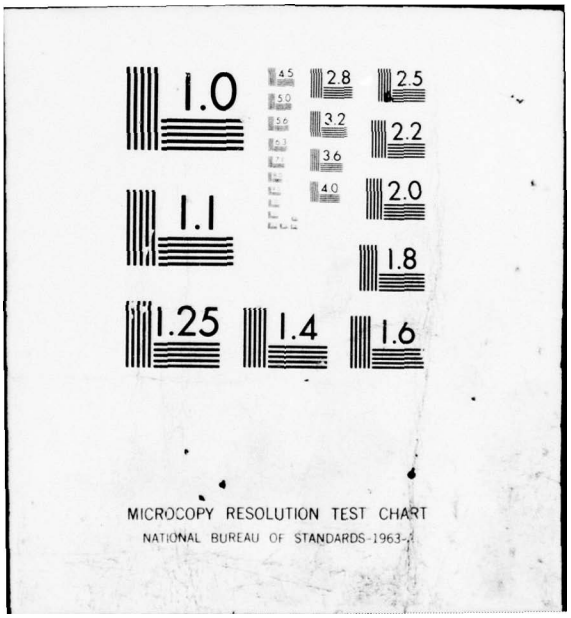
AD-A072 450 HONEYWELL INC MINNEAPOLIS MN SYSTEMS AND RESEARCH CENTER F/G 9/2
A FAULT TOLERANCE ASSESSMENT OF DAIS.(U)
MAR 79 W L HEIMERDINGER, K M FANT F33615-77-C-1232
UNCLASSIFIED 79SRC13 AFAL -TR-79-1007 NL

1 OF 2

AD A072450


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AFAL-TR-79-1007

A FAULT TOLERANCE ASSESSMENT OF DAIS

WALTER L. HEIMERDINGER
KARL M. FANT

HONEYWELL, INCORPORATED
SYSTEMS AND RESEARCH CENTER
2600 RIDGWAY PARKWAY
MINNEAPOLIS, MINNESOTA 55413

MARCH 1979

TECHNICAL REPORT AFAL-TR-79-1007
Final Report - September 1978 - March 1979

AIR FORCE AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Barry D. Baxley

BARRY D. BAXLEY, 1Lt, USAF
Project Engineer
System Concepts Group
Avionic Systems Engineering Branch

Joseph G. Jolda

JOSEPH G. JOLDA, Capt, USAF
Technical Manager
System Concepts Group
Avionic Systems Engineering Branch

FOR THE COMMANDER

Raymond E Siferd

RAYMOND E. SIFERD, Colonel, USAF
Chief
System Avionics Division

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFAL/AAA-2 ,W-PAFB, OH 45433 to help us maintain a current mailing list".

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

AIR FORCE/56780/12 July 1979 — 140

62204F

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: AFAL-TR-79-1007	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): A Fault Tolerance Assessment of DAIS.	5. TYPE OF REPORT & PERIOD COVERED: Sept 78 - March 79 Final Report. Sep 78 - Mar 79
	6. PERFORMING ORG. REPORT NUMBER: 79SRC13
7. AUTHOR(s): Walter L. Heimerdinger, Karl M. Fant	8. CONTRACT OR GRANT NUMBER(s): F33615-77-C-1232
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Honeywell, Inc. Systems and Research Center 2600 Ridgway Parkway, Minneapolis, MN 55413	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 2003-02-43
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Avionics Laboratory, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio 45433	12. REPORT DATE: March 1979
	13. NUMBER OF PAGES: 136
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): 121 P.	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Fault tolerance analysis	Graphical system models	Error recovery
Fault tolerance models	Fault detection	DAIS
Digital system analysis	Fault recovery	LOGOS
Functional faults	Error detection	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A functional/graphical approach is used to assess the fault tolerance of the Digital Avionics Information System (DAIS), a real-time federated multi-computer digital avionics system. Two separate assessments are made; first the functional/graphical approach is used to assess the fault tolerance of the DAIS architecture, then the utility of the functional/graphical approach itself is considered. The functional/graphical approach used has two complementary aspects; a functional characterization of fault phenomena, and a graphical approach to the representation of system behavior, both for normal

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD A071317

79 08 06 032

operations and in response to faults. The graphical representation used is based on the LOGOS system description schema, which uses two associated graphs, a control graph and a data graph, to represent a process or activity. The study extends earlier functional/graphical studies of individual, isolated fault tolerance mechanisms, to the assessment of a complex large scale digital system. The assessment begins by modeling top level DAIS control functions and includes DAIS master executive functions and top level functions from a potential DAIS application.

The assessment located fault tolerance problem areas involving multiple instances of control and diversion of control. While somewhat difficult to produce, the graphical model is a powerful tool for summarizing the fault tolerance of DAIS.

PREFACE

This report documents the use of a functional/graphical approach for the assessment of the fault tolerance of the Digital Avionics Information System (DAIS) currently being implemented at the Air Force Avionics Laboratory. The functional/graphical approach used was developed under the sponsorship of the Office of Naval Research and the Air Force Office of Scientific Research under contract numbers N00014-75-C-D011 and F44620-75-C-0053, respectively.

Much of the effort involved the application of the LOGOS system description facility developed at Case Western Reserve University. Much of the work in developing the graphical models presented in this report was done by Dr. Charles W. Rose, Robert A. Gingell, and Dennis A. Risen. Dr. Larry L. Kinney provided much of the effort of interpreting the operation of the DAIS executive, especially for bus recovery operations.

LT Ronald C. Durbin, AFAL/AAA-2 supervised the technical effort for the Air Force.

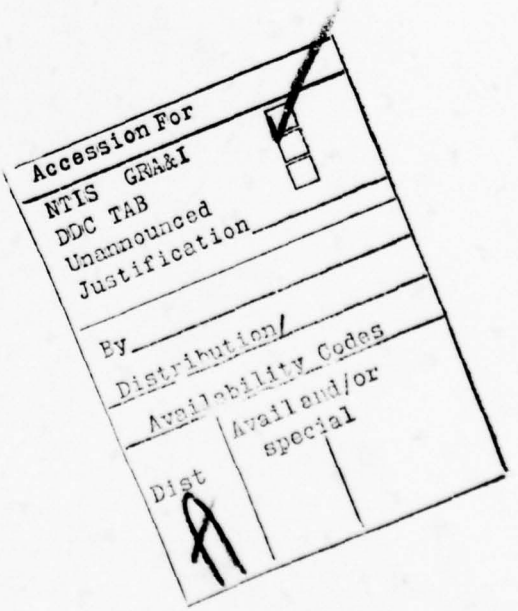


TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I

INTRODUCTION

CONTRACT OBJECTIVES

This report documents a case study in the use of a functional/graphical approach to the assessment of fault tolerance in digital control systems.

As indicated by its name, the functional/graphical approach has two complementary aspects: a functional approach to the characterization of fault phenomena and a graphical approach for the representation of the behavior of the system in response to faults. This new approach was developed to augment existing tools that cope poorly, if at all, with modern multicomputer fault tolerant system designs. We refer to the functional/graphical approach as a fault tolerance assessment tool rather than an analysis tool because it is at best an aid in an essentially heuristic process in its present form. Nevertheless, we believe that it has potential to be the nucleus of a systematic fault tolerant systems analysis procedure.

The vehicle for this case study is the Digital Avionics Information System (DAIS), a multicomputer digital avionics system based on the military 1553 digital multiplexing standard currently being implemented at the Air Force Avionics Laboratory. The DAIS architecture centralizes a number of the activities needed to implement avionics functions such as timing control, input/output, and task control in an integrated hardware/software core avionics system that can support a number of varied avionics applications.

DAIS is an example of a major new trend in avionics system architecture in which all digital avionics functions are performed by a single integrated set of computers and terminal interface units are configured to provide easily expanded computational capacity and a high degree of overall system availability through fault tolerance.

DAIS is a prototype for the next generation of avionics systems in which applications functions are distributed in an interconnected pool of digital computers and terminals. This new approach has enormous advantages and distinct problems in comparison to older approaches in which each function was permanently associated with a dedicated computer. The greatest advantage is the flexibility to redistribute applicable functions to balance the computational load and to accommodate growth in the life of the avionics system. This ability to redistribute functions can also be exploited for fault tolerance; critical functions can be relocated if the processor currently supporting them fails. Thus DAIS is one of the first of a generation of multicomputer avionics systems in which system reconfiguration will be a major mechanism for achieving high reliability and availability. DAIS provides a challenging vehicle for the trial of the functional/graphical fault tolerance assessment approach because of the substantial role the DAIS core hardware and software plays in maintaining the integrity of critical applications functions.

RATIONALE FOR FAULT TOLERANCE ANALYSIS

Fault tolerance is the ability of a digital system to continue to operate successfully despite failures in one or more system components. The overall goal is increased system reliability--more precisely system availability--

since the key concern of the user is that the system function correctly when the user needs it. The traditional approach to electronic system reliability has relied on rigorous component selection and screening, an approach that Avizienis has termed "fault intolerance,"[1] because it attempts to construct systems in which hardware failures never occur. Modern designs tend to use a judicious mixture of both fault intolerance and fault tolerance techniques.

Fault tolerance schemes first appeared in analog systems where linear summing techniques could be used to mask outputs from failed circuits. Fault tolerance mechanisms in digital systems originally focused on hardware faults, but as system functionality has become increasingly dependent on software, schemes to deal with software failures have emerged.

Fault tolerance in avionics systems has usually been implemented on an ad hoc basis within each separate subsystem. Now, with the emergence of integrated, centralized avionics systems such as DAIS, the central system architecture must provide an extremely high level of fault tolerance. Centralized avionics systems provide a potential source of catastrophic failure; if the core system fails, all functions implemented in it are lost. On the other hand, these systems allow application software to have access to data from a wide variety of alternate sources for comparison with and substitution for outputs from faulty external electronics. Thus the avionics digital computer has moved from a role as an internal component in selected

---

[1] A. Avizienis, "Architecture of Fault-Tolerant Computing Systems," 1975 International Symposium on Fault-Tolerant Computing, Paris, France, June 1975.

subsystems to a central role in an integrated system that is essential to mission success. Centralized integrated avionics systems such as DAIS have become as much communication networks as computing facilities. They are the cement that holds the variety of avionics subsystems together and they are the central interface for the pilot to control them.

The flexibility for reconfiguration is not bought without a price, however. Interconnected processors can suffer from mutual interference arising from contention between the processors for common communication resources, or from faults that cause a processor to generate a stream of meaningless messages to its neighbors. Hopkins[2] aptly calls such a processor a "babbler." The tradeoff between reconfigurability and isolation is always present in integrated systems such as DAIS; the rich interconnection network needed to allow effective function partitioning for load sharing and reconfiguration poses a problem in the isolation of failed units.

A more pervasive issue posed by integrated multicomputer systems such as DAIS is the location of the overall control of the system. The major choices are centralized or decentralized system control.

Systems with centralized control assign a single agent in the system to be aware of the overall state of the system, including the health of each major system unit. This agent monitors the performance of major system

---

[2] A. L. Hopkins, Jr., "Design Foundations for Survivable Integrated On-Board Computation and Control," Proceedings of the Joint Automatic Control Conference, San Francisco, CA, pp. 232-237, June 1977.

components and receives error information to allow it to detect faulty units. Generally, this agent is provided with the facilities to enforce any reconfiguration decisions it might make upon the rest of the system. Clearly, this agent, be it hardware, software, or a combination of the two, is critical to the operation of the entire system.

Figure 1 outlines the DAIS core system architecture. A DAIS core system is comprised of from one to four digital processors, each of which may have up to 64 K words of memory. Each processor is interfaced to two identical MIL-STD 1553A digital data busses via a specially designed Bus Control Interface Unit (BCIU). If no alternative agent is provided, then failure of the centralized controller is equivalent to failure of the system. Such unreplicated critical resources are usually termed "hard core."

The decentralized approach to system control distributes information on the overall system state among several agents. Each agent makes a decision on the health of the system and the desired configuration based on information passed to it. Such an approach can keep vital functions operating in a portion of a system in which some controlling agents have failed. The unsolved problem in decentralized systems is how to maintain a consistent system configuration when no single deciding agent has a complete, up-to-date picture of the entire system. Decentralized systems suffer from a variation of the Uncertainty Principle; by the time the various control agents have exchanged enough information to establish consistent state information, the information is no longer completely valid. Decentralized systems also are difficult to synchronize since they have no single authoritative time reference in the system.

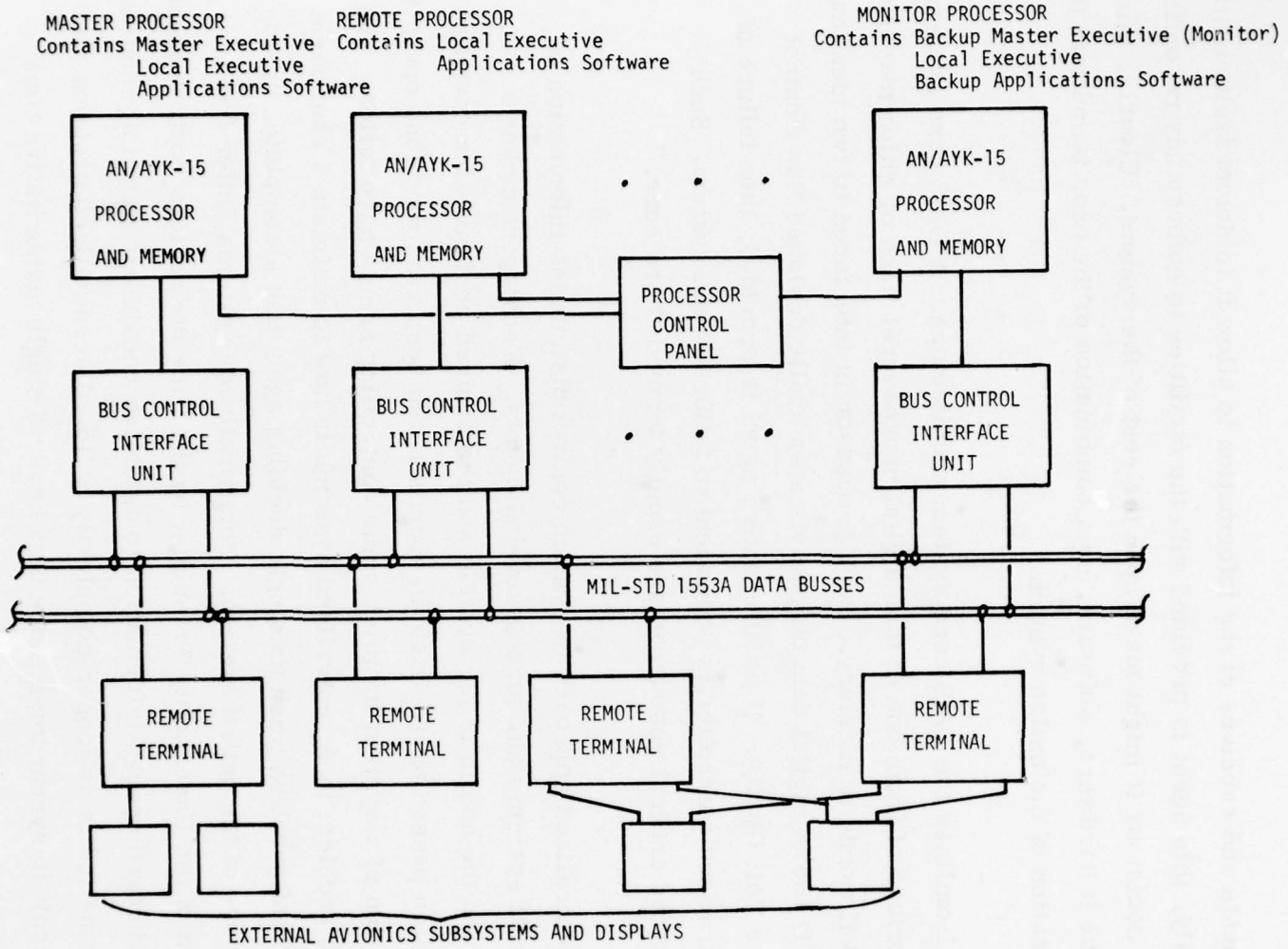


Figure 1. DAIS Core System Architecture

The DAIS architecture uses the centralized approach. All system control and timing decisions (except the decision to reconfigure, a decision reserved solely for the pilot) are made by a single software master executive. To avoid total reliance on this single complex piece of software, an additional copy of the master executive, the monitor executive, is provided in fault tolerant DAIS configurations. Only one executive controls the system. The master normally maintains control, relinquishing control to the monitor either explicitly, by issuing a "takeover" message, or implicitly, by failing to perform certain operations monitored by the monitor.

DAIS relies on a specialized "hard core" hardware unit for overall system control, the pilot's Processor Control Panel (PCP). This panel allows the pilot to exclude any processor and its associated BCIU or any bus from the active DAIS configuration by removing power to the selected unit. However, the PCP cannot specify that a particular processor be included in the remaining configuration; failures in processor self tests or memory sum checks may still exclude a powered processor from a new configuration.

PROBLEMS IN FAULT TOLERANCE ANALYSIS

Despite the need for increased confidence in electronic systems, fault tolerance analysis is becoming increasingly difficult as systems grow in complexity.

First, consider the problem of defining a fault. The IEEE Standard Dictionary of Electrical and Electronics Terms[3] defines a fault as "a physical condition that causes a device, a component, or an element to fail to perform in a required manner; for example a short-circuit, a broken wire, an intermittent connection." Notice the emphasis on physical phenomena that cause permanent changes in system behavior. This is consistent with the design of many analog control systems where major functions are directly implemented with dedicated physical devices so that physical faults are directly related to changes in system performance.

A more comprehensive term in the same dictionary is failure, defined as "the termination of the ability of an item to perform its required function." The term failure is no longer tied to physical phenomena. A failure is in some sense a "functional fault." In old-style direct-implementation systems, there is still nearly a one-to-one relationship between faults and failures; in newer, more complex digital systems, we may encounter system failures without physical faults.

The direct relationship between faults and system performance has been the main emphasis in most fault tolerance analysis work to date. In analog systems the main concern has been with the sensitivity of system outputs to the radical changes in the performance of individual elements that could

---

[3] IEEE Standard Dictionary of Electrical and Electronics Terms, IEEE Standard 100-1972

occur if they become faulty. Work in digital systems has concentrated on the changes to logical behavior induced by failed gates and on techniques to detect and mask out such failues.

Two interrelated technology developments have recently had profound effects on the implementation of modern electronic control systems; they are large-scale integrated (LSI) circuits and digital computers. LSI circuits have eliminated many of the external interconnections in older designs, one of the major sources of faults. They have allowed complex units such as digital computers to be constructed from a reasonably small number of physical devices with an overall reliability comparable to much simpler circuits in previous technologies. The present economics of integrated circuit technology dictate the fabrication of systems from a limited number of general purpose units. The devices may be extremely complex as in microcomputers, but they must have wide applicability.

Digital computers reinforce the effects of LSI. They create a market for large quantities of identical logic units such as memory arrays, and they provide an extremely versatile general purpose vehicle for implementing electronic systems. Consequently, modern control systems designers are beginning to use complex general-purpose components (digital computers), that compare favorably in overall reliability with older and simpler circuit elements, to implement systems with stringent reliability requirements.

The effect of these changes on fault tolerance strategies has been profound. Designers no longer have access to low-level circuit elements such as logic gates. In many cases, they may not even know the gate configuration used by the LSI designer in implementing a particular function. Thus a number

of low level techniques for coping with individual physical faults are no longer available. This may be appropriate since the concentration of large numbers of circuit elements raises the liklihood that a single physical fault can simultaneously affect several functional elements. Instead of combinatorial techniques, fault tolerance implementers are turning to algorithmic techniques that rely on the versatility and computation capacity of digital computers to detect, isolate, and circumvent failures. Thus a designer may use a software algorithm to circumvent the failures that may result from hundreds of different physical faults. As the designer becomes increasingly insulated from physical circuit details, he increasingly designs against failures, as defined earlier, rather than faults.

Expansion of the definition of faults from physical faults to failures or functional faults allows a hierarchical approach to fault tolerance; instead of constructing mechanisms to deal with individual physical device failures, a designer may create mechanisms at one level that deal with small portions of the system and superimpose broad, high-level mechanisms covering a major part of the system at a higher level.

The DAIS architecture is typical of many new fault tolerant system architectures; the avionics applications designer is provided with a collection of somewhat general-purpose computers connected with a communications network of busses and bus control interface units that allow the designer to distribute functions among several computers and to switch between software configurations, if required. The application designer is encouraged to rely on error handling mechanisms built into the core system to deal with

transient or localized errors and to assume that automatic configuration management mechanisms will deal with high-level failures by switching to an alternate processor/software combination.

While the application designer will continue to be responsible for exploiting application-specific information to detect and circumvent functional faults, he will have to rely on features built into the DAIS hardware/software core system for much of the fault tolerance of the system.

The overall fault tolerance of the DAIS-based avionics will depend upon two factors: the effectiveness of application-specific fault tolerance mechanisms and of the DAIS core system. While fault tolerant DAIS avionics systems have been proposed and are in various stages of design, none have been implemented yet. Thus this study will assess only the effectiveness of fault tolerance features in the DAIS core system. We will briefly discuss the impact of some faults on a subset of the avionics functions performed in a DAIS-based design for an Integrated Digital Advanced Medium STOL Transport (IDAMST) avionics system and compare a reconfiguration scheme proposed for this system with the DAIS baseline reconfiguration scheme. However, we will not assess any fault tolerance schemes unique to the IDAMST application.

The mechanisms analyzed in the DAIS system deal with failures of system elements to perform correctly and are concerned only indirectly with specific physical fault phenomena. Thus this assessment could be referred to as a failure tolerance assessment rather than a fault tolerance assessment. However, because the differentiation between faults and failures given in the standard dictionary is not widely recognized and because the work leading

to this effort has introduced the term "functional fault," we will use the more pronounceable term "fault tolerance." In the remainder of this report we will consider a fault to be the failure of a system component to perform as specified.

DEVELOPMENT OF A FUNCTIONAL/GRAPHICAL ASSESSMENT APPROACH

The functional/graphical fault tolerance assessment approach used in this study resulted from a concern at Honeywell that existing techniques for evaluating the behavior of fault tolerant digital computer systems did not adequately deal with two major design trends: the use of multiple computers operating concurrently to provide protective redundance, and the increasing use of software mechanisms to implement these mechanisms.

In recognition of the new role of software in fault tolerance, a search was begun for an approach that could deal with system designs in a structured top-down fashion, in keeping with newly learned lessons in managing software complexity. The need to deal with dynamic interactions between concurrent control streams suggested that the approach should be based on techniques for the specification and analysis of concurrent systems. An attempt was made to adapt as many existing system specification and analysis results as possible to the fault tolerant system analysis problem.

Since the common denominator of many current hardware and software representations is the directed graph, and since directed graphs are an extremely natural way to show structural relationships, a graphical approach was chosen. Under Office of Naval Research and Air Force Office of

Scientific Research sponsorship, several potential graphical representation schemes were considered and two were chosen for further evaluation; LOGOS, a modeling system developed at Case Western Reserve University, under ARPA funding as a design aid for large-scale computer systems[4] and Petri Nets, a form of directed graph conceived by Carl A. Petri in Germany in 1962 to describe communications with automata.[5] Both LOGOS and Petri Net models supplement conventional directed graphs with a labeling or marking to record dynamic state information. We refer to these augmented directed graphs as labeled graphs.

LOGOS and Petri Net graphs were used to explicitly model several common fault mechanisms, fault detection schemes, and fault recovery schemes. This first crude application of labeled graphs demonstrated that these models could effectively represent many common faults in digital structures. It was also observed that identical faulty structures could appear at several levels of implementation. For example, the same faulty pipeline structure could appear in a hardware request-acknowledge logic chain, a software sequence, or a series of high-level system commands from an operator. This observation led to the development of a functional approach to fault definition in which failure of a hardware or software system element to

---

[4] C. W. Rose, "LOGOS and the Software Engineer," Proceedings of the AFIPS Fall Joint Computer Conference, December, 1972, pp. 311-323.

[5] C. A. Petri, "Kommunikation mit Automaten," Schriften des Reinsch-West Falischen Institute, Instrumentelle Math. und der Universitat Bonn, No. 2, Bonn, 1962.

perform correctly replaced physical device failure as the criteria for defining a fault. Jack and Heimerdinger summarizes these results.[6]

Subsequent activities defined and described six classes of functional faults and defined several graphical properties of labeled graphs related to fault phenomena. During this work, it was discovered that neither Petri Net nor LOGOS models could adequately describe fault tolerant phenomena in real-time systems without the addition of a notation for time.[7] Subsequent research efforts developed approaches for reducing labeled graph models to create a higher level model while still retaining important graph properties.[8]

While all of these efforts reinforced the conclusion that a systematic approach to fault tolerant systems analysis could be built upon a combination of labeled graphs and functional fault classes, all efforts to date involved detailed studies of individual fault phenomena and an important question remained unanswered; could this approach be successfully applied to a large-scale complex system? This analysis effort addresses that question.

---

[6] L. A. Jack, W. L. Heimerdinger, M. D. Johnson, "Theory of Fault Tolerance--1974-5 Annual Report," Contract No. N00014-75-C-0011, Honeywell Systems and Research Center, Minneapolis, Minnesota, September, 1975

[7] W. L. Heimerdinger, L. A. Jack, "A Graph Theoretic Approach to Fault Tolerant Computing--1975-76 Annual Report," Contract No. F44620-75-C-0053, Honeywell Systems and Research Center, Minneapolis, Minnesota, March, 1976.

[8] L. A. Jack, W. L. Heimerdinger, Y. W. Han, L. L. Kinney, "Theory of Fault Tolerance--1976 Annual Report, Volume I," Contract No. N00014-75-C-0011, Honeywell Systems and Research Center, Minneapolis, Minnesota, December, 1976.

SECTION II

TECHNICAL APPROACH

GRAPHICAL MODEL

Section I discussed the following advantages of graph models for the description and analysis of fault tolerant phenomena: they are declarative rather than procedural; they reflect the actual structure of the algorithms being modelled; they are precise and unambiguous; they are concise; and particularly important to the study of distributed intelligence systems, they allow concurrency to be described naturally and explicitly. In this section the specific graph model used to perform the fault tolerance assessment of DAIS is introduced, together with the functional fault classification scheme employed.

The graph model chosen is based upon the system description facility of the LOGOS design environment[9] which was developed at Case Western Reserve University. A copy of the LOGOS system is available on the AFAL AVSAIL (Avionic System Analysis and Integration Laboratory) computer facility.

---

[9] Rose, op. cit.

LOGOS Description Language is a directed graph/string representation system based fundamentally on the works of Petri,[10] Slutz,[11] Karp and Miller,[12] and Luconi,[13] and is extended to cover the needs of a hardware/software system design automation environment. A complete description of the representation may be found in work done by Rose, Bradshaw, and Katzke.[14,15,16]

---

[10] C. A. Petri, "Kommunikation mit Automaten," Schriften des Reinsch-West Falischen Inst., Instrumentelle Math. und der Universitat Bonn, Nr. 2, Bonn, 1962.

[11] D. R. Slutz, "The Flow Graph Schemata Model of Parallel Computation," Doctoral Thesis, M.I.T., Cambridge, Mass., September 1968.

[12] R. M. Karp and R. E. Miller, "Parellel Program Schemata," Journal of Computer and System Sci. 3, pp. 147-195, 1969.

[13] F. L. Luconi, "Asynchronous Computational Structures," Doctoral Thesis, M.I.T., Cambridge, Mass., January 1968.

[14] C. W. Rose, "A System of Representation for General Purpose Digital Computer Systems," Jennings Computing Center Report No. 1113, Case Western Reserve Univ., Cleveland, Ohio, August 1970.

[15] F. T. Bradshaw, "Structure and Representation of Digital Computer Systems," Jennings Computing Center Report No. 1114, Case Western Reserve Univ., Cleveland, Ohio, January 1971.

[16] S. W. Katzke, "A Graph-Oriented Data Structure Language," Jennings Computing Center Report No. 1126, Case Western Reserve Univ., Cleveland, Ohio, June 1973.

A description of a process is formed by two directed graphs: a control graph (CG) and a data graph (DG). Together they form a schema called an "activity." An example activity is shown in Figure 2.

The control graph represents the structure of the control flow; it sequences, initiates, and synchronizes the data operations in the data graph. The data graph represents the data structures of the algorithm and the transformations

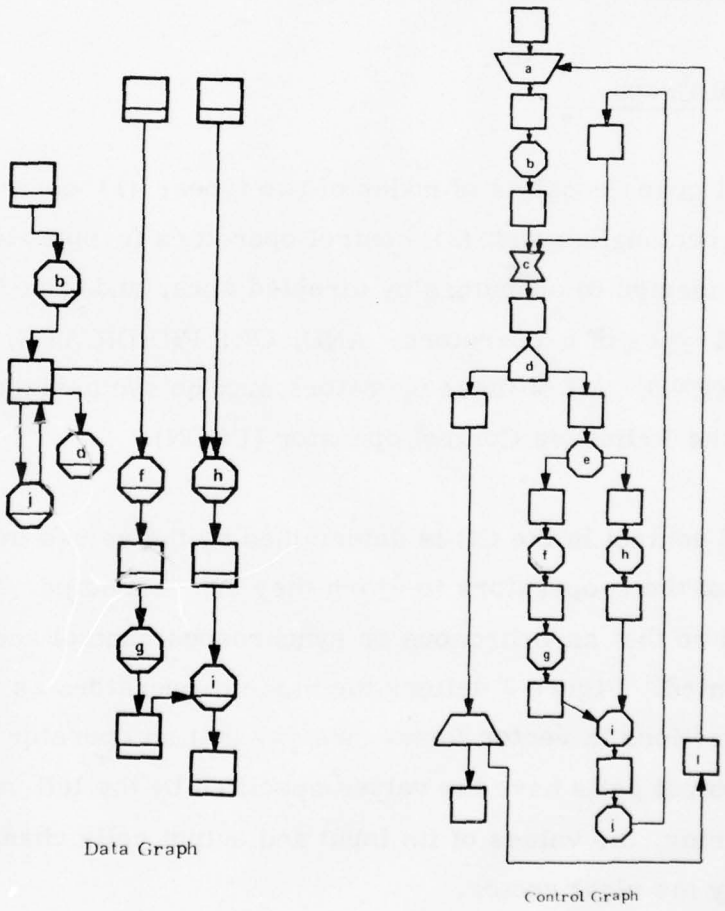


Figure 2. Example of an Activity

made upon their values. A data operator (transformation) is enabled to occur when its associated control operator is enabled to occur. The association is indicated either by a dotted arc from the control operator to the data operator or by giving both operators the same label. The separation of control and data allows the specification of the data structures and data operations to be essentially declarative and independent of the eventual sequencing. Thus, various sequencing schemes can be evaluated without totally redefining the data structures involved.

The Control Graph

The control graph consists of nodes of two types: (1) control-cells (c-cells), denoted by rectangles; and (2) control operators (c-operators). C-cells must be connected to operators by directed arcs, and *vice-versa*. There are several types of c-operators: AND, OR, PREDICATE, BLOCKHEAD, and BLOCKEND. All of these operators may be synthesized from a single operator, the Primitive Control operator (PCON).

The flow of control in the CG is determined by the values in the c-cells and the nature of the c-operators to which they are connected. The c-operators are defined so that asynchronous or synchronous control and data flow can be represented. Figure 3 defines the basic c-operators as well as their transfer functions in vector form. We say that an operator "fires" when its input and output cells have the values specified by the left vector, and as a result of firing, the values of its input and output cells change to the values specified by the right vector.

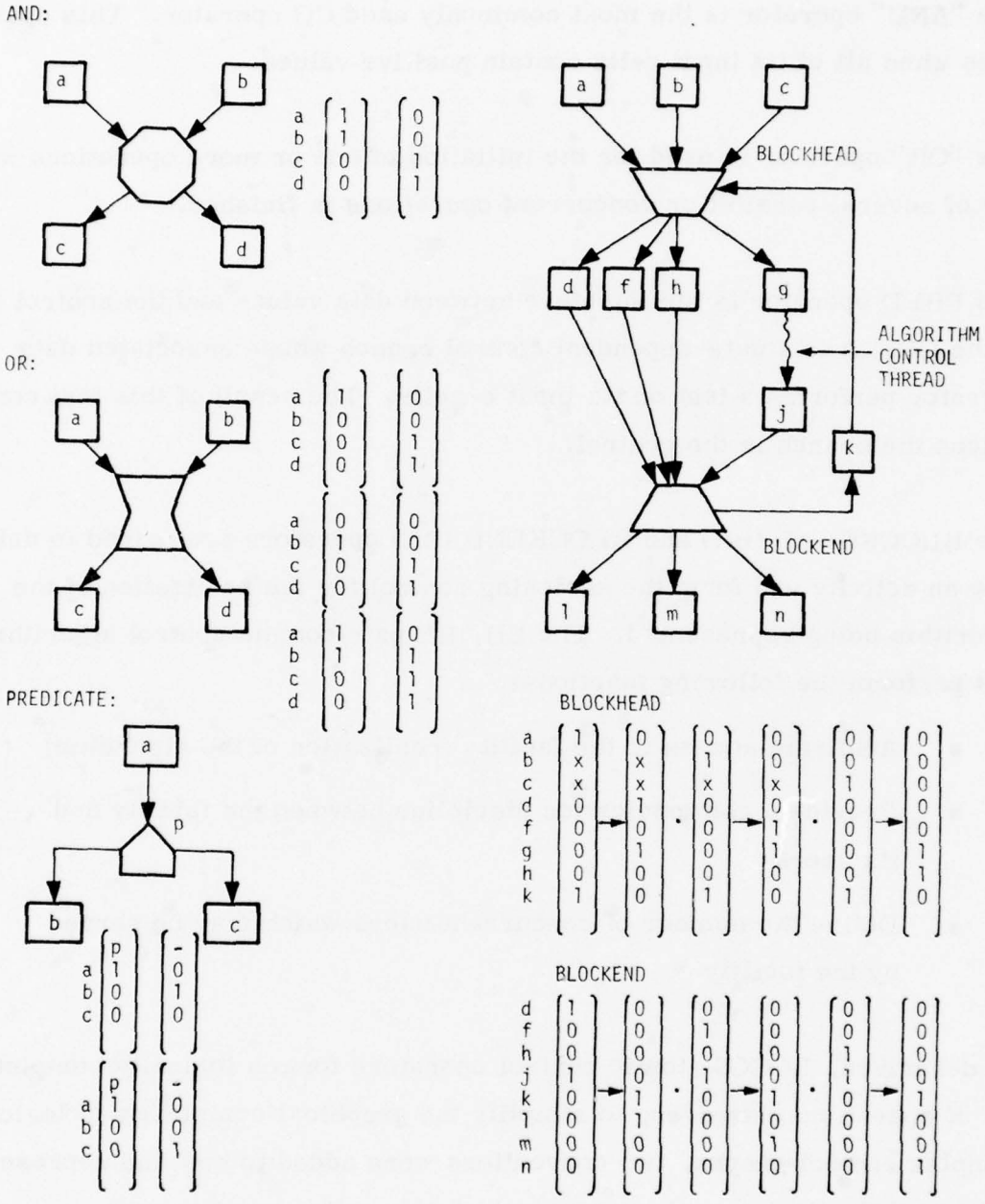


Figure 3. LOGOS Atomic Control Operators

The "AND" operator is the most commonly used CG operator. This operator fires when all of its input cells contain positive values.

The "OR" operator is used for the initiation of one or more operations when any of several parallel or concurrent operations is finished.

The PRED operator is the interface between data values and the control flow in the CG. It is a data-dependent control branch whose associated data operator performs a test on its input c-cells. The result of this test conditions the branch in the control.

The BLOCKHEAD (BH) and BLOCKEND (BE) operators are paired to delineate an activity and form the enclosing control for the realization of the algorithm being represented. The BH, BE pair contain control algorithms that perform the following functions:

- Arbitrate access to the facility (realization of the algorithm)
- Provide a communication discipline between the facility and its users
- Define the number of concurrent users which may be served by the facility

As described, LOGOS atomic control operators form a logically complete set of operations; however, to simplify the graphical complexity of logically complex control graphs, two conventions were added to both the representation and the design environment: read-only arcs and zero arcs.

A read-only arc is denoted by arrows on both ends and may be used only between c-cells and c-operator inputs. This is a shorthand notation for connecting a c-cell as both an input to, and output from, a c-operator. Thus, the effect on the c-cell of the c-operator firing is to instantaneously decrement and increment the cell, leaving it unchanged.

A zero arc is denoted by dotting the arc. It, too, can only be used as an input arc to a c-operator. Its effect is that of inversion; rather than requiring a "one" in the connected c-cell to satisfy the enabling condition for its c-cell input, the zero arc specifies that a "zero" in the c-cell is the required value.

Thus, a LOGOS AND operator can have regular or "one" arcs, read-only arcs, or zero arcs connected to its inputs, resulting in an arbitrary enabling condition. The only constraint is that at least one input arc to all LOGOS operators must be a regular arc in order to prevent multiple initiations of the control operator and its associated data operators without changes to the data operator inputs.

If an OR operator has a single output and does not have a data operator associated with it, and if the input paths to the operator are mutually exclusively enabled, then the operator may be removed by the following procedure:

1. Remove the c-cells which are inputs to the OR operator,
2. Connect the "dangling" output arcs from the OR operator predecessors to the output c-cell of the OR operator, and
3. Remove the OR operator.

In this way, the remaining c-cell will be set to a "one" whenever any one of the mutually exclusive control operators of which it is an output fires.

If a single c-cell is shared as an input cell by two or more control operators, one of the following conditions will hold when the c-cell contains a "one:"

1. None of the connected operators is enabled,
2. Exactly one of the connected operators is enabled, or
3. More than one of the connected c-operators is enabled.

The results in the first two cases has already been dealt with in the c-operator transition vectors shown in Figure 3. In the third case, however, a conflict or race exists; only one of the multiple operators can fire because the "one" in the c-cell is a consumable resource. Unless the race is resolved by a timing constraint (discussed later in this section), the operator to be fired is selected at random. Control graphs which are constructed so that a race never occurs are called persistent. Another class of control graphs is called weakly persistent. Weakly persistent control graphs may contain races, but are constructed so that any operator which loses a race (becomes temporarily disabled) is again enabled to fire and does so. Only control graphs which are at least weakly persistent can be analyzed with the LOGOS analysis package.

The Data Graph

The semantics of the formal LOGOS data graph are specified graphically and textually. The graphical portion of the semantics is the topology of the data graph. It is formed by the interconnection of data cells (domain or

input values) to data operators and data operators to data cells (range or output values) and represents the structure of the data operations. The textual description is a graph-oriented data structure language.[17] This language allows the extensible definition of data structure types or modes and the declaration of instances of these structures. It is also used to define the expected types of input or domain data structures and results or range data structures of a data operator. Textual data graph semantics were not used in the DAIS model and therefore are not discussed in this report.

Hierarchical Design

If a large system on the order of DAIS were to be described using the LOGOS representation as introduced thus far, the graphs would quickly become unmanageably large and complex. A representation which requires "flat" or single-level modelling is clearly not suited to describing realistically-sized systems.

Graph models in general, and LOGOS in particular, are well-suited to hierarchical modelling in which the detail of arbitrarily complex sequential and/or parallel operations can be hidden when the design is being considered at a higher level, or when the structure of a particular operation is not of interest. This approach is consistent with structured design and software engineering practices.

---

[17] Katzke, op. cit.

Hierarchical representation in LOGOS is presently achieved by recognizing that, though a data operator is enabled, reads its inputs, performs its transformations, and writes its outputs in an arbitrarily long (but indivisible) operation, many operations have internal storage so that the reading of input, performance of transformations, and writing of output may be separated distinctly in time.

LOGOS allows the separation of initiation and termination of these complex data operators in the control graph. This is shown in Figure 4a. C-operator "$\overline{a}$" initiates the read portion of data operator "a" when it is enabled. When the input data values have been read, the data operator allows $\overline{a}$ to transform its input and output c-cells. When the input cells to "a_t", "4," and "3" are "one," the data operator is enabled to terminate and, upon completion of its computation, it writes its output cells and "a_t" and transforms its c-cells.

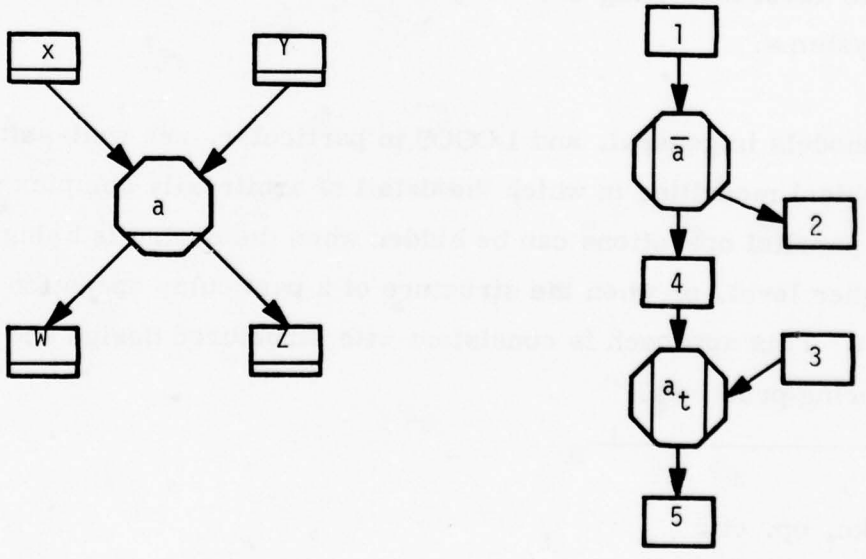


Figure 4a. Separation of Initiation and Termination

Figure 4b represents a pipeline constructed in this way. This data operator/control operator communication is easily achieved synchronously or asynchronously in hardware and by the sequential nature of most processors in software. Note that c-cell a in Figure 4b will contain a "one" whenever data operator "A" is in the process of computation. By relieving the constraint that c-cells may contain only "ones" and "zeros" and allowing non-negative integers, multiple initiations of operations with sufficient internal structure or storage is allowed. The hierarchical nature of the LOGOS design language lies in the data operator. Once it is possible to separate the initiation and termination of operators, it is then possible to construct arbitrarily complex computations and imbed them within the data operator. Such a data operator is referred to as a Complex Data Operator or CDO.

If data operator A in Figure 5a were a complex data operator, its internal structure might appear as in Figure 5b. Dotted control cells INIT, RT, WE, and TERM correspond to X, W, V, and Z in the main control graph. The dotted data cells in Figure 5b are formal parameters corresponding to the input and output data cells in Figure 5a.

Iterative refinement can be continued by "pushing on" data operator D in the expanded graph. In this way, the structure and function of each data operator can be elaborated on until it is in the terms of system-defined or user-defined primitive operations.

A similar structure exists for expressions called Complex Expression Operators or CEDs. A CED has no internal control graph.

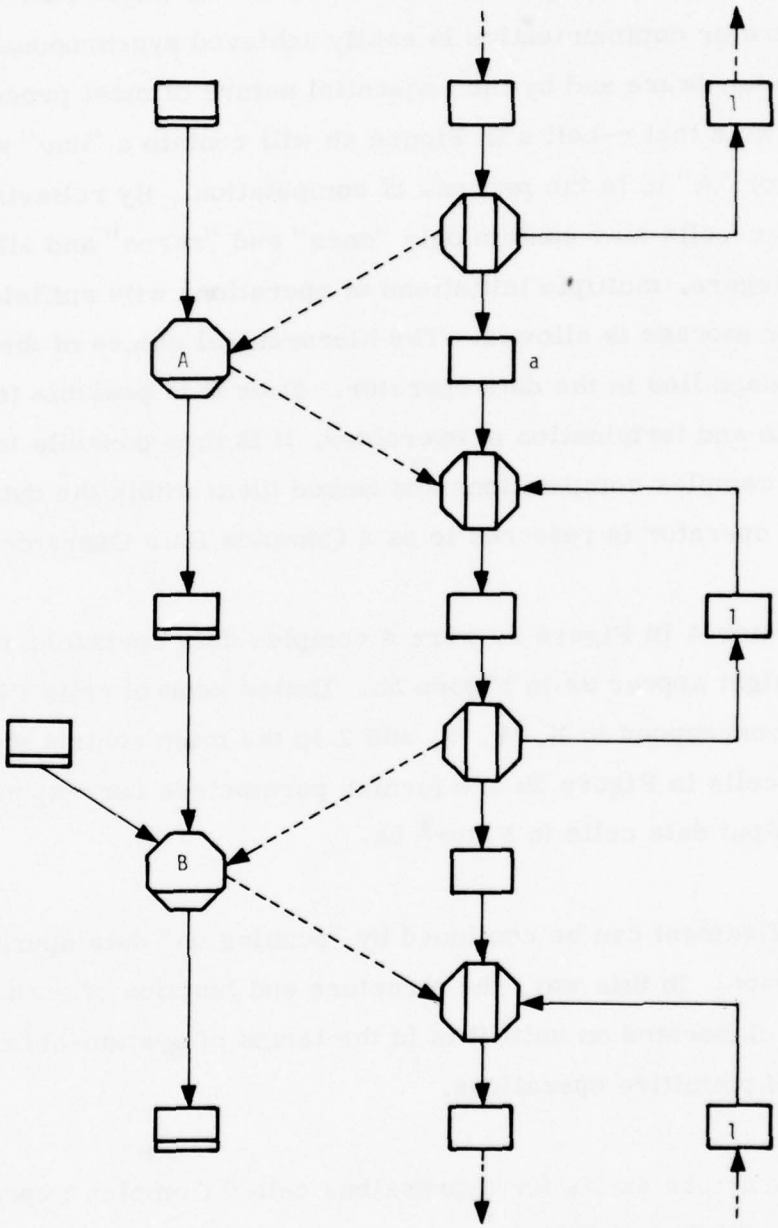


Figure 4b. Pipeline

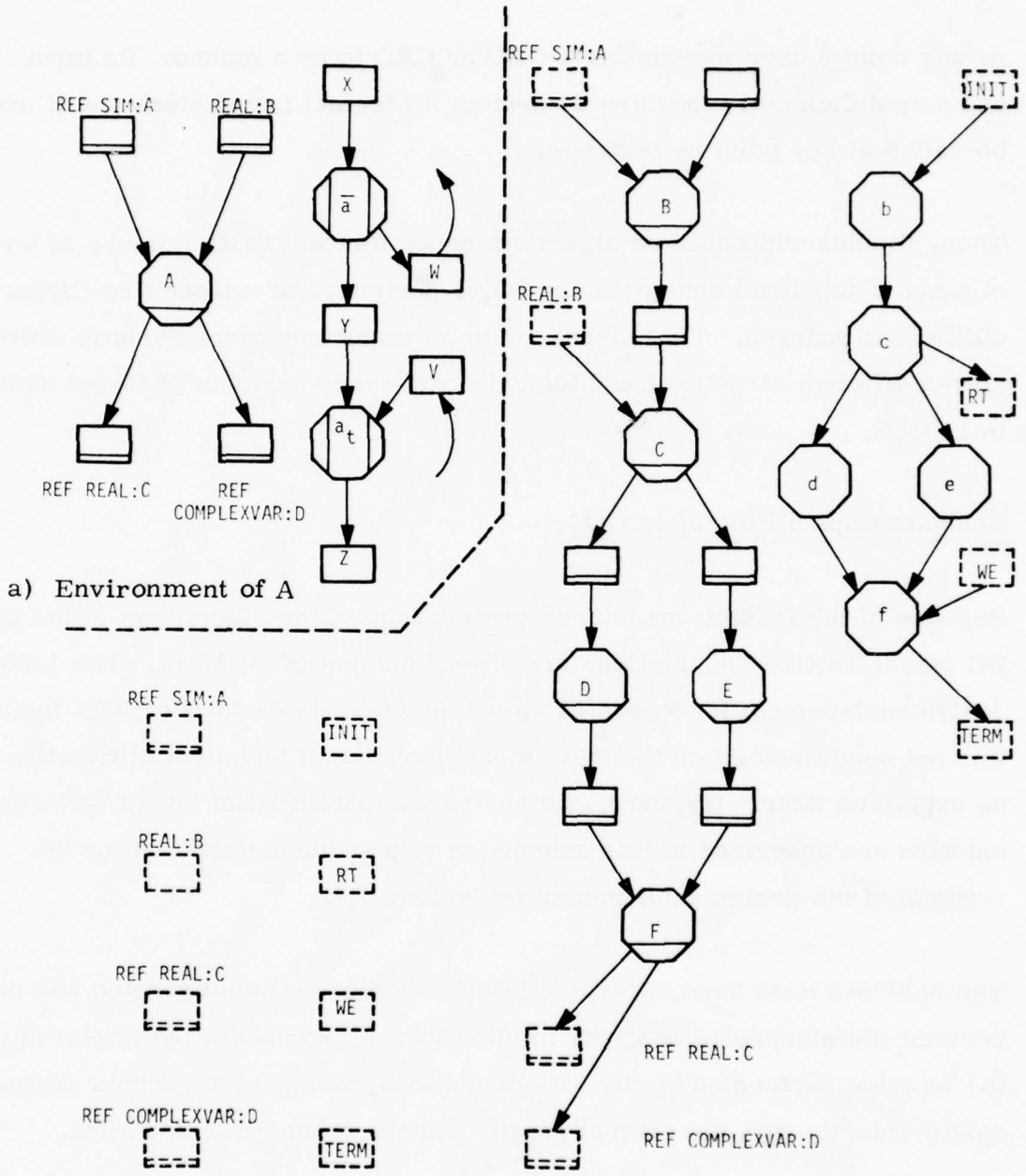


Figure 5. Hierarchical Design

At any point a user may define a CDO or CEO to be a macro. Its input and output data cell types are defined as its formal parameters and it may be called at any point by instance.

Thus, the description of an algorithm or system in LOGOS may be in terms of a set of top-level schemata and fully, partially, or unspecified CDOs, CEOs, and macros. This hierarchy is an important characteristic which allows efficient structural and logical consistency analysis of target systems in LOGOS.

Analysis Capabilities of LOGOS

Because of the explicit structural descriptions of the algorithms in the CG/DG pairs, LOGOS descriptions are a good subject of analysis. The LOGOS design environment incorporates an extensive analysis facility; this facility was not used to evaluate the DAIS model because of technical difficulties to be explained later. However, the analysis capabilities of the LOGOS representation are described in this subsection to provide background for the critique of the design environment in Section VII.

The analyses have been separated into two classes: uninterpreted and interpreted. Uninterpreted analysis implies that no interpretation is placed upon the function performed by the data operations; thus, uninterpreted analyses deal primarily with the control graphs and are topological in nature.

The existence of concurrency in an algorithm raises the question of determinacy--that is, whether multiple activations of a parallel activity with a given initial control state (contents of its cells) and data values will result in the same final values in a set of "result" locations.

A determinate activity is one that is free of pathological race conditions which can occur as a result of concurrency. Figure 6 contains examples of these races. Data operators a and b are enabled to occur in parallel and, depending which completes first, data cell x will have a different history or sequence of values written into it. Thus, data cell z will also have a different time history of values. This is called a write-write race. In the same graph, d-operations b and d are concurrent and d-cell w will have different time histories if b writes into y before d reads it, or vice-versa. This is a read-write race. Both races are well known to both software and hardware designers.

A scheme proposed by Karp and Miller[18] (using a vector addition system) has been modified in LOGOS to answer the questions of determinacy by creating the tree of attainable states representing the dynamic behavior of the graphs.

Two other pathological control topologies are shown in Figure 6. Control operator e is a predicate and since f is an AND, f will never be enabled to fire. This is called a hung state and can occur in a number of very subtle ways. Since c-operator b may activate the OR and exit the BLOCKEND, it is possible that other control/data operator pairs might be enabled or computed after the activity is deactivated or even reinitiated. This pathology is called proper termination. The same basic vector addition analysis system is used to test for these conditions as well as for "repetition freeness" (a necessary condition for determinacy) and recursion in a system of activities.

---

[18] Karp and Miller, op. cit.

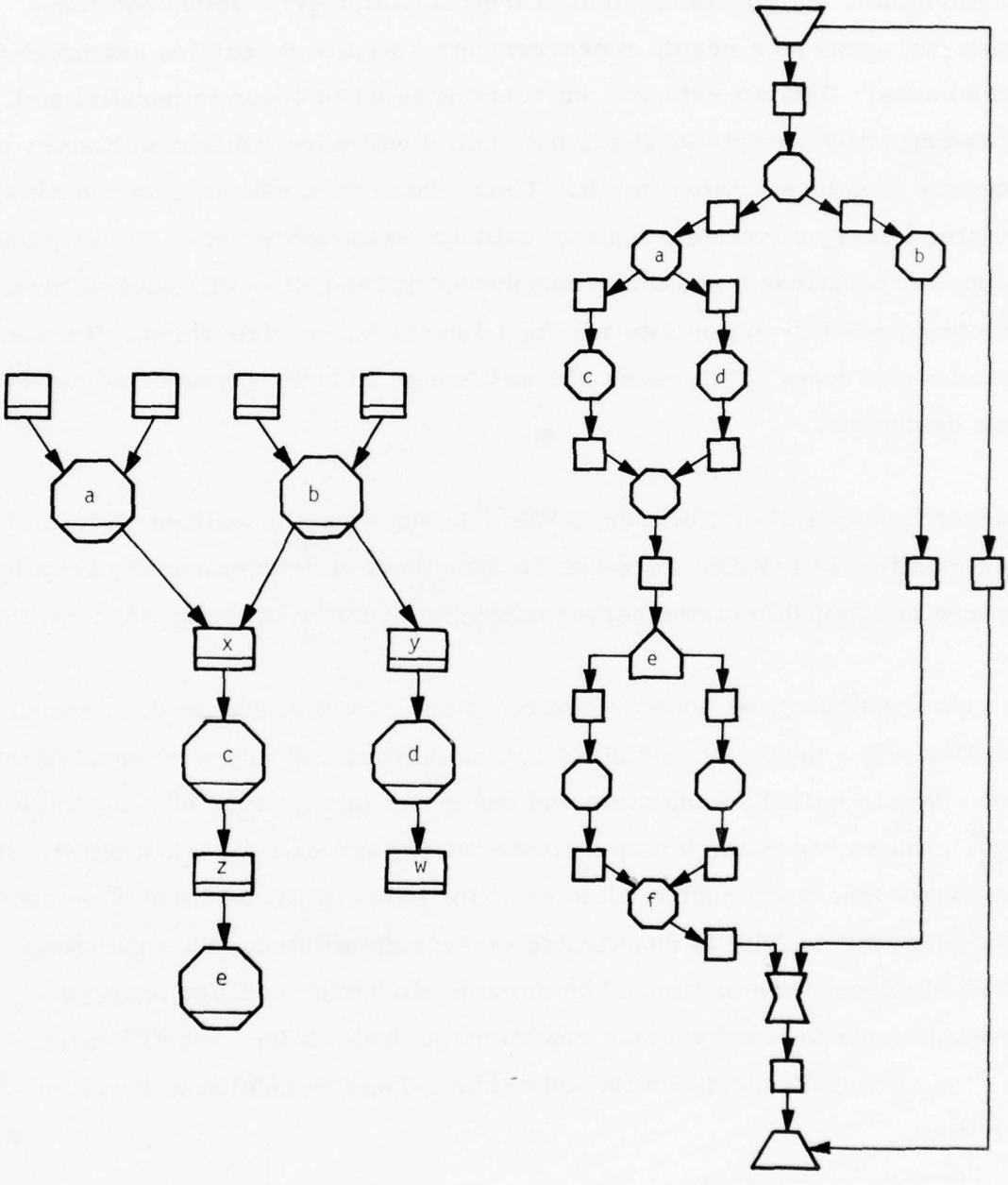


Figure 6. Anomalies Example

LOGOS uses the hierarchical structure of target system specification to perform its analysis in an incremental way. If a CDO or CEO has no internal structure defined at the time of analysis, it is analyzed as a primitive. When its structure is defined, analysis of its global environment is carried out only to the extent that its internal structure modifies its initial global behavior specification. Similarly, if the specification of a CDO, CEO, or macro changes, reanalysis is automatically performed only in areas where the effect of changes has propagated.

Interpreted analysis deals with the correctness of the algorithm used in implementing activities. Interpreted data analysis algorithms based on the functional attributes of the data operators are performed. For example, a data operator must access data structures of the appropriate type and compute results that correspond to the type of output data structure to which it is connected.

Although the current LOGOS system description facility is powerful and embodies most of the features required for the description and analysis of fault tolerance in real-time systems, it suffers from two critical shortcomings. First, it is an asynchronous representation. Synchroniety and explicit time relationships among operations cannot be expressed. Second, the definition of data operators is uninterpreted. That is, the function of a user-defined data operator at any level above a primitive is not formally represented. Therefore, it is not possible to determine the functional effect of a data fault.

To overcome these problems, two enhancements to the LOGOS description facility were developed on this contract. They were not implemented in the design environment. The first is the inclusion of explicit time in both the representation and analysis algorithms. The second is the inclusion of English language descriptions of the function of data operators. While the first is a solution to the timing problem, the second is an informal aid to the human analyst. A complete solution to the functional description problem is beyond the scope of this contract.

Timing In LOGOS Graphs

The existing LOGOS places no interpretation on operation timing. If its analyses verify a schema property, this property will hold for all assignments of execution timing for each component operation. Neither the representation nor the analysis recognize time in a quantitative manner. However, a real-time system must satisfy timing as well as algorithmic specifications. A time fault exists if an operation can occur too soon, too late, or not at all. A real-time fault tolerant system must sense time faults and take action to restrict their propagation. To remedy this shortcoming, the representation has been enhanced to include time interpretation, allowing analysis of time-dependent systems--for instance, systems that are correct under some, but not all, timing interpretations.

In the original LOGOS design, each control operator has an associated transition vector, which defines the enabling conditions and the state transition when the operator occurs. An operator may fire at any time while it is enabled, but is under no time constraints to fire (permissive). Once

enabled , an operator may become disabled by some other transition in the schema. Once disabled, an operator will eventually become reenabled and occur (weak persistent). Operator occurrence and the associated state transition are instantaneous.

The new enhanced control operator is not permissive. Once enabled, an operator will occur immediately unless immediately disabled by schema weak persistence. The state transition at operator occurrence is divided into two phases. The operator initiates by acquiring its resources (decrementing its input counters) and terminates by incrementing its output counters. The time between initiation and termination is given by a pair of time bounds of the form [L:U], where L is the lower bound and U is the upper bound.

Either L, U, or the entire [L:U] may be omitted. Default values for L and U are 0+. The [min:max] model for timing is exactly what is required for time fault analysis. It is also simple and closed under the necessary arithmetic calculations. The methods developed here could probably be applied to performance modelling, where time distributions rather than bounds would be more suitable.

Each primitive control operator has a specified or default execution time bound. Analysis computes a time bound for each schema for use in the collapsed representation analysis. The computed time bound [L(C) : U(C)] is compared with optional user-specified [L:U] for the schema. For validity, we must have $L <= L(C)$, $U(C) <= U$. The user specification, rather than the computed bounds, is used when analyzing the collapsed schema in its parent schema.

Data Graph Interpretation

With the current LOGOS system, one can model a control fault in a schema and analyze the schema to see if it still behaves properly. The interpreted control graph provides the mechanism to record the fault and its propagation. The data graph has not been interpreted because it may have large or even infinite state space, which would make most analysis questions undecidable. Thus, LOGOS currently cannot model or analyze data faults and their propagation.

Data interpretation calculus was limited to a simple English language description of the function of data operations. While not analytically useful, these descriptions do provide the human analyst with a functional model of DAIS. The consistency of CDO elaborations and the effect of functional faults in the data graph can be informally assessed using these descriptions.

SECTION III

DAIS SYSTEM DESCRIPTION

This section contains an overall summary of the DAIS core system hardware and software. It also summarizes key DAIS design concepts and terminology.

The DAIS program introduces a number of significant new concepts in avionics systems and, quite naturally, these concepts are being implemented in a phased approach. The current DAIS program is concentrating on the demonstration of a complete integrated avionics suite without system level fault tolerance features such as backup/recovery and reconfiguration. Consequently, a number of key design decisions remain to be made regarding these features. This assessment of DAIS is based on the current DAIS system design as specified in DAIS System Control Procedures and DAIS Mission Software Product Specification-- Executive, Volumes I and II.[19,20,21]

---

[19] "DAIS System Control Procedures," Air Force Avionics Laboratory, Document No. MA 201 200

[20] "DAIS Mission Software Product Specification--Executive, Local Executive," Part II, Volume I, Intermetrics, Inc., prepared for Air Force Avionics Laboratory, Document No. SA 201 302, 27 August 1976

[21] "DAIS Mission Software Product Specification--Executive, Bus Control," Part II, Volume II, Intermetrics, Inc., prepared for Air Force Avionics Laboratory, Document No. SA 201 302, 1 November 1977

These documents specify the system design and operational software configuration rules for DAIS and also specify the top level functionality of the high level fault tolerance mechanisms.

The DAIS design assessed here is consistent with these documents but necessarily includes assumptions regarding the most likely future implementation of some features. In reading this report, it is important that the reader keep in mind that the primary purpose of this contract was to demonstrate the application of the functional/graphical assessment approach to a nontrivial contemporary fault tolerant computer architecture. DAIS was selected as the assessment vehicle because it embodies many of the newer design trends in integrated real time control systems that are important to fault tolerance and are difficult to analyze.

THE DAIS CORE SYSTEM

Digital Avionics Information System is a real time, federated multiple processor system with fault tolerant features. A complete DAIS avionics system consists of an integrated hardware/software core system supporting a mission-dependent complement of sensors, actuators, and displays and mission-dependent application software. As shown in Figure 1, the DAIS core system hardware consists of from one to four AN/AYK-15 digital computers (processors), each with its own central memory, and several special Remote Terminal (RT) interface units which interface with sensors, controls, and displays. The processors and RTs are interconnected by a pair of 1553A multiplex data busses. The RTs connect directly to the busses while the processors are connected via bus control interface units.

DAIS is a centrally-controlled system in that a single processor is designated the master processor. The master processor is responsible for monitoring system status, error management, configuration management, and handling all bus traffic. The BCIU serving the master processor is responsible for initiating and monitoring all transactions on the bus. All processor/BCIU pairs are identical and any one can serve as the master processor/BCIU. BCIUs have direct memory access (DMA) to their associated processor's central memory and all bus traffic occurs via DMA. The master BCIU also reads instructions via DMA from the master processor central memory. Each BCIU is controlled by its processor via a programmed I/O interface which allows setting and reading BCIU internal registers.

The DAIS architecture is unique among multiple processor architectures in that every potential interaction between system elements must be defined at compile time instead of interpreted at run time. The user must define all tasks in a given DAIS configuration and specify all possible interactions between tasks and between tasks and external devices. Control interactions are specified by defining sets of events to govern task activation and by using a signal facility in the core software to transmit event information. Data interactions are specified by declaring common data pools (compools) and by specifying the rules under which remote copies of compools are updated from master copies by the core software. Every possible message for a given configuration is represented by a BCIU instruction in the master processor. A special external linkage editor and simulator system known the Partitioning Analyzing and Linkage Editing Facility (PALEFAC) allows the applications designer to generate the tables to specify the needed interactions and to verify the consistency of the resulting structure.

Thus DAIS systems are table driven and the structure of a particular DAIS mission software system is contained in tables as much as in algorithms. This feature of DAIS minimizes unpredictable interactions and makes system level analysis much more straightforward than it would be in a system that determines interactions at run time.

The DAIS executive is comprised of two distinct parts: a master executive and a set of local executives. A local executive resides in each processor including the master processor and is responsible for handling task management, messages, compool blocks, and exception conditions. The master executive resides only in the master processor and is responsible for all bus traffic and monitoring system status. The BCIU instruction lists are a part of the master executive.

Time in the DAIS system is measured in minor and major cycles. A major cycle is typically one second and consists of an integer power of two minor cycles, usually 128. Minor cycles are numbered beginning with 0 from the beginning of each major cycle. Minor cycles are used for detailed task and message control and the major cycle is the cyclic repetition period for the system. The master executive manages minor cycle events by transmitting a series of bus messages to all other processors signalling the beginning of a new minor cycle and specifying the number of the new minor cycle.

The system allows three different types of messages which vary in how their transmission is handled. There are synchronous, asynchronous, and critically timed messages.

Synchronous messages are assigned to particular minor cycle numbers and are automatically transmitted during each minor cycle which matches their assigned numbers. Numbers are assigned on a period and offset basis. For instance, a synchronous message with period of 4 and offset of 11 will be transmitted every fourth minor cycle beginning with minor cycle 11 of each major cycle.

Asynchronous messages are transmitted upon request. An asynchronous transmission can be requested by any task in any processor. The request is routed to the master executive either directly in the master processor or by requests returned in the status word of a bus transaction by other processors. Asynchronous messages have priority over synchronous messages.

Critically timed messages are asynchronous messages which may be assigned a minor cycle number and offset in milliseconds into the minor cycle. This is the only instance where a time unit finer than the minor cycle is recognized. Critically timed messages are only transmitted from the master processor to remote terminals.

At the beginning of each minor cycle the list of synchronous messages for that minor cycle is begun. If an asynchronous message request appears, the synchronous messages are halted, the asynchronous message is transmitted, and then the synchronous messages begin again. The synchronous messages for each minor cycle must be completed before the next minor cycle can begin, so synchronous messages are guaranteed to be transmitted in a particular minor cycle. If the synchronous messages are not completed

when the timer indicates the time for the next minor cycle to begin, the minor cycle is delayed. A theoretical minor cycle number is maintained based only on the clock, and an actual minor cycle number is maintained based on the clock and completion of the synchronous messages. The actual minor cycle number can lag behind the theoretical minor cycle number. If this lag gets too large, the master executive will take recovery action.

Compool blocks are the system buffer areas which hold message data. Compool block types correspond to message types; thus synchronous compool blocks buffer synchronous messages and asynchronous compool blocks buffer asynchronous messages.

The content of a single asynchronous message may be required in more than one processor, so there must be multiple copies of some compool blocks. Whenever a copy of an asynchronous compool is updated, all of the copies must be updated. The local executive in the processor originating the update must signal the master executive to send messages to update the other copies in other processors.

Tasks are coordinated with message activities by means of event signals. There are four kinds of system events:

1) A minor cycle event signals the beginning of a new minor cycle.

2) A task activation event signals the activation of a task.

3) A compool update event signals that the local copy of asynchronous compool has been updated.

4) An explicit event is a user defined event that can be explicitly signaled by a task via a signal statement in the program.

Each task can be assigned an event condition set such that its activation is conditioned on the occurrence of particular conditions. For instance, a task might be activated only during certain minor cycles, when a certain compool is updated, when another task signals it, or upon some combination of these events.

Application tasks are structured in hierarchical relationships. All tasks form a single hierarchy with strict father-son relationships. A father can schedule and cancel any or all of its descendents. Once scheduled, a task is controlled via event conditions and may be activated many times before being cancelled.

The great father of all tasks is called the master sequencer. Directly subordinate to the master sequencer are the request processor, configurator, and subsystem status monitor. The request processor manages requests from the pilot while subsystem status monitor tracks subsystem failures. Both of these tasks communicate with the configurator which controls the current system functionality by scheduling and cancelling families of tasks.

SECTION IV

DAIS FUNCTIONAL DESCRIPTION

Since the top-down approach to fault tolerance assessment concentrates on the high-level functions of a system, it is important to describe these functions succinctly and accurately. In this section, we use a series of graph models to represent the high-level functions of a DAIS system and their interactions. Key operators in the graphs indicate the DAIS activity corresponding to the operator. A narrative for each graphical model uses the model to explain the sequencing of the corresponding top-level functions.

TOP LEVEL DAIS SYSTEM CONTROL FUNCTIONS

Figure 7 is a top-level functional labeled graph of the DAIS system. Control flow is portrayed in the center of the figure, with data structures and data operators at the sides. Since the figure portrays DAIS functionality rather than details of implementation, the pilot is included as part of the system.

At the top level of abstraction, a DAIS avionics system is a collection of functions that are activated and cancelled by pilot inputs within mode and configuration constraints. These functions produce outputs, generally on system display units that are observed by the pilot and are used to determine new input requests. The center portion of Figure 7 shows the operation of the DAIS mode and configuration logic, which determines the allowable set of functions.

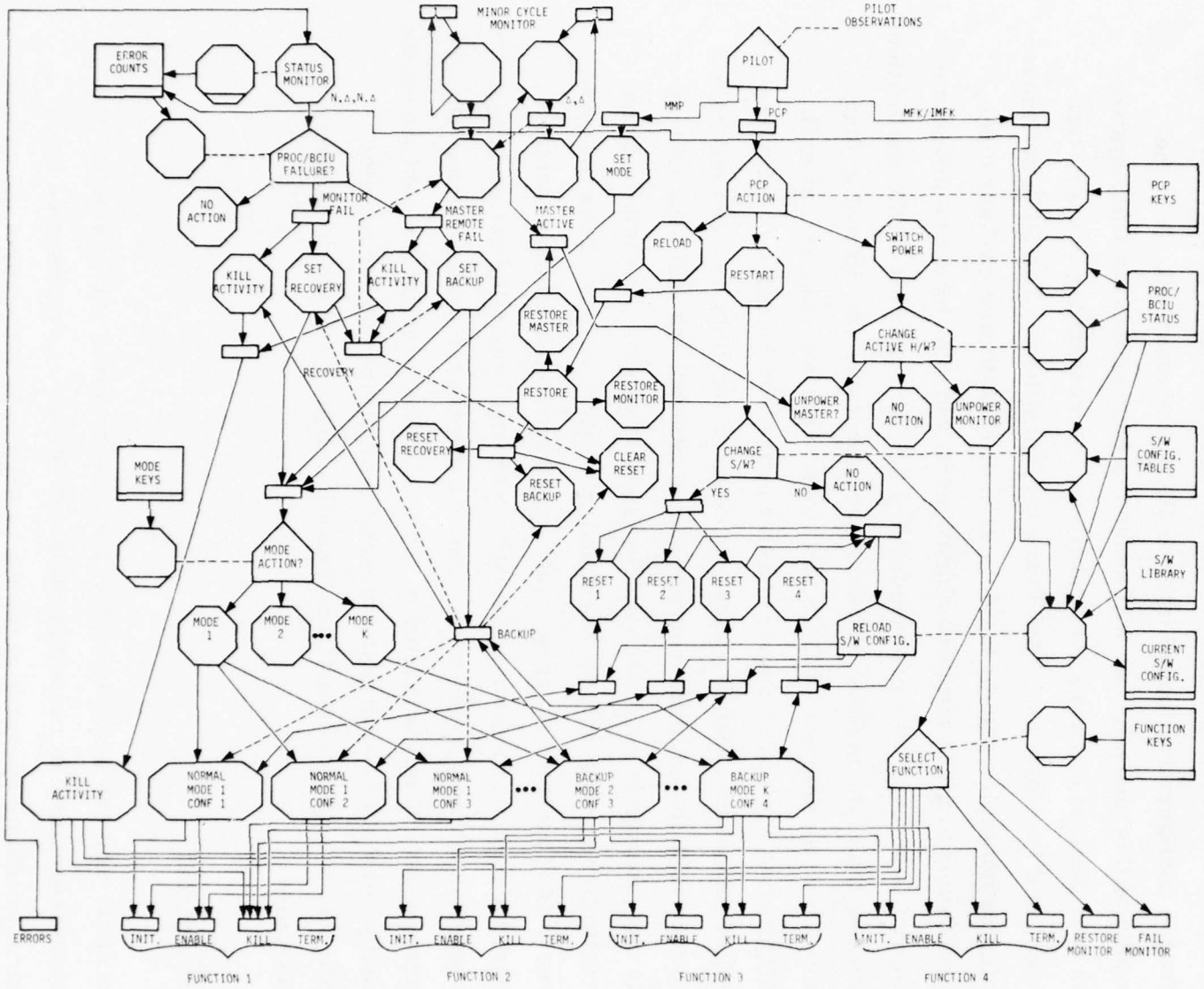


Figure 7. DAIS Control Top Level System Functional Description

The major determining factor in the functions available to the pilot is the software configuration loaded into the DAIS processor set. For the purpose of this analysis, we will assume a four-processor DAIS system with four distinct software configurations, corresponding to one-, two-, three-, and four-processor hardware configurations. The DAIS software configuration tables permit the definition of 15 software configurations, coresponding to all possible combinations in which at least one processor is active. However, the processors are indistinguishable to the applications software; thus, the distinct software configurations are limited to four. Since this is an analysis of DAIS fault tolerance, we assume that each configuration with more than one active processor includes at least one monitor processor. Since no logic to resolve conflicts between multiple monitors is included in the current documentation, we will assume that only one monitor is used when discussing this graph. The implications of multiple monitors will be discussed in a later section. Thus the two-, three-, and four-processor software configurations contain a master processor and one monitor processor, with any remaining active processors assigned as remote processors. The one-processor configuration, of course, contains only a master processor. The three larger configurations actually consist of two disjoint subsets; a single-processor monitor configuration and a "normal" configuration made up of the remaining processors.

Pilot Actions

The pilot, who has a major role in the operation of a DAIS avionics system (especially in reconfiguration activities), is represented by a control predicate operator at the top of Figure 7. No control inputs are

shown to this predicate since the pilot is essentially a source of spontaneous stimuli. Each pilot input is steered to a specific output from the pilot predicate operator, depending on data observed by the pilot from system outputs. This feedback is symbolized in Figure 8 by the pilot observation data operator that reads the contents of output data cells and governs the decisions of the pilot predicate operator. Here we use a graphical predicate operator to represent an external, but essential portion of the overall flow of control in a DAIS environment.

The pilot predicate operator has three outputs, each representing one of the three interfaces from the pilot to DAIS: 1) the pilot control panel (PCP), 2) the master mode panel (MMP), and 3) the multifunction keyboard (MFK). The integrated multifunction keyboard (IMFK) is functionally identical to the MFK for input purposes.

Pilot Control Panel Actions

The pilot interface with the greatest influence on system behavior is the PCP, which controls power to each processor/BCIU pair, thus determining the maximum processor/BCIU configuration possible. The PCP also provides reload and restart interrupts to all processors.

The pilot first determines the maximum possible software configuration by applying power to selected processor/BCIU pairs. Presumably the pilot provides power to all processor/BCIU pairs in the system--four in a maximum configuration of the current design.

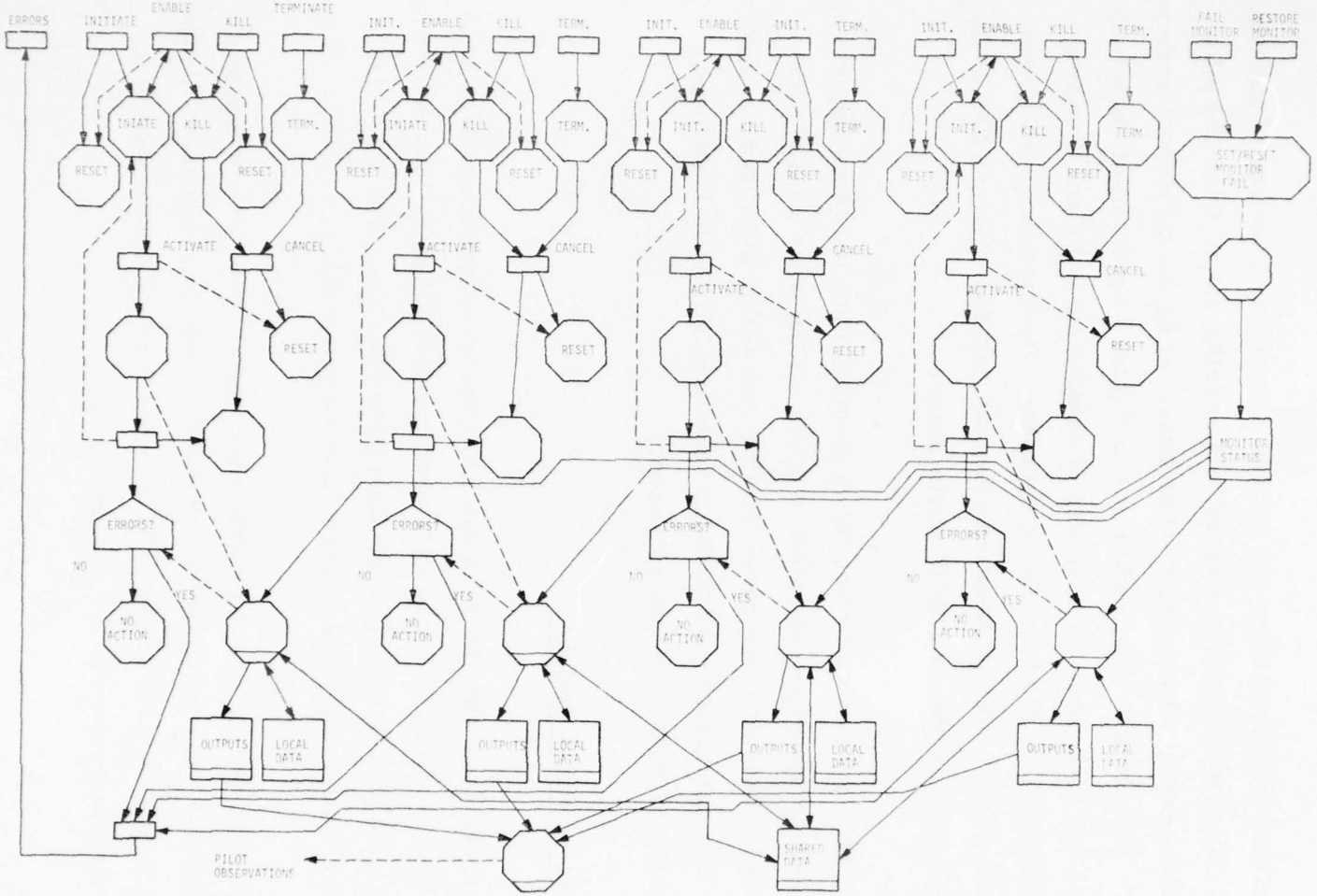


Figure 8. DAIS Top-Level Applications Functional Description

The current hardware set is determined by four GO discretes, one for each processor. Each processor sets its GO bit to ON if it can successfully complete a series of self-tests following the initial application of power. Once the pilot initiates the software configuration processes with either the LOAD/RELOAD or START/RESTART PCP switch, the processors leave their GO bits on until the processor or BCIU subsequently fails a self-test or is specifically disabled by another controlling processor. A GO bit set to OFF can only be restored by the removal and reapplication of power to the processor/BCIU pair.

After the system has been initialized, processor/BCIU power switching affects system functionality only if the switching action deletes a processor/BCIU pair from the currently active configuration subset. If the monitor processor or its BCIU is turned off, an enabling count is removed from the righthand operator of a pair of timed operators that represent the minor cycle monitor (the portion of the monitor executive responsible for monitoring master executive activity).

The minor cycle monitor is conceptually a timer that triggers after a preset delay if not periodically reset. This function requires the use of graphical operators having designated, finite firing times. In this case a pair of time operators are used. The rightmost operator claims the initial value count of 1 from its input c-cell and completes a cycle by sending a count to an output c-cell after a time delay of Δ units. In DAIS Δ = 40 msec, the time allotted to a minor cycle. The rightmost two operators continue to cycle, representing the transmission of minor cycle messages by the master, as long as a count remains in the C-cell designated MASTER ACTIVE. The timer structure is connected to the

MASTER ACTIVE c-cell with a read-only link, so operation of the timer structure does not change the count in this cell.

The leftmost operator fires in a similar fashion, but with delay NΔ, where N is an integer parameter that specifies the number of minor cycle periods allowed to elapse before the minor cycle monitor declares the master dead. If the leftmost operator outputs a count to the c-cell that acts as input to both sides of the structure at a time when there is no count in the c-cell to its right, then the lower left operator in the structure fires to indicate that the master has failed. A zero link from the RECOVERY c-cell inhibits this activity when recovery is in effect since the monitor processor does not operate during recovery.

If power switching removes power from a currently active monitor or remote processor, the effect on system-level activities is more indirect. The newly unpowered processor/BCIU pair no longer responds with status words when addressed by the master BCIU. The resulting error responses are accumulated by the system status monitor portion of the master executive until an error count threshold is exceeded. Failure of a remote processor/BCIU is treated identically to failure of a master; that is, the system switches to a backup configuration.

When the pilot requests a START/RESTART action on the PCP, the initialization software compares the software configuration currently loaded in the DAIS processor set with the maximum configuration allowed by the currently active hardware. If the existing software load passes a checksum test and matches the current active processor configuration, then the processors are initialized with this software configuration with no further

action. If the software load module does not produce the correct checksum or if it does not match the hardware configuration, then the CHANGE S/W predicate calls for a software reload. The pilot can unconditionally cause a reload by triggering the LOAD/RELOAD switch on the PCP instead of the START/RESTART switch.

The software reload procedure begins by overlaying the previous software configuration. This is represented by the action of the reset operators that eliminate the c-cell count (if any) representing the previous software configuration.

The software reload process, symbolized by the RELOAD S/W CONFIG predicate operator, is then completed. This predicate operator represents a complex series of system actions in which the largest software configuration allowed by the GO bit configuration is loaded and checksummed. Any processors that are unable to successfully load and checksum the required software reset their GO bits and the software reload process is repeated. The result is a complete new software configuration. As stated before, we assume that one of four distinct configurations will be selected, and the RELOAD S/W CONFIG predicate places a single count in exactly one of the four configuration c-cells. Only the RELOAD S/W CONFIG predicate and the four RESET operators can change the count in the configuration c-cells; all other operators connected to these c-cells are connected via "read-only" arcs that allow the configuration information to condition other operations but do not change the stored values.

Mission Mode Actions

The second major factor that determines the allowable functions is the mission mode. The pilot uses the master mode panel (MMP) keys to select exactly one of the allowable mission nodes. In Figure 7 this is represented by the MODE ACTION predicate operator, which outputs exactly one mode command for each stimulus, based on the setting of the MMP mode keys. Although it appears that the MMP is the only avenue for entering mode requests, it is also possible for the pilot to use the IMFK for this purpose.

The current DAIS MMP design allows 15 master modes. Selection of a master mode may automatically invoke specified application functions and may automatically terminate incompatible functions. A master mode may also allow the pilot to initiate and terminate a specified set of functions via the MFK or IMFK.

TOP-LEVEL DAIS APPLICATION FUNCTIONS

Figure 8 shows the structure of four hypothetical applications functions. Since selection of a mode may activate or terminate a function either unconditionally or conditionally in response to pilot IMFK inputs, the representation of the set of applications functions must provide a control interface that allows for conditional or unconditional function activity. In Figure 7, each function interfaces to the control structure through four c-cells: INITIATE, ENABLE, KILL, and TERMINATE. As the reader can verify by applying the LOGOS firing rules to Figure 8, a function will become active only if a count is placed in the ACTIVATE c-cell.

This occurs only if ENABLE and INITIATE simultaneously have nonzero counts. A count in a TERMINATE c-cell stops function activity by transferring a count to the CANCEL c-cell. A function deactivated by a TERMINATE count can be re-activated by placing a new count in the INITIATE c-cell for the function. If, however, the KILL c-cell is used to deactivate a function, then new counts will be required in both ENABLE and INITIATE c-cells before the function will again be activated. Furthermore, the RESET operators attached to the INITIATE c-cells enforce a requirement that the ENABLE count arrive before the arrival of an INITIATE count. This guarantees that INITIATE counts are not "remembered" from previous mode settings. The RESET operators connected to KILL and CANCEL c-cells prevent a similar situation for KILL counts; that is, a count placed in a KILL c-cell will disable a function that is currently active but will not remain to affect a future activation count.

The relationship between software configurations, master modes, and functions (specialist functions, equipment functions, and display functions) are application-dependent in DAIS. The row of operators marked NORMAL/MODE 1/CONF 1, etc. in Figure 7 show a hypothetical set of these mode/configuration-to-function relationships. In Figure 7, mode 1 unconditionally enables and initiates function 1 when software configuration 1 or 2 is selected, but unconditionally disables function 1 when software configuration 3 is loaded, possibly because of resource constraints. Mode 2 of software configuration 3 also unconditionally disables function 1, while enabling but not initiating functions 2 and 3. These two functions will be activated only if the pilot explicitly requests them through the SELECT FUNCTION predicate. Functions 2 and 3 are unconditionally deactivated in backup mode K with software configuration 4.

In the top-level system graphs represented by Figure 7 and 8, we are concerned only with system capabilities that are visible outside of the system. Each of the functions in Figure 8 may require a combination of interlocking DAIS application software modules such as controller functions, detailed computational functions (specialist functions), equipment monitoring and control functions (EQUIP functions), data transfer functions (asynchronous or synchronous compool updates), and display functions (DISP functions) to produce the observed top-level outputs. At this level, the only concern with the allocation of functions to system resources is whether a function is available in the currently active software configuration, which is determined by the state of the backup/recovery process, by the set of available processor/BCIU pairs, and by the pilot's actions in initiating a reload or restart. All of these constraints are top-level changes in system behavior that are usually immediately observable outside of the system.

SECOND-LEVEL DAIS APPLICATION FUNCTIONS

The second level of system detail deals with the individual operational applications software functions used to create top-level software functions and with the interactions between functions. These relationships are application-dependent, a situation that posed problems in this assessment, since no complete applications for a fault tolerant DAIS system have been fully designed.

Although the goal of this study was to assess the fault tolerance of the DAIS core system, the interactions between this system and application functions are an important factor in overall fault tolerance. No fault

tolerant DAIS applications systems have been fully designed or implemented, but preliminary designs for a few fault tolerant applications have been proposed. The specifications for one application, Integrated Digital Avionics for the Medium STOL Transport (IDAMST) given in the IDAMST OFP Specification[22], were examined and used to create the second-level model presented here. The second-level example concentrates on the applications functions necessary to accomplish one of the main mission functions of a STOL transport--air drop cargo delivery. Enough functions were included to show the flow of information from sensor systems such as the inertial navigation system, the AN/ARN 118 TACAN receiver, the attitude and heading reference system, and the separate flight control system; through the various specialized calculations such as navigation input selection algorithms, navigation filters, drop zone calculation logic and computer air release point calculations; to the waypoint steering and drop zone warning functions that provide information to the crew. This functional chain involves a significant portion of the IDAMST application functions. Peripheral applications functions, such as the equipment functions (EQUIPS) that provide the interface to change peripheral equipment settings and parameters, were omitted in the second-level model to keep the model size manageable.

Figure 9 is the second-level model of a DAIS system configured for the selected IDAMST functions. The data blocks at the top of the model represent data obtained from the various external input sources. These include:

---

[22]"Computer Program Development Specification for IDAMST Operational Flight Program Application, Software Type B5," Part I, Air Force Avionics Laboratory, 30 July 1976.

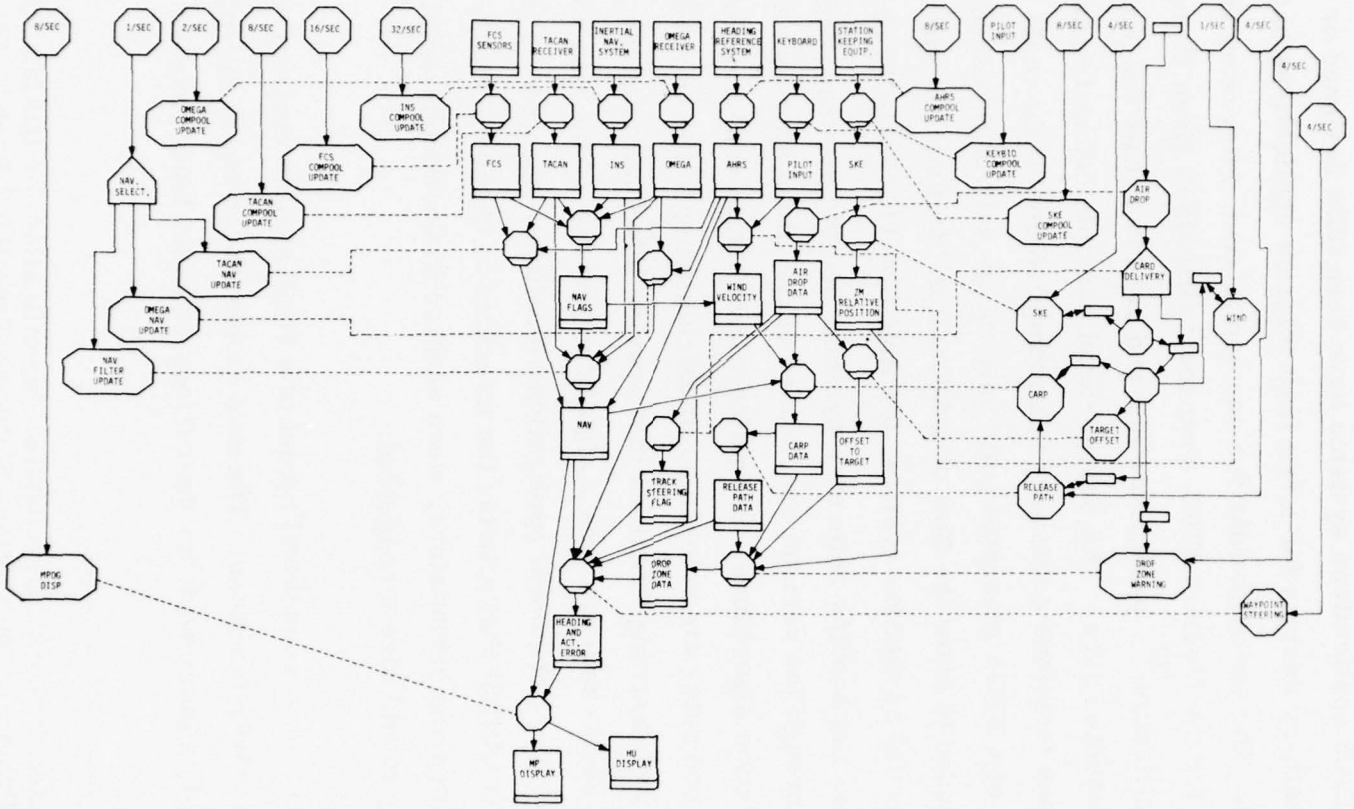


Figure 9. IDAMST/DAIS Application Structure

- Flight control system (FCS) sensors to provide airspeed, barometric altitude, and aircraft angle of attack inputs
- An AN/ARN 118 TCAN receiver to provide range rate and bearing information to selected ground stations
- Inertial navigation system (INS) inputs to provide aircraft position, velocity, and attitude information
- OMEGA receiver inputs to provide aircraft latitude, longitude, and velocity information when provided with true airspeed and magnetic heading data
- A 6000A attitude and heading reference system (AHRS) to provide heading, pitch, roll and turn rate information
- A data entry keyboard (DEK) to receive data and command inputs from the crew
- AN/APN-169B station-keeping equipment (SKE) inputs to provide the range and bearing from a drop zone marker (ZM)

Inputs from each of these sources enter the DAIS system at an RT and are transferred to a DAIS processor for use in the following applications functions:

- Navigation selection--compares inputs from the various sources of navigation information and selects the specialist function to be used for the next position and velocity estimate
- TACAN navigation update--uses TACAN inputs to determine velocity and position
- OMEGA navigation update--copies the position and velocity provided by the OMEGA equipment.

- Navigation filter update--reads data from the INS, OMEGA, and TACAN; combines them to provide an optimal estimate of position and velocity based on selection criteria provided by navigation select.
- Multipurpose display generator display module--transfers navigation and cargo drop information to the crew displays.
- Air drop operational sequencer--activated by the crew to initiate functions required for air drop operations and to pass crew-provided drop parameters to these functions.
- Cargo deliver controller--enables the specialized software functions for air drop cargo delivery in the correct order.
- SKE computations--coverts SKE information to latitude and longitude.
- Wind computation--determines wind velocity based on true air speec and estimated ground speed.
- Computed air release point (CARP) computation--determines the point in space at which cargo is to be ejected.
- Target offset calculation--determines the difference in latitude and longitude between a target and an indicator.
- Cargo release path calculation--determines the course the aircraft is to follow while ejecting cargo.
- Drop zone warning--provides warning signals and timing information to the crew to enable them to eject cargo at the correct time.

- Waypoint steering--determines deviations from the course required for air drop and displays this information to the crew. This provides the information needed by the pilot to successfully follow the required air drop course.

The above information is included here to provide the reader who may not have access to DAIS documentation with a basic understanding of functions performed in a DAIS application. Details of the function of each of the modules described above are contained in the IDAMST specification.[23]

An assessment of DAIS core system functionality should not involve the internal functioning of the individual modules listed above; instead, it is important to determine the role of the core system in sequencing these functions and in moving data to and from them.

Figure 10 shows the essence of these sequencing and data flow relationships. Detailed function names have been replaced by abbreviations, since we will be interested only in structural relationships from this point on. The reader can determine the identity of the function corresponding to an operator or data item in Figure 10 by matching it with its counterpart in Figure 9.

Data items, A1, B1, C1, D1, E1, F1, and G1 are assumed to be resident in RTs. The IDAMST specification calls for the automatic transfer of data from the RTs to compools in the processors by the master executive; the shading of the row of data operators across the top of Figure 10 indicates that these are master executive functions.

---

[23] Ibid.

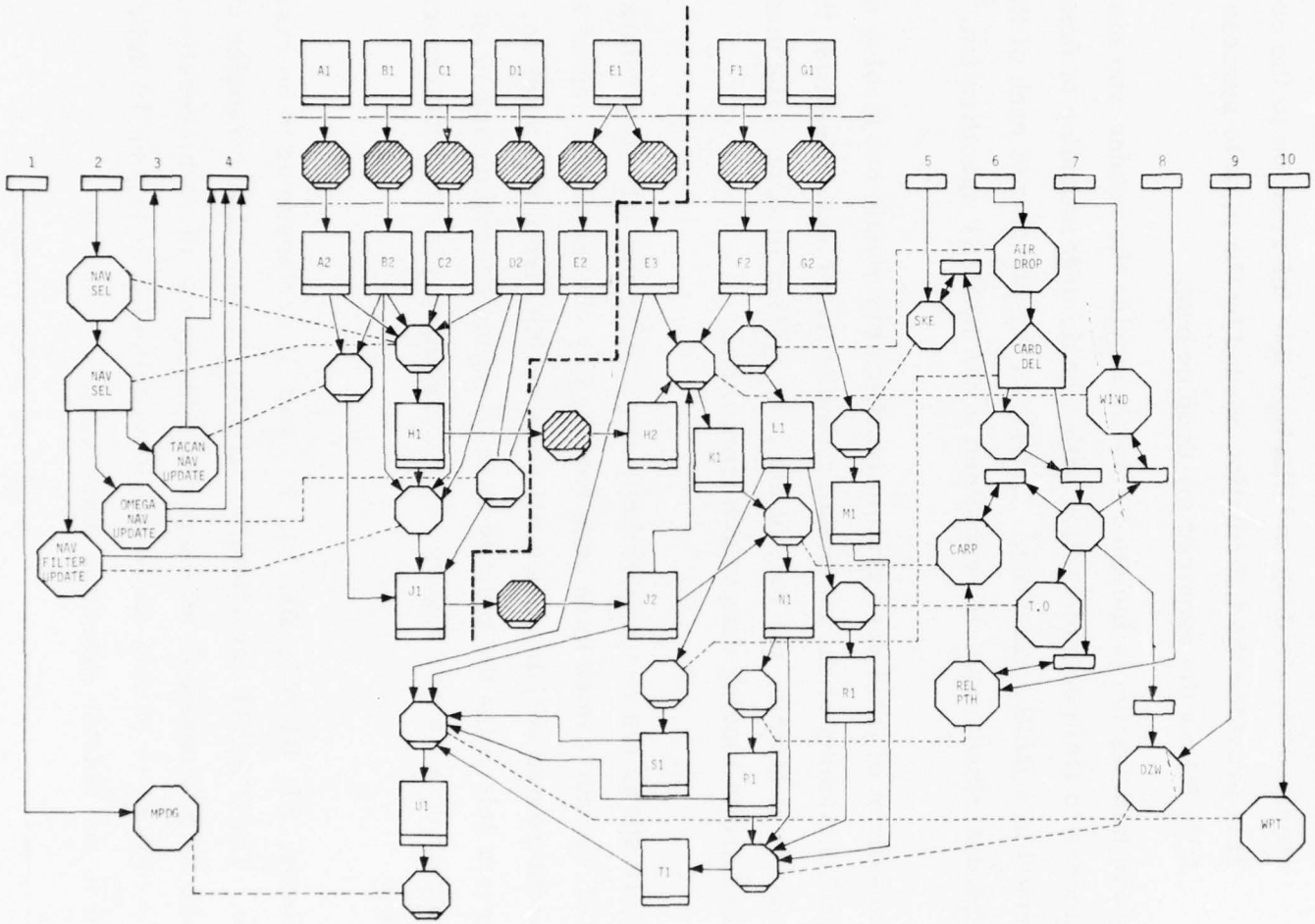


Figure 10. Second-Level IDAMST Application Function for DAIS

SECOND-LEVEL DAIS MASTER EXECUTIVE FUNCTIONS

An important area of concern in DAIS is the master executive's support of applications that are distributed over more than one processor. Although the configurations recommended for IDAMST place all of the functions described here in a single processor, we have exercised the applications designers' prerogative to locate the functions that generate aircraft position, attitude, and other navigation information in processor 1 and the functions that use this information for cargo delivery in processor 2. The boundary between the two processors is indicated by the diagonal dashed line in Figure 10.

This distribution of functions allows us to include an important DAIS core system feature in our hypothetical model; asynchronous compool updates. In Figure 10, data items H1 and J1 represent decision information generated in the navigation selection computations and the current values of key navigation parameters, respectively. Both data items represent information that is generated in processor 1 and used in processor 2. These data items are generated by synchronously scheduled processes , making it possible to create a scheme whereby the copies in processor 2 are updated synchronously. However, for the purpose of illustration, we will assume that compools H and J are asynchronous compools. This means that the remote copies H2 and J2 will be updated whenever their corresponding master copy (H1 and J1 in this case) is updated. The needed control information is provided in Figure 9 by c-cells 3 and 4. C-cell 3 receives a count whenever the navigation select function completes an update of H1. C-cell 4 receives a count whenever any of the three functions that are capable of changing J1 does so.

The partitions just described separate the IDAMST applications functions into three regions: processor 1, and processor 2, and the RTs. The control graph to the left of Figure 10 represents the sequencing of tasks in processor 1 accomplished by the local executive in processor 1, in response to counts exchanged through c-cells 1, 2, 3, and 4. There is a direct correspondence between these counts and messages exchanged with the master executive. Similarly, the control graph to the right represents the functions active in processor 2 that are stimulated by counts received through c-cells 5, 6, 7, 8, 9, and 10. The shaded data operators connecting H1 and H2 and J1 and J2 represent asynchrounous data transfers provided upon request by the master executive while the remaining shaded operators represent data transfers provided automatically at prespecified time intervals (synchronously) by the master executive.

We are now in a position to model the behavior of the master executive in relation to the hypothetical applications environment just defined. Figure 11 shows these major functions. The c-cells and data items arrayed at the bottom of Figure 11 represent the portions of the applications functions on processors 1 and 2 that are visible to the master executive. The data items arrayed at the sides represent data stored in RTs.

Figure 11 clearly shows that the central source of activity in a normally operating DAIS system is the master executive synchronous instruction list (SIL). This portion of the master executive issues synchronous messages to fire operators T_A, T_B, T_C, ..., T_G in accordance with the predefined synchronous instruction list (SIL). These synchronous messages must compete for the bus with asynchronous message requests received from applications tasks such as the navigation tasks in processor 1. This

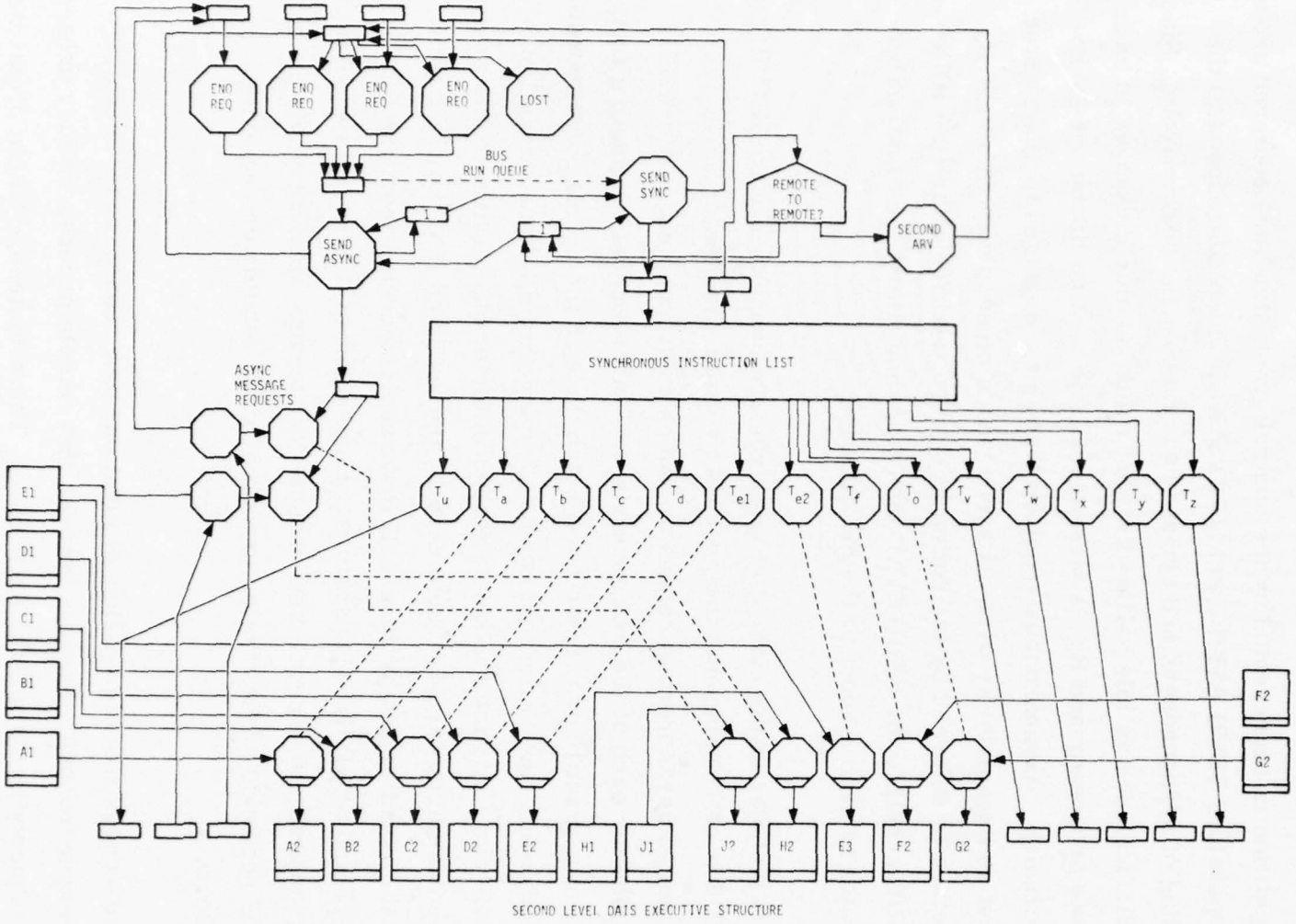


Figure 11. Second Level DAIS Executive Structure

competition is represented by the control graph structure centered around the operators SEND ASYNC and SEND SYNC, which correspond to the sending of asynchronous and synchronous messages, respectively. Since the SIL includes an idle polling list that sends dummy messages to each remote processor and RT, a message will be sent on the bus at every opportunity. Asynchronous requests that are recognized by the master executive have priority over all synchronous messages. This priority structure is enforced by the dotted link (zero link) from the BUS RUN QUEUE c-cell to the SEND SYNC operator that allows the SEND SYNC operator to send a count to the SIL function.

The remainder of the control graph structure connected to the BUS RUN QUEUE c-cell represents the process by which the master executive recognizes asynchronous requests. An asynchronous request c-cell is provided for each of the four processors. All requests originating from tasks in a specific processor must enter through the c-cell corresponding to that processor. Note that counts arriving at the leftmost c-cell may pass directly into the run queue without interference, while counts arriving at the remaining three request c-cells are accepted into the run queue only if a count exists in the c-cell that receives counts from the SEND ASYNC and SEND SYNC operators. Each count in this c-cell represents an opportunity to accept a new request that occurs when a remote processor or RT responds to the master executive while sending or receiving a message.

Asynchronous messages allow at most one new request to be accepted, while synchronous messages between two remote processors may present two requests to the master executive. The remote-to-remote predicate in Figure 11 shows this situation.

At this level of detail, the "opportunity" c-cell enables a conflict between all asynchronous requests awaiting entrance into the run queue. The LOGOS firing rules allow a random selection of the request to win the conflict. In DAIS, however, the selection is not random, but is determined by the identity of the sender and receiver of the previous message. Figure 12 shows the details of this process. In some cases the opportunity is offered to processors that have no asynchronous requests. In this case the opportunity is lost, represented by a firing of the operator LOST in Figure 11. The receiver of each asynchronous message gets the next opportunity to enter a request into the run queue, in "daisy chain" fashion. The expanded model in Figure 12 represents this process by keeping track of the destination involved in each request.

Figure 12 includes the details necessary to completely determine the behavior of the bus request logic. Each of the single asynchronous request c-cells for an individual processor has been replaced by three c-cells, one for each possible remote destination. (It is not possible to generate an asynchronous receive request).

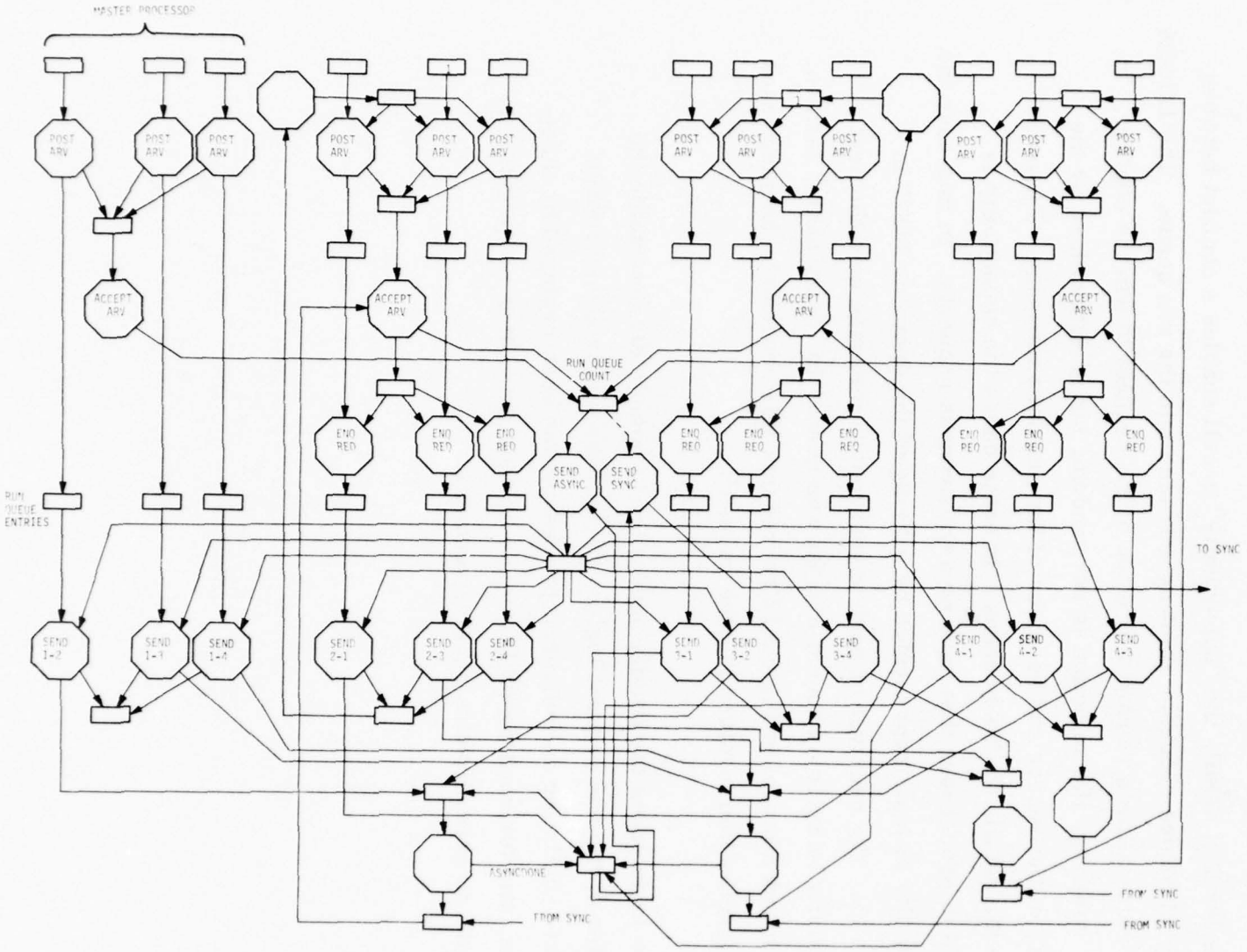


Figure 12. Third Level Model of DAIS Bus Message Allocation

SECTION V

DAIS FAULT TOLERANCE MECHANISMS

This section summarizes the various mechanisms included in the DAIS design for fault tolerance purposes. These mechanisms will be referenced in the discussion of DAIS functional faults in Section VI.

DAIS PROCESSOR/BCIU FAULT TOLERANCE MECHANISMS

One processor can be configured as a single processor system and assigned to monitor the health of the master processor. Such a processor is called a monitor processor. If the master fails to show activity within given time frames or if the master specifically instructs the monitor, the monitor processor will take over. This is known as backup.

The switch to backup is done automatically without any pilot intervention after the fault has been detected and isolated. In the backup mode of operation, only the processor that was previously the monitor is operational; hence, a degraded system performance results. The monitor processor takes control of the system by flagging its BCIU as Master and assuming the master executive functions. It also activates mission critical applications software functions resident in the monitor processor. The pilot is notified of the failure condition and normal system operation is continued at major frame 0 minor cycle 0. Normally, the pilot will execute a reconfiguration procedure at a later time.

Note that backup is initiated for both a master processor/BCIU failure and a remote processor/BCIU failure. The latter failure requires backup since the applications software is distributed among the master and remote processors; hence, a failure of any one can cause improper system operation. Entering the backup mode of operation is similar in the two cases except that the monitor initiates backup when the master fails while the master initiates backup when a remote processor fails.

Recovery is initiated when a monitor processor/BCIU failure is detected. The master processor maintains control of the system, deletes the monitor from the bus communication list, and notifies the pilot of the failure. Normal system operation continues in the current minor cycle since recovery requires only a short time to effect. All system functions are maintained and no degradation occurs except that backup is no longer available as a mode of operation.

System reconfiguration is initiated by the pilot. When he is informed of a processor/BCIU failure, he can remove power from the failed unit by depressing the appropriate button on the Processor Control Panel (PCP). He can then initiate reconfiguration by toggling the LOAD/RELOAD switch on the PCP. After the BCIU of a processor has completed its current operation, the processor invokes the startup ROM and sets a flag to indicate a LOAD/RELOAD interrupt. The startup ROM includes the self-testing loop that does a self-test of the processor and a self-test of the BCIU. At the end of the loop, the processor is placed in a no-op loop if any errors have occurred, or else, upon recognition of the LOAD/RELOAD interrupt, it proceeds to configuration identification.

The objective of configuration identification is to determine which processors are operational and, consequently, identify what the configuration will be. Each processor builds a startup status block that contains several bits of status information regarding the processor and its software and tables. Each processor then enters a delay loop proportional to the processor ID, with the lowest numbered processor becoming the control processor and placing its BCIU in master mode.

The control processor builds a configuration index by attempting to communicate with each processor, asking it to send its startup status block. If the control processor is successful in building the configuration index, the third phase of reconfiguration is entered; if not, another processor's delay loop will time out and it will assume the role of control processor.

The third phase of reconfiguration is the software loading and verification. The mission software is loaded into the processors according to the system configuration. Each configuration consists of an absolute load module for each processor.

Fifteen separate configurations will be included in the mass storage file of a system to allow different configurations for all possible combinations of 1, 2, 3, or 4 processors. The control processor uses the configuration index to index the startup configuration table.

The startup configuration table contains the load module IDs for each processor. The control processor attempts to load its module and verifies all programs and tables (using check sums) as required. It then directs the

loading of modules into other processors. Once all the modules are loaded and verified, the master processor initializes the processors, remote terminals, and BCIUs, and resumes normal system operating using the new configuration.

FAULT TOLERANCE PROCEDURES FOR OTHER DAIS CORE UNITS

Failures to DAIS core units other than processors or BCIUs are handled directly by the master executive (or the monitor if backup is in effect).

The 1553A multiplex bus system uses dual redundant busses. In the event of a bus element failure, the system will automatically direct all activity to the other element and continue operation.

Critical RTs are also dual redundant. For example, two separate RTs drive the pilot's console. In the event of a redundant RT failure, the system will direct all traffic to the duplicate RT and continue operation. In the event of the failure of an RT that is not duplicated, the master executive will delete all BCIU instructions associated with the failed RT, essentially deleting it from the system.

The master executive performs bus message error management and terminal failure management. Each terminal on the bus provides mechanisms for detecting and recording message error conditions and terminal failures. The RTs and BCIUs record information used by the master in error analysis in the following registers:

Status Code Register
Built-in Test Word Register
Last Command Register

The Last Command Register stores the last command executed by the terminal. The Built-In-Test Register contains status information about a terminal's failure and also message failures. The Status Register contains summary information about message failures.

The master can be informed of a message error either when a remote does not return a status word for a message (most common way) or if a status word is returned indicating an error. The master can acquire information regarding the message error by using mode commands. These mode commands are used to perform some corrective actions.

Whenever an error is detected that relates to bus message transmission, Bus Error Recovery attempts to identify the error and correct the error if possible. A retransmission of a bus message that contained an error is then attempted. If the bus error was due to a permanent hardware failure, Configuration Management is called to configure a new system that does not use any failed hardware units.

If an error is detected during a normal bus message transmission, it is called a primary error. However, the Bus Error Processing, while trying to identify the source of the error and to initiate a retransmission of the message, may encounter additional bus related errors. These are called

secondary errors and they are handled in a slightly different way than primary errors. Both types are discussed below.

Bus errors are recognized by bus control; the possible types are defined by Table 1. Bus control records primary errors for the receiver and for the transmitter separately.

TABLE 1
ERROR TYPES

Name	Description
BUSY	Busy remote processor
DATA'ERR	An error during data transmission
FAIL'REF	A remote processor attempt to transmit to a failed Remote Terminal
MSG'ERR	The receiver detected data transmission problems
RT'PARITY	A parity error in an RT to subsystem transfer
STATUS'ERROR	A terminal did not correctly reply with a status word
TERM'FAIL	A terminal's self test procedure failed

The general flow followed by the master executive in Bus Error Processing is shown in Figure 13. For primary errors, the master executive records the error and initializes the bus control queue (run queue).

In the case of asynchronous or critically timed messages, part of initialization is to remove from the input queue all entries of the type ASYNC'DONE or MODE'DATA to prevent redundant entries after the message is retransmitted. (Terminal failure or remote terminal parity errors do not require

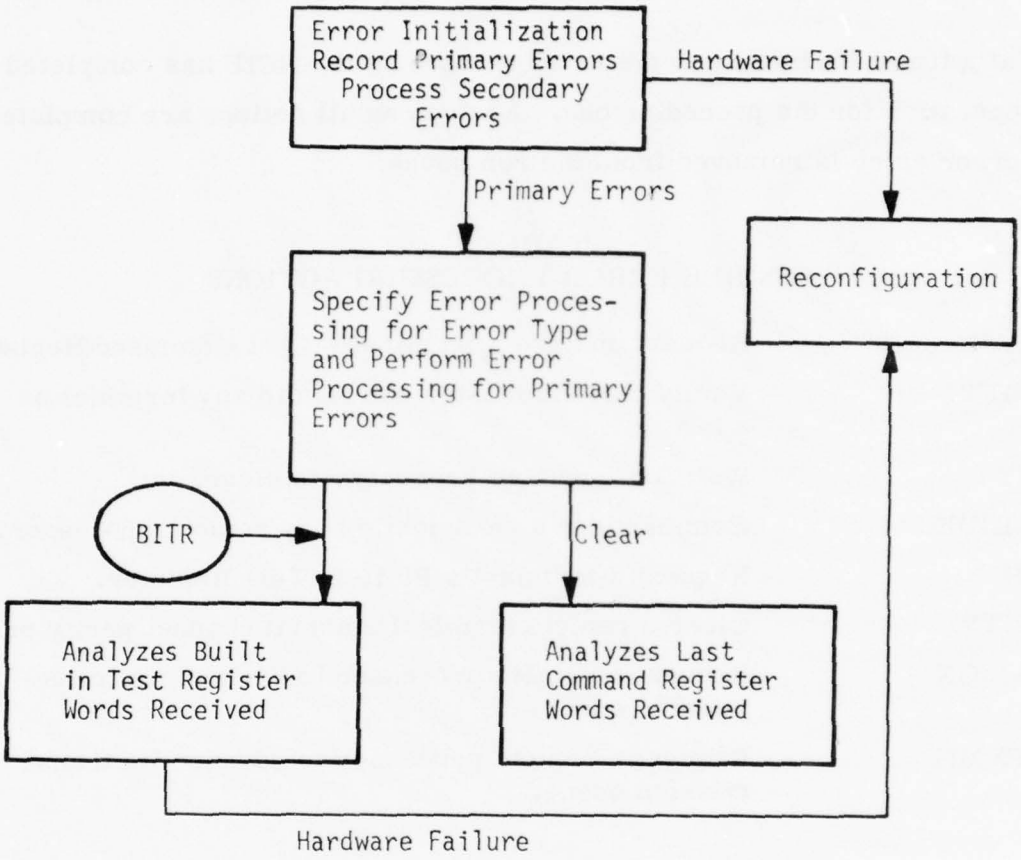


Figure 13. Bus Error Processing Routines

retransmission and, hence, this part of the initialization is not required for them.) The remainder of the initialization places a new entry on the run queue. (Note that if the error is a secondary error, it is processed directly rather than placing it on the run queue. This is discussed later.)

As the run queue entry is processed, the master executive interprets the error type and builds an activity list of corrective actions. The possible corrective actions are listed in Table 2. The corrective actions are initiated

one at a time, each one being started as soon as the BCIU has completed the operation for the preceding one. As soon as all actions are completed, the error entry is removed from the run queue.

TABLE 2
POSSIBLE ERROR PROCESSING ACTIONS

CLEAR	Request analyze a terminal's Last Command Register.
VERIFY	Verify that the master can talk to any terminal on a bus.
BUSY	Wait for a remote processor to clear.
OVERIDE	Commandeer a chronically busy remote processor.
BITR	Request a terminal's Built-in Test Register.
PARITY	Clear a remote terminal's serial channel parity errors.
REALIGN	Request a remote processor to back up its transmission queue.
SKIPXMIT	Request a remote processor to advance its transmission queue.

If the action is CLEAR, then the terminal is requested to send its Last Command Register contents and its status; however, if the terminal is the master, the Last Command Register is already available. When the Last Command Register has been returned, it is analyzed to see if the transmitter queue must be realigned.

If the action is VERIFY, then the master attempts to talk to other terminals on the bus associated with message error by requesting their status. If it can, the bus is presumed okay; otherwise, it is assumed to have failed. In the latter case, the system will be reconfigured.

If BITR is the action specified, the terminal is requested to send its BITR (Built-in Test Register) contents. When the BITR is returned, it is analyzed to check for message errors and terminal failures. If there are message errors and the terminal was a receiver, processing is simply continued; however, if the terminal was a transmitter, this is a fatal error and the master halts. In the case of terminal failures involving the connection to the bus, the corresponding terminal's bus is failed. For other terminal failures, such as a power or DMA failure, the terminal is failed. In all of the terminal failure cases, reconfiguration is initiated.

When secondary errors occur during the processing of a primary error, they are processed directly without placing an entry on the run queue. The particular actions taken for a secondary error depend upon the corrective action that was in process for the primary error when the secondary error occurred.

First, consider the CLEAR, BUSY, and BITR actions for a primary error. If 'STATUS' ERR or DATA ERR occurs while performing a CLEAR, BUSY, or BITR action for a primary error, then that action is retried up to some maximum number of times. If the maximum is reached, the bus is presumed failed or the corrective actions are attempted using the other bus. (When both buses fail, the system fails.) A secondary BUSY error during primary error BUSY results in a retry for a maximum number of times after which the OVERIDE action is added to the others specified for the primary error. If the primary action was CLEAR or BITR, the secondary error is fatal. A TERM 'FAIL' secondary error causes a BITR action to be added to those for the primary error unless it is already included. An RT'PARITY

secondary error causes the PARITY action to be added to those for the primary error. A MSG'ERR secondary error from a transmitter termina is fatal but from a receiver terminal it is ignored.

Secondly, consider that a VERIFY action is being performed for the prima error. Then a STATUS'ERR secondary error causes the bus to be failed. Other secondary errors are ignored.

Finally, consider the REALIGN or SKIPXMIT actions for a primary error STATUS ERR, BUSY, or MSG'ERR secondary errors cause the CLEAR a REALIGN actions to added, if not already included, for the primary error Other secondary errors are ignored.

SECTION VI

FUNCTIONAL FAULT ASSESSMENT

In this section we use the graphs developed in Section III as an assessment tool. First we examine the behavior of the graph to see if the system contains potential problem areas as designed. We then examine the graphs to see the faults in the individual graph functions. We are concerned with four faults associated with control while data faults are considered in a single class. The fault classes and the behavior in a graph model associated with each class are:

1. Loss of Control (LOC)

   An operator fails when enabled. This corresponds to a function in the system that fails to perform under the conditions called for in the design.

2. Multiple Instance of Control (MIC)

   An operator spontaneously produces an unwanted output. This corresponds unexpected activation of a mechanism in the system.

3. Diversion of Control (DOC)

   A predicate operator or other control structure diverts a control thread to a path not allowed for the present system state. This corresponds to an incorrect decision branch in software or to a malfunction in hardware signal steering logic.

4. Late Control (LC)

   A control stream does not complete within time constraints. This may occur at the top levels of abstraction in the system where an individual function may require thousands of steps to complete.

5. Data Error (DE)

   A data cell does not contain a legal data value. In this assessment, we limit data errors to situations where a stored data item does not pass an integrity check, such as a sum check.

The following tables summarize functional faults in the DAIS functional graphs. Table 3 catalogs faults determined from an inspection of the Figure 7. Tables 4, 5, 6, and 7 refer to Figures 8, 10, 11, and 12 respectively. Each functional fault entry includes the fault class involved, the location of the fault in the system, a brief functional description of the fault, the probable effects of the fault on DAIS application functions and the DAIS fault tolerance mechanisms (if any) that deal with the fault.

TABLE 3
TOP LEVEL SYSTEM CONTROL FUNCTIONAL FAULTS
(REF: FIGURE 7)

Fault Type	Location	Description	Effects	Applicable DAIS Mechanism
LOC	PCP	Unable to switch power on	Loss of processor during reconfiguration	Reconfiguration software deletes unpowered processor from configuration
LOC	PCP	Unable to switch power on	None-unless processor fails with BCIU in in master mode	Backup for master or remote processor failure; recovery for monitor
LOC	PCP	Loss of restart interrupt	Requires software reload on restart	Use reload
LOC	PCP	Loss of reload interrupt	Pilot unable to force software reload without power switching	Use restart and switch power if necessary to change configuration
LOC	Mode Actions	Loss of mode select	Pilot unable to switch mission modes and thus may be denied certain functions	Pilot can substitute MFK for MMP. Can reload system software
LOC	Function Select	Loss of function select	Pilot loses access to certain functions	Pilot can use alternate keyboard for hardware failures; can reload software for software malfunctions
LOC	System Status Monitor	No response to excess error counts	Lurking fault; no problem if errors do not actually occur.	Software reload-reinitializes system and can provide new software copy.
LOC	Minor Cycle Monitor	No response when master ceases activity	System will fail to operate; only pilot initiated reload or restart can restore activity	Pilot must initiate restart or reload

TABLE 3
TOP LEVEL SYSTEM CONTROL FUNCTIONAL FAULTS (continued)

Fault Type	Location	Description	Effects	Applicable DAIS Mechanism
MIC	System Status Monitor	Spurious monitor or master failure action	Forces backup or recovery as applicable. System degrades if backup occurs.	Backup/Recovery
MIC	Minor Cycle Monitor	Spurious master failure action	Forces backup and system degradation	Backup
DOC	PCP	Wrong processor powered off (Usually misplaced finger)	Backup or recovery invoked depending on processor. Easily corrected.	Backup/Recovery Reload or restart needed to restore configuration
DOC	PCP	Reload/Restart Interchanged	May cause unneeded reload with extra delay-otherwise No effect.	Reload
DOC	Software Reload Management	Wrong software modules reloaded	May result in smaller configuration than desired.	Reload

TABLE 4
TOP LEVEL APPLICATIONS FUNCTIONAL FAULTS
(REF: FIGURE 8)

Fault Type	Location	Description	Effects	Applicable DAIS Mechanism
LOC	Scheduler	Function fails to initiate	May cause faults in other functions --result cannot be analyzed at this level	None
LOC	Scheduler	Function stops without cancel request	May cause faults in other functions-- result cannot be analyzed at this level	None
MIC	Functions	Function initiates spontaneously	Application dependent-- may interfere with existing functions	None--but switching modes should deactive spurious function

TABLE 5
SECOND LEVEL APPLICATION FAULTS
(IDAMST EXAMPLE - REF: FIGURE 9)

Fault Type	Location	Description	Effects	Applicable DAIS Mechanism
LOC	Omega, TACAN, FCS, INS, or AHRS	Data from affected function no longer current	Bad data can be rejected by NAV Select if the minor cycle number written on each compool update is monitored	Minor cycle header in compool. Pilot may try restart. Switching power to cause backup, recovery, or reconfiguration difficult because pilot has no knowledge of location of processor with faulty module
LOC	NAV Select	No stimulus to NAV Select function	NAV compool no longer correct. Data for air drop calculations no longer valid	As above
LOC	NAV Select Function	Nav Select fails to stimulate next function	As above	As above
LOC	MPDG DISP Function	No stimulus to MPDG DISP function	Display "freezes." Pilot may not notice lack of activity	As above
LOC	Air Drop Function	No stimulus received for Air Drop Function	No air drop functions appear	Pilot requests function again, possibly on alternate keyboard
LOC	Cargo Delivery Function	Cargo Delivery function does not receive stimulus from Air Drop or Does not activate subsidiary functions	As above	As above
LOC	SKE Function	No stimulus to SKE function	Zone marker position not updated; will be detected only if Drop Zone Warning calculations check minor cycle information in SKE compool	Minor cycle header in compool

TABLE 5
SECOND LEVEL APPLICATION FAULTS
(IDAMST EXAMPLE - REF: FIGURE 9) (continued)

Fault Type	Location	Description	Effects	Applicable DAIS Mechanism
LOC	WIND Function	No stimulus to WIND function	Wind velocity compool not updated.	Minor cycle header in compool
LOC	Release Path Function	Release Path function not scheduled or not stimulated	Release path data not updated and Cargo Air Release Point function not activated. CARP data not updated. Drop Zone Warning can detect only by minor cycle information	Minor cycle header in compool
LOC	Target Offset Function	No stimulus to target offset	No target offset data calculated. Drop Zone Warning not activated	Event conditions
LOC	Drop Zone Warning function	No stimulus from minor cycle timing or function not scheduled or Target Offset function does not complete	No air drop information	Crew can observe fault and request function again; reload/ restart if required
LOC	Waypoint Steering Function	No stimulus to Waypoint Steering Function	No heading data to crew displays	As above
MIC	All applications functions	Application functions stimulated too often	No effect on data because of interlocks in compool read/write routines. Lower priority functions may suffer loss of control because of time demands of spurious functions. Fault would be detected as LOC	Compool read/write interlock

TABLE 6
SECOND LEVEL SYSTEM CONTROL FAULTS
(REF: FIGURE 11)

Fault Type	Location	Description	Effects	Applicable DAIS Mechanism
LOC	Send synchronous message function	Synchronous Instruction List (SIL) not executed	Majority of system activity stops; no synchronous compool updates. Proper operation of minor cycle monitor causes backup. If recovery in effect, system fails until pilot initiates restart or reload.	Minor cycle monitor, backup. Restart/reload if recovery in effect
LOC	Send asynch-rounous message function	Asynchronous message not sent. Lack of async message complete reply stops SIL activity	As above	As above
LOC	Asynchronous request lost	Async. request not recognized. Source of this condition analyzed at third level	Async. message and other async. messages emanating from same processor not sent. In IDAMST example, NAV information not transferred to Processor 2	None

TABLE 7
THIRD LEVEL SYSTEM CONTROL - ASYNCHRONOUS MESSAGE CONTROL
(REF: FIGURE 12)

Fault Type	Location	Description	Effects	Applicable DAIS Mechanism
LOC	Asynchronous Request Vector (ARV)	ARV not recognized - can occur in "normal" core system if processor containing request is not addressed	Asynchronous message delayed indefinitely	None
LOC	Run Queue	Asynchronous run queue entry lost	Asynch. message delayed until ARV is recognized again - no lasting effect	None
LOC	Send function	Transmitter does not recognize when message complete	ntinued repetitions ame message	None
MIC	ARV	Spurious ARV's generated; may be due to faulty executive or "babbling" user task	roblem originates remote, substantial bus overhead created, depending upon number of messages directed to faulty processor. If problem originates in master, SIL will be suspended, causing backup or if recovery is in effect system failure	Backup, recover to handle first order effects. If problem is due to faulty task, pilot may have to try to find configuration mode that does not include the faulty task

The majority of the functional faults tabulated in the previous section ultimately produce changes in system behavior visible to the ultimate DAIS fault tolerance mechanism, the pilot. The pilot's main recourse is the restart/reload function, which produces an entirely new configuration. The pilot has a substantial disadvantage in this process, because he can only affect the new configuration indirectly, by switching processor/BCIU power. In general, he will not be able to identify a faulty processor easily. For example, if a remote processor fails, then its GO bit is turned off as part of the backup process, thus changing the color of an indicator on the PCP, but the backup process also turns off the GO bit of the original master processor and all other processors except the monitor processor, thus denying the pilot the identity of the original culprit.

MAJOR FUNCTIONAL FAULTS

A few of the functional faults tabulated are serious enough to merit further discussion. These are; loss of control continuity during reload, multiple restart/reload interrupts, and asynchronous message overload.

Loss of control continuity during reload occurs when reconfiguration takes place under the supervision of any processor except the monitor processor in a system with more than two active processors. Consider the situation that occurs when such a system reconfigures while in the recovery state. The control processor for reconfiguration will be the original master processor, which does not contain a complete set of applications software, but requires the services of a remote processor to continue to provide critical functions. Under the current DAIS reconfiguration procedures, the control processor cannot support applications functions alone and cannot start a new applications function set

until a new configuration has successfully been established. This may require as many as three attempts to load software from mass stroage for one or more configurations. With tape mass storage, this process may consume several seconds, during which critical functions are not available. Of greater concern is the possibility that data transfer errors from the mass storage may preclude the loading of any configuration. In this circumstance, the system becomes inoperative for the remainder of the mission. Because of this, failure of a monitor is potentially a greater problem for the pilot than failure of a master or a remote. In the latter case, degradation of system functions immediately occurs, as the system goes into the backup state, but the pilot can initiate reconfiguration and allow the reconfiguration process to proceed without losing critical functions.

Two approaches should be considered to alleviate this problem. First, Read Only Memory (ROM) could be used to store alternative configurations. This would eliminate the reliance upon external mass storage for reconfiguration, and may eliminate the need for initial checksums. Second, applications requiring more fault tolerance could be configured with two monitor processors. The second monitor would be configured the same as the first except that it would wait a greater number of missed minor cycles before assuming system control. In effect, a second backup state would be available to provide a control processor capable of supervising the reconfiguration of the remaining processors and of providing critical functions as specified in the DAIS system requirements.

Multiple restart/reload interrupts are lethal to DAIS applications functions. It is conceivable that multiple restarts could be spaced far enough apart

that the system could complete the relatively rapid restart process and resume normal operation quickly enough that the pilot would not observe a major loss in functionality. However, closely spaced reload interrupts will preempt normal functionality unless the system is forced into backup and the monitor processor is used to provide critical functions while continuing reconfiguration as a background process. In this case, the system would remain permanently in a degraded configuration, but would remain operational.

Asynchronous message overload occurs when an endless stream of asynchronous message requests enters the run queue. As shown in Figure 12, this fault is unique to the master processor. Applications tasks in the master processor have direct access to the run queue and can thus continuously enter asynchronous message requests. Since asynchronous messages always take priority over synchronous messages, the effect of this overload is to stop all activity associated with minor cycles. If this condition persists, the system eventually will skip enough minor cycles that the difference between the actual minor cycle and the "theoretical" minor cycle will exceed a system design parameter. When this occurs, the master executive will force the system into backup. Since all asynchronous message requests to the monitor executive are also entered directly into the run queue, the condition may occur during backup, causing an immediate loss of system activity.

Applications tasks in a remote processor cannot produce this overload condition because each request originating in a remote must wait for at least one other message to address the remote before the request is recognized. It is recommended that a "software throttle" be provided in the master executive to limit the number of asynchronous requests allowed during a given processing period.

SECTION VII

USE OF THE AFAL LOGOS SYSTEM

One reason for using a LOGOS-based analysis model was the availability of an experimental computer-based LOGOS system at AFAL. Since this system is not a production system, the graphs used for this effort were created and assessed manually. A few typical models were transferred to the AFAL LOGOS system to determine its potential as a support tool for assessment efforts such as this. This section summarizes the capabilities of this LOGOS system and summarizes our experience in using it on this contract.

SYSTEM CAPABILITIES

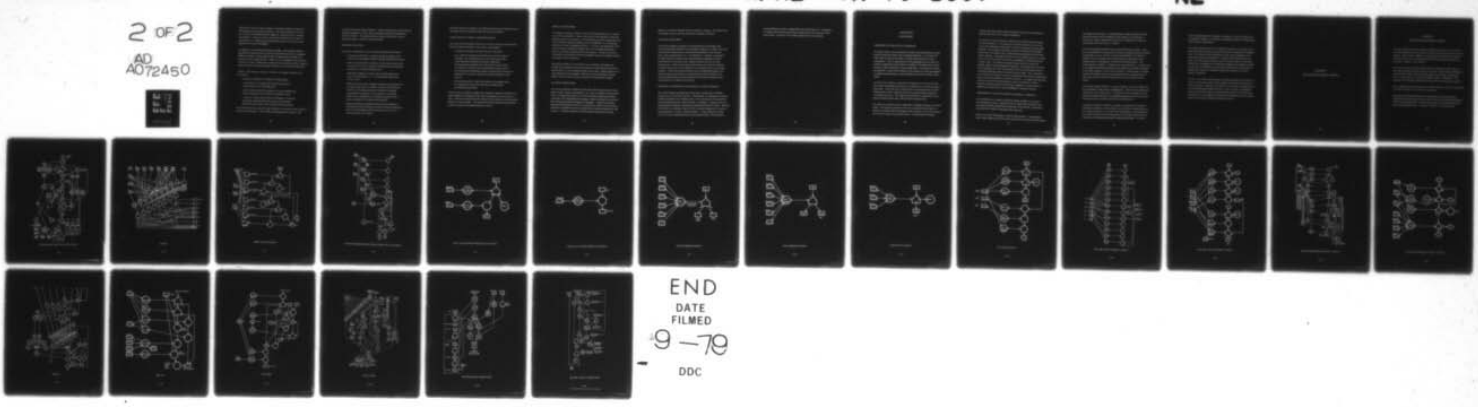
The current Logos System is implemented on the DEC PDP-10/TOPS-10 system used in the AFAL Avionics System Analysis and Integration Laboratory (AVSAIL). It is a reimplementation of the original LOGOS current system capabilities test implementation effort carried out between 1972 and 1974 on the CASE-10 TENEX System at Case Western Reserve University. It is a single-user system for the creation, maintenance, and analysis of graphical models, but has no "generation" facility to turn the models into actual systems.

GRAPHICAL EDITING FACILITIES

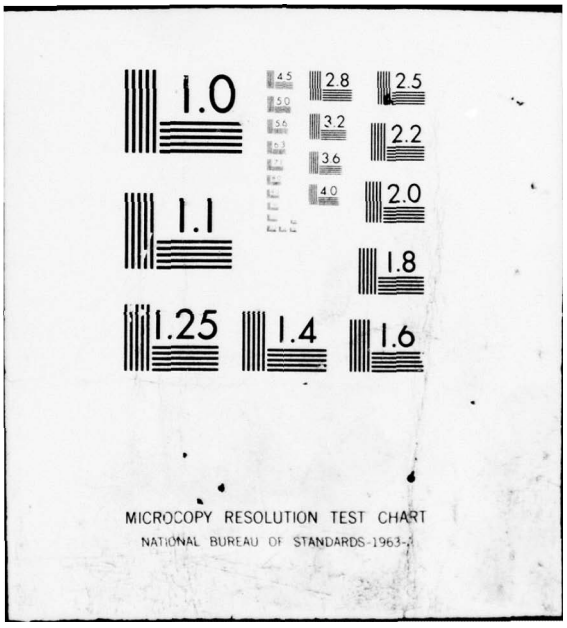
The system can create and edit Logos graphs through a interactive graphical/textual user interface called the Logos Design Package (LDP).

AD-A072 450 HONEYWELL INC MINNEAPOLIS MN SYSTEMS AND RESEARCH CENTER F/G 9/2
A FAULT TOLERANCE ASSESSMENT OF DAIS.(U)
MAR 79 W L HEIMERDINGER, K M FANT F33615-77-C-1232
UNCLASSIFIED 79SRC13 AFAL -TR-79-1007 NL

2 OF 2
AD
A072450



END
DATE
FILMED
9-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

LDP allows a user to "draw" graphs on a graphical display terminal by identifying objects and their positions. The user's inputs may be from one or all of the following devices: light pen, position cursor, and text specification. (There are a few cases where not all forms of input are appropriate, for example when specifying the name of an object cursor position is not very meaningful.)

In addition to the graphical interaction package, LDP provides database specification functions with the Graph Oriented Data Structures Language (GODSL) compiler, which together with the graph syntax specifies the structure and accessing of data. GODSL is a text language which performs a number of syntax checks when statements are entered, and other analysis, using the general analysis packages invoked via the ANALYZE command.

Finally, LDP contains a number of features for helping a designer use the system:

- extensive facilities for "walking" through graphs;
- a "graph move" command which allows relocation of nodes and the arcs connecting them;
- a query feature which the designer can use at (almost) any time to determine what actions are available to him;
- a "macro" facility which allows the designer to represent commonly repeated operations as data or expression operators and to insert these operators into his graphs.

The system permits any picture under display to be copied to other graphics devices on the system. In the TOPS-10 implementation at AFAL, this

consists only of the Calcomp plotter. Picture information is sent from the graphics packages to a file, which is postprocessed into commands to drive the plotter. The postprocessor accepts a scale factor so that plots from page to wall size can be produced.

ANALYSIS FACILITIES

The system implements the basic analysis functions listed below.

- At graph creation time, extensive syntax checking so that errors do not propagate through a design until global analysis.
- Mode matching of data structures at each end of a data transmission path, even if a variety of expressions and accessing operations are applied along the path.
- Expression analysis: to ensure that cyclical paths do not occur in an expression, i.e., an expression which does I+(J*K+(J*K+(...
- Block structured vector addition analysis which evaluates the reachable state tree of a schema, but only on the basis of "blocks" in the design which has the effect of reducing computational complexity of the analysis to something which can be calculated in a reasonable amount of time, a major advantage of the Logos representation.
- Global determinacy analysis between blocks through the determination and intersection checks of the data structures which are (potentially) read and written by one or more simultaneously active blocks.

The major items not present in the analysis but which are supported in the representation are procedure calls and recursion analysis.

LIMITATIONS OF CURRENT IMPLEMENTATION

The representation implemented by the system is missing various useful items described elsewhere in this report, specifically:

- no facility to specify transition times for operators;
- as a result of constraints imposed by the current analysis packages, the representation can not model real-world synchronization problems beyond the "pipelining" synchronization provided by the WE and RT control cells; the ability to reference control information between two or more operators would allow models to become more layered as opposed to the very flat, spread out models produced in this contract;
- data interpretation to reduce the number of reachable states in a design and to test fault-tolerance by deliberately inserting erroneous data.

The fact that the system is single user presents a number of problems in a multi-designer environment in that only one designer may be active at a time. This was experienced in the course of this contract and is discussed further in a later section. The rest of this section will deal with specific problems extant in the implementation at AFAL.

DISPLAY LIMITATIONS

The display subsystem within the current LOGOS package has a tendency to be slow and awkward. Much of this stems from attempts to make the display systems very general. The original intent of the funding agencies was to make the system available over the ARPANET, to a community which had a variety of graphics devices. The size and complexity of the system software required to define and drive a virtualized graphics device is a major overhead factor in the system. It may be far more reasonable in the future to orient the graphics towards a readily available and simpler form of graphics.

The devices which are in use tend to be of moderate resolution and lacking in the amount of memory necessary to hold picture descriptions. This becomes a very annoying problem when large, complex pictures are being displayed as the designer must worry about managing a limited display resource while trying to manage and comprehend a design.

DATABASE LIMITATIONS

The current database is limited in that about 3500 specific objects (arcs, nodes, labels, names, declarations, etc.) may be handled by the system. This is a design limitation in the LEAP associative database package contained in the implementation language, which is Stanford University's SAIL (Stanford Artifical Intelligence Language). Should the database problems described later not have occurred, the capacity of the system would very probably have been exceeded by the schema produced for this contract. A further problem is that the system maintains the entire

database in primary storage while the system is running. This means only one designer may be actively working on a design at any time.

ANALYSIS LIMITATIONS

The current analysis packages are missing features (procedure and recursion analysis) which the representation supports, as well as all of the representation enhancements developed specifically for this contract (time, data interpretation, etc.) Furthermore, a number of techniques for improving the execution time and eliminating redundant analyses performed by the existing Vector Addition System have not yet been implemented. These would reduce analysis time to permit the analysis of schema which can not be analyzed in a reasonable time by the current analyses. These and various niceties such as background analysis (the current analysis runs only as a foreground task) and the ability to do extensive exploration and interaction with the analysis packages would vastly improve the utility of the system.

SUMMARY OF EXPERIENCE WITH MODEL ON THIS CONTRACT

The LEAP database packages described above caused major problems during the use of the AFAL system. Specifically, the management routines would unexplicably start making errors in the construction of the database after the database became about 25% full. In general, it appeared that the LOGOS-specific database construction routine performed correctly. Memory limitations in the displays used (GT-40's) severely hampered the creation and management of large, complex displays. Larger displays at AFAL (GT-44's) were available during second and third shifts, but difficulties

in determining the proper configurations prevented their use. In general, the usage of the system represented a much greater test than any it had faced before, and much of the system appeared to work properly.

SECTION VIII

CONCLUSIONS

ASSESSMENT OF DAIS FAULT TOLERANCE

As stated earlier, any assessment of system level fault tolerance using today's assessment tools must be largely subjective. It is clear that DAIS relies heavily on the pilot to control top level fault recovery mechanisms. This is appropriate for the applications now forseen for DAIS; the gathering of information for display to the pilot and the transmission of pilot requests to various equipments. The pilot can monitor the behavior of most of the functions of the system by observing displays or equipment performance, thus providing a top level of system supervision.

The DAIS core system is well protected against Loss of Control faults. The design consistently uses silence or inactivity as a means for signalling faults. Thus most error routines halt the processor and its BCIU, allowing other units to discover the error through missing responses or through time-outs. This approach minimizes the potential for interference between faulty units and correctly operating units.

The DAIS core system is more vulnerable to Multiple Instance of Control faults. The master executive error retry procedures have limits to prevent an undue number of message retry attempts, but the core system has no way to detect an abnormal number of asynchronous message

requests until the master executive gives up because of an inability to keep up with minor cycle timing constraints.

The system is also vulnerable to Diversion of Control faults involving terminal addresses. The system has numerous checks to insure that a correctly addressed remote BCIU or RT that misbehaves is promptly isolated. The isolation mechanism consists of deleting all messages addressed to the suspect unit, presumably disconnecting that unit from system activity. This approach does not successfully deal with a remote BCIU or RT that becomes an alias by responding to messages directed to other units. The spurious response generated by the alias will usually conflict with the legitimate response, a situation that will invoke error handling, but which will probably result in the elimination of the correctly operating remote unit rather than its alias. This situation is not corrected by backup or recovery procedures since the RT configuration tables remain unchanged by backup or recovery. A configuration manager capable of remotely switching power to RTs is required, because a persistent aliasing unit may conceivably respond to a large number of addresses on both busses, thus strangling the multiplex system.

ASSESSMENT OF THE FUNCTIONAL/GRAPHICAL APPROACH

The introduction of a model necessarily makes analysis of a system a two step process; first, construction of the model with verification that the model actually summarizes the behavior of the system, and second, analysis of the model to determine fault tolerance.

There are several advantages to this two step process. The graphical model makes apparent both the extent and the level of system detail studied.

The model may be used as a communications medium between the fault tolerance analyst and the designers of the system to be analyzed. It also may be used to explain the interaction of complex fault tolerance mechanisms with the rest of the system.

Over thirty graphical models were created during this contract. Some are not included here because they deal with an isolated portion of system functionality or are at too low a level of system detail. We began to model at the bus message level of detail and then worked up to less detailed levels, searching for the top level of system detail. The graph depicted in Figure 7 resulted from many iterations. In most cases, we attempted to eliminate portions of the models that could be represented in another graph at a lower level of detail. In some cases, we added functions to models when it became apparent that the omitted function interacted directly with other functions at the level of detail of that model.

The top level model, Figure 7, was once smaller, but did not fully treat the functionality required for reconfiguration. When this was added, the model grew to its present size. We believe that the complexity of this graph is probably necessary to completely represent all of the top level DAIS functions that involve system level fault tolerance.

A number of engineers skilled in computer architecture and software reviewed the graphical models after a short tutorial on interpretation of the graphical models and on basic DAIS nomenclature. They generally were able to deduce the top level behavior of the DAIS system from the model and an accompanying English language description. We found

their interpretation of the models to be uniform. Thus we believe that the graphical model is a relatively unambiguous method of summarizing fault tolerance information.

The models presented here were difficult to produce, requiring several iterations before a balance was struck between complexity and completeness. We attribute these difficulties in part to the complexity of the DAIS fault tolerance mechanisms, involving interactions between the pilot, the core system hardware, and the various software executives, and in part to inexperience in producing multiple level graphical models. We feel that the problems associated with inexperience with graphical models will continue to diminish rapidly with continued use of the graphical approach.

We know of no other fault tolerance assessment approach that allows the study of interactions between concurrently active sources as diverse as human operators, hardware, and software in as natural a representation as the labelled graph model used here. There is still a vast distance between our present tools and really effective tools for fault tolerance assessment, but the approach successfully demonstrated in this contract is one of the most promising directions to pursue.

APPENDIX A

DETAILED DAIS GRAPHICAL MODELS

APPENDIX A

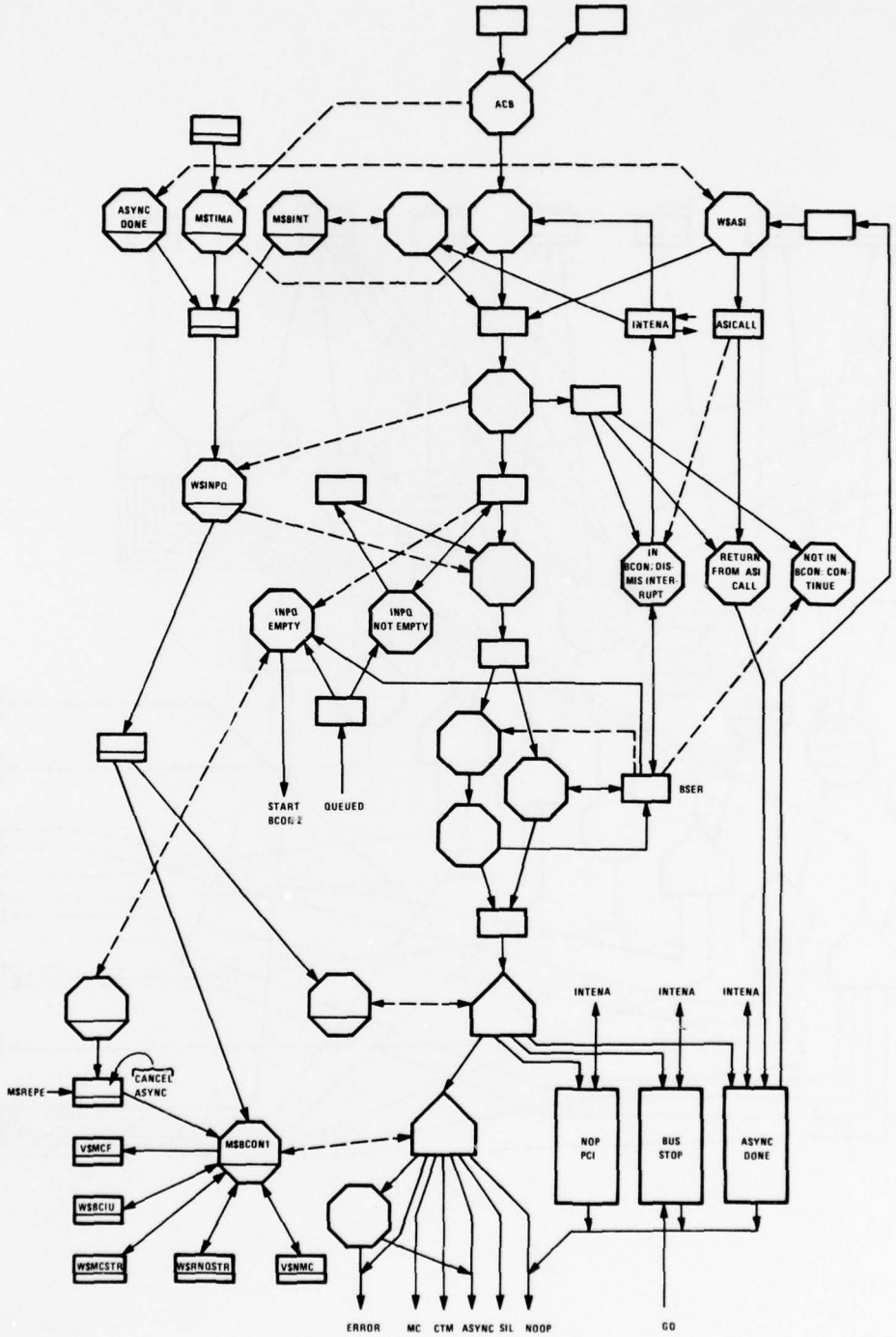
DETAILED DAIS GRAPHICAL MODELS

The upper level DAIS system models contained in Section IV were based on an understanding of the operation of the DAIS system gained from lower level models. Most of these models were generated from the DAIS Bus Control Executive flow charts in the DAIS Mission Software Product Specification - Executive, Volume 2: Bus Control.

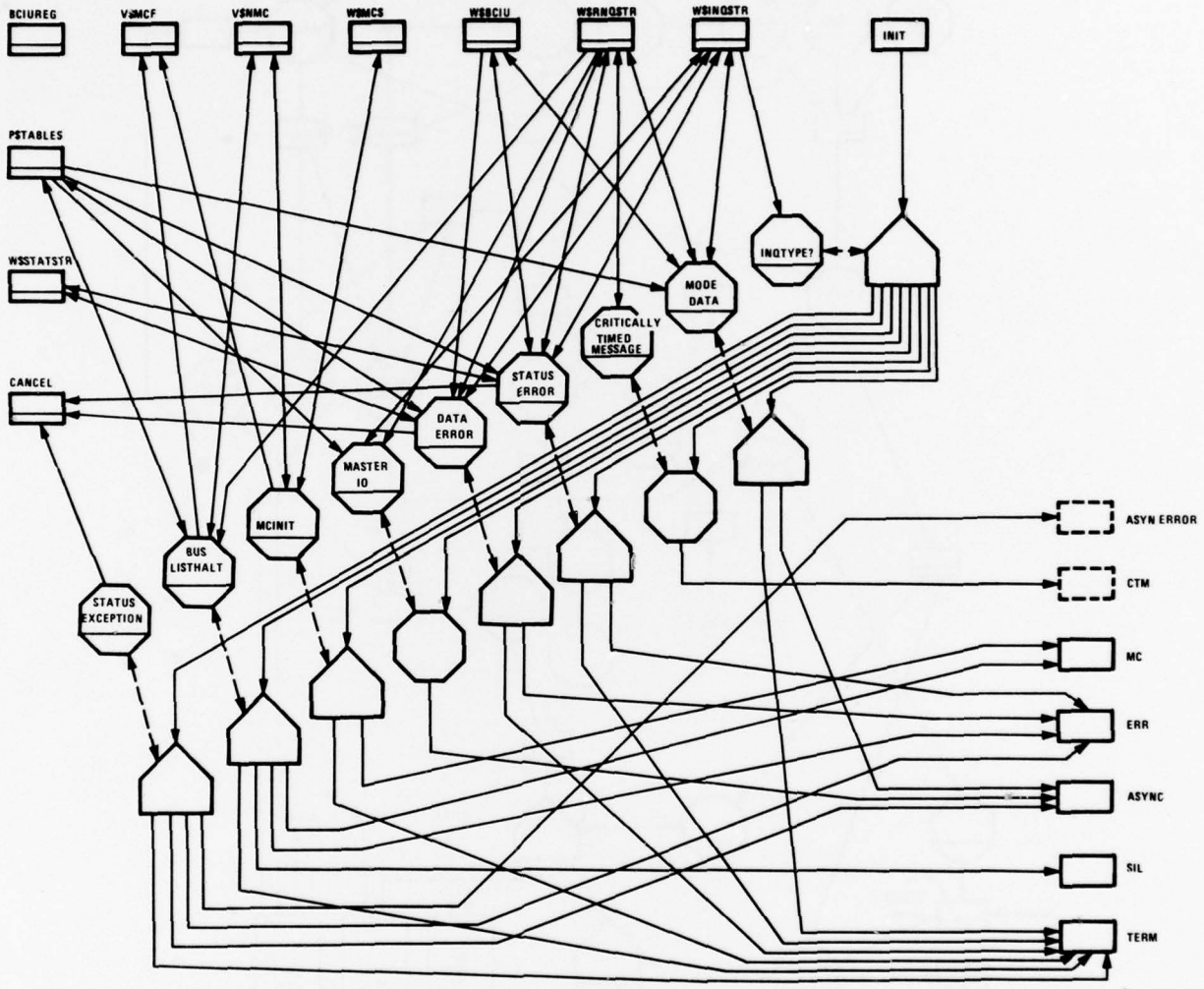
The first model is a top level model of the bus control executive software. The major component of this software is the Bus Control Routine (M$BCON). The operation of this routine naturally splits into two major components; these are portrayed in two separate models, BCON-1 and BCON-2. The majority of the detailed models show the behavior of subroutines in BCON-1.

Three additional master executive bus control routines, NOP'PCI, BUS'STOP, and ASYNC'DONE are also modeled individually.

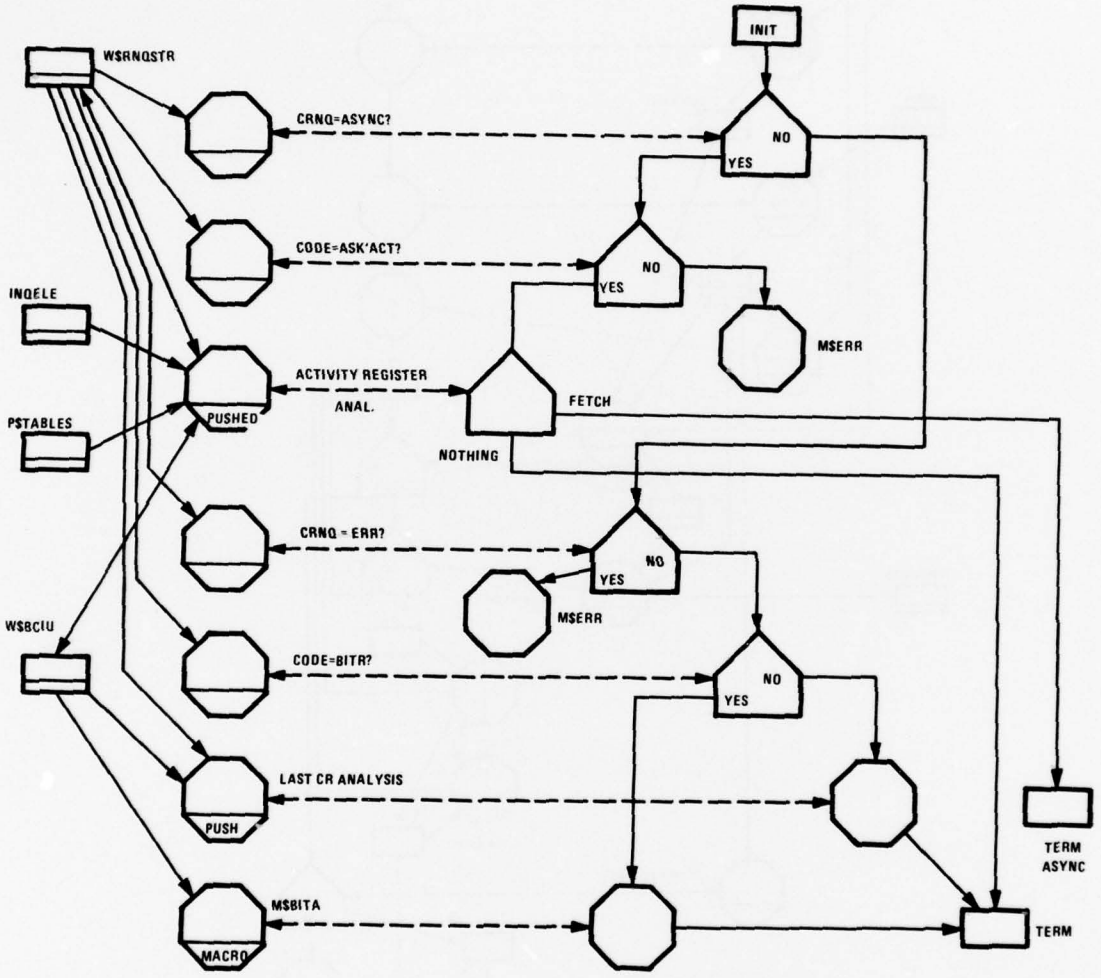
The final two detailed models show the operation of two local executive routines directly concerned with bus control; these are X$ATR0 and X$AREC. These were generated from the flow charts given in the DAIS Mission Software Product Specification - Executive, Volume 1: Local Executive.



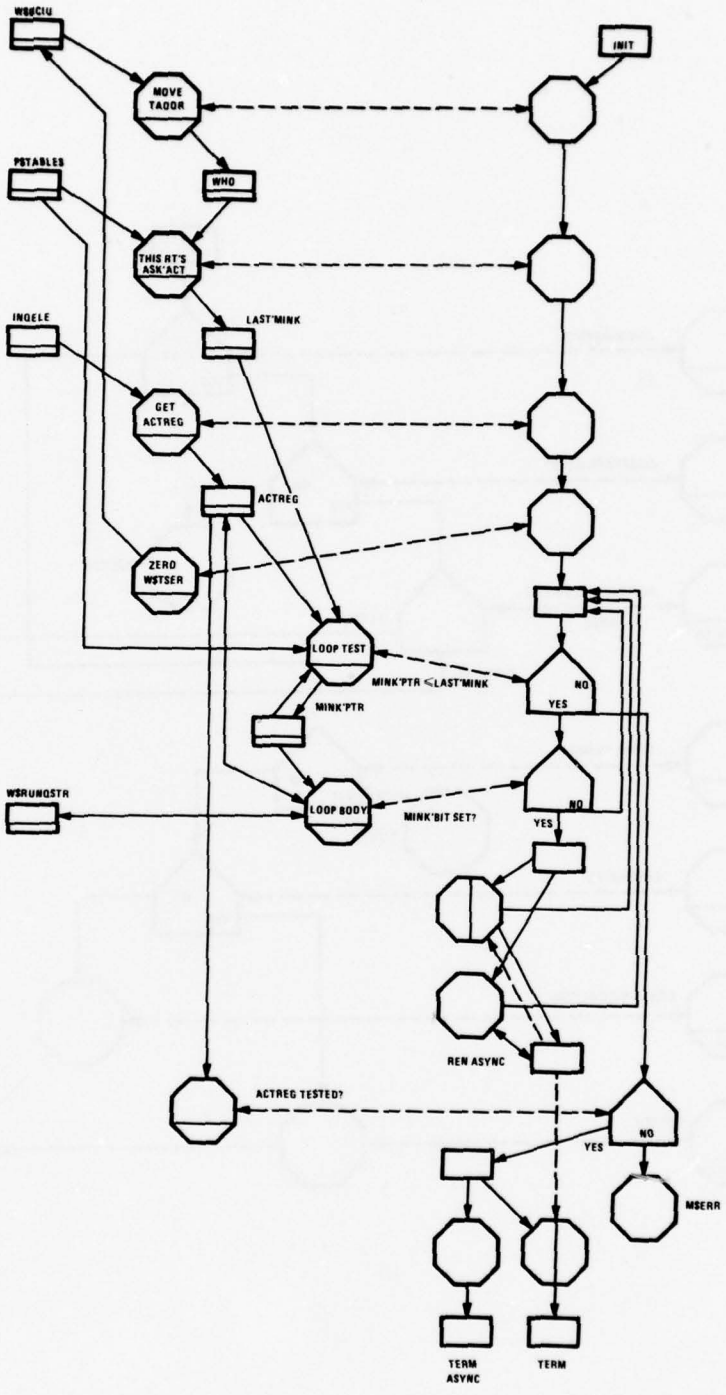
DAIS MASTER EXECUTIVE (TOP LEVEL)


BCIUREG
VSMCF
VSNMC
WSMCS
WSSCIU
WSRNQSTR
WSINQSTR
INIT
PSTABLES
WSSTATSTR
CANCEL
INQTYPE?
MODE DATA
CRITICALLY TIMED MESSAGE
STATUS ERROR
DATA ERROR
MASTER IO
MCINIT
BUS LISTHALT
STATUS EXCEPTION
ASYN ERROR
CTM
MC
ERR
ASYNC
SIL
TERM

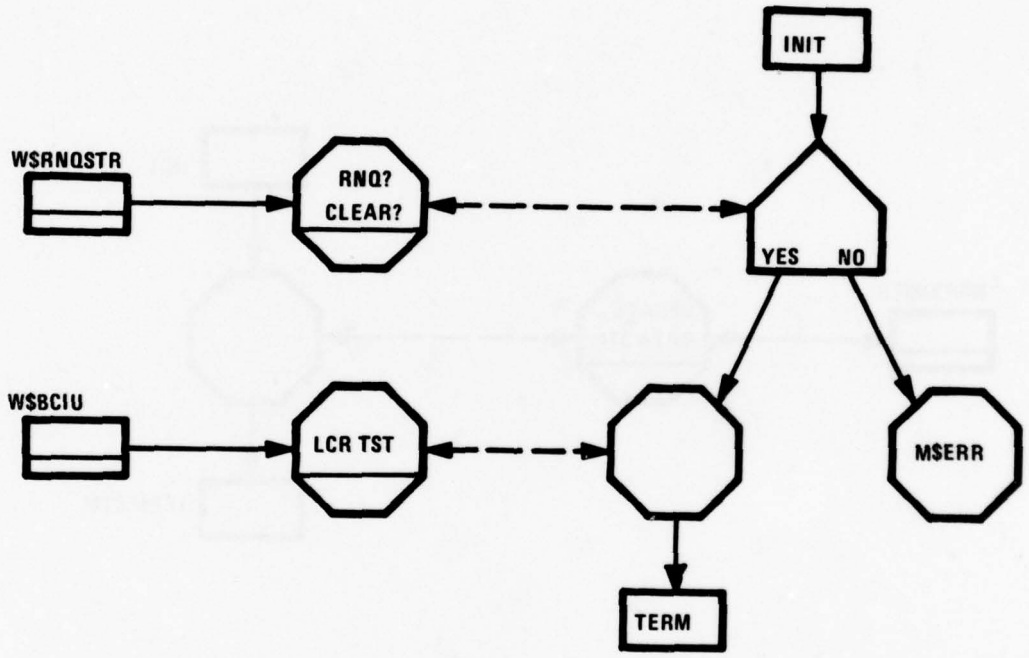
BCON-1



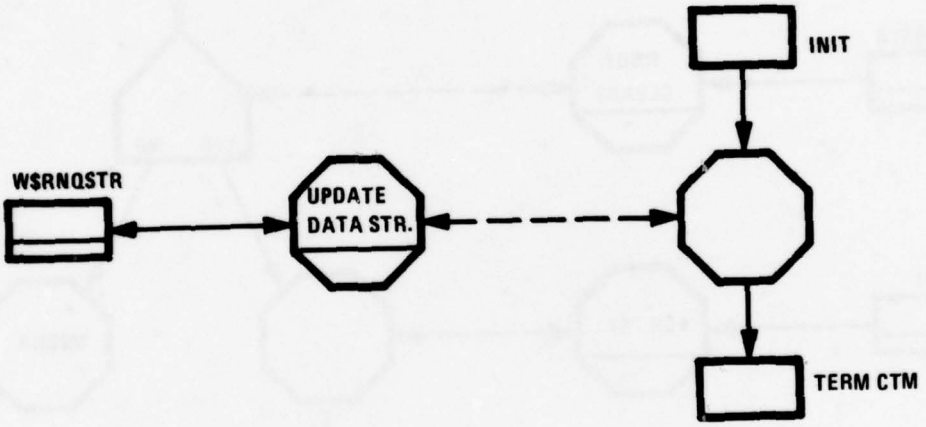
MODE' DATA IN BCON-1



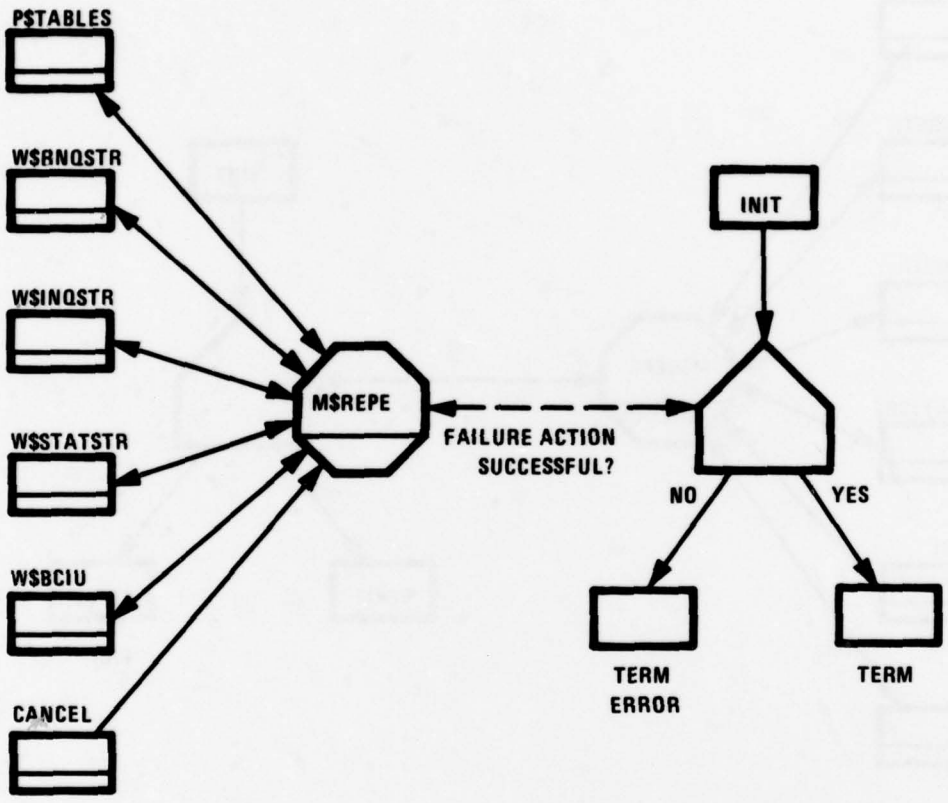
ACTIVITY REGISTER ANALYSIS IN MODE DATA IN BCON-1



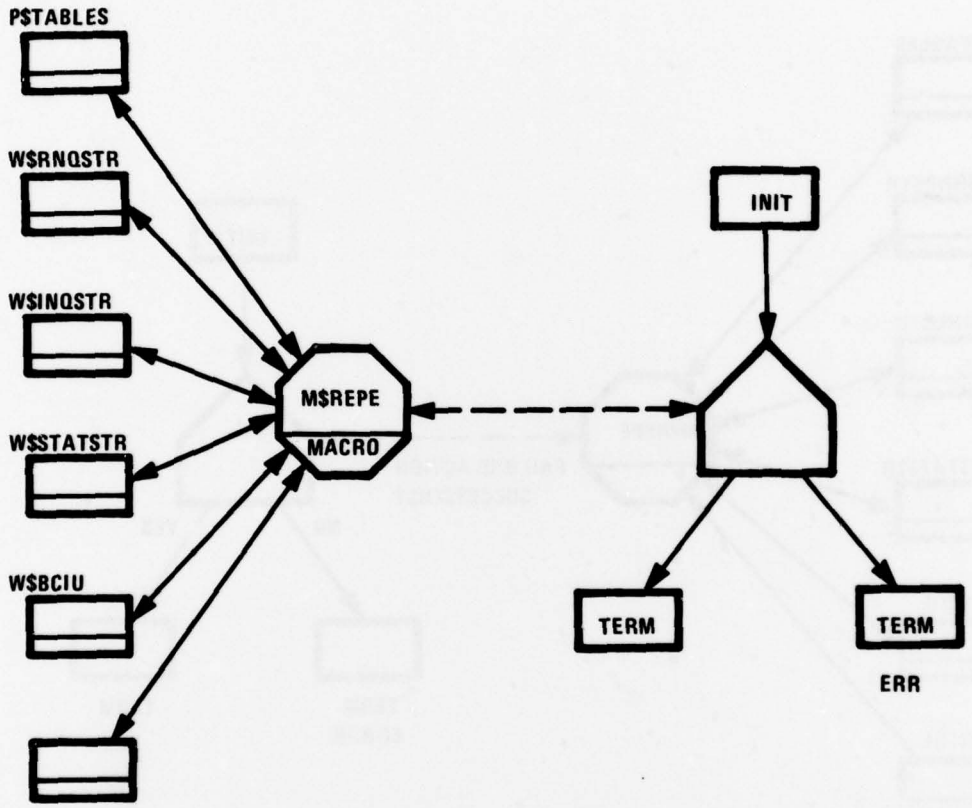
LAST CR ANALYSIS IN MODE DATA IN BCON-1



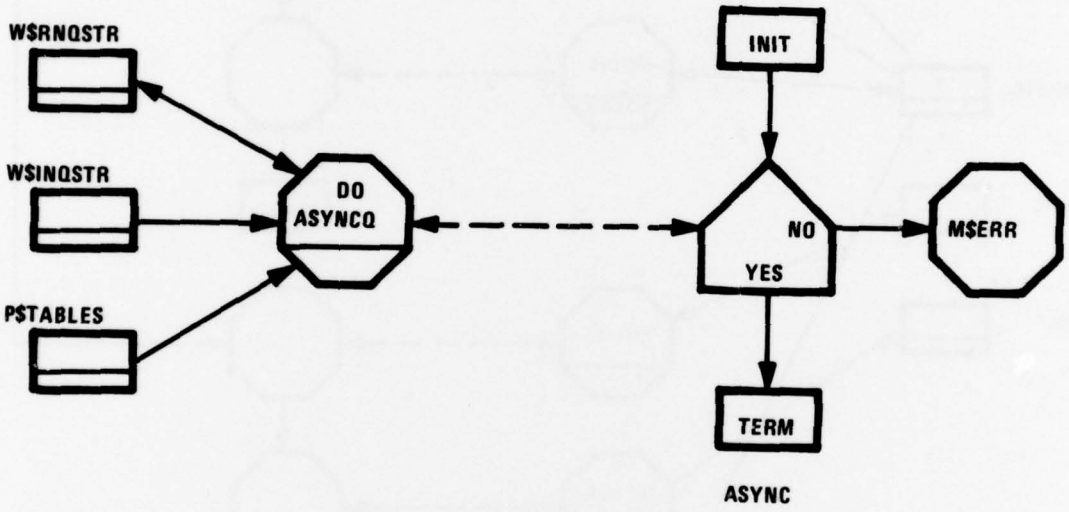
CRITICALLY TIMED MESSAGE IN BCON-1



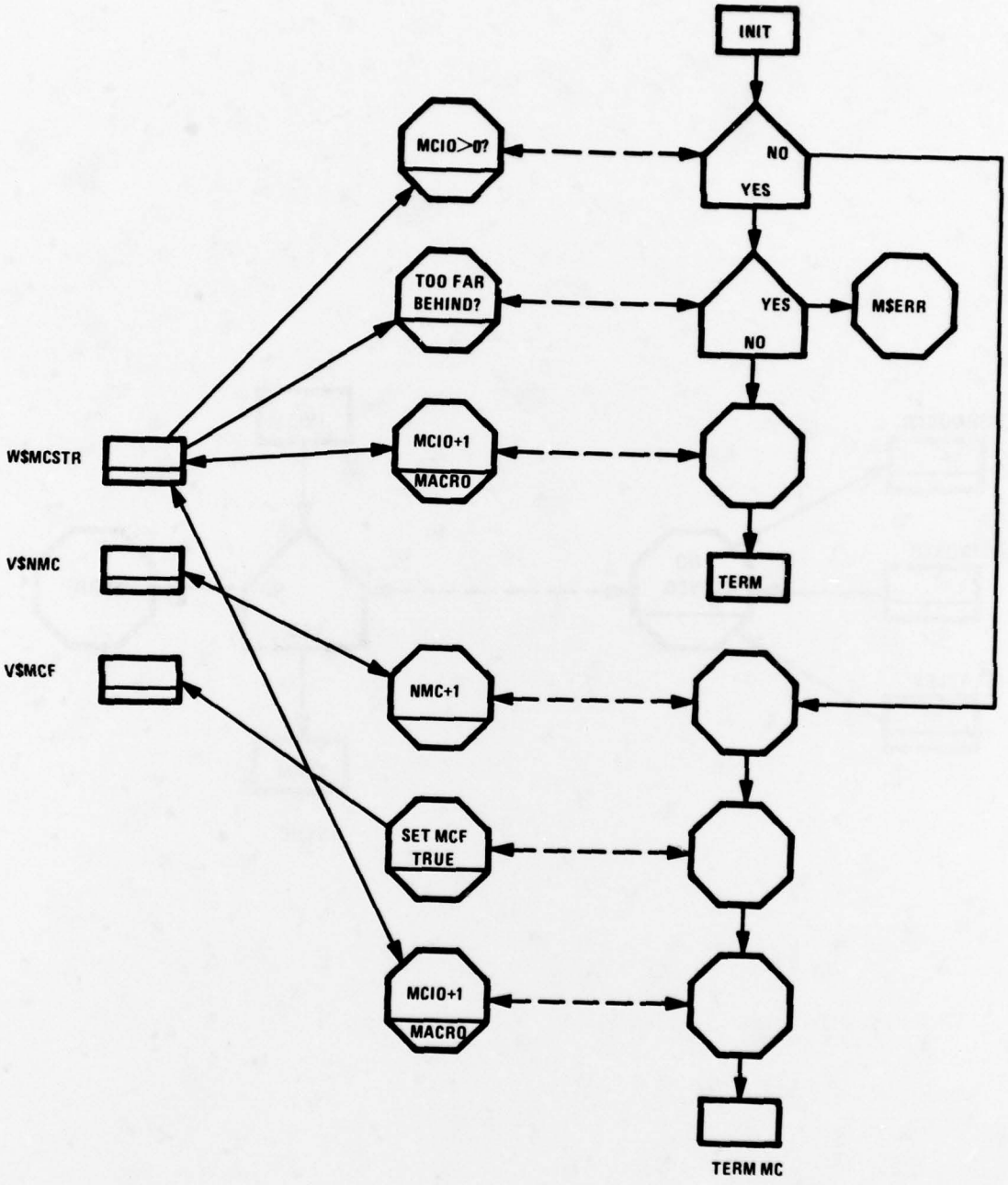
STATUS ERROR IN BCON-1



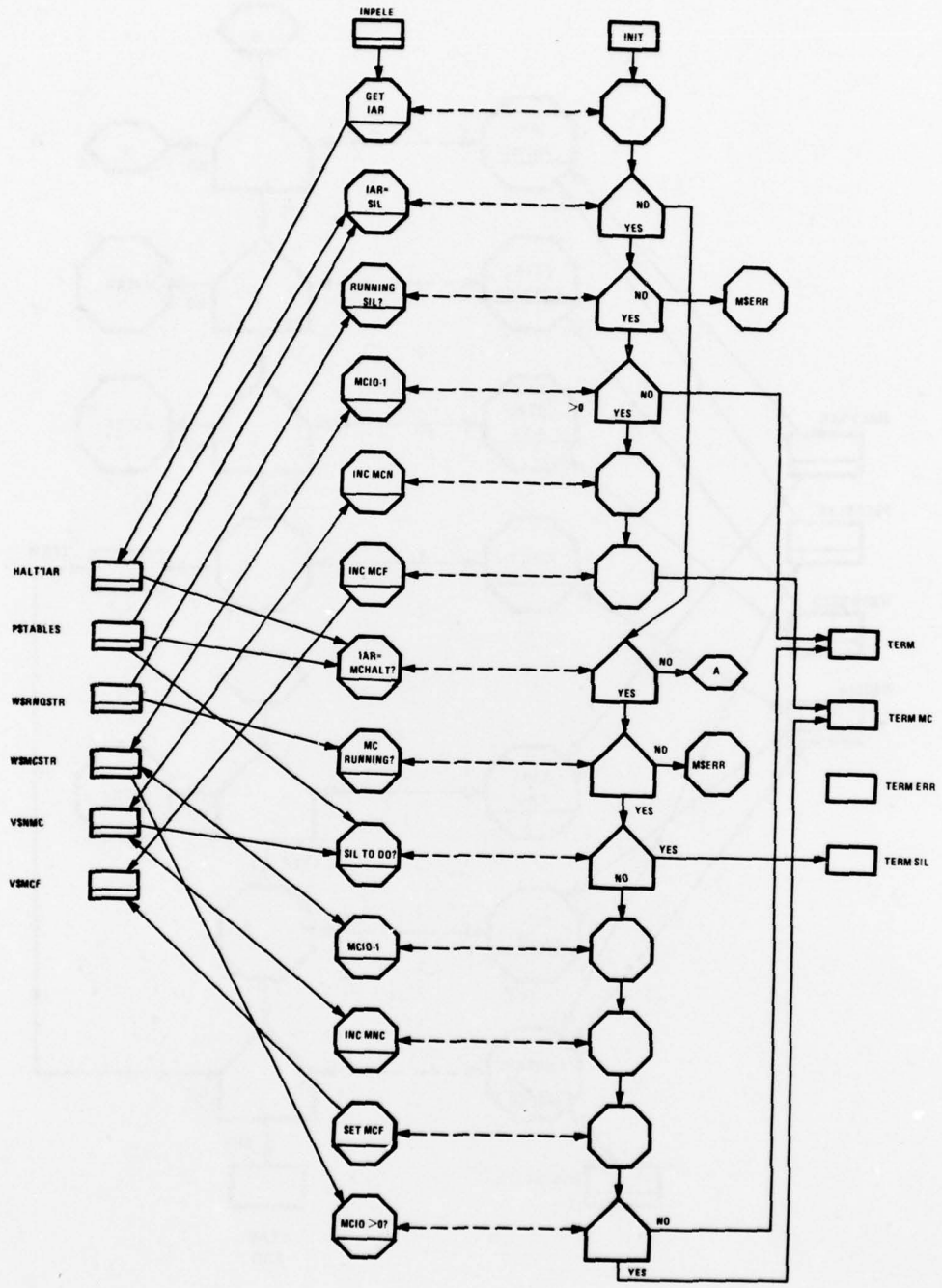
DATA ERROR IN BCON-1



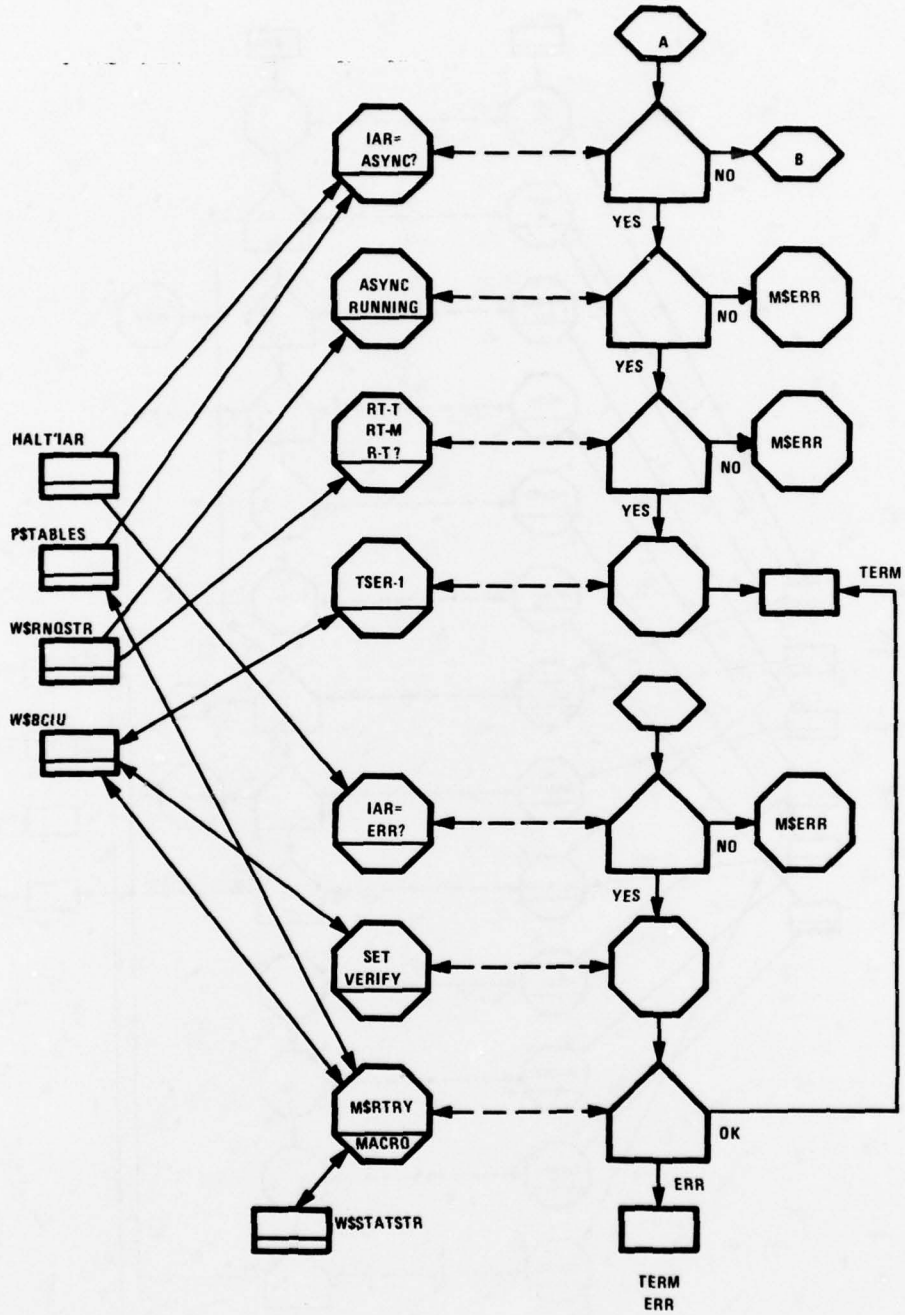
MASTER IO IN BCON-1



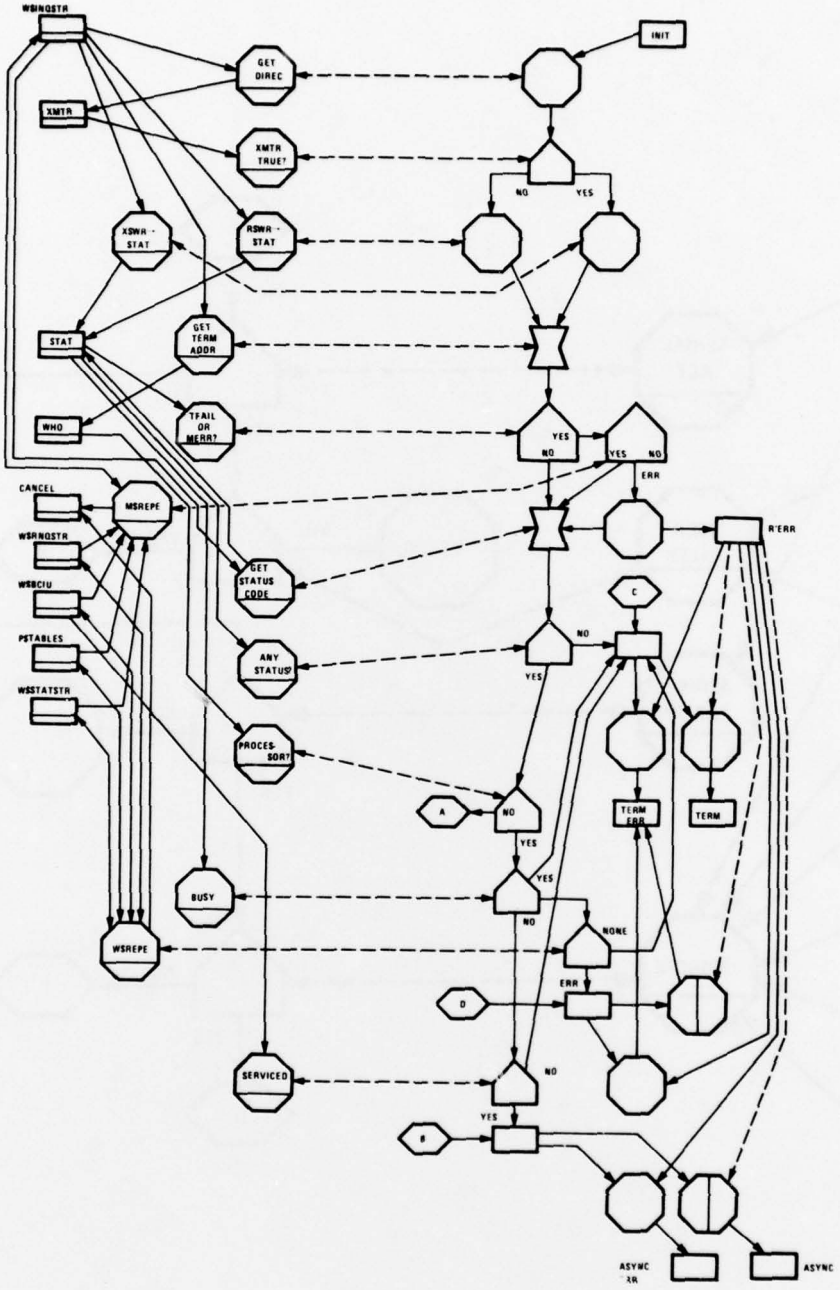
MC INIT IN BCON-1



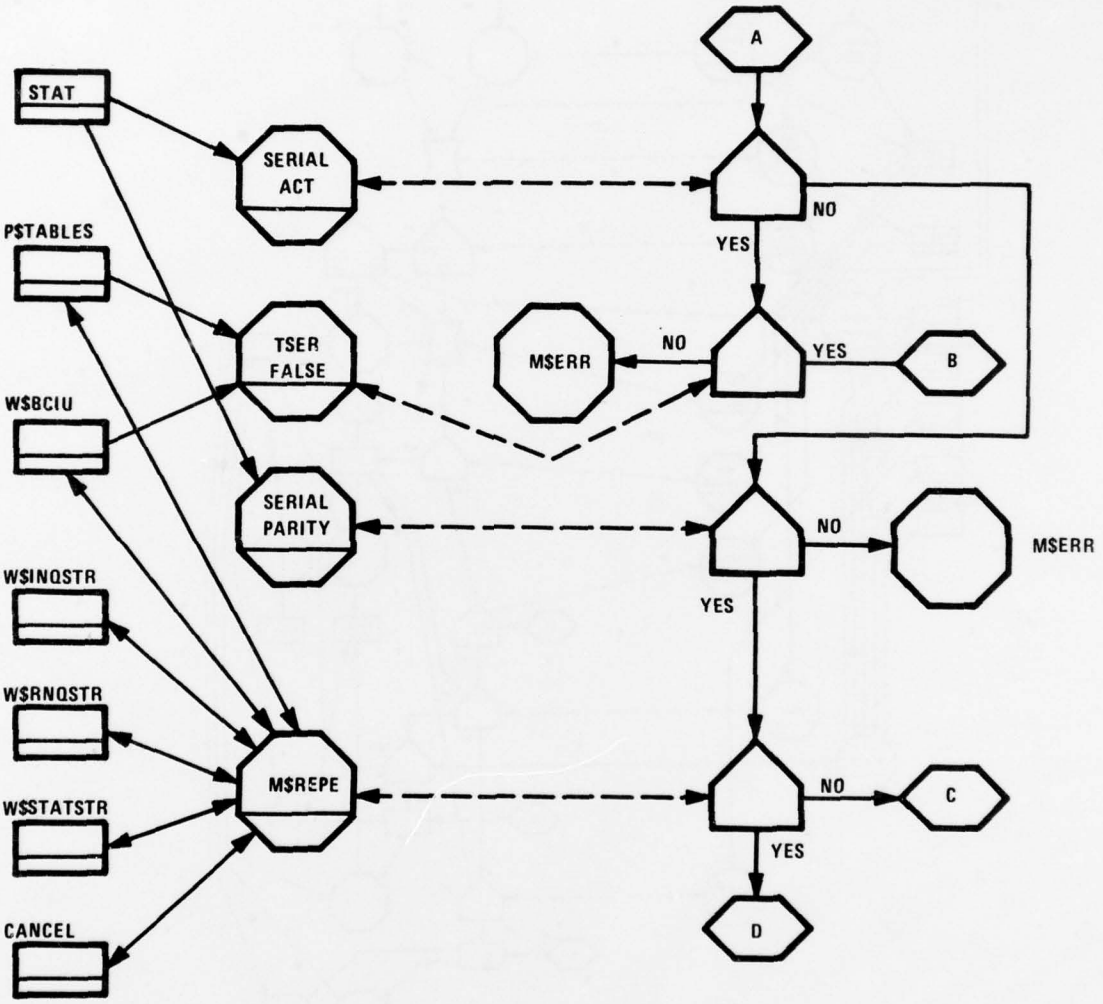
BUS LIST HALT IN BCON-1 (PAGE 1)



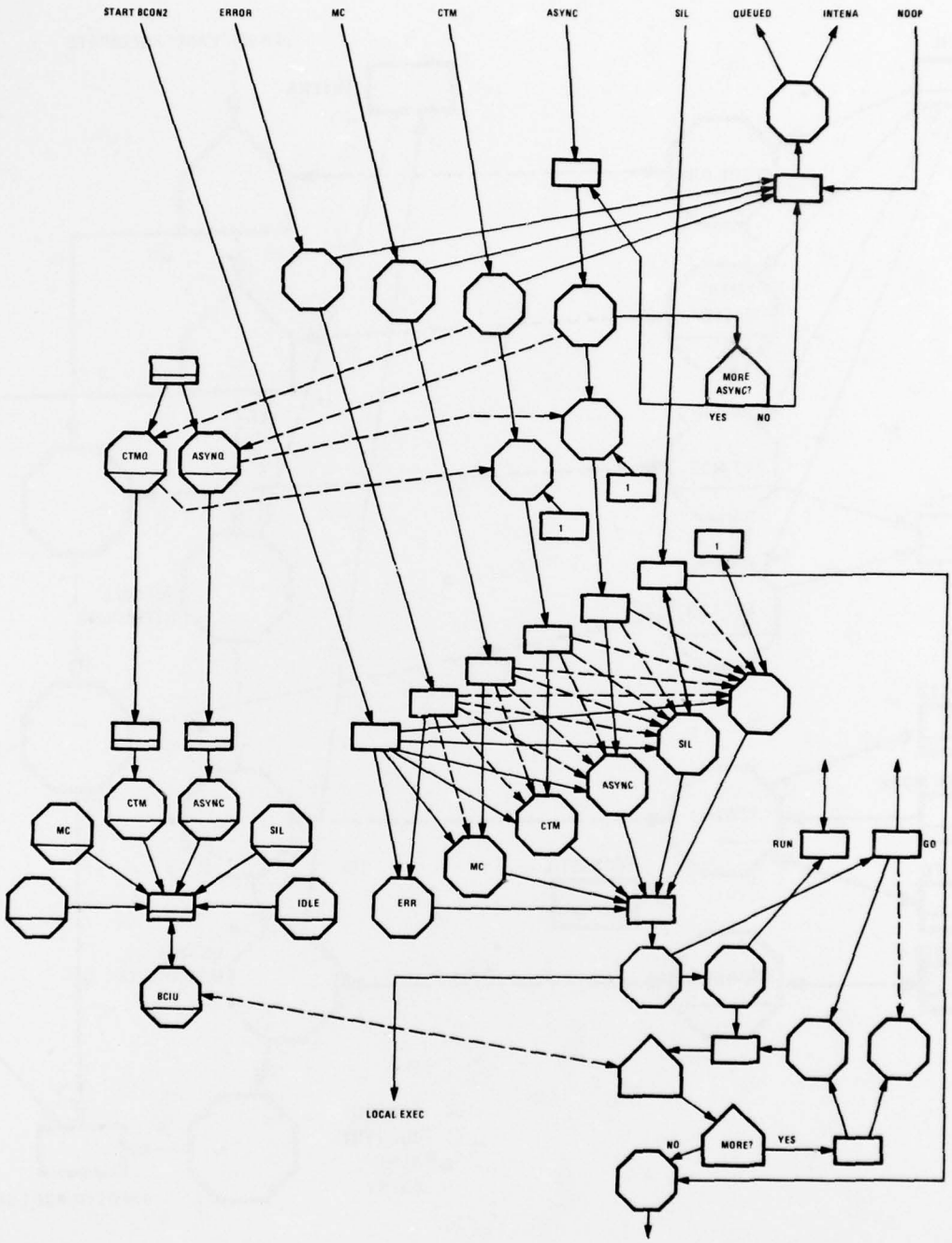
BUS LIST HALT IN BCON-1 (PAGE 2)



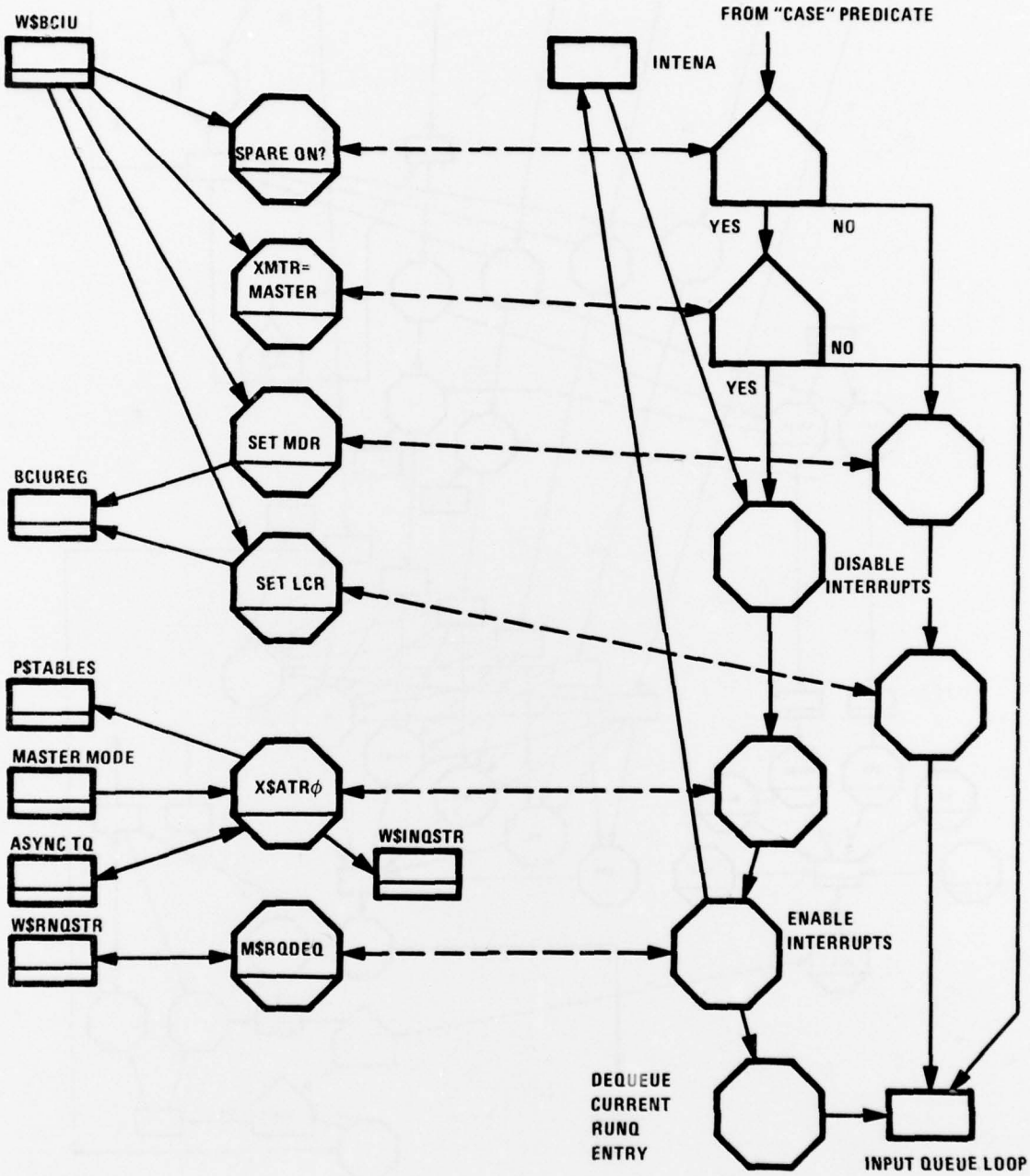
STATUS EXCEPTION IN BCON-1 (PAGE 1)



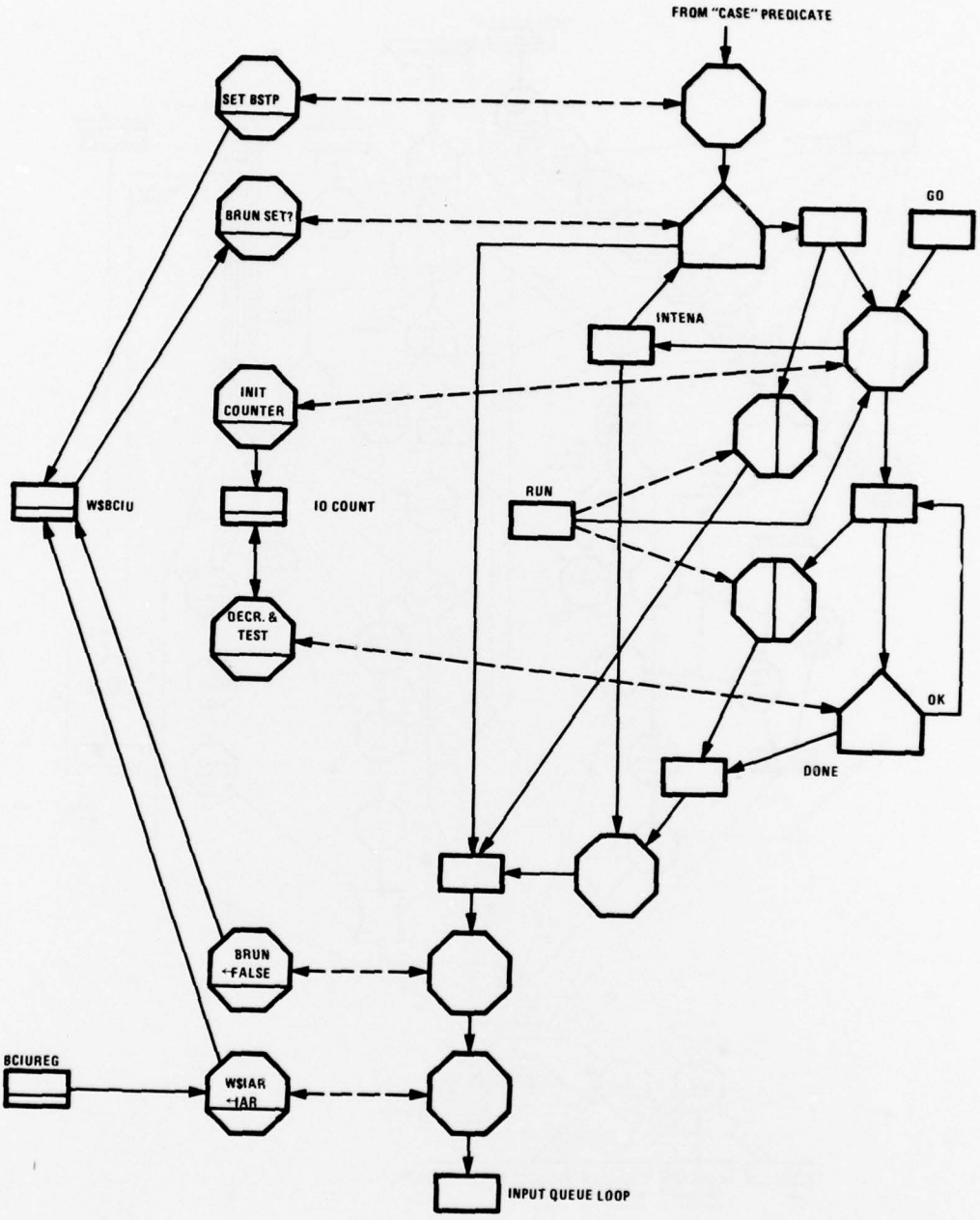
STATUS EXCEPTION IN BCON-1 (PAGE 2)


START BCON2
ERROR
MC
CTM
ASYNC
SIL
QUEUED
INTENA
NOOP
MORE ASYNC?
YES
NO
CTMQ
ASYNQ
1
1
1
CTM
ASYNC
SIL
MC
IDLE
ERR
SIL
ASYNC
CTM
MC
RUN
GO
BCIU
LOCAL EXEC
NO
MORE?
YES

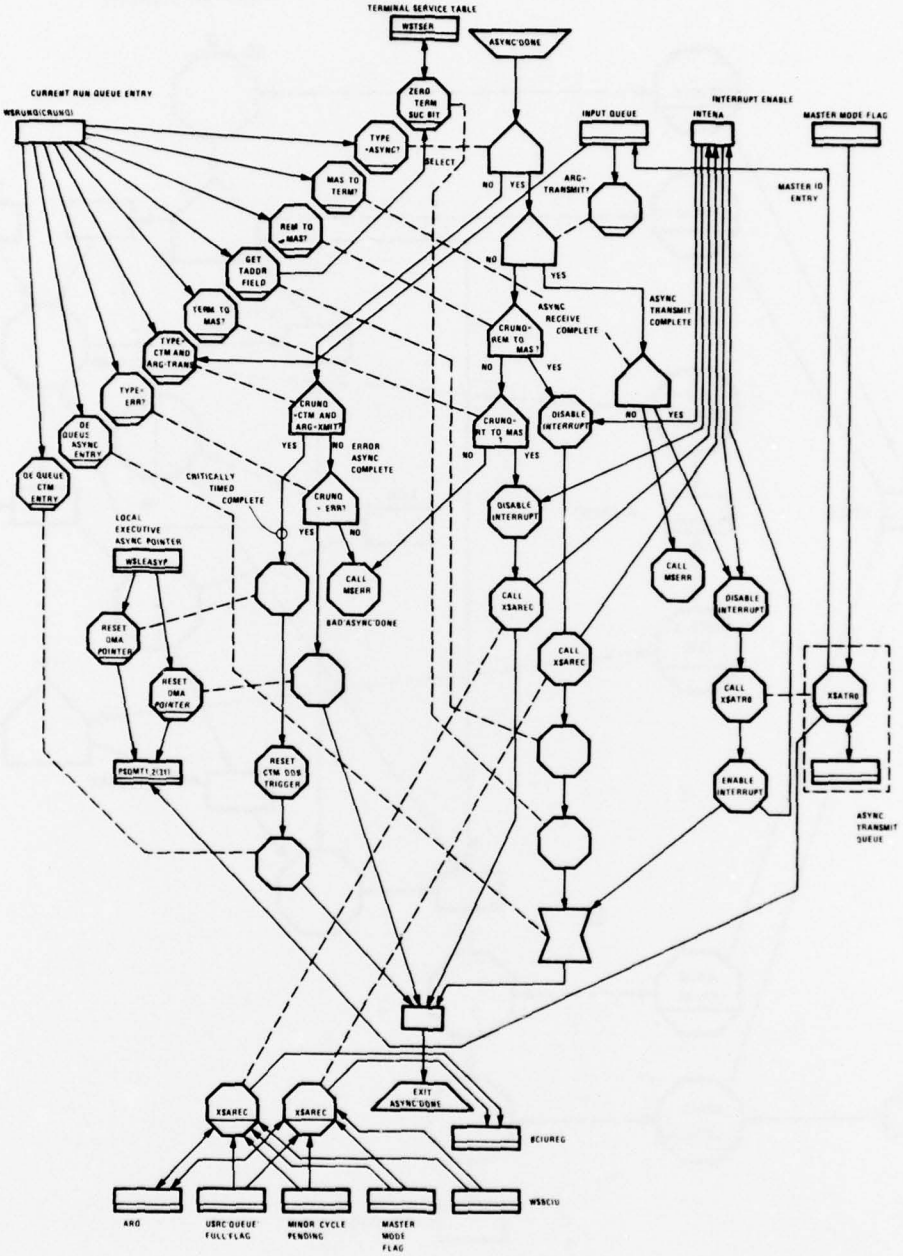
BCON-2


W$BCIU
FROM "CASE" PREDICATE
INTENA
SPARE ON?
YES
NO
XMTR=
MASTER
NO
YES
SET MDR
BCIUREG
SET LCR
DISABLE
INTERRUPTS
P$TABLES
MASTER MODE
X$ATRϕ
W$INQSTR
ASYNC TQ
W$RNQSTR
M$RQDEQ
ENABLE
INTERRUPTS
DEQUEUE
CURRENT
RUNQ
ENTRY
INPUT QUEUE LOOP

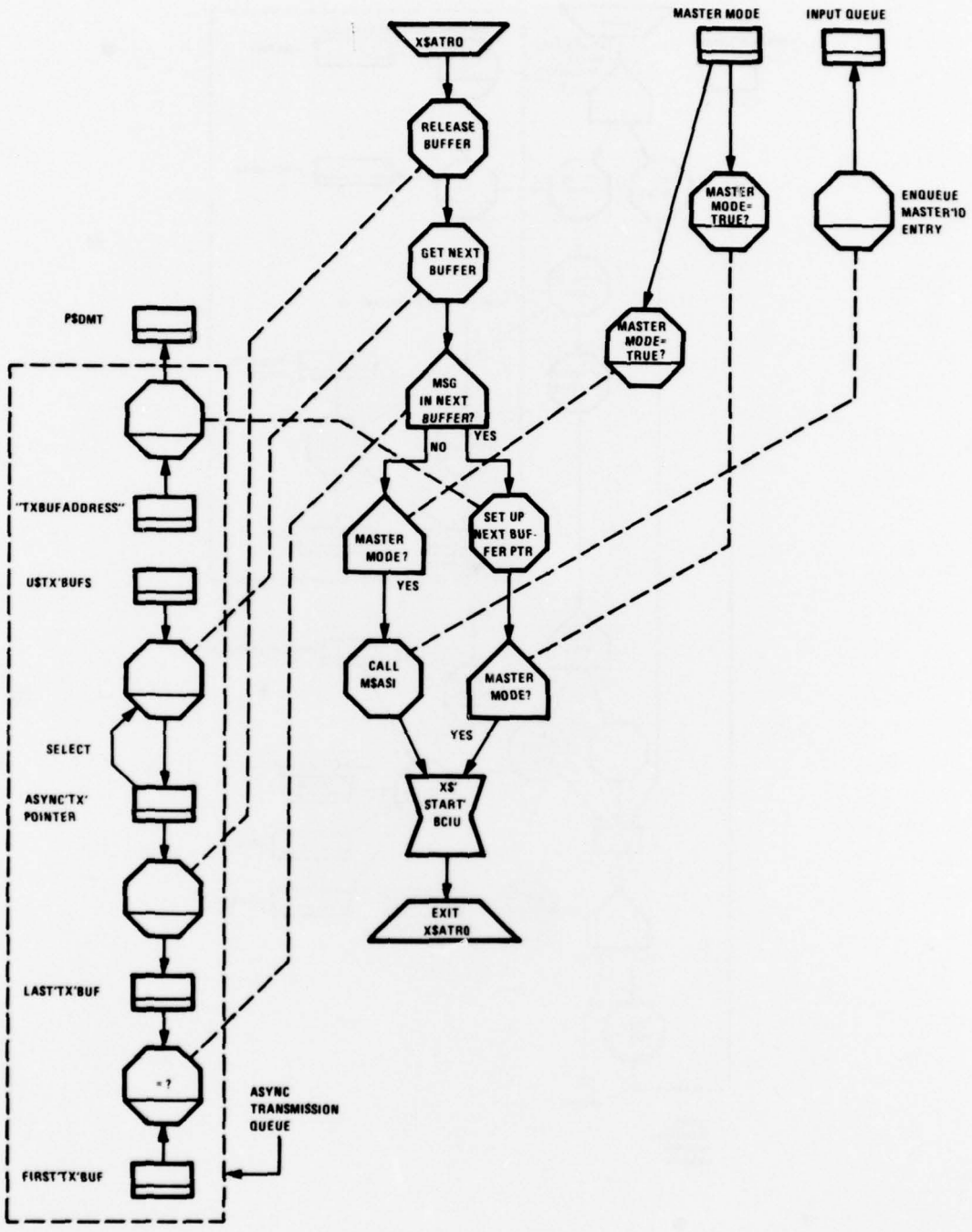
NOP PCI


FROM "CASE" PREDICATE
SET BSTP
BRUN SET?
GO
INTENA
INIT COUNTER
WSBCIU
IO COUNT
RUN
DECR. & TEST
OK
DONE
BRUN ←FALSE
BCIUREG
WSIAR ←IAR
INPUT QUEUE LOOP

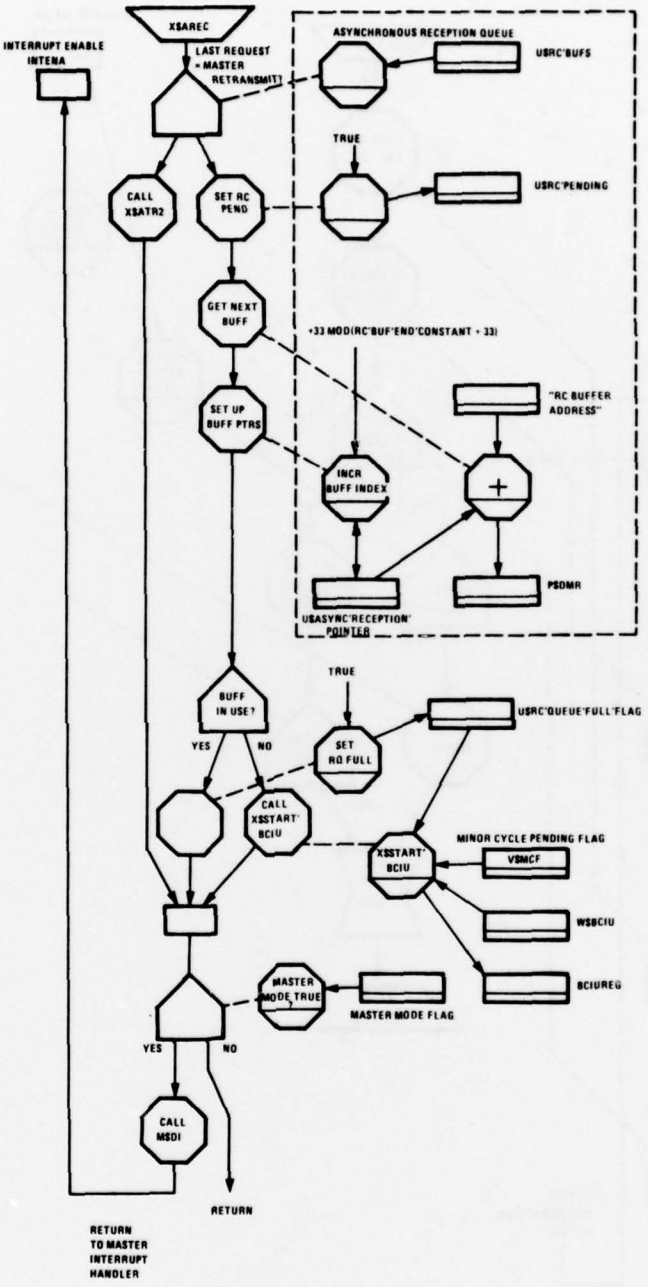
BUS STOP


TERMINAL SERVICE TABLE
WSTSER
ASYNC'DONE
CURRENT RUN QUEUE ENTRY
WSRUNQ(CRUNQ)
ZERO TERM SUC BIT
TYPE =ASYNC?
SELECT
INPUT QUEUE
INTERRUPT ENABLE
INTENA
MASTER MODE FLAG
MAS TO TERM?
REM TO MAS?
ARG=TRANSMIT?
NO
YES
MASTER IO ENTRY
GET TADDR FIELD
TERM TO MAS?
CRUNQ=REM TO MAS?
ASYNC RECEIVE COMPLETE
ASYNC TRANSMIT COMPLETE
TYPE=CTM AND ARG=TRANS
TYPE=ERR?
CRUNQ=CTM AND ARG=XMIT?
CRUNQ=PT TO MAS?
DISABLE INTERRUPT
DE QUEUE ASYNC ENTRY
DE QUEUE CTM ENTRY
ERROR ASYNC COMPLETE
CRITICALLY TIMED COMPLETE
CRUNQ=ERR?
LOCAL EXECUTIVE ASYNC POINTER
WSLEASYP
CALL MSERR
BAD ASYNC'DONE
RESET DMA POINTER
CALL XSAREC
CALL XSATRB
ENABLE INTERRUPT
ASYNC TRANSMIT QUEUE
PSDMTT(2)(3)
RESET CTM DDS TRIGGER
EXIT ASYNC'DONE
BCIUREG
WSBCIU
ARG
USRC QUEUE FULL FLAG
MINOR CYCLE PENDING
MASTER MODE FLAG

ASYNC DONE



X$ ATR0 (LOCAL EXECUTIVE)



X$ AREC (LOCAL EXECUTIVE)

☆U.S.Government Printing Office: 1979 — 657-002/682