the select process is a function of the output of all the computers. What follows therefore is a necessary condition for the properties of Equation 4-1).

Next consider the case in which computer C_J has a shorter computation frame period by an amount δT such that after execution of n computation frames by computer C_I, computer C_J has executed exactly (n + 1) computation frames, i.e.,

$$n \delta T = T \tag{4-2}$$

Looked at another way, computer C_J would execute exactly one more computation frame than C_I every nT seconds.

It is shown in Appendix A that due to the timing error, computer C_J will generate an error having the following recurrence relation:

$$u(j + 1)nT = A^{n+1} u(jnT) + A^n (A - I) u(jnT) + A^n By(jnT) \tag{4-3}$$

It is shown (also in Appendix A) that to guarantee convergence of this error, it is necessary that the control law being computed (by all the computers) be asymptotically stable.

(This result can in fact be obtained in a more general manner by arguing that the outputs of the deselected computers are uncontrollable such that to insure convergence of the error, the control law being executed must be asymptotically stable.)

The effects of the timing error can be illustrated by considering the example of Figure 7 which depicts a simple second order system employing a stable, metastable and unstable control as shown. The system response (to a unit impulse) is the same for each controller and is shown in Figure 8a. Figure 8b shows time histories of the errors that would exist in a deselected computer having a 10% timing* error (i.e., $\frac{\delta T}{T} = 0.1$.).

To generalize these examples, it is clear that by employing an unstable or metastable control algorithm, a deselected microcomputer can, as a result of timing errors alone, accumulate excessive data errors or possibly be incapacitated (e.g. as a result of overflows).

Consequently, in RAMP (more specifically in the TAFCOs algorithm discussed briefly in Section II), the control laws are designed to be asymptotically stable. This design policy, which

*In the practical application such errors are typically much smaller, e.g., .01% to .001%.

permits control of data errors due to timing, has an equally important role in RAMP's tolerance to intermittent faults.

Before discussing this aspect, the subject of faults and fault tolerance is first explored in the following sections.

V. FAULT TOLERANCE, MASKING AND IDENTIFICATION

In the previous section, the RAMP concept has been illustrated using triplicated computer sets such that each computer (with the exception of those interfaced to the sensor inputs) has three buffer memories.

In terms of the general concept of RAMP (Section II) each computer will have N (= 2n + 1) memories containing data generated from N redundant computers. (In general, more than one redundant set may input to a given computer in which case there will be as many mrmories as there are computers. This case becomes obviously included in the discussion that follows.)

Each of the N memories will in turn contain a total of K (real number) data values placed there by the corresponding transmitting computer. This is illustrated in Figure 9 where for example D_{jk} corresponds to the jth data value in the kth buffer memory.

Now where there are no timing errors (i.e. differing clock rates in the transmitting computers) or faults (in the transmitting computers, data transmission paths, and/or the buffer memories), the data in a given row of Figure 9 will be

identical. However, where timing errors or faults exist, these data values can differ.

The basic approach in RAMP is to use these data in providing fault tolerant performance of the system. Before discussing the specific strategies employed by RAMP, it is necessary to consider the nature of the faults themselves.

First, it is assumed that the faults result from random hardware failures.* I.e., common mode or "generic" sources of failure arising from design mistakes, external effects due to heating or EMI, fabrication mistakes, etc. have all been accounted for in the system design and development process.

Second, the faults being considered may be either permanent or intermittent (Reference 5). (Intermittent faults are a crucial issue in the "real world" implementation of systems such as RAMP and are discussed in Section VII.)

Third, it is assumed that the hardware faults experienced in a given module are confined to the module and do not propagate (i.e. the microcomputer modules are fully autonomous). Note that this latter assumption is readily confirmed in practice by simply enumerating the input/output hardware failure modes for

*For the electronic components of the type employed in microelectronic systems such as RAMP, a constant failure rate model is employed. See for example, Reference 9.

each microcomputer module and verifying that none effects the function of the remaining microcomputers in the network.

Given these types of faults, RAMP employs two basic methods for realizing fault tolerant performance:

1) Mid-Value Select

Due to variations in the clock rates in the individual microcomputers the data values corresponding to the rows of Figure 9 will include errors due to timing. I.e., the data values will be dispersed along the real line as shown in the example of Figure 10.

The basic strategy employed by all the microcomputers in RAMP is to select, as the correct data value, the mid-value of these dispersed operands (e.g. D_{j2} in Figure10).* More sharply stated, given the set of N (= 2n + 1) distinct values (some subsets of which may be equal) in a given row of Figure 9, a value will be selected such that there are exactly n distinct values greater than or equal to it and exactly n distinct values less than or equal to it. The postulate here is that this mid-value will approximate, with a known error, the correct value

*Mid-values for the sensor and command values are selected as well.

in spite of any kind of failure in up to n of the transmitting microcomputers. (A detailed argument behind this is reviewed in Appendix B.)

The net result is that for a given set of data values, only one value is selected and 2n are ignored. I.e., the selection process "masks" the results of any failed units and the remainder of the "good" units.

2) In-Flight Fault Identification

The fundamental strategy behind RAMP is to rely on the known reliability of redundant hardware to safely execute flight-critical operations. Specifically, RAMP does not employ any software other than the mid-value select to compensate for faults experienced in flight.

Faults experienced during flight are however identified but for two different purposes:

a) To alert the flight crew that failure(s) has occurred. Here, the decision to continue or to abort flight critical operations with the degraded system is a crew decision.

b) To provide a flag and record for system

maintenance of both permanent and intermittent faults experienced during the flight operations.

The basic approach to in-flight fault identification is simple threshold detection which is described as follows.

Referring back to Figure 10, it has already been noted that some of the data values will be dispersed as a result of timing errors in the computer clocks. Based on analysis and flight test results, an expected maximum range can be determined for the dispersion of these data. Correspondingly, maximum ranges can be established for each redundant sensor and redundant command* inputs. During flight, the actual range of each data value is determined and compared to the corresponding, predetermined maximum range. Where the actual range exceeds that maximum, the flight crew and maintenance recorder are signalled.

To this point in the discussion, little has been said about system reliability which is of course the major concern with RAMP. Reliability modelling and

* Digital command inputs must of course be identical.

techniques for deriving reliability estimates are well covered in the literature (e.g. References 9 and 10) and will not be discussed in this paper with one exception. This is the fact that to estimate the reliability of RAMP for flights of given duration (e.g. ten hours maximum) it is necessary to determine, at the outset of flight, the existence and nature of any faults residing in the system.

As a result, the RAMP system must be tested prior to flight. This is discussed in the next Section.

VI. RAMP PREFLIGHT TEST

It has been pointed out in the previous section that in order to derive valid estimates for the in-flight reliability of a given implementation of RAMP, a preflight test is required. Ideally, one seeks to design a test that will verify that no faults exist. Moreover, the time required to perform the test must be short (in the RAMP application the preflight test time should be under ten minutes). Now it is well known in digital systems practice that the problem of fully testing LSI components of the type employed in RAMP is intractable to the extent that the required test times have astronomical dimensions. Instead, one settles for testing to a given "coverage", defined as a percentage of all possible faults that can be uncovered with a given test method. (See for example Reference 11).

Before proceeding with the discussion of this subject it is appropriate to point out that the problem of designing tests to a known coverage and more important proving that such coverage has been obtained is at present an open research question. This applies not only to RAMP but in fact to LSI-based systems at large. What follows therefore is a description of

the authors' approach to this problem in the context of designing and verifying the preflight test for RAMP.

Before discussing the approach to fault testing, it is necessary to look more closely at the nature of the faults themselves. Recalling that the microcomputer modules in RAMP do not propagate their failures, concern is confined to the ways a given microcomputer module can fail.

To do this, a "top-down" or fault tree (Reference 9) approach is currently being investigated. This approach is explained by the following example.

Consider a single microcomputer module executing the control algorithm given by Equation 6-1. This is illustrated in Figure 11 a, where the control algorithm consists of the repetitive execution of the arithmetic replacement statement,

$$u = Au + By. \qquad (6\text{-}1)$$

It has already been pointed out that each RAMP module operates recursively, inputting data, performing mid-value selection, performing threshold detection (to flag inflight errors), executing the control algorithm, and outputting results. This process is illustrated by the program shown in Figure 11 b in which all the foregoing functions with the exception of the control

algorithm are performed by procedure (i.e. subroutine) calls. The TIMEOUT procedure is invoked as a part of this code in order to establish the computation frame time T (Section II). Hence, the program of Figure 11b is endlessly executed every T seconds.

To begin the discussion of faults from a "top-down" viewpoint note first that the microcomputer module executing the algorithm of Figure 11 has just one failure mode: it will not generate a correct y(t) for some u(t). It is the postulate that in RAMP modules this single failure mode can arise from only three kinds of faults:

1) Faults that cause data alteration.

2) Faults that result in improper execution of the designed code.

3) Faults that cause excessive timing errors.

Considering the first of these, it is further postulated that under the assumption that the code is being correctly executed, the microcomputer module data can be altered only as a result of one or more of the following:

a) Faults in the path(s) connecting the input data y(t) to the CPU (i.e. accumulator)

b) Faults in the path(s) connecting the CPU (i.e. accumulator) to the output of the module.

c) Faults in the constant data (e.g. A and B in Figure 11) store.

d) Faults in the read/write data (e.g. u) temporary store.

e) Faults in the CPU that correctly evaluate arithmetic and/or logical expressions for some operand pairs but not others. In the example at hand such faults would include those that give invalid results for u for some values of y but not others.

Next, in considering improper program execution, we first note that in RAMP the instructions are fixed, i.e. each module is preprogrammed to repetitively execute a predetermined, exact sequence of instructions. The instructions moreover fall into two categories:

a) Those that are data-sensitive. I.e. the path taken in the program flow is determined by the data being processed. (Data-sensitive instructions are illustrated in Figure 11c.)

b) Those that are data-insensitive (e.g. the procedure

CALLS, and JUMP TO TOP instructions in Figure 11b).

Correspondingly, improper program execution can be traced to:

a) Faults in the CPU that correctly execute data-sensitive instructions for some data values but not others.

b) Faults in machine code store, instruction counters, address decoders, read/write stack, etc.

Finally, excessive timing errors can be traced to faults in the timing reference (e.g. crystal), oscillator amplifier failures, etc.

The discussion of the example is summarized in Figure 12 which the authors postulate embraces all the possible ways a RAMP module can fail.

The objective of the preflight test therefore is to test for the faults depicted at the base of the Figure. Since the preflight test is initiated immediately after system start-up (Section VIII), the system will initially "appear" synchronized. (For example, if the clock frequencies of all the microcomputers are within a readily achievable .001% of one another, a system with a 50 ms computation frame rate will "appear" to be synchronized for some 80 minutes after system startup.) Hence, the preflight

test will be carried out during this period of apparent synchronization.

In the current work, the plan is to carry out the preflight test for each module in two steps:

1) A self-test of each module would be initially performed to:

   a) test the CPU for faults associated with evaluation of arithmetic and logical operations and the testing for faults in data sensitive instructions.

   b) test the read/write store.

This test would be carried out concurrently by all the microcomputers.

2) An input/output test. To perform this test, the RAMP structure would be supplemented with two microcomputer modules, one that inputs test patterns to the sensor/command microcomputers and a second to monitor outputs of the servo microcomputers.*

   Given these additional microcomputers, and referring back to the basic structure of RAMP as depicted in Figure 3, the network structure under test would have each set of redundant microcomputers

* This latter computer would also be employed for in-flight fault identification.

receiving test inputs from at least one microcomputer and each set would likewise be monitored by at least one microcomputer. In executing the input/output test, each microcomputer in a redundant set would accordingly:

a) Generate checkwords signalling success or failure of the self-test.

b) Receive and verify input test patterns validating data input paths.

c) Transmit all the contents of its constant store.

d) Receive a sequence of data inputs that test all paths in the instruction code and transmit the results to a monitoring computer (i.e. to the next set of redundant computers or the microcomputer monitoring the servo microcomputers' outputs).

e) Transmit its value of time.

f) Transmit test patterns to check the downstream computers inputs.

g) Generate data patterns that will exercise all the instruction paths in the downstream computers.

Recalling that the system appears synchronized during this test, each microcomputer monitoring a

redundant set must receive identical data from each microcomputer in that set. Success of the preflight test is accordingly achieved by a "bit-by-bit" match of these data for the full duration of the test.

Returning to the subject of coverage, the foregoing test covers all the faults depicted in Figure 12. Given the validity of the postulate that these represent all possible faults, the actual coverage that can be attained by this testing procedure is governed by:

1) The coverage that can be achieved in the CPU and read/write store self-tests.
2) The probability of compensating faults, e.g. a failed CPU test routine that signals a "success" checkword.

The foregoing preflight test approach is currently being investigated with the current laboratory version of RAMP (Section VIII). Results will be presented in the future.

VII. INTERMITTENT FAULT TOLERANCE

In Section V it was noted that faults can be classified as being either permanent or intermittent.

Of these two types of faults, the permanent faults are best understood: the bulk of the available electronic component failure rate data and failure mechanism models are based on permanent faults, they are more readily uncovered by testing, and their effects on system performance easily determined.

Intermittent faults on the other hand are less understood yet in the "real world" application account for the majority of failures in computer systems (some 80% to 90% as estimated in Reference 12).

For the purposes of the discussion that follows, these intermittent faults can be inclusively classified as being either permanent or transient. In the context of the microcomputer systems of the type that would be employed in RAMP such permanent faults are those in which an intermittent fault has altered normal program flow to the extent that the microcomputer enters an infinite loop, enters a halt state, etc. I.e., the microcomputer system is "crashed". With the transient faults however, normal program flow is resumed after disappearance of the

intermittent fault but usually with the consequence that the data being processed have been altered. These two consequences of intermittent faults are depicted in Figure 13 which also shows how these relate to the faults discussed in the previous Section.

Note in the figure that intermittent timing faults result in transient system faults. This particular type of fault has been indirectly discussed in Section IV, i.e., by virtue of use of an asymptotically stable control law, the errors introduced by an intermittent timing fault will, in time, converge to zero. Now it should be clear from Figure 13 that this will also be the case for all the other intermittent faults that produce transient system faults. I.e. the RAMP microcomputer module is inherently fault tolerant to these types of intermittent faults.

Permanent system faults, i.e., those resulting from broken program flow, can be accommodated by the simple expedient of using external hardware* to detect a loss in program flow and force the microcomputer into a restart or retry (Reference 12) operation. This can be done in a (probably endless) variety of ways. Hence, the following describes one such approach mainly to illustrate the simplicity and effectiveness of the basic idea.

---

* As will be seen, such external hardware is already available in current microcomputer components.

First, note that to initiate operation of a microcomputer, the device is normally supplied with an external reset signal (or pulse) that results in the start of the computer program at some predesignated location (i.e. address). Since each RAMP module will be expected to perform different functions (e.g. the preflight test, processing of the flight algorithm, etc.) the module will receive (i.e. have deposited in buffer memory) not only data, but a command word that when fetched will indicate the specific function the module should be performing at that time. Hence, to start system operation, a RAMP module will be given a hardware reset and its program will "look" at the command word (which following the previous section, will signal the module to begin the preflight test). At the completion of the preflight test, the microcomputer module would receive a different command word to signal start of processing of the control algorithm.

Given this, a retry or restart due to broken program flow can be achieved using the hardware structure shown in Figure 14a. The idea here is to supplement the program employed in the RAMP module with output instructions such that under normal program flow, a pulse is output during each computation frame.

I.e. in the unfailed condition, the microcomputer will generate a pulse train of period equal to the computation frame period T_1. Each of these pulses in turn will retrigger a monostable multivibrator as shown in the figure. As long as this pulse train is maintained, the multivibrator will output a constant level (OFF) as shown in Figure 14b. In the event of broken program flow, it can be expected that this pulse train will cease. Now the monostable multivibrator is adjusted such that at a time T_2 ($T_2 > T_1$) after the last pulse (Figure 14b) its output level shifts (ON) as shown. This output in turn is used to gate an astable multivibrator (Figure 14a) which in this gated condition generates a train of pulses having period T_3 ($> T_1$) as shown in Figure 14b. The function of each of these pulses is to reset the microcomputer such that it begins processing at the reset address, fetches the command word and recognizing that it should be executing the control algorithm begins processing the algorithm from some predesignated initial state (e.g. $u = 0$ in Equation 6-1). Successful resumption of computation restores the pulse train driving the monostable multivibrator causing its output to return to the OFF level. The astable multivibrator in turn is gated off as shown.

Finally, it can be expected that the algorithm will initially be computing incorrect states following the retry. However, based on its convergence properties, it will, in time, reach the correct state.* Hence, the above reset circuitry plus the convergence properties of the control law provide a RAMP module with the inherent high tolerance to intermittent faults enjoyed by most analog systems.

---

* I.e., as a result of the intermittent failure the module will be deselected and hence be running "open" loop.

VIII. LABORATORY IMPLEMENTATION OF RAMP

In order to verify its underlying concepts, RAMP has been implemented in the laboratory as shown in Figure 15. Here, as shown a pair of redundant microcomputer triplets are used to generate trajectory commands and provide trajectory and attitude control of a rotorcraft plant mechanized on an analog computer. (Only the pitch plane has been mechanized in the laboratory; a full, six axis version of RAMP is currently under development.)

The basic microcomputer module employed in the laboratory is shown in Figure 16 along with its basic specifications. This structure was selected by the authors as being representative of what in the 1985-1990 design period may be available on a single chip or, at most, a three chip family (e.g. see Reference 16).* Each of the modules are implemented on a single card, the six modules interconnected on a wired backplane.

In the laboratory, the basic microcomputer module of Figure 16 is employed as a sensor microcomputer (the first set

---

* The laboratory version employs a 16-bit architecture with NMOS technology. Future, single chip versions may be available in 32-bit architectures with faster (e.g. SOS) technology. See Reference 13.

of triplets in Figure 15) and a servo computer (the second set of triplets). The structure of these modules is depicted in Figure 17 and their operation explained as follows:

1) Each of the three sensor modules (Figure 17 a) selects (analog) outputs from the rotorcraft plant and converts these to digital representation. (Non-redundant sensor inputs have been used to date.) The computer then generates trajectory commands (range and velocity) and executes a trajectory control algorithm based on the errors between these commands and the actual aircraft states. The outputs of the trajectory control algorithm are attitude control commands which, along with the measured attitude control states (pitch attitude and pitch rate) are written to the buffer memories of the servo modules. (Refer again to Figure 3 for the structure of the interconnect between modules transmitting and receiving buffered data.) In addition to those command and data values, a "data ready" signal is transmitted to indicate to the receiving (servo) microcomputer

that the memory has been written. The purpose of this signal is to circumvent the conflict of the receiving microcomputer trying to read a memory simultaneously being written by a transmitting computer.

2) Each servo computer in turn reads the contents of its buffer memories, performs the midvalue select, threshold detection, executes the attitude control algorithm, and generates an analog control (Figure 17b). Control outputs feed a model of a "fly-by-wire" actuator (Figure 15) which in turn outputs a single control to the aircraft plant.

The above implementation of RAMP has verified that the basic concepts behind RAMP can be realized in practice specifically:

1) Minimum Complexity Realization

One guage of complexity in a fault-tolerant computational system is the amount of "overhead" that is required to provide fault tolerance. In terms of computational resources, one is concerned principally with the demands on available program memory and execution time. Figure 18 shows the fractions of

program memory and execution time that constitute the overhead of the current laboratory version of RAMP as would be applied in the six-degree-of-freedom application. (In the figure "FAULT" refers to the midvalue-select, threshold-detect code, "COMM" to the code for writing and reading buffer memories, "OTHER" referring to the modules mainline program, initialization procedures, etc. "AVAILABLE" refers to that portion of the resource which can be used for the modules intended application.)

Note that the pre-flight test is not included in the figure (the current version being evaluated employs approximately .5K of program memory and requires approximately 4 minutes to execute).

2) Asynchronism

Clock rates can and have been varied in the laboratory to demonstrate the convergence properties of the asymptotically stable control algorithms.

3) Mid-Value Select; Threshold Detection

The mid-value select strategy has, to date, been employed in the laboratory simulations with

no effects on system stability or accuracy. The coverage of the threshold detection process is still under investigation and will be reported in the future.

4) Intermittent Fault Tolerance

The reset circuit method described in Section VII has been implemented and tested successfully in the laboratory.

IX. CONCLUSION

As explained earlier, the RAMP is targeted to the VLSI microcomputer components that are projected to be commercially available in the late 1980's. I.e., implementation of RAMP using present-day components is not practical chiefly due to the large volumes required for packaging. Correspondingly, the reliability levels that can be achieved by RAMP will depend upon the reliabilities of these future components. LSI semiconductor failure rates (on a per-gate basis) have continued to decrease with improvements in screening and processing technology. It is the industry's goal to continue this trend, in the face of the new and forceful challenges which VLSI technology will present (Reference 14). Hence, the viability of RAMP as a means of realizing ultra reliable avionics systems hinges on these future developments.

The concepts underlying RAMP, in particular the use of autonomous, asynchronous, intermittent-fault tolerant modules for control has broad, immediate application. Correspondingly,

these concepts present several new research challenges for both the systems theorist and experimentalist in the general areas of distributed asynchronous microcomputer networks, testing and fault masking and identification.

APPENDIX A - EFFECT OF TIMING ERRORS IN PARALLEL ASYNCHRONOUS FLIGHT CONTROL COMPUTERS

As described in the main body of the text, it is the function of each microcomputer in the RAMP structure to mid-value select operands from the upstream set of redundant computers. To simplify the discussion of the effects of asynchronism consider the case of two parallel computers M_S and M_D employed in a control configuration as shown in Figure A1. As seen in the figure, M_S is the computer selected for the control, i.e., is operating in the closed loop. Computer M_D is deselected and is hence operating open loop.

As also explained earlier, each of a given set of parallel computers in the RAMP structure has n_o cache memories such that the nth cache memory in each computer in the set is simultaneously written by the nth upstream computer. This process is represented in Figure A1 by a sample-and-hold function as shown (i.e., only the selected upstream computer is depicted). Computers M_D and M_S in turn sample the held values as shown. This sampling process is depicted in Figure A2 wherein the samplings by M_D and M_S are shown by impulse functions $i_D(t)$ and $i_S(t)$. For the development that

follows, it is assumed that the selected computer M_S samples at the same rate that y(t) is sampled with each sample taken at a time immediately following the sampling of y(t). Computer M_D samples at a higher rate as shown such that at times iT and (i + n)T computers M_S and M_D simultaneously sample the input. Between these times, computer M_D samples an input value immediately preceeding that sampled by M_S. Note that in the period nT, computer M_D takes exactly one more sample (and hence one additional computation) than M_S.

Now both computers will execute the same control law given by the following recurrence equation:

$$u\left[(i + 1)T\right] = Au\,(iT) + By\left[(i + 1)T\right] \qquad (A1)$$

In what follows it is shown that Equation (A1) must be stable in order that the deselected computer not accumulate unacceptable errors due to its asynchronism. Stability of Equation (A1) is however not required for stability of the closed loop system. The existence and form of the error is arrived at inductively in the following.

At time iT, let the selected and deselected computers respectively compute control values $u_S(iT)$ and $u_D(iT)$. Moreover, assume that the deselected computer has an error

u_e (iT) due to the asynchronism such that,

$$u_D(iT) = u_S(iT) + u_e(iT) \qquad \text{(A2)}$$

Now compute

$$u_e\left[(i + n)T\right] = u_D\left[(i + n)T\right] - u_S\left[(i + n)T\right]$$

as follows.

First, note that in solving Equation (A1),

$$u_S\left[(i + n)T\right] = A^n u_S(iT) + \sum_{k = i}^{n+i-1} A^{n+i-k-1} By\left[(k+1)T\right] \qquad \text{(A3)}$$

Through study of Figure (A2) it can be seen that

$$u_D\left[(i + n)T\right] = A^{n+1} u_D(iT) + A^n By(iT) + \sum_{k = i}^{n+i-1} A^{n+i-k-1} By\left[(k + 1)T\right] \qquad \text{(A4)}$$

From (A2),

$$u_D(iT) = u_S(iT) + u_e(iT)$$

Substituting this into Equation (A4) and subtracting Equation (A3) the error at time (i + n)T is obtained:

$$u_e\left[(i + n)T\right] = A^{n+1} u_e(iT) + A^n (A - I) u_S(iT) + A^n By(iT) \qquad \text{(A5)}$$

The assumption that $u_e(iT)$ is an error due to asynchronism is correct since at $t = 0$, $u_e(0) = 0$ (i.e. the first sample is taken simultaneously by the computers) and at time $t = T$,

$u_\epsilon(T)$ arises solely from the asynchronism and is in general non-zero.

Equation (A5) may be written in final form:

$$u_\epsilon[(j+1)nT] = A^{n+1}u_\epsilon(jnT) + A^n[A - I]\,u_S(jnT) + A^n By(jnT) \qquad (A6)$$

Now for (A6) to be stable matrix A^{n+1} must have eigenvalues λ_{ϵ_i} such that,

$$|\lambda_{\epsilon_i}| < 1 \qquad \forall i \qquad (A7)$$

Correspondingly, for Equation (A1) to be stable matrix A must have eigenvalues λ_{s_i} subject to the same condition.

But

$$\lambda_{\epsilon_i} = (\lambda_{s_i})^{n+1}$$

i.e.,

$$|\lambda_{\epsilon_i}| = |\lambda_{s_i}|^{n+1}$$

<u>Hence, to guarantee convergence of the error in the deselected computer due to asynchronism, it is necessary that the control law being computed be asymptotically stable.</u>

Note further that when n is large (as is the case in practical applications) Equations (A1) and (A6) have the same natural response. E.g. at time t = mnT (m an integer)

$$u_S(mnT) = A^{mn}u_S(0).$$

Correspondingly,

$$u_{\mathcal{C}}(mnT) = A^{m(n+1)} u_{\mathcal{C}}(0)$$

$$\tilde{=} u_S(mnT).$$

Hence, the rate of convergence of the asynchronism error introduced into the deselected computer is the same as that of the control law being computed.

Finally, looking at the last two terms on the right of Equation (A6), the magnitude of the error in the deselected computer can be reduced by one or both of the following:

1) Increasing the rate of convergence in the control law.

2) Increasing n(i.e. reducing the timing error).

APPENDIX B - MID-VALUE SELECT FOR FAULT MASKING

Section V described the strategy of selecting the mid-value of a set of N = (2n + 1) data values as a means of masking faults in the RAMP network. The following paragraphs explain in greater detail the justification for this approach.

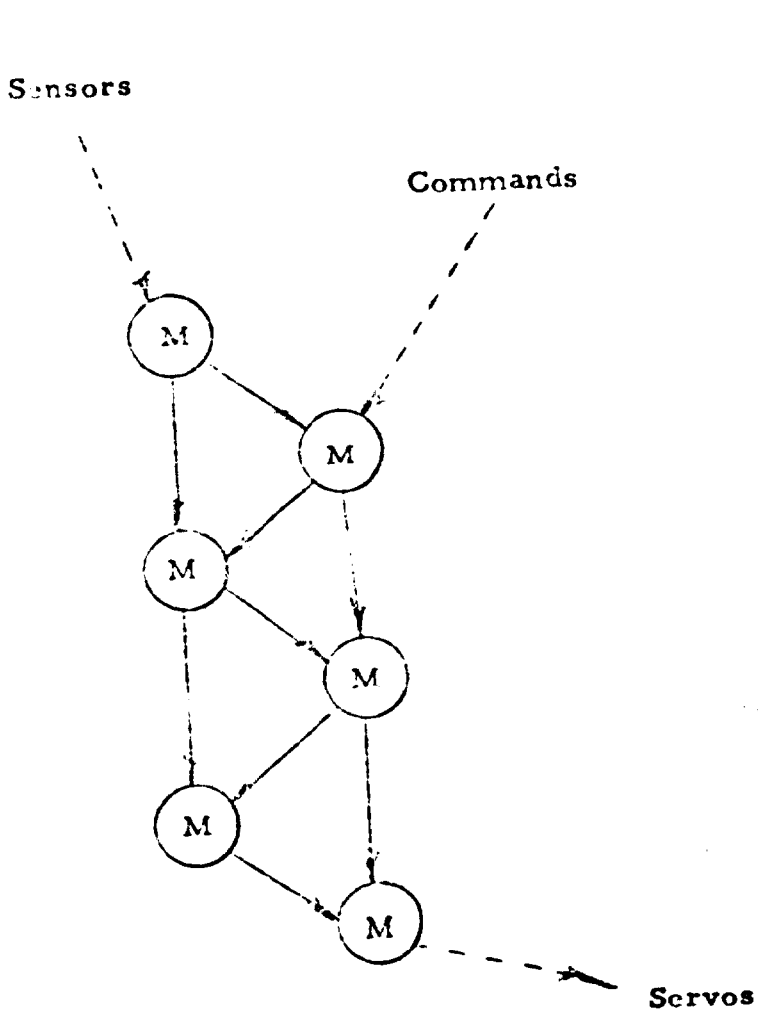
Recall that the effect of the timing errors in the set of N computers is to disperse the data values over some range 2ϵ about that value which would be obtained with no timing error. This is illustrated in Figure B1 in which each of the N values are classified as being correct or incorrect as follows. A correct value lies within $\pm \epsilon$ of the value that would be obtained with no timing error; all other values are incorrect. Given no faults (i.e. timing error only) all values would be correct. On the other hand a data value generated by a faulted computer could be in general either correct or incorrect.

It can now be argued that given up to n failures in the N computers, the mid-value will always be correct: If the mid-value originated from a non-failed computer, it is obviously correct. If on the other hand it originated from a faulted computer it will still be correct. The reason for this is that the mid-value by definition has exactly n distinct values greater

than or equal to it and exactly n distinct values less than or equal to it. Hence, if the mid-value is generated from a faulted computer there must remain up to n - 1 values originating from faulted computers. The mid-value must accordingly lie between two correct values and therefore be correct.

This argument also shows why N must be odd since in using an even number N = 2n of computers* only (n - 1) failures can be tolerated, the same condition that would correspond to use of an odd number, N - 1 of computers. I.e., the use of an even number of computers is uneconomic.

* Given an even number of computers there is of course no mid-value. One would instead select the median value of the two innermost values on the real line.



M = Microcomputer

- ANALOG-LIKE MODULES
- ASYNCHRONOUS
- ELECTRICALLY ISOLATED
- COMMUNICATION CONFINED TO MESSAGE/DATA TRAFFIC

FIGURE 1 - RAMP DISTRIBUTED NETWORK CONCEPT

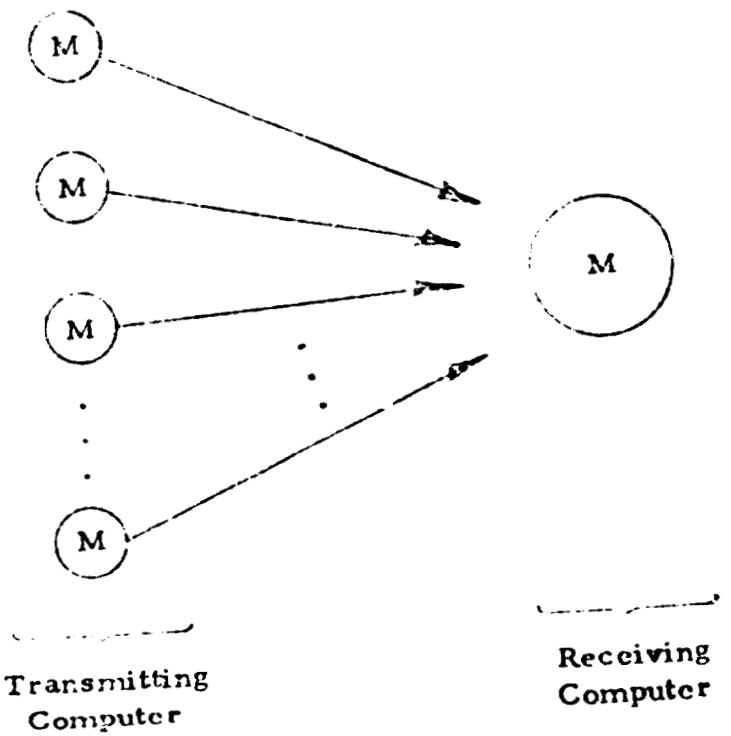
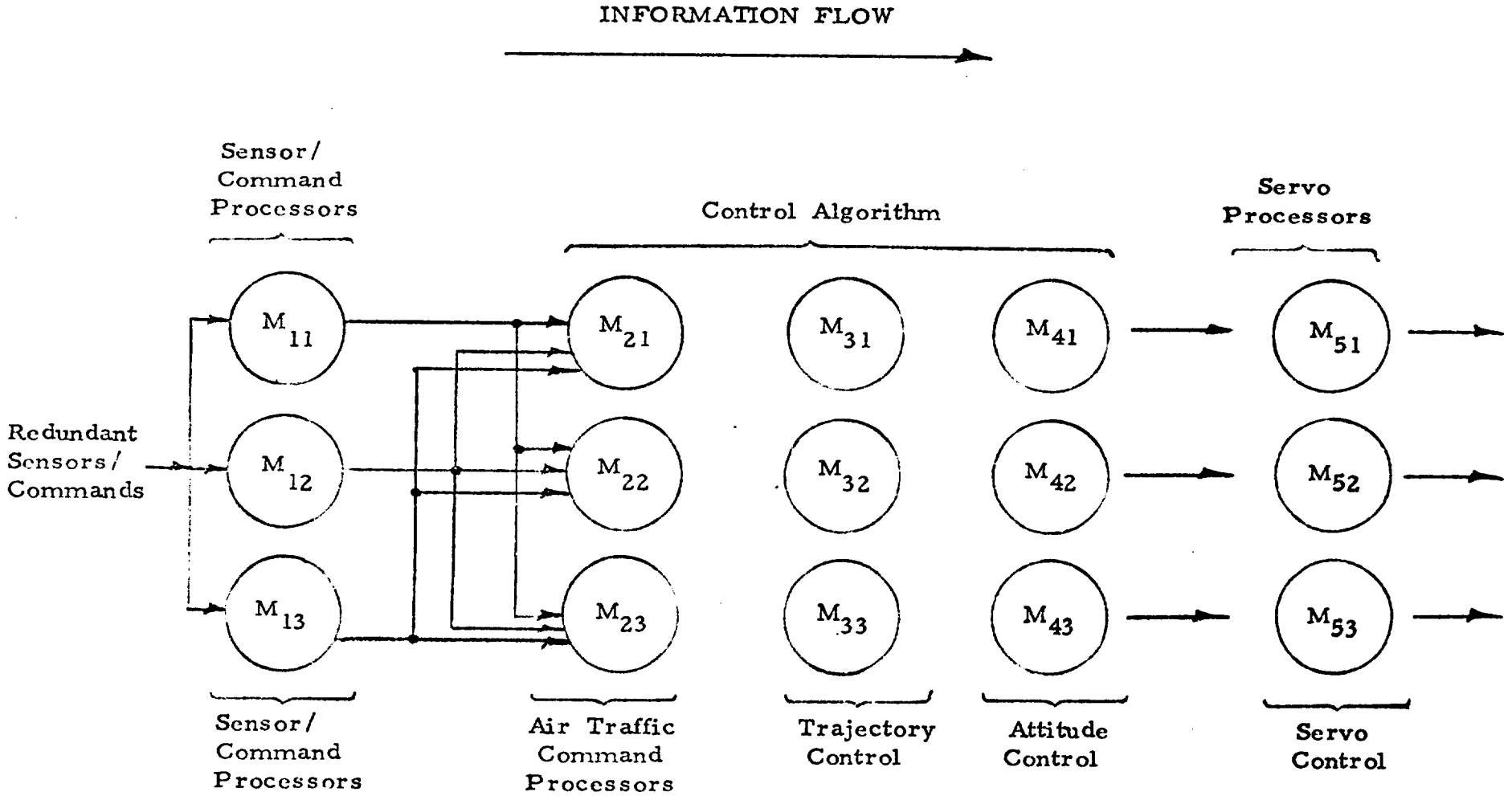


FIGURE 2 - REDUNDANT MICROCOMPUTERS



Note: Intercommunication connection shown only for leftmost computers

FIGURE 3 - SPECIFIC RAMP CONTROL SYSTEM UNDER EXPERIMENTATION

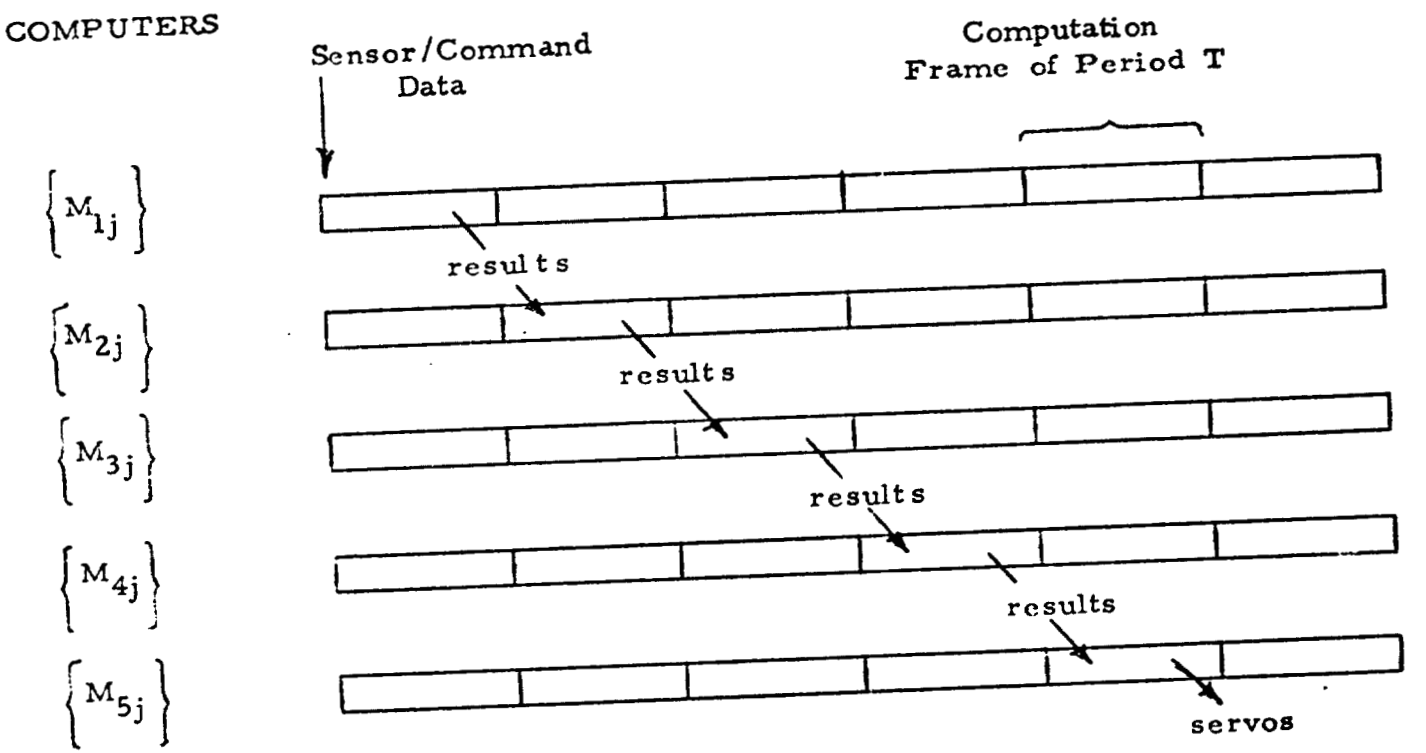


FIGURE 4 - INFORMATION FLOW IN EXPERIMENTAL RAMP STRUCTURE OF FIGURE 3.

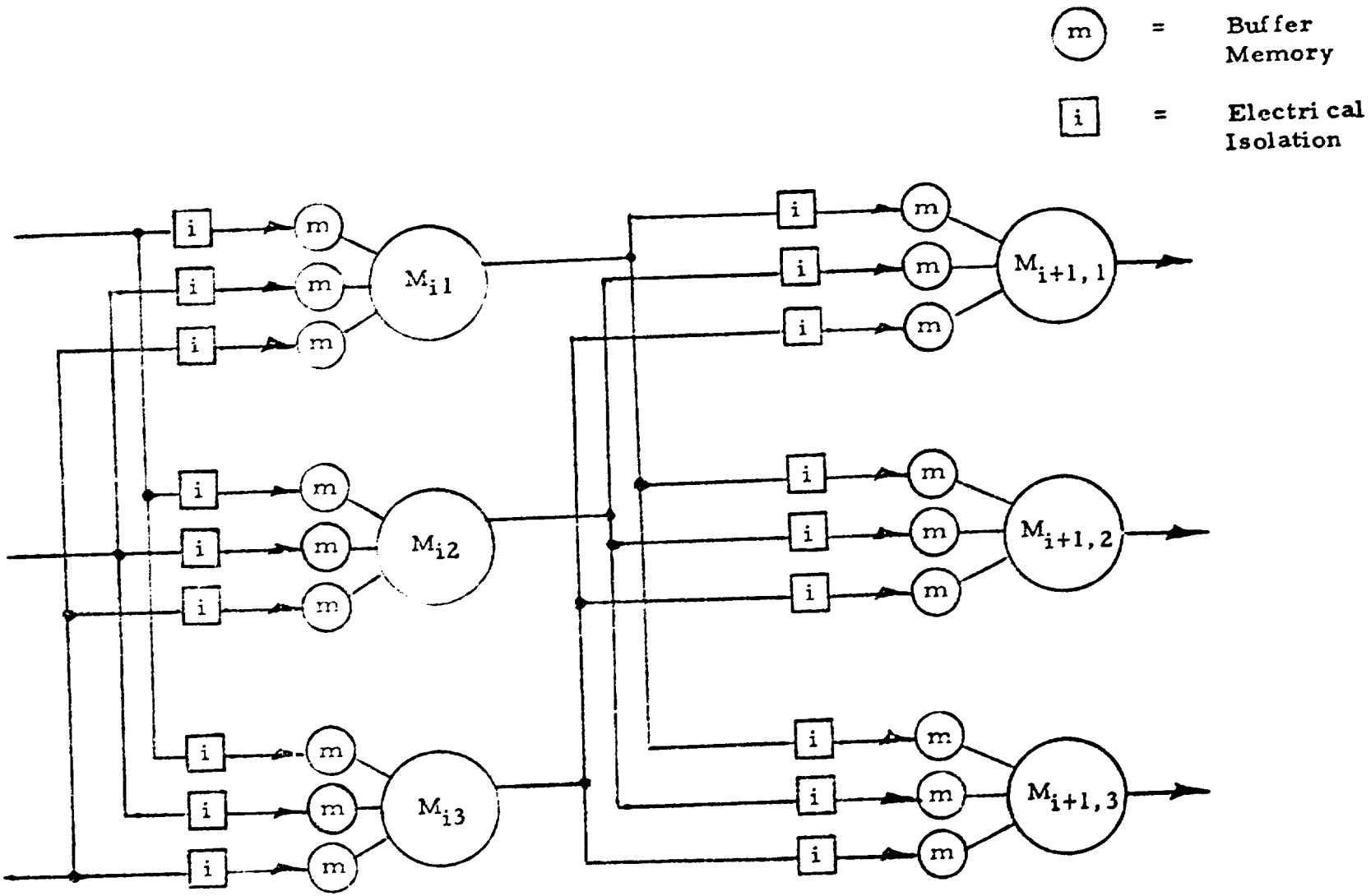


FIGURE 5 - RAMP COMMUNICATIONS PATHS AND BUFFER MEMORIES

Control Law:

$$u\left[(i+1)T\right] = Au(iT) + By\left[(i+1)T\right]$$

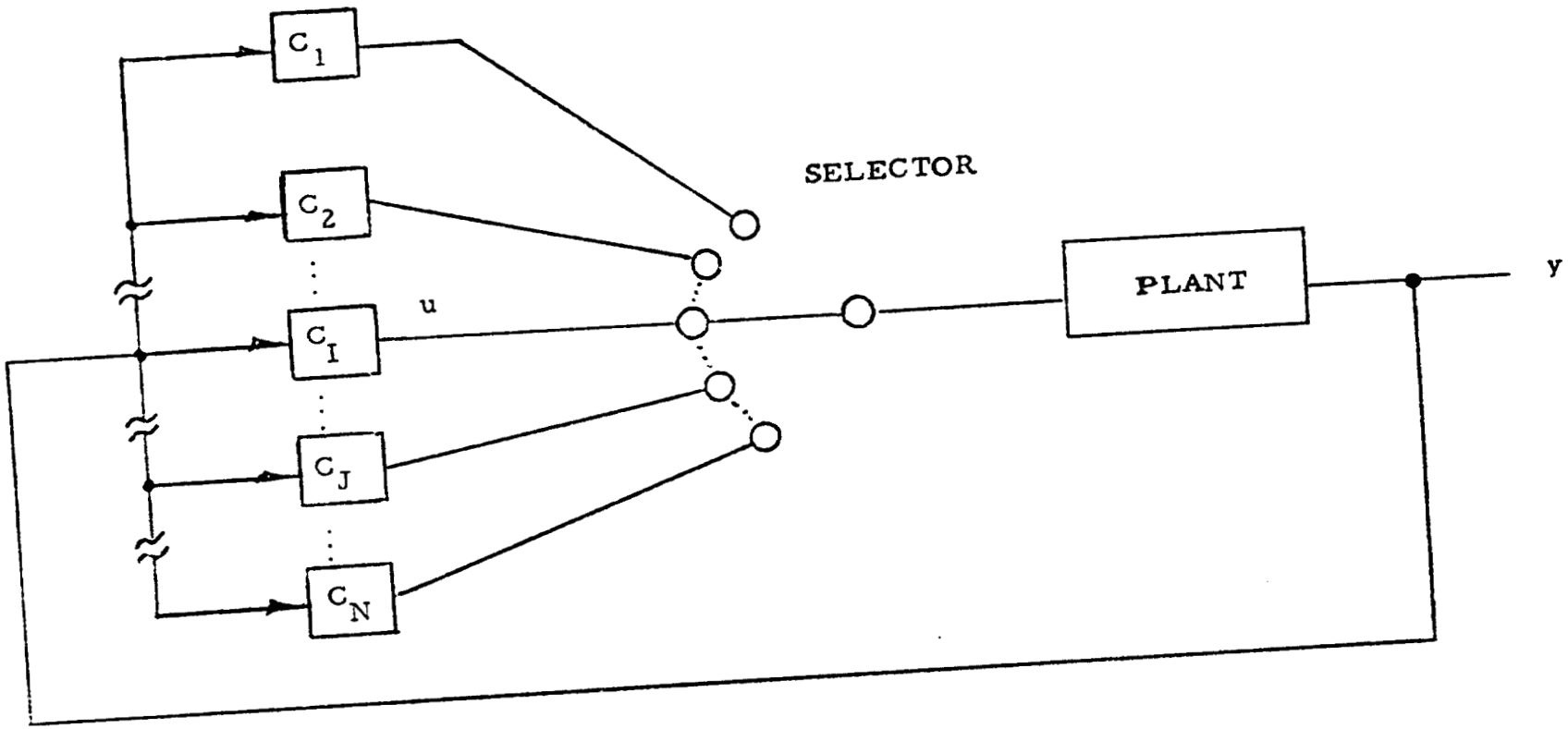
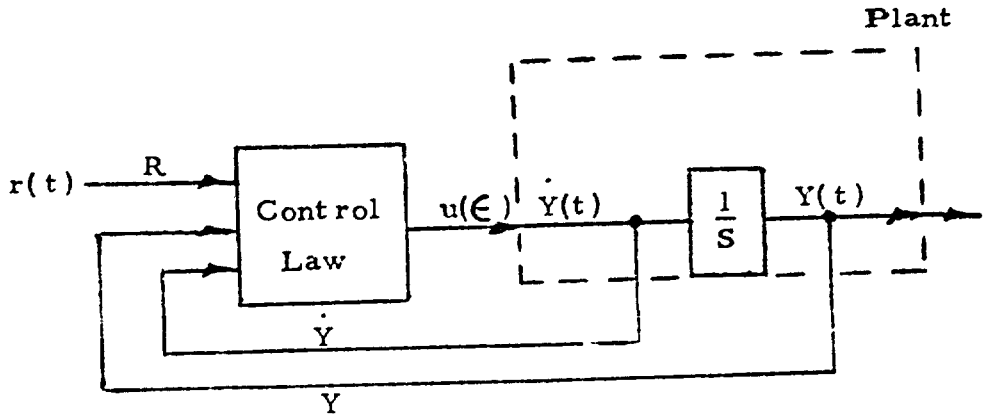
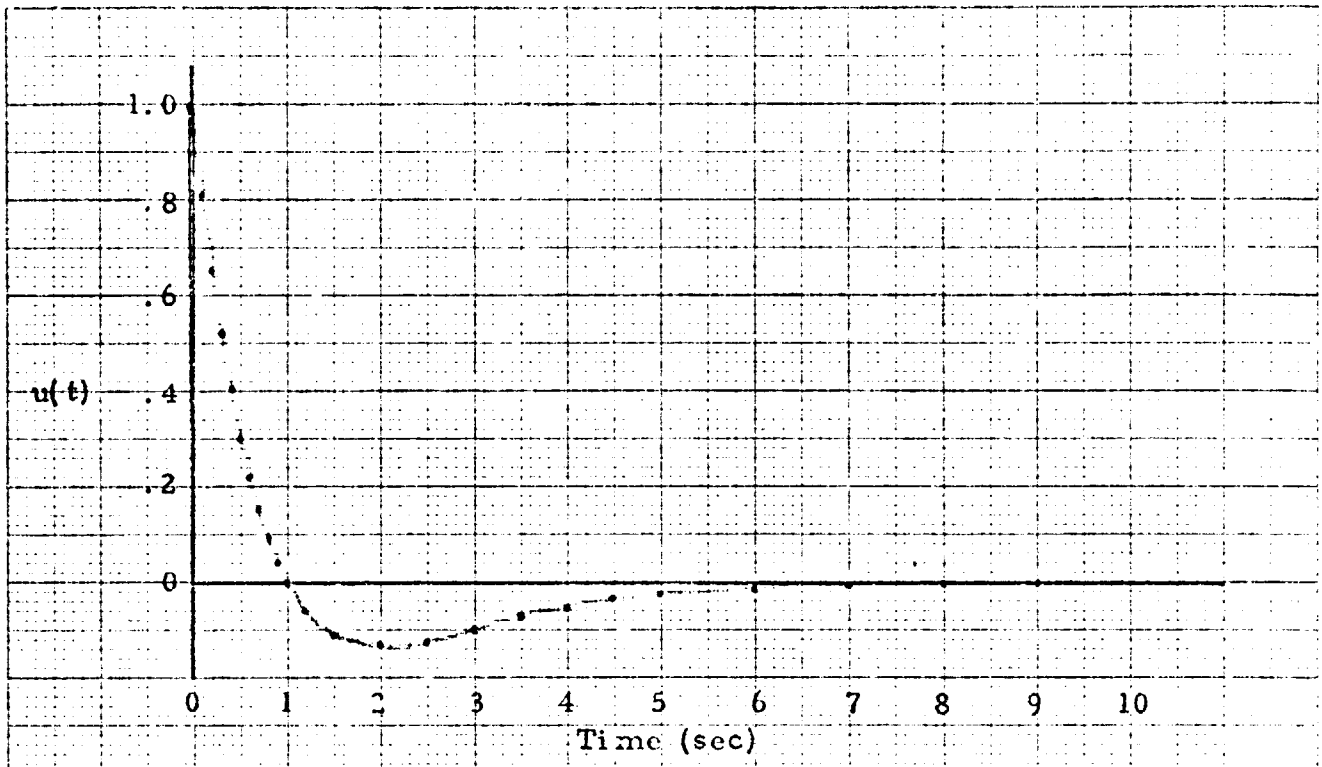


FIGURE 6 - PARALLEL ASYNCHRONOUS CONTROL COMPUTERS

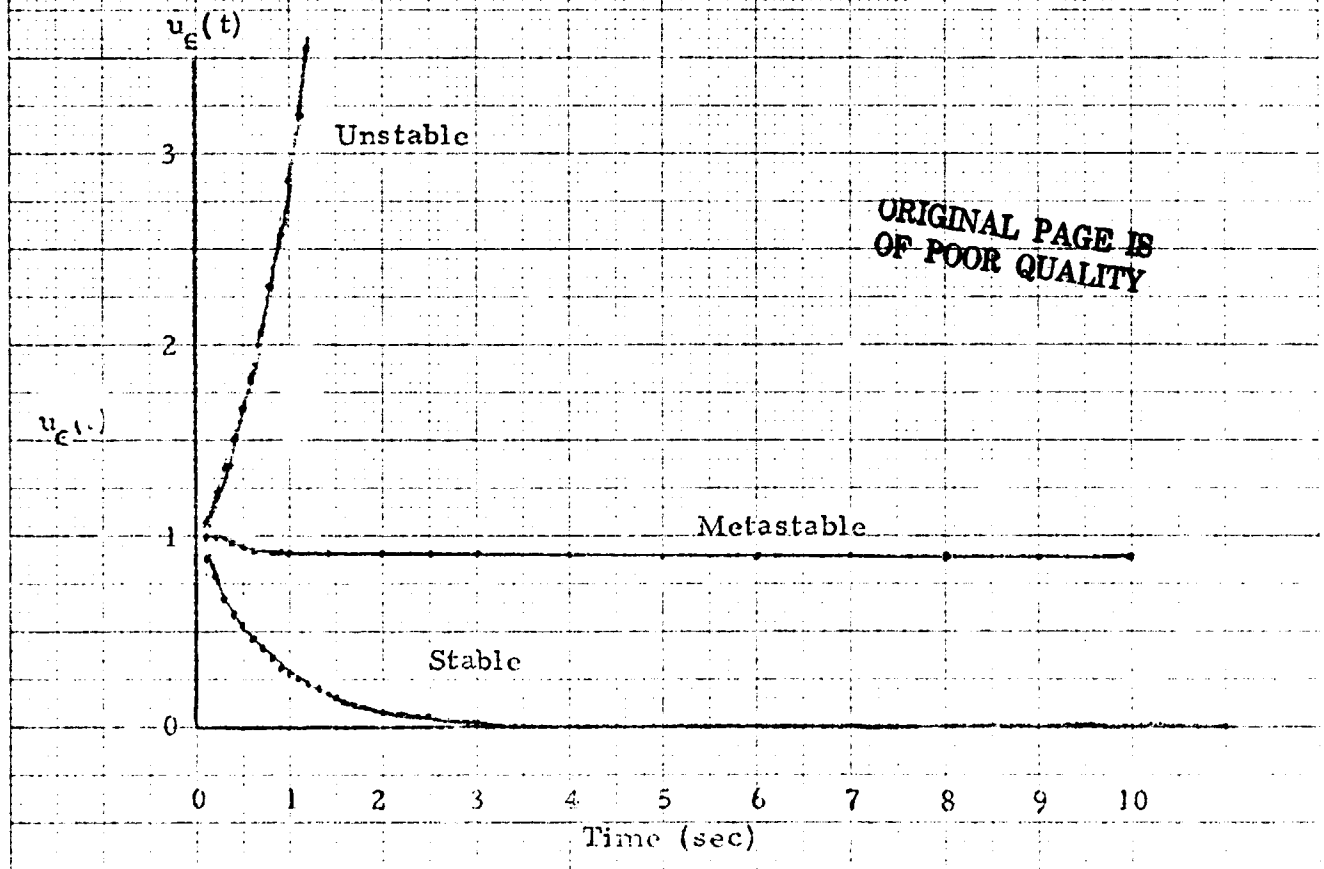


CASE	CONTROL LAW
STABLE	$u(k+1) = \frac{1}{1+T}\mu(k) + [R - \dot{Y} - Y]\frac{T}{1+T}$
METASTABLE	$u(k+1) = u(k) + [R - 2\dot{Y} - Y]T$
UNSTABLE	$u(k+1) = \frac{1}{1-T}u(k) + [R - 3\dot{Y} - Y]\frac{T}{1-T}$

FIGURE 7 - EXAMPLE OF STABLE, METASTABLE AND UNSTABLE CONTROL LAWS

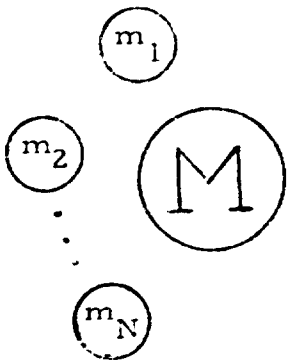


(a) UNIT IMPULSE RESPONSE



(b) ERRORS DUE TO ASYNCHRONISM

FIGURE 8 - RESPONSE OF SYSTEM OF FIGURE 7



Memory Contents

Memory Location	m_1	m_2	... m_k ...	m_N
1	D_{11}	D_{12}	... D_{1K} ...	D_{1N}
2	D_{21}	D_{22}	... D_{2K} ...	D_{2N}
⋮	⋮	⋮	⋮	⋮
j	D_{j1}	D_{j2} ...	D_{jk} ...	D_{jN}
⋮	⋮	⋮	⋮	⋮
K	D_{K1}	D_{K2} ...	D_{Kk} ...	D_{KN}

FIGURE 9 - BUFFER MEMORY CONTENTS

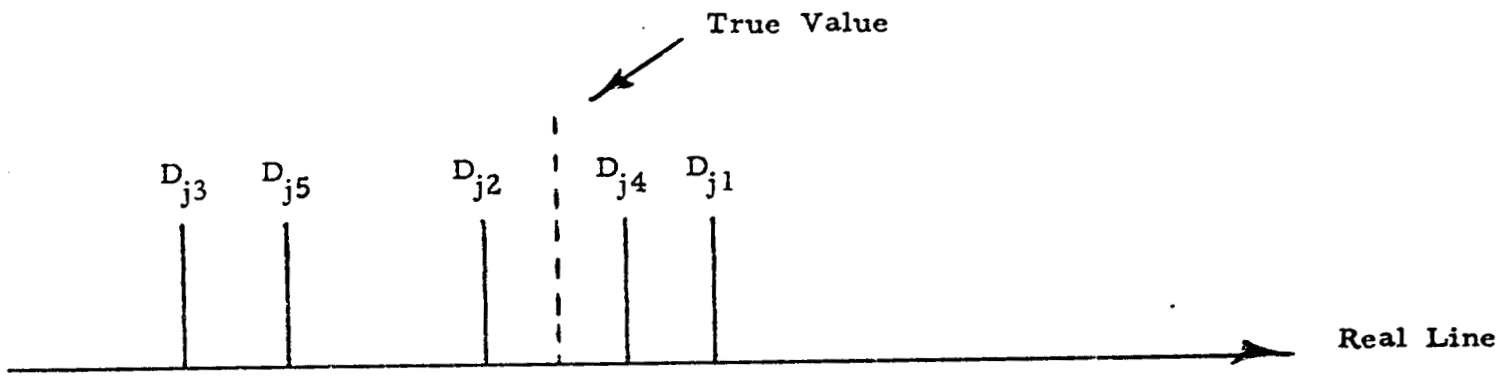
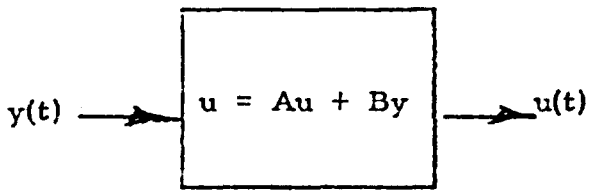


FIGURE 10 - DISPERSION OF DATA VALUES DUE TO ASYNCHRONISM FOR N = 5

(a) Control Algorithm



(b) Example Program:

```
TOP:  CALL INPUT REDUNDANT Y VALUES
      CALL MIDVALUE SELECT Y
      CALL THRESHOLD DETECT Y ERRORS
           u = Au + By
      CALL OUTPUT u
      CALL TIMEOUT
           JUMP TO TOP
```

(c) Midvalue Select Procedure:

If $Y_1 \leq Y_2 \leq Y_3$, MIDVAL = Y_2

If $Y_2 \leq Y_1 \leq Y_3$, MIDVAL = Y_1

etc.

FIGURE 11 - EXECUTING OF EXAMPLE CONTROL ALGORITHM

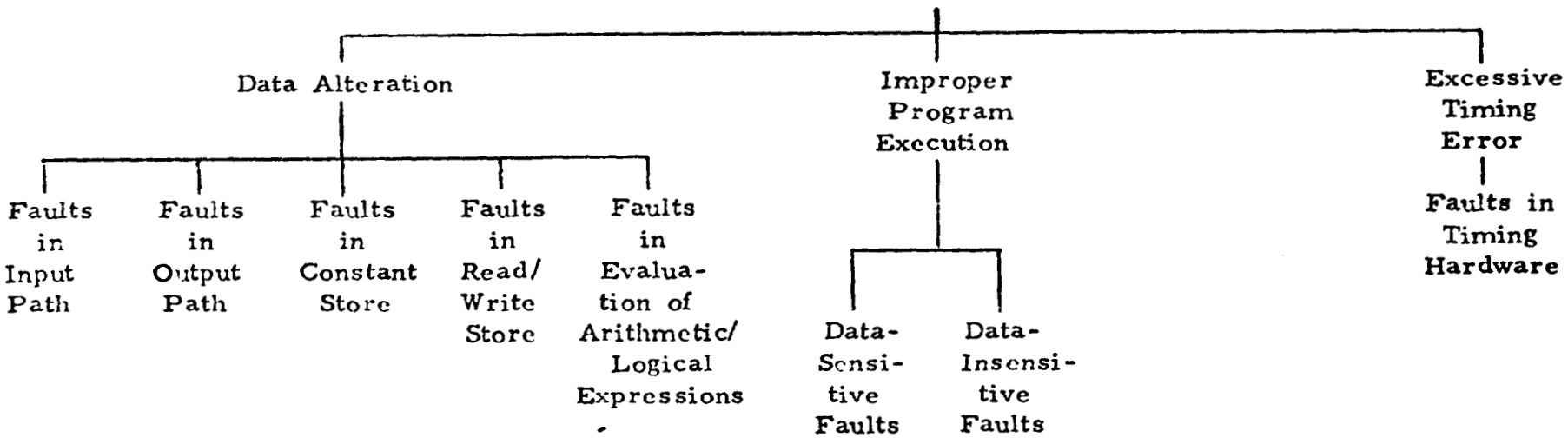


FIGURE 12 - RAMP MICROCOMPUTER MODULE FAULT TREE

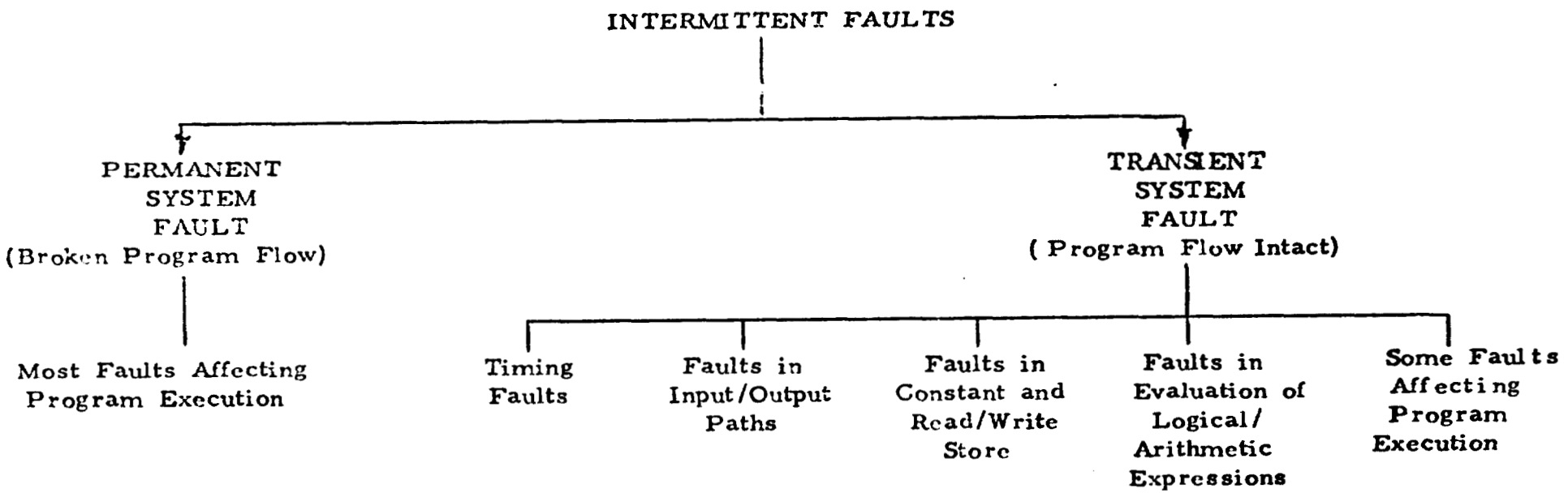
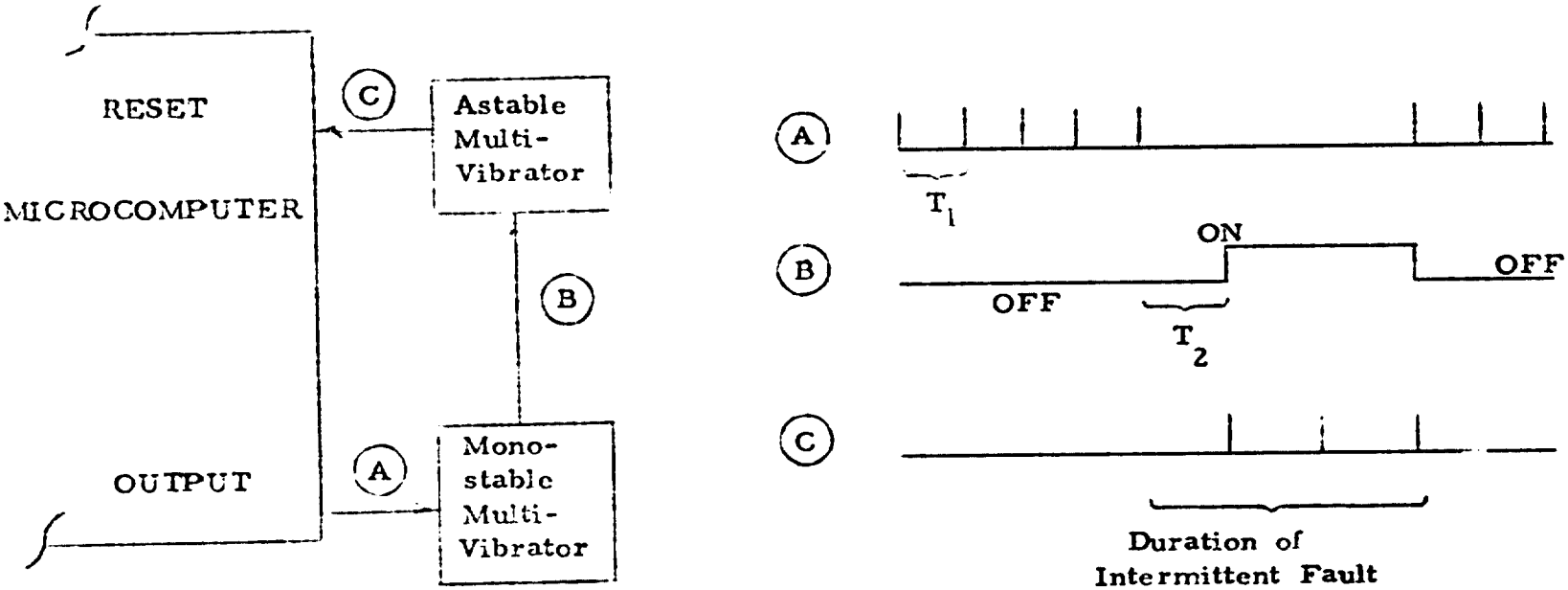


FIGURE 13 - CHARACTERIZATION OF INTERMITTENT FAULTS IN THE MICROCOMPUTER SYSTEM



14a - Circuit

14b - Waveforms

FIGURE 14 - RESTART CIRCUITRY USED TO REESTABLISH PROGRAM FLOW

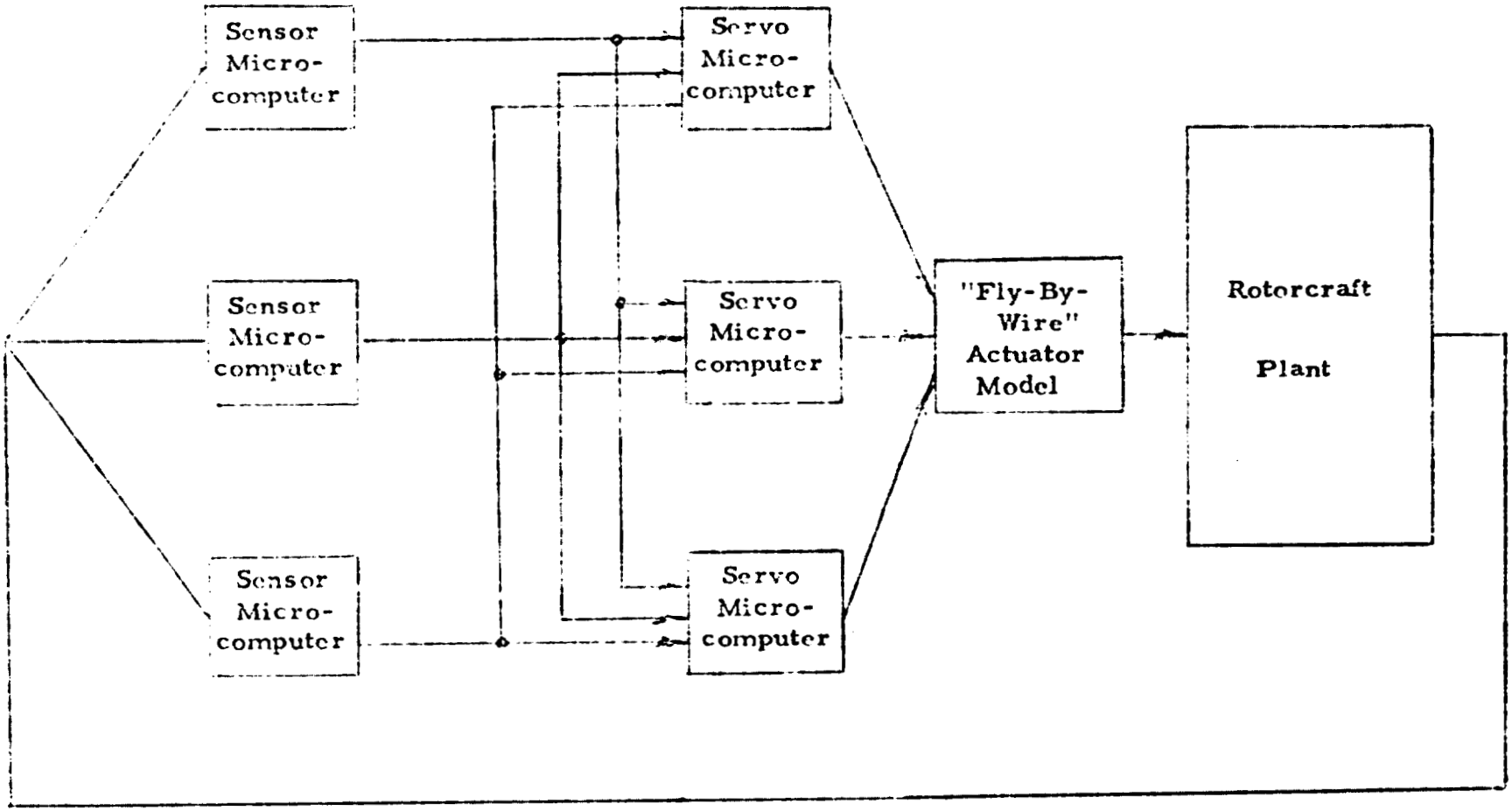
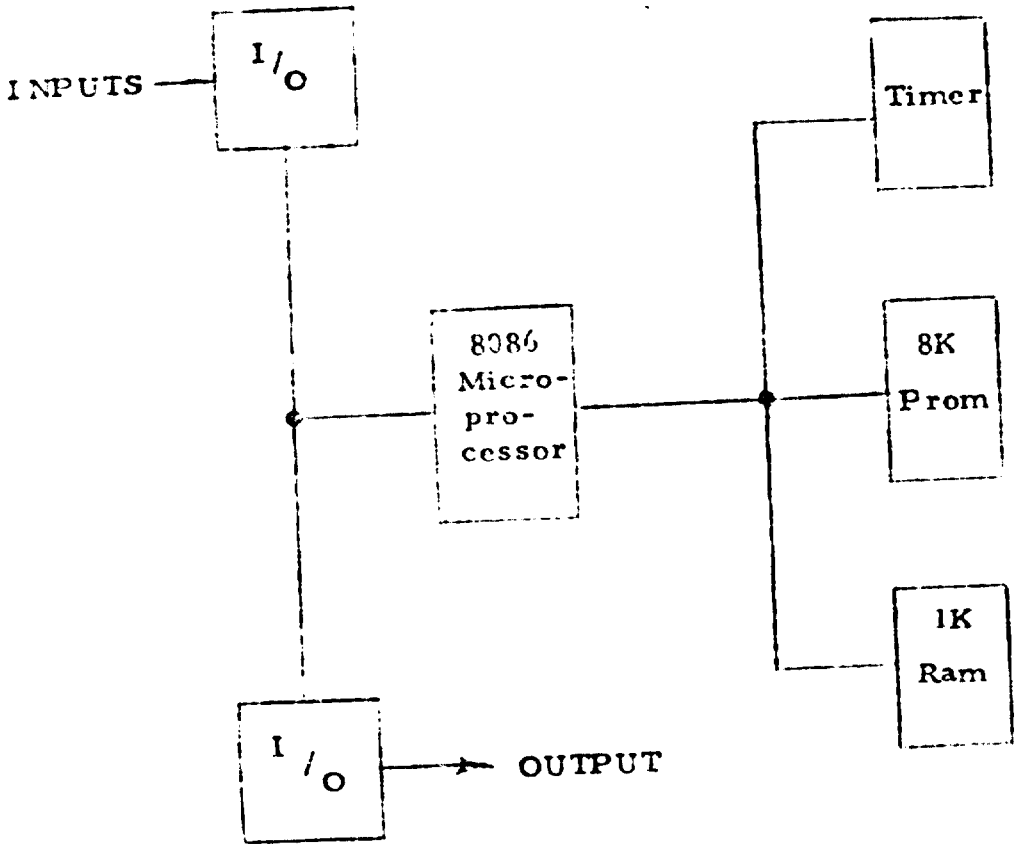


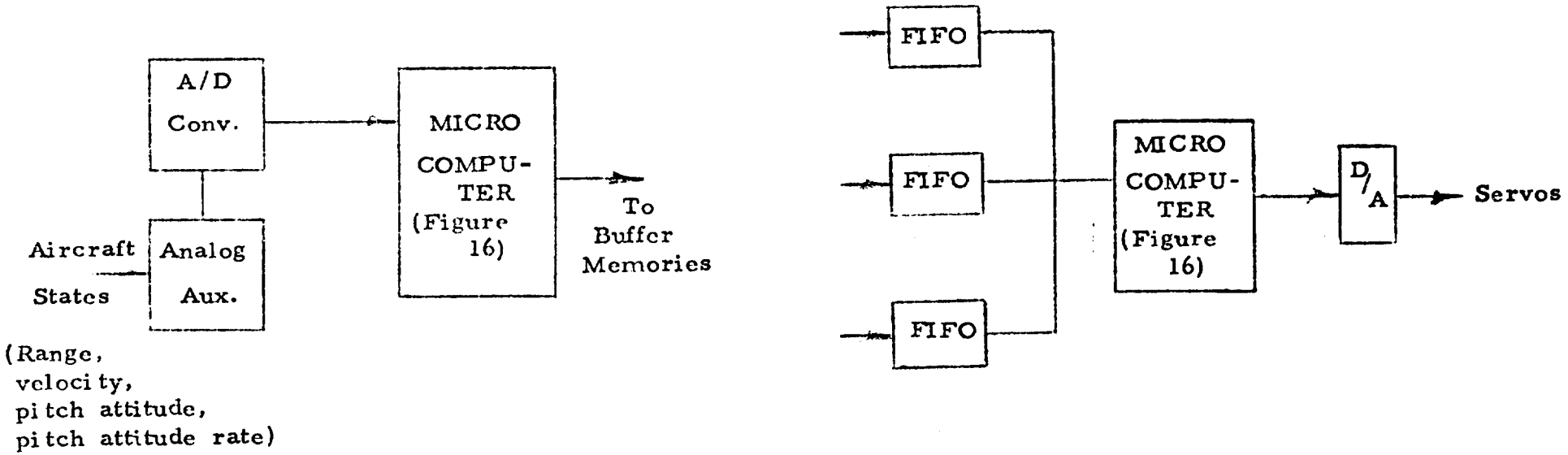
FIGURE 15 - RAMP LABORATORY CONFIGURATION



BASIC SPECIFICATIONS

Technology:	NMOS
CPU Arch:	16 bit
Program Memory:	oK bytes
Ram Memory:	1K bytes

FIGURE 16 - BASIC LABORATORY VERSION OF RAMP MICROCOMPUTER MODULE



(a) Sensor Module

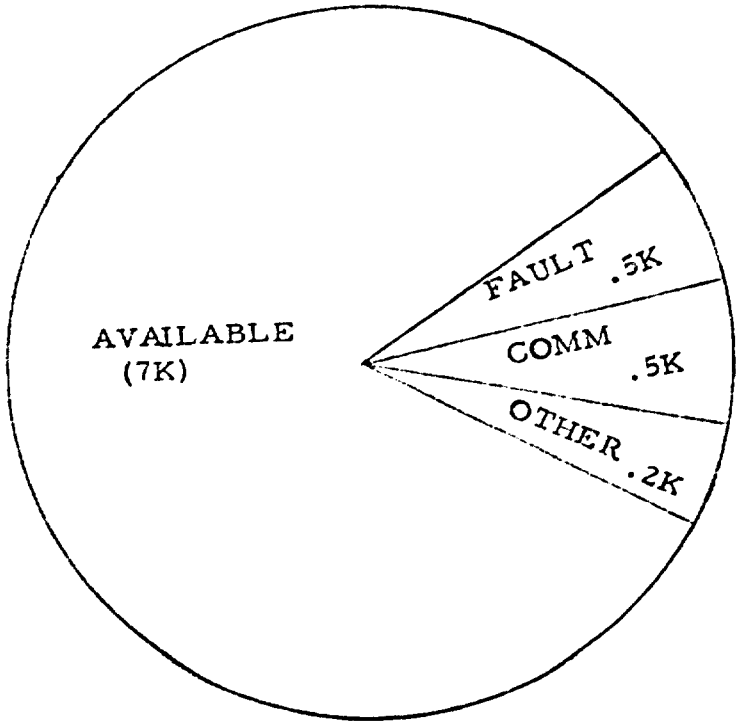
(Also generates trajectory commands and provides trajectory control)

(b) Servo Module

(Also provides attitude control)

FIGURE 17 - MICROCOMPUTER MODULE TYPES EMPLOYED IN LABORATORY VERSION OF RAMP

PROGRAM MEMORY (8.2K Available)



TIME (50 ms Computation Frame)

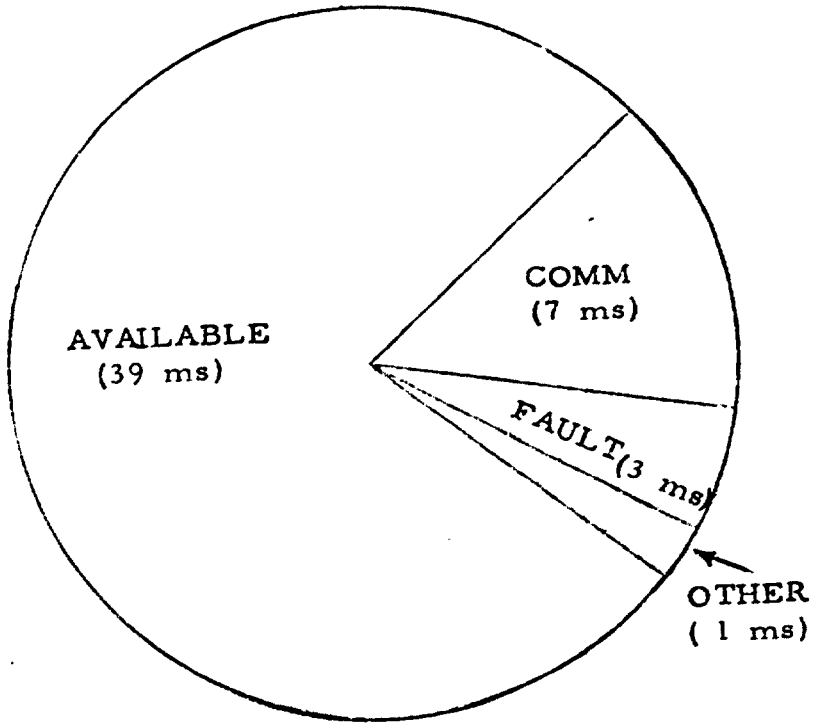


FIGURE 18 - ALLOCATION OF MICROCOMPUTER MODULE RESOURCES TO SIX DEGREE OF FREEDOM FLIGHT CONTROLLER

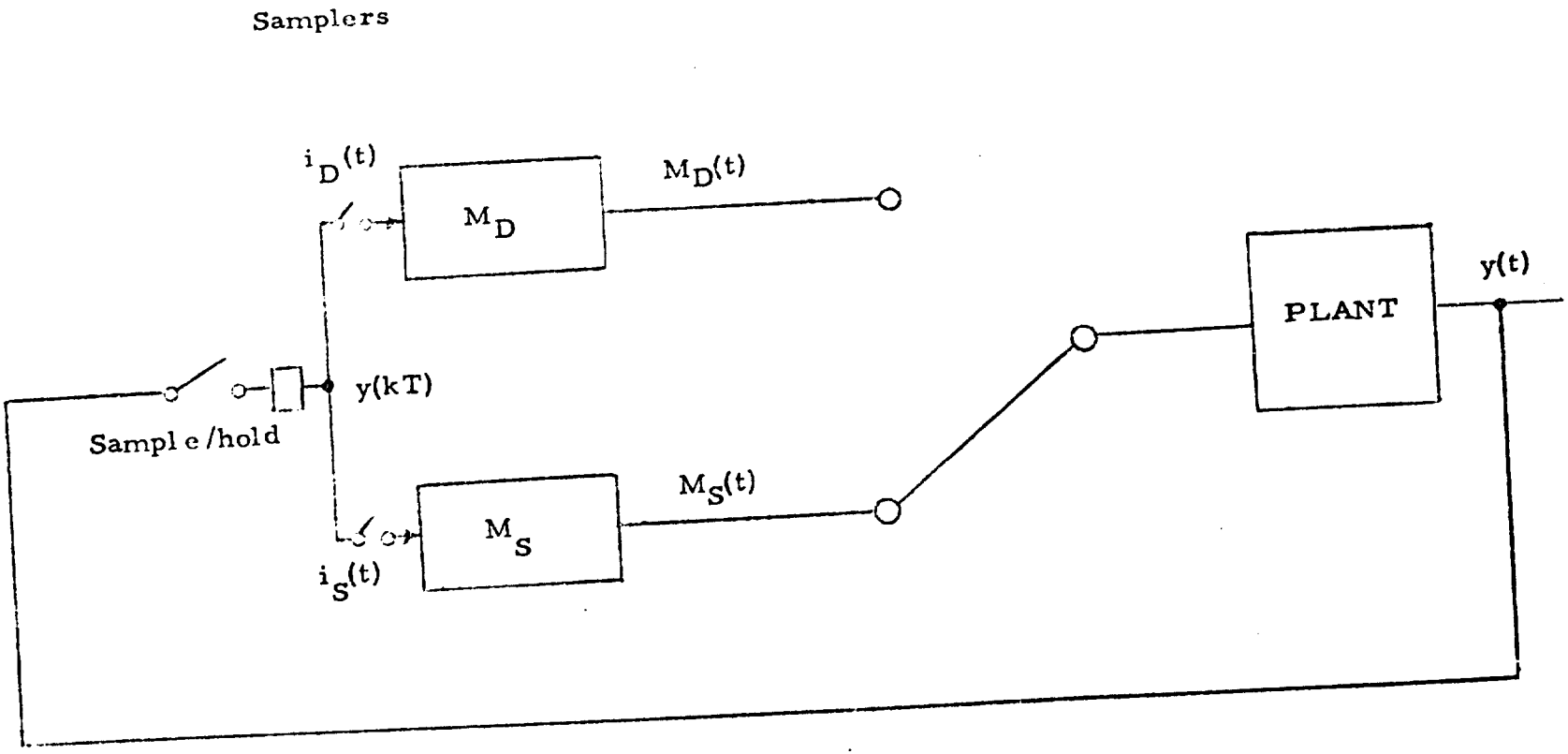


FIGURE A1 - PARALLEL, REDUNDANT, ASYNCHRONOUS MICROCOMPUTERS

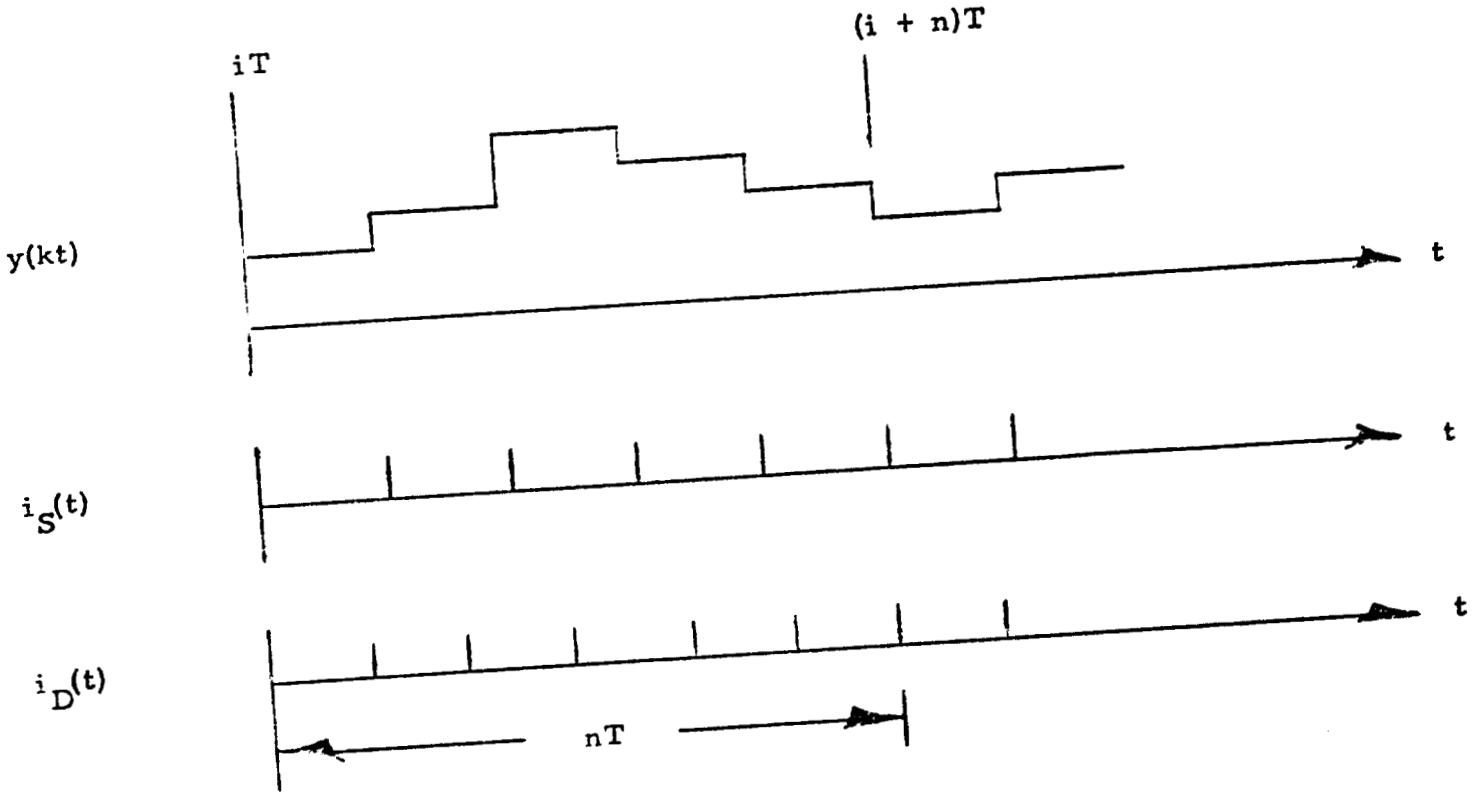


FIGURE A2 - DATA SAMPLING FOR SYSTEM OF FIGURE B1

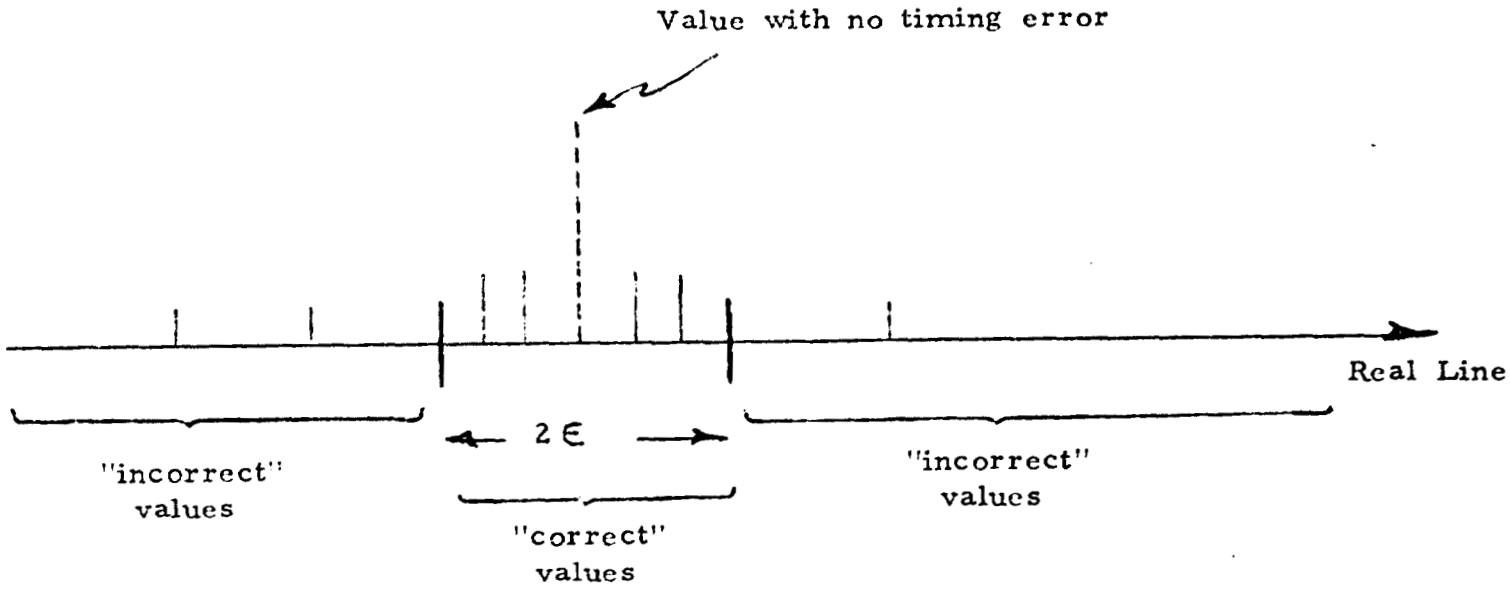


FIGURE B1 - MID-VALUE SELECT

REFERENCES

1. G. Meyer and L. Cicolani, "A Formal Structure for Advanced Automatic Flight Control Systems", NASA TN D-7940, 1975.

2. G. Meyer and L. Cicolani, "Application of Nonlinear Systems Inverses to Automatic Flight Control Design-System Concepts and Flight Evaluations", AGARDO-GRAPH on Theory and Applications of Optimal Control in Aerospace Systems, 1980.

3. J. Wensley, et. al., "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control", Proc. IEEE, Vol. 66, No. 10, pp. 1240-1255, October, 1978.

4. A. Hopkins, et. al., "FTMP - A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft", Proc. IEEE, Vol. 66, No. 10, pp. 1221-1239, October, 1978.

5. R. Yeh, Ed., "Applied Computation Theory, Analysis Design and Modelling", pp. 352-359, Prentice Hall, 1976.

6. T. Cunningham, et. al., "Fault Tolerant Digital Flight Control with Analytical Redundancy", USAF Technical Report AFFOL-TR-77-25.

7. W. Dunn and G. Meyer, "A Fault-Tolerant Distributed Microcomputer Structure for Aircraft Control Systems", AIAA Guidance and Control Conference, Palo Alto, California, 1978.

8. R. Smyth, "State of the Art for Digital Avionics and Controls", AGARD Conference Proceedings No. 272, pp. 1-1 to 1-20, May, 1979.

9. Military Standardization Handbook 217B, "Reliability Prediction of Electronic Equipment", September, 1974.

10. R. T. Anderson, "Reliability Design Handbook", Cat. No. RDH-376, IIT Research Institute, Chicago, Illinois, 1976.

11. W. Bouricius, et. al., "Reliability Modelling Techniques for Self-Repairing Computer Systems", Proc. 24th National Conference, Association for Computing Machinery, ACM Publication P-69, 1969.

12. O. Tasar, "A Study of Intermittent Faults in Digital Computers", Proc. National Computer Conference, pp. 807-811, 1977.

13. D. Patterson and C. Sequin, "Design Considerations for Single-Chip Computers of the Future", IEEE Journal of Solid State Circuits, Vol. SC-15, No. 1, pp. 44-52, February, 1980.

14. V. Ohm, "Reliability Considerations for Semiconductor Memories", IEEE Spring COMPCON, San Francisco, 1979.