FAA WJH Technical Center
00092838

DOT/FAA/CT-86/33

FAA TECHNICAL CENTER
Atlantic City International Airport
N.J. 08405

N-Version Software Demonstration for Digital Flight Controls

D.B. Mulcare
L.A. Barton

Lockheed-Georgia Company
A Division of Lockheed Corporation
Marietta, Georgia 30063

April 1987

Final Report

This document is available to the U.S. public through the National Technical Information Service, Springfield, Virginia 22161.

U.S. Department of Transportation
Federal Aviation Administration

DOT/FAA/CT-86/33

N-VERSION SOFTWARE SPECIFICATION, DESIGN, AND DEMOMSTRATION FOR DIGITAL FLIGHT CONTROLS

Dennis B. Mulcare and Lynn A. Barton

Prepared for the

Federal Aviation Administration

Under Contract NAS2-11853

Lockheed-Georgia Company Engineering Report LG86ER0163

29 May 1987

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
DOT/FAA/CT-86/33		

4. Title and Subtitle	5. Report Date
N-Version Software Demonstration for Digital Flight Controls	March 1987
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
D. B. Mulcare, L. A. Barton	DOT/FAA/CT-86/33
9. Performing Organization Name and Address	10. Work Unit No.
Lockheed-Georgia Company Marietta, Georgia 30063	NAS2-11853
	11. Contract or Grant No.
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address U.S. Department of Transportation Federal Aviation Administration Technical Center Atlantic City Airport, New Jersey 08405	Contractor Report
	14. Sponsoring Agency Code

15. Supplementary Notes

Point of Contact: W. E. Larsen/MS 210-2
Ames Research Center
Moffett Field, CA 94035

16. Abstract

This report illustrates how four independently developed versions of digital flight controls applications software might be used in a quadruplex system architecture. This approach to software fault tolerance is called N-version software. Here each computer channel has distinct versions of Ada programming units performing the same functions concurrently. Since intermediate software results are voted to detect and isolate discrepant computations, cross-channel synchronization occurs at each voting plane. The demonstration of this system was based on a high-level software design, English language specifications, and associated Ada program unit specifications parts. The demonstration was performed in non-realtime on a single VAX 8600 computer using an Ada multitasking test harness to effect voting plane synchronization and test case application and analyses.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Ada Programming Language, Digital Flight Controls, Multitasking Test Harness, N-Version Programming, Quadruplex Redundancy, Software Fault Tolerance, Software Specification, Software Testing	Unlimited Subject Category 38

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified		

*For sale by the National Technical Information Service, Springfield, Virginia 22161

Foreword

This report describes the specification, design, and testing a digital flight control system (DFCS) software that has been prepared under an FAA-sponsored program entitled "Methods for the Verification and Validation of Digital Flight Control Systems," as Subtask 4.5.2.1 of Contract NAS2-11853, Modification 1. The intent has been to conduct an N-version programming demonstration illustrative of DFCS software fault tolerance for a quadruplex architecture. Accordingly, four independently developed versions of applications software were coded and demonstrated in respective DFCS channels.

Considerable background information is presented, largely of a system or software design nature. First, higher level software encompassing the N-version software is described, including a multitasking test harness and the foreground executive programs for the four DFCS channels. Coded in Ada R , the interfaces for this software were set up for the insertion of the N-version applications modules and the associated software voters. These applications modules were then developed in accord with the respective DFCS program unit specifications.

This report has also been published as Lockheed-Georgia Company Engineering Report LG86ER0163.

R Registered Trademark, U.S. Government Ada Joint Program Office

TABLE OF CONTENTS

Section	Topic	Page
1.0	INTRODUCTION	1
1.1	Software Fault Tolerance	1
1.2	Demonstration Guidelines	4
1.3	Ada Programming Language	7
1.4	Anna Specification Language	8
2.0	SYSTEM DESIGN	9
2.1	System Description	9
3.0	SOFTWARE DESIGN	19
3.1	DFCS Software Design	19
3.2	DFCS Applications Software	25
4.0	MULTIRATE EXECUTIVE DESCRIPTION	39
5.0	SELECT_MODES PROCEDURE SPECIFICATION	41
6.0	ASSESS_CHANNEL PROCEDURE SPECIFICATION	45
7.0	GIVE_STATUS PROCEDURE SPECIFICATION	49
8.0	MANAGE_AL_INPUTS PROCEDURE SPECIFICATION	53
9.0	CALC_AUTOLAND PROCEDURE SPECIFICATION	57
10.0	MANAGE_IL_INPUTS PROCEDURE SPECIFICATION	65
11.0	CALC_INNER_LOOP PROCEDURE SPECIFICATION	69
12.0	ASSESS_SYSTEM PROCEDURE SPECIFICATION	75
13.0	GIVE_WARNING PROCEDURE SPECIFICATION	79
14.0	TEST HARNESS SET-UP	83
14.1	Test Harness Operation	83
14.2	N-Version Voter Synchronization	96
14.3	Closed-Loop Simulation	96
14.4	DFCS Software Development	96

TABLE OF CONTENTS (Cont'd)

Section	Topic	Page
14.5	Ada Compilation Dependencies	104
15.0	RESULTS AND CONCLUSIONS	105
15.1	N-Version Software Demonstration	105
15.2	Methodology Extensions	105
15.3	Test Harness Flexibility	107
15.4	Conclusions	108
References		R-1
APPENDIX A	Channel 3 Applications Sofware Listings	A-1

LIST OF ILLUSTRATIONS

Number	Title	Page
1	Project Task Flow Diagram	2
2	Overall Test Harness Organization	3
3	Approaches to Software Fault Tolerance	5
4	Potential Drawbacks of Software Fault Tolerance	6
5	System Block Diagram	10
6	Top-Level System Logic	11
7	System Signal Flow	12
8	System-Level Signal Summary	13
9	Computer Input/Output Organization	15
10	Computer Cross-Channel Signal Summary	17
11	Software Input Signal Flow	18
12	Overall DFCS Flight Program Organization	20
13	Top-Level DFCS Package Listings	
	(a) Package CHANNEL_RESOURCES	21
	(b) Package SYSTEM_RESOURCES	22
14	Multirate Foreground Executive Flow Diagram	23
15	Foreground Procedure Timing Diagram	24
16	DFCS Data Flow Diagram	26
17	Call/Usage Graph	27
18	DFCS Applications Package Listings	
	(a) Package CONTROL_LAWS	28
	(b) Package DFCS_LOGIC	29
	(c) Package DFCS_RESOURCES	33
	(d) Package N_VERSION_VOTERS	36
	(e) Package VOTING_PLANES	37
19	N-Version Voting Requirements	38

LIST OF ILLUSTRATIONS (Cont'd)

Number	Title	Page
20	Procedure RUN_FOREGROUND_1 Listing	38
21	Autoland Control Law Block Diagram	58
22	Inner Loop Control Law Block Diagram	70
23	Overall Test Program Call Graph	84
24	DFCS Program Call Graph (In Test Harness)	85
25	Test Harness Program Unit Listings	
	(a) Procedure RUN_TEST_EXEC	86
	(b) Procedure START_TESTING	86
	(c) Procedure APPLY_INPUTS	87
	(d) Package TEST_RESOURCES	88
	(e) Task Body TEST_EXEC	91
	(f) Procedure XCHK_SYNCH_1	94
	(g) Task Body CHNL_1_SYNCH	95
26	Ada Multitasking Communication	97
27	Closed-Loop Simulation Block Diagram	98
28	Procedure SIMULATE_FLIGHT Listing	99
29	Overall Compilation Dependencies	102
30	Project Critique	106

1.0 INTRODUCTION

A set of software program unit specifications was generated via the process depicted in Figure 1 for use in an exploratory investigation of software fault tolerance using the N-version programming approach. The resultantly developed software is representative of a scaled-down flight control system (see Section 2.0) with a critical pitch-axis fly-by-wire (FBW) function. Accordingly, a double fail-operational, quadruplex system architecture was postulated to furnish requisite system reliability. Each of the four DFCS channels, moreover, incorporated a different version of applications software as independently developed by a different programmer.

The overall DFCS software structure, or the multirate executive program and its called procedure interfaces, however, was essentially the same in each channel per Section 4.0. Each DFCS executive contains calls to N-version program units, which in turn usually include calls to voters for cross-checking the intermediate computations of all the channels. Central to the N-version demonstration, these program units were developed using the Ada programming language in accordance with a set of applications software module specifications, which are presented in Sections 5.0 through 13.0.

Each of the program units was constructed so that it could be run in a single channel test harness on a stand-alone basis for unit testing and de-bugging, or as part of the total program for integrated 4-version testing. The latter entails the voting of the four versions of the DFCS software running effectively in parallel on a single VAX 8650 for the N-version software demonstration and evaluation. Hence, a non-realime multitasking test executive program with suitable integral test drivers was devised (see Section 14.0) to enable convenient software integration and valid N-version evaluation testing.

Figure 2 summarizes the organization of the multitasking test harness, where Ada tasks are denoted by the parallelogram shaped boxes. Task TEST_EXEC performs or directs all of the automated test functions, such as input test data application and results processing. The software for each of the four DFCS channels runs within an associated DFCS_EXEC task, which are coordinated such that synchronization occurs at each software voting plane. If a channel output is outside of permissible limits, it is assigned the voter selected value so that the erroneous state is not propagated. Note that the DFCS_EXEC tasks replace the top-level flight software, which is not germane to the problem at hand, so that the four DFCS channels can run logically in parallel.

1.1 Software Fault Tolerance

Concern over the potential for generic or common-mode software faults in critical systems has prompted rather widespread interest within the aerospace industry in software fault tolerance. While the enabling technology appears to be in place, it remains to demonstrate and assess all aspects of fault-tolerant software for critical DFCS applications. Various attributes of DFCS software, moreover, present some challenging demands. Temporal constraints, such as difference equation iteration rates or maximum fault detection/recovery times, are of particular concern.

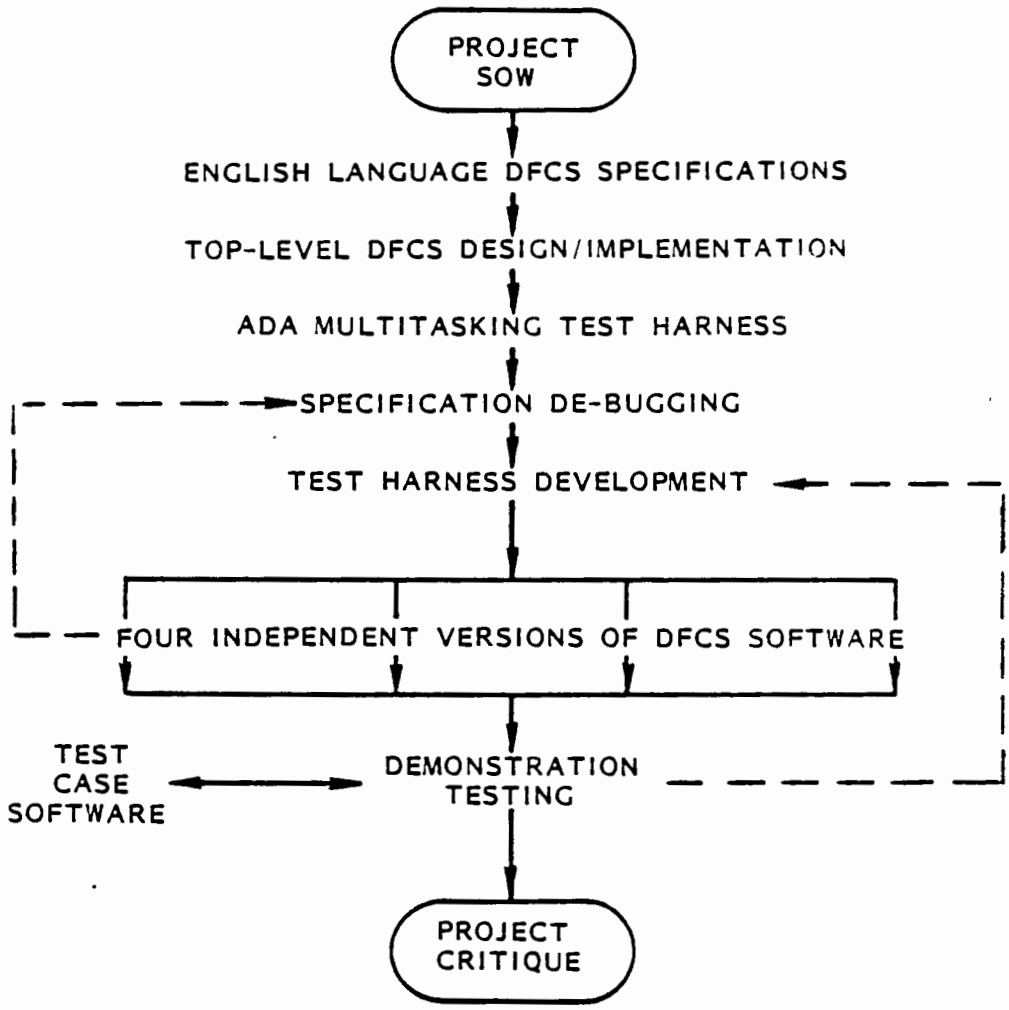


Figure 1 - Project Task Flow Diagram

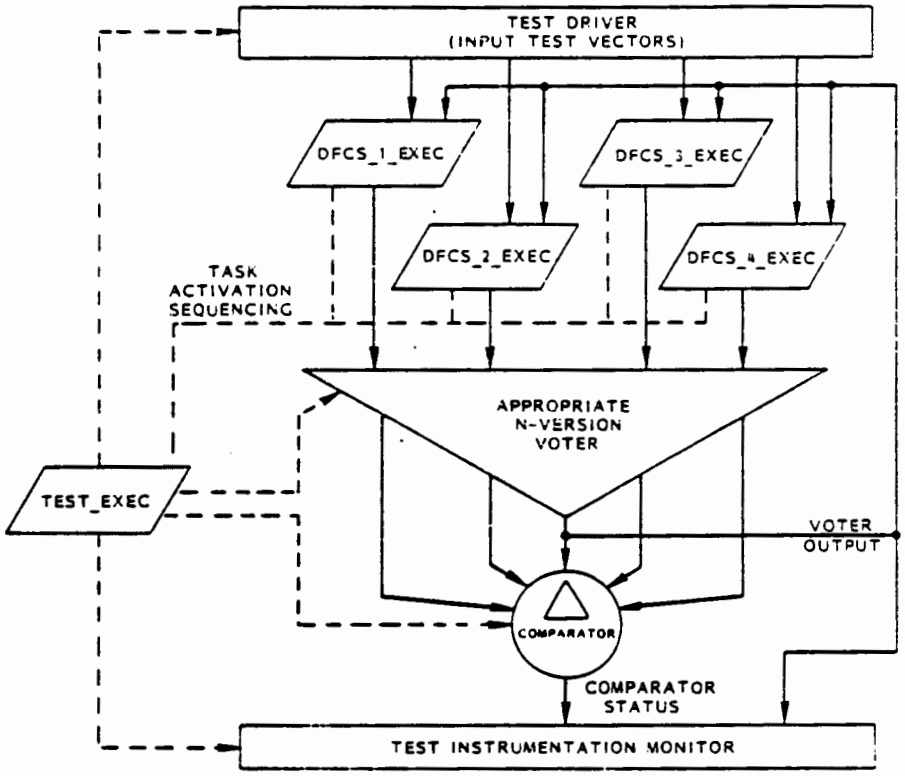


Figure 2 - Overall Test Harness Organization

The two primary approaches to fault tolerance, N-version software and recovery blocks (Ref. 1), are depicted in Figure 3. Both involve dissimilar versions of software performing the same function(s). In the case of N-version software, the versions must be developed independently. They run logically in parallel, and the version outputs are submitted to a voter/comparator for selection of the proper result. Recovery block alternates run logically in a sequence, which needs to be invoked only to the extent that alternate versions fail their acceptance test. Normally, some level of degradation in performance is accepted with among successive Alternates to ensure continuing operation.

Of these two approaches, N-version holds strong appeal for most types of DFCS software. The aforementioned time constraints are a dominant factor in such a preference. Hence, the DFCS application program modules under this investigation, were implemented using the N-version method . As suggested in Figure 3, the voter/comparator is a potential single point of failure in the N-version approach. As a consequence, dissimilar voters are sometimes used to obviate this prospect, but the compounding of complexity is appreciable, so only single voters were used in this investigation.

As with all software fault tolerance development efforts, strong emphasis was placed on establishing definitive, high quality software specifications (e.g., see Ref.2). Completeness, accuracy, and lack of ambiguity are in general essential to the realization of fault-tolerant software, so the prospects for demonstrating and evaluating N-version software are crucially dependent on the software specifications. For example, aspects such as maximum time allowances for voted code segments, as well as specific modes and responses for voting, must be completely and precisely stipulated.

Despite all initial efforts, some deficiencies existed in the specifications. Their rectification was rather time-consuming, but the variety of questions raised by different programmers did force corrections to the specifications that might otherwise might not have been so thorough. Simlarly, software de-bugging was facilitated by the the N-version approach. Overall, software fault tolerance has some drawbacks, inherent or potential, as summarized in Figure 4. Still, the net benefits appear worth pursuing.

1.2 Demonstration Guidelines

The DFCS demonstration software was coded using DEC's Ada compiler for the VAX VMS operating system at the Lockheed-Georgia Company. The DFCS software and the descriptions in this memorandum are intended to be essentially in accord with DO-178A, i.e., the documentation is to be illustrative of compliance without necessarily being exhaustive. Configuration control, error logging, and delineation of software development phases are to be observed in an orderly manner that supports and enhances the value of the results of the investigation.

The following assumptions were adopted at the outset to expedite but in no way compromise, the conduct of the investigation:

- No Flight Control Computer Operating System
- No Bus Management Software Functions
- No Hardware-Related Instructions

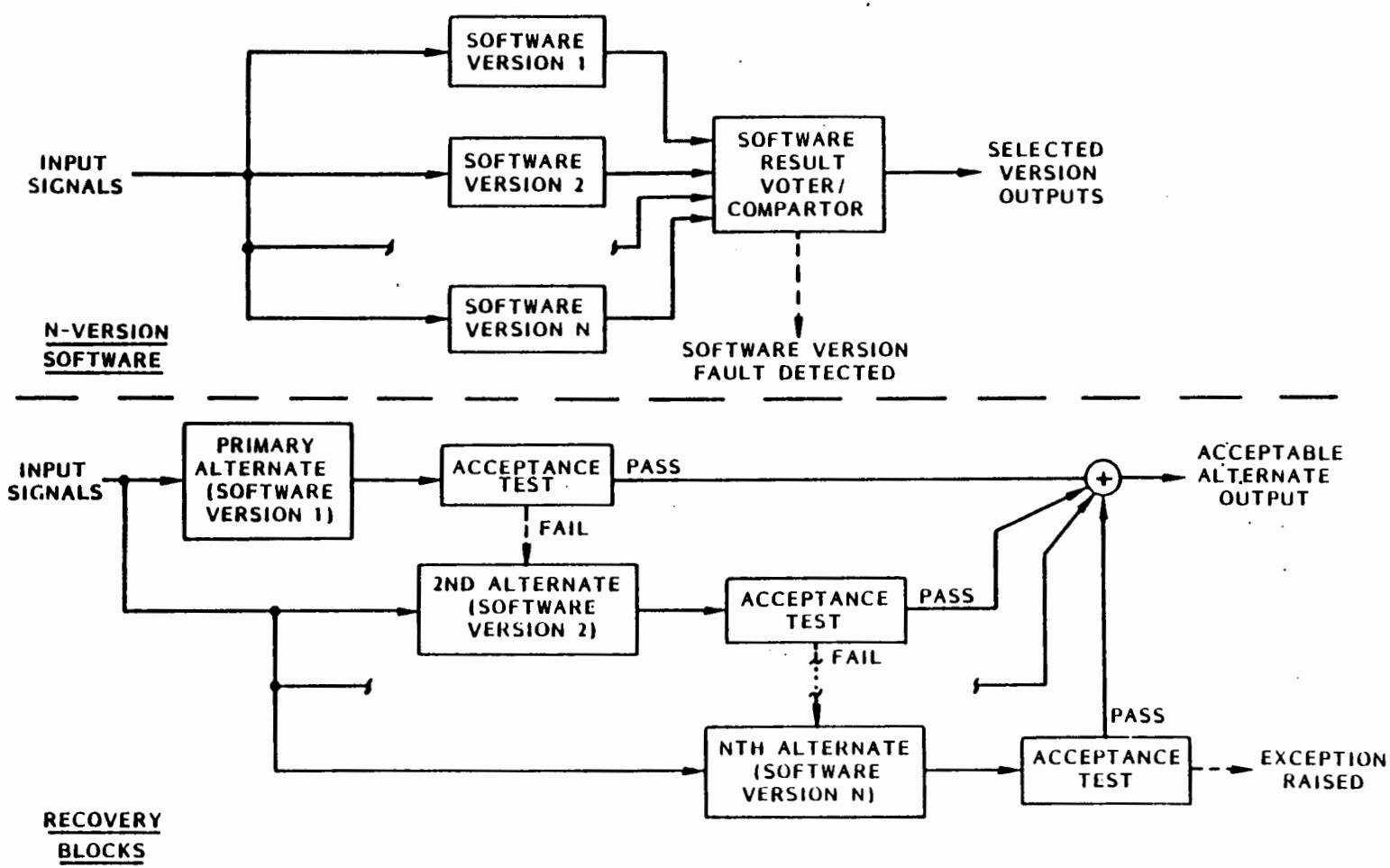


Figure 3 - Approaches to Software Fault Tolerance

PROBLEM	GENERAL APPROACH	TECHNICAL APPROACH	RESULTS
Software Fault Propensity	Software Fault Tolerance	N-Version Software	Implementation Demonstration
Software Fault Tolerance Drawbacks	Tailored Development Strategy	Program Structuring Techniques	Methodology Extensions
N-Version Demonstration Mechanism	Uniprocessor Test Harness	Ada Multitasking	Flexible and Versatile Test Harness

Figure 4 - Potential Drawbacks of Software Fault Tolerance

- o Limited Pitch Axis Functions Only
- o No Fixed Point Arithmentic, and Hence No Variable Scaling.

After program unit testing and de-bugging, several stages of testing were conducted: integration and demonstration testing. Due to specification discrepancies detected during coding, the specification parts of this document were revised before consistency among versions was established. Most of these discrepancies were incompletely or incorrectly specified logic.

1.3 Ada Programming Language

There are strong indications that the Ada programming language (Ref. 3) will experience widespread usage in civil aviation in the near term. This would be based primarily on the merits of the language itself, rather than on the U.S. Department of Defense's influence. Despite its drawbacks, the Ada programming language has no viable competition now for use in digital flight system applications. Of course, the language is still developmental with regard to support of flightworthy computers, but the associated problems should be correctable with adequate funding from military programs. Two particularly significant problems associated with Ada are the overhead of tasking and machine language code insertions. Neither factor was applicable to the problem addressed here. Tasking was used for simulation purposes, but not for DFCS software per se.

Here the use of Ada facilitated the conduct of the N-version software demonstration by enabling explicit definition of program units specification parts, precise definition of their software interfaces, the construction of the multitasking test harness, and non-interference observation of test results through Ada package importation by test units.

Specification benefits naturally derive from the two-part composition of Ada program units, which involve a specification part and a corresponding body part. The intent here is to use Ada packages and procedure specification parts to define the fault-tolerant software modules. Each specification part defines a particular interface and its available services, and is refective of and consistent with the overall design of the program. Hence, this document includes Ada specification parts as the precise, lower-level portions of the respective module specifications. The N-version programmers used them to implement the DFCS functions and services in the associated body parts in the form of executable Ada code.

Although the imposition of a well-defined program design tends to eliminate many types of software faults, those that might remain would seem likely to be more restricted to those types that are detectable by the N-version software voters. This would obviously be desirable from both experimental and architectural standpoints. Note that the Ada specification parts are only one component of the module specifications; English text and analytical diagrams, for example, were used as well. Follow-on activities will investigate the use of comments expressed in the Anna (annotated Ada) specification langauge. Logic specification checks for completeness and test case generation will also be pursued.

FAA WJH Technical Center
00092838

1.4 Anna Specification Language

In general, formal specification has been identified as the key to rationalizing the software development process (Ref. 4). In the case of fault-tolerant software, moreover, formal specification would seem necessary to eliminate a class of faults that cannot be tolerated, namely software faults originating in specifications. By definition, such faults lie outside the safeguards of software fault tolerance, which it is charged with ensuring specification conformity during operation, under the assumption that the specification is correct. Thisproperty can be affirmed to some extent by the verifiability inherent in formal specifications.

The Anna (annotated Ada) specification language (Ref. 5) appears to be a significant advance in specification technology for practical systems. Despite its as yet developmental status, Anna is considered mature and promising enough to merit a limited trial application. This seems feasible because: Anna statements are of the form of actual Ada comments, so they are ignored by an Ada compiler; in many cases they resemble Ada source code, so they are comparatively readable; and above all Anna specifications need not be complete, so they can be used to the extent desired for any particular program unit.

Although the processing of Anna statements normally involves associated, but currently unavailable, support software for automated consistency checks, the addition of semantic definition to Ada specifications alone is expected to yield more than ample return for the effort expended. In particular, Anna holds promise of providing the high quality specifications that are so are vital to fault-tolerant software.

Eventually, it should be possible to obtain the Anna support software, and it would doubtlessly prove informative to evaluate its static consistency checking as well as to apply its dynamic run-time checks during simulation testing. Exceptions raised by the run-time checks might well prove useful in the conduct or analysis of testing. From a fault avoidance standpoint, both of these types of checks should improve software quality in general, and from a fault tolerance standpoint, the dynamic checks might serve as acceptance tests in recovery block mechanizations.

2.0 SYSTEM DESIGN

As a framework and context for the software program unit specifications, a DFCS design was systematically developed that illustrates the precision and accountability appropriate for critical functions. Here only certain pitch-axis functions were levied as requirements in order to suitably bound the scope of the software development effort. Accordingly, the following system functions were included:

- Augmented Fly-by-Wire (AFBW) for a Negative Static Margin Transport - double fail-operational redundancy
- Autoland (Glideslope and Flare Modes) - single fail-operational redundancy
- Vertical Navigation and Altitude Hold (Growth Provisions Only) - fail-passive.

Inclusion of growth provisions was based on a potential interfacing with a navigation estimation algorithm that Battelle developed under this same contractual task to explore recovery block software fault tolerance.

2.1 System Description

The above requirements, especially the AFBW function, would typically result in a quadruplex system architecture as depicted in Figure 5. The redundancy levels and interconnections shown are representative of current industry practice, based on the safety and reliability requirements associated with the above DFCS functions. Four parallel MIL-STD-1553B multiplex (MUX) buses are assumed for system interconnection, and the computer cross-channel buses are asynchronous broadcast buses like ARINC 429.

Top-level system logic requirements in terms of MIL-F-9490D Operational States (Ref. 6) are summarized in Figure 6. This logic, which reflects system safety based on the interaction between redundancy margins and airplane flying qualities status, is most appropriate for an AFBW function. This system state logic, whose definition is expanded and applied in subsequent sections on fault and annunciator logic, was ultimately be implemented in N-version software modules.

The system-level signal flow for a single channel, which is typical of all channels except for the routing of dual or triplex signals, is given in Figure 7. Distribution of these signals is clarified in Figure 4 or in the software interface definitions. The individual system-level signals are characterized in Figure 8.

Each flight computer is postulated to be identical, with an input/output processor (IOP) for transferring and formatting external signals and a central processor unit (CPU) for flight software computation. As depicted in Figure 9, the two processors operate autonomously and share two sections of memory. Only the IOP can write the memory addresses assigned to input variables, and the CPU can only read them. Similarly, only the CPU can write the output addresses, and the IOP can only read them. The input data refresh rate is assumed to be high enough such that associated data skew or phase lag are not serious concerns.

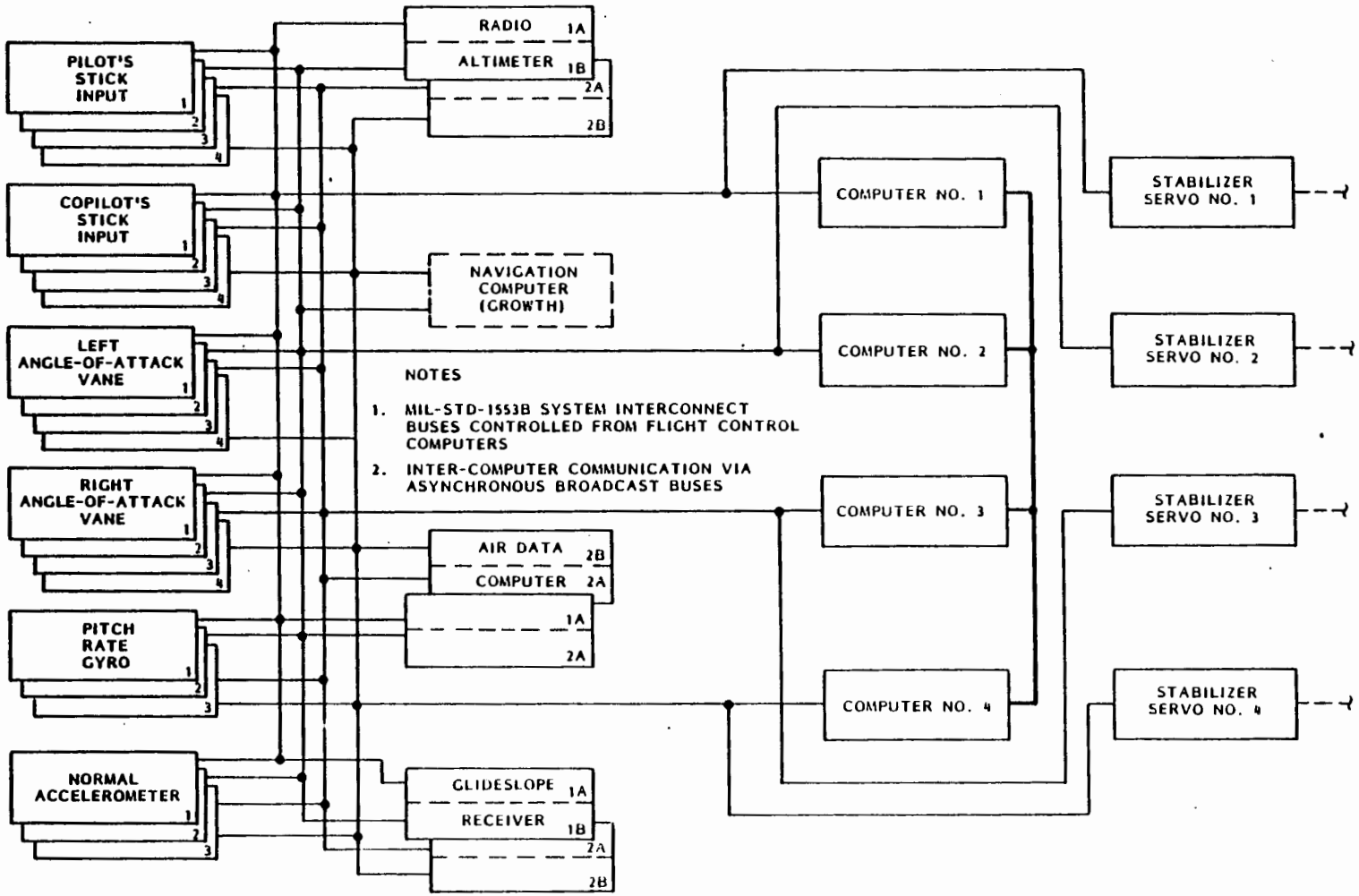


Figure 5 - System Block Diagram

IDENTIFIER	DEFINITION	SIGNIFICANCE
OPERATIONAL_STATE_1	System Operational State I	Full Capability
OPERATIONAL_STATE_2	" " " II	Limited "
OPERATIONAL_STATE_3	" " " III	Marginal "
OPERATIONAL_STATE_4	" " " IV or V	Unsafe "
NORMAL_FLYING_QLTY	Normal Flying Qualities	Level 1
DEGRADED_FLYING_QLTY	Degraded " "	" 2
MARGINAL_FLYING_QLTY	Marginal " "	" 3
UNFLYABLE	Unflyable " "	Less than Level 3
DOUBLE_FAIL_OP	Double Fail Operational	4 Indep. Paths
SINGLE_FAIL_OP	Single " "	3 " "
FAIL_UNSAFE	Fail Unsafe	2 "
DEPLETED	Inadequate Resources	0 or 1 "

```
OPERATIONAL_STATE_1  =  DOUBLE_FAIL_OP * NORMAL_FLYING_QLTY

OPERATIONAL_STATE_2  =  DOUBLE_FAIL_OP * DEGRADED_FLYING_QLTY +
        SINGLE_FAIL_OP * ( NORMAL_FLYING_QLTY + DEGRADED_FLYING_QLTY )

OPERATIONAL_STATE_3  =  MARGINAL_FLYING_QLTY * not UNSAFE +
        FAIL_UNSAFE * not UNFLYABLE

OPERATIONAL_STATE_4  =  DEPLETED + UNFLYABLE
```

```
DOUBLE_FAIL_OP  = MIN_3_SERVOS_ENGAGED * COMPUTERS_VALID *
                  STICKS_VALID * AOA_PAIRS_VALID

SINGLE_FAIL_OP  = MIN_2_SERVOS_ENGAGED * MIN_3_COMPUTERS_VAL *
                  NOM_4A_STICKS_VAL * MIN_3_AOA_PRS_VAL *
                  ( not MIN_3_SERVOS_ENGAGED + not COMPUTERS_VALID +
                  not STICKS_VALID + not AOA_PAIRS_VALID )

FAIL_UNSAFE  =  1_SERVO_ENGAGED + XCT_2_COMPUTERS_VAL +
                NOM_2A__STICKS_VAL + XCT_2_AOA_PRS_VAL

UNSAFE  =  MAX_1_COMPUTER_VAL + MAX_1A_BUS_VAL + MAX_2A_STICKS_VAL +
           MAX_1_AOA_PR_VAL
```

```
NORMAL_FLYING_QLTY  =  MIN_2_PRGYROS_VAL * MIN_2_AOA_PRS_VAL

DEGRADED_FLYING_QLTY  =  MAX_1_PRGYRO_VAL * MIN_2_AOA_PRS_VAL

MARGINAL_FLYING_QLTY  =  MIN_2_PRGYROS_VAL * ZRO_AOA_PR_VAL

UNFLYABLE  =  MAX_1_PRGYRO_VAL + ZRO_AOA_PR_VAL
```

Figure 6 - Top-Level System Logic

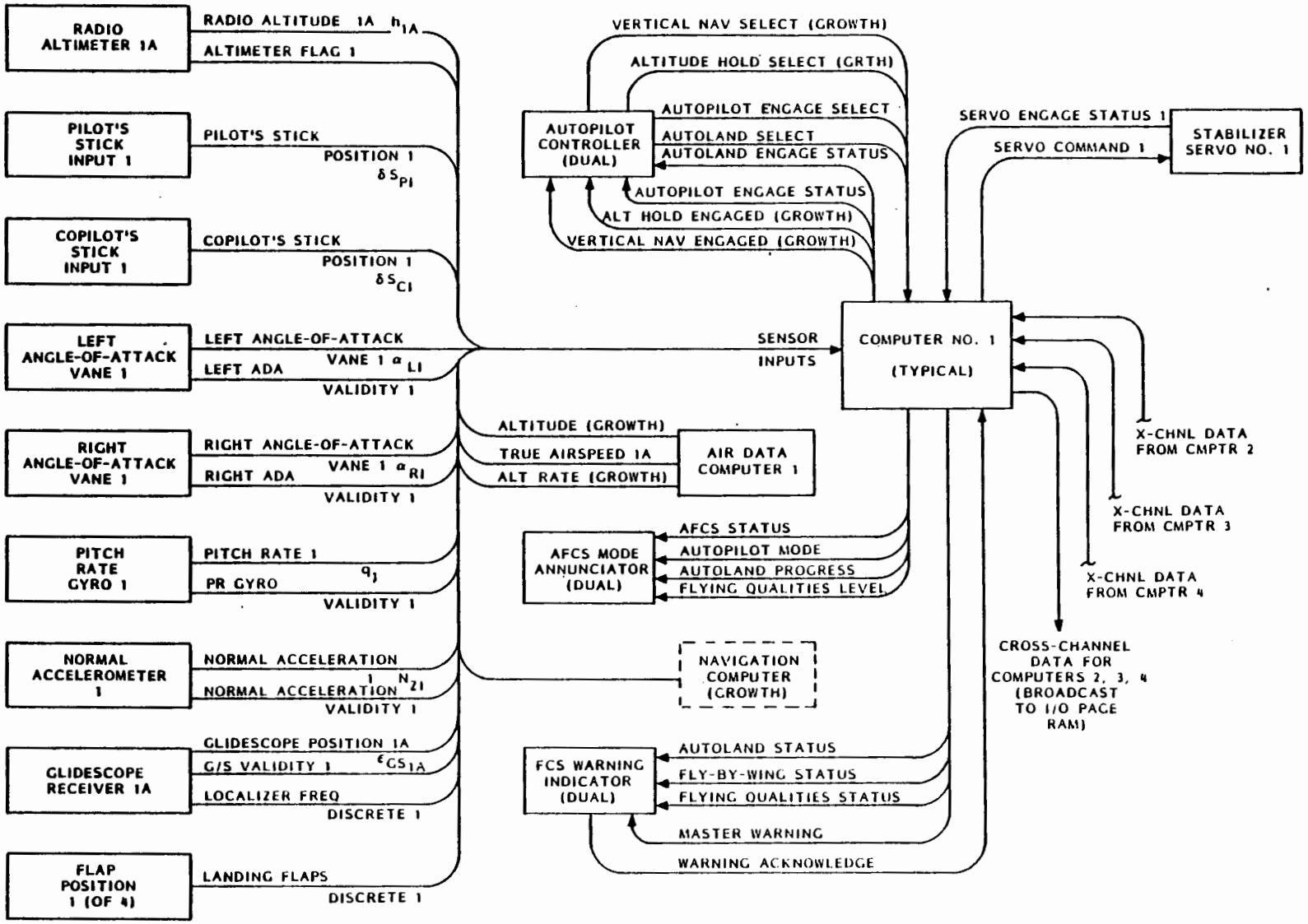


Figure 7 - System Signal Flow Diagram (Typical Channel)

IDENTITY	TYPE	RANGE	IDENTIFIER
Pilot's Stick Command No.1	Float	0.5,-1.5, deg	P_STICK_CMD(1)
Pilot's Stick Command No.2	"	"	P_STICK_CMD(2)
Pilot's Stick Command No.3	"	"	P_STICK_CMD(3)
Pilot's Stick Command No.4	"	"	P_STICK_CMD(4)
Copilot's Stick Command No.1	"	"	CP_STICK_CMD(1)
Copilot's Stick Command No.2	"	"	CP_STICK_CMD(2)
Copilot's Stick Command No.3	"	"	CP_STICK_CMD(3)
Copilot's Stick Command No.4	"	"	CP_STICK_CMD(4)
Left Angle-of-Attack No.1	"	+50,-10 deg	LEFT_AOA(1)
Left Angle-of-Attack No.2	"	"	LEFT_AOA(2)
Left Angle-of-Attack No.3	"	"	LEFT_AOA(3)
Left Angle-of-Attack No.4	"	"	LEFT_AOA(4)
Right Angle-of-Attack No.1	"	"	RIGHT_AOA(1)
Right Angle-of-Attack No.2	"	"	RIGHT_AOA(2)
Right Angle-of-Attack No.3	"	"	RIGHT_AOA(3)
Right Angle-of-Attack No.4	"	"	RIGHT_AOA(4)
Pitch Rate No.1	"	+/- 30 deg/sec	P_RATE_GYRO(1)
Pitch Rate No.2	"	"	P_RATE_GYRO(2)
Pitch Rate No.3	"	"	P_RATE_GYRO(3)
Normal Acceleration No.1	"	+3 G, -1 G	NORM_ACCEL(1)
Normal Acceleration No.2	"	"	NORM_ACCEL(2)
Normal Acceleration No.3	"	"	NORM_ACCEL(3)
True Airspeed No.1A	"	100-1000 fps	TRUE_AIRSPEED(1)
True Airspeed No.1B	"	"	TRUE_AIRSPEED(2)
True Airspeed No.2A	"	"	TRUE_AIRSPEED(3)
True Airspeed No.2B	"	"	TRUE_AIRSPEED(4)
Radio Altitude No.1A	"	-20, +2500 ft	RAD_ALTITUDE(1)
Radio Altitude No.1B	"	"	RAD_ALTITUDE(2)
Radio Altitude No.2A	"	"	RAD_ALTITUDE(3)
Radio Altitude No.2B	"	"	RAD_ALTITUDE(4)
Glideslope Deviation No.1A	"	+/- 1.4 deg	GS_BEAM_DEV(1)
Glideslope Deviation No.1B	"	"	GS_BEAM_DEV(2)
Glideslope Deviation No.2A	"	"	GS_BEAM_DEV(3)
Glideslope Deviation No.2B	"	"	GS_BEAM_DEV(4)
Pilot's Stick Validity No.1	Boolean	1 => Valid	P_STK_VAL(1)
Pilot's Stick Validity No.2	"	"	P_STK_VAL(2)
Pilot's Stick Validity No.3	"	"	P_STK_VAL(3)
Pilot's Stick Validity No.4	"	"	P_STK_VAL(4)
Copilot's Stick Validity No.1	"	"	CP_STK_VAL(1)
Copilot's Stick Validity No.2	"	"	CP_STK_VAL(2)
Copilot's Stick Validity No.3	"	"	CP_STK_VAL(3)
Copilot's Stick Validity No.4	"	"	CP_STK_VAL(4)
Left Angle-of-Attack Validity No.1	"	"	L_AOA_VAL(1)
Left Angle-of-Attack Validity No.2	"	"	L_AOA_VAL(2)
Left Angle-of-Attack Validity No.3	"	"	L_AOA_VAL(3)
Left Angle-of-Attack Validity No.4	"	"	L_AOA_VAL(4)
Rt Angle-of-Attack Validity No.1	"	"	R_AOA_VAL(1)
Rt Angle-of-Attack Validity No.2	"	"	R_AOA_VAL(2)
Rt Angle-of-Attack Validity No.3	"	"	R_AOA_VAL(3)
Rt Angle-of-Attack Validity No.4	"	"	R_AOA_VAL(4)

Figure 8 - System-Level Signal Summary (1 of 2)

IDENTITY	TYPE	RANGE	IDENTIFIER
Pitch Rate Gyro Validity No.1	Boolean	1 => valid	P_RATE_VAL(1)
Pitch Rate Gyro Validity No.2	"	"	P_RATE_VAL(2)
Pitch Rate Gyro Validity No.3	"	"	P_RATE_VAL(3)
Normal Accelorometer Validity No.1	"	"	N_ACCEL_VAL(1)
Normal Accelorometer Validity No.2	"	"	N_ACCEL_VAL(2)
Normal Accelorometer Validity No.3	"	"	N_ACCEL_VAL(3)
True Airspeed Validity No.1A	"	"	TAS_VAL(1)
True Airspeed Validity No.1B	"	"	TAS_VAL(2)
True Airspeed Validity No.2A	"	"	TAS_VAL(3)
True Airspeed Validity No.2B	"	"	TAS_VAL(4)
Radio Altimeter Validity No. 1A	"	"	RAD_ALT_VAL(1)
Radio Altimeter Validity No. 1B	"	"	RAD_ALT_VAL(2)
Radio Altimeter Validity No. 2A	"	"	RAD_ALT_VAL(3)
Radio Altimeter Validity No. 2B	"	"	RAD_ALT_VAL(4)
Glideslope Validity No.1A	"	"	GS_BEAM_VAL(1)
Glideslope Validity No.1B	"	"	GS_BEAM_VAL(2)
Glideslope Validity No.2A	"	"	GS_BEAM_VAL(3)
Glideslope Validity No.2B	"	"	GS_BEAM_VAL(4)
Autopilot Selection A & B	Enumerated	5 modes	MODE_SEL
Autoland Category A & B	"	3 categrorles	"
Autopilot Engagement A	"	5 modes/3 categories	ANNUN_V1, ANNUN_V2
Autopilot Engagement B	"	"	ANNUN_V3, ANNUN_V4
Autoland Progress A	"	6 states	ANNUN_V1, ANNUN_V2
Autoland Progress B	"	"	ANNUN_V3, ANNUN_V4
Flying Qualities Level A	"	3 levels	ANNUN_V1, ANNUN_V2
Flying Qualities Level B	"	"	ANNUN_V3, ANNUN_V4
Autoland Status A	"	3 states	#ARN_V1,#ARN_V2
Autoland Status B	"	"	#ARN_V3,#ARN_V4
Fly-by-Wire Status A	"	"	#ARN_V1,#ARN_V2
Fly-by-Wire Status B	"	"	#ARN_V3,#ARN_V4
Flying Qualities Status A	"	2 states	#ARN_V1,#ARN_V2
Flying Qualities Status B	"	"	#ARN_V3,#ARN_V4
Master Warning A	"	3 states	FLASH_#ARN_V1, FLASH_#ARN_V2
Master Warning B	"	"	FLASH_#ARN_V3, FLASH_#ARN_V4
Acknowledge warning	boolean	1 => Acknowledge	ACKNOWLEDGE
Servo Engage Status No.1	Record	3 components	SERVO_1
Servo Engage Status No.2	"	"	SERVO_2
Servo Engage Status No.3	"	"	SERVO_3
Servo Engage Status No.4	"	"	SERVO_4
Servo Command No.1	Float	-11, +2 deg	STAB_SERVO_CMD_V1
Servo Command No.2	"	"	STAB_SERVO_CMD_V2
Servo Command No.3	"	"	STAB_SERVO_CMD_V3
Servo Command No.4	"	"	STAB_SERVO_CMD_V4

Figure 8 - System-Level Signal Summary (2 of 2)

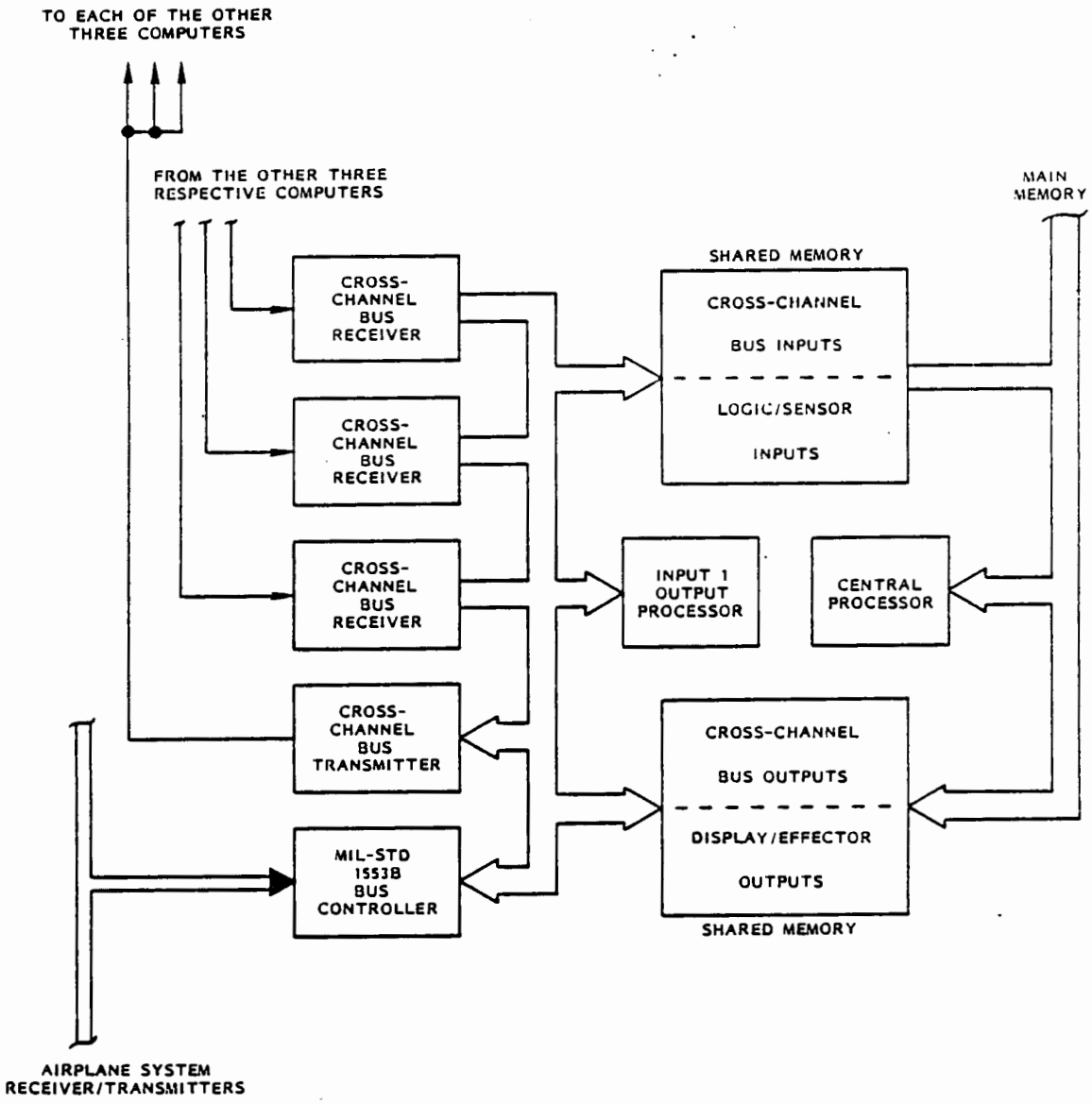


Figure 9 - Computer Input/Output Organization (Same for All Computers)

Figure 10 lists all of the DFCS computer cross-channel signals and summarizes their salient characteristics. Note that some logic input signals require a dedicated discrete input for a practical design, e.g., to provide responsiveness in the real-time coordination of resources. As far as the flight software is concerned, all input, output, or cross-channel signals could be made available as local data objects. However, for test observability or software voting, the level of visibility of these objects was raised. Figure 11 shows the interaction between the IOP/CPU shared memory and the input signal that must be accomplished by the flight software. The latter is specified in Sections 8.0 and 10.0 as part of the N-version test article.

IDENTITY	TYPE	RANGE	IDENTIFIER
Left Angle-of-Attack No.1	Float	+50,-10 deg	LEFT_AOA(1)
Left Angle-of-Attack No.2	"	"	LEFT_AOA(2)
Left Angle-of-Attack No.3	"	"	LEFT_AOA(3)
Left Angle-of-Attack No.4	"	"	LEFT_AOA(4)
Right Angle-of-Attack No.1	"	"	RIGHT_AOA(1)
Right Angle-of-Attack No.2	"	"	RIGHT_AOA(2)
Right Angle-of-Attack No.3	"	"	RIGHT_AOA(3)
Right Angle-of-Attack No.4	"	"	RIGHT_AOA(4)
Channel Status No.1	Boolean	1 => valid	CHNL_STATUS_V1
Channel Status No.2	"	"	CHNL_STATUS_V2
Channel Status No.3	"	"	CHNL_STATUS_V3
Channel Status No.4	"	"	CHNL_STATUS_V4

NOTE: All sensor inputs and their validity signals are cross-bused by the I/O Processor. The logic signals are the result of software computation.

Figure 10 - Computer Cross-Channel Signal Summary

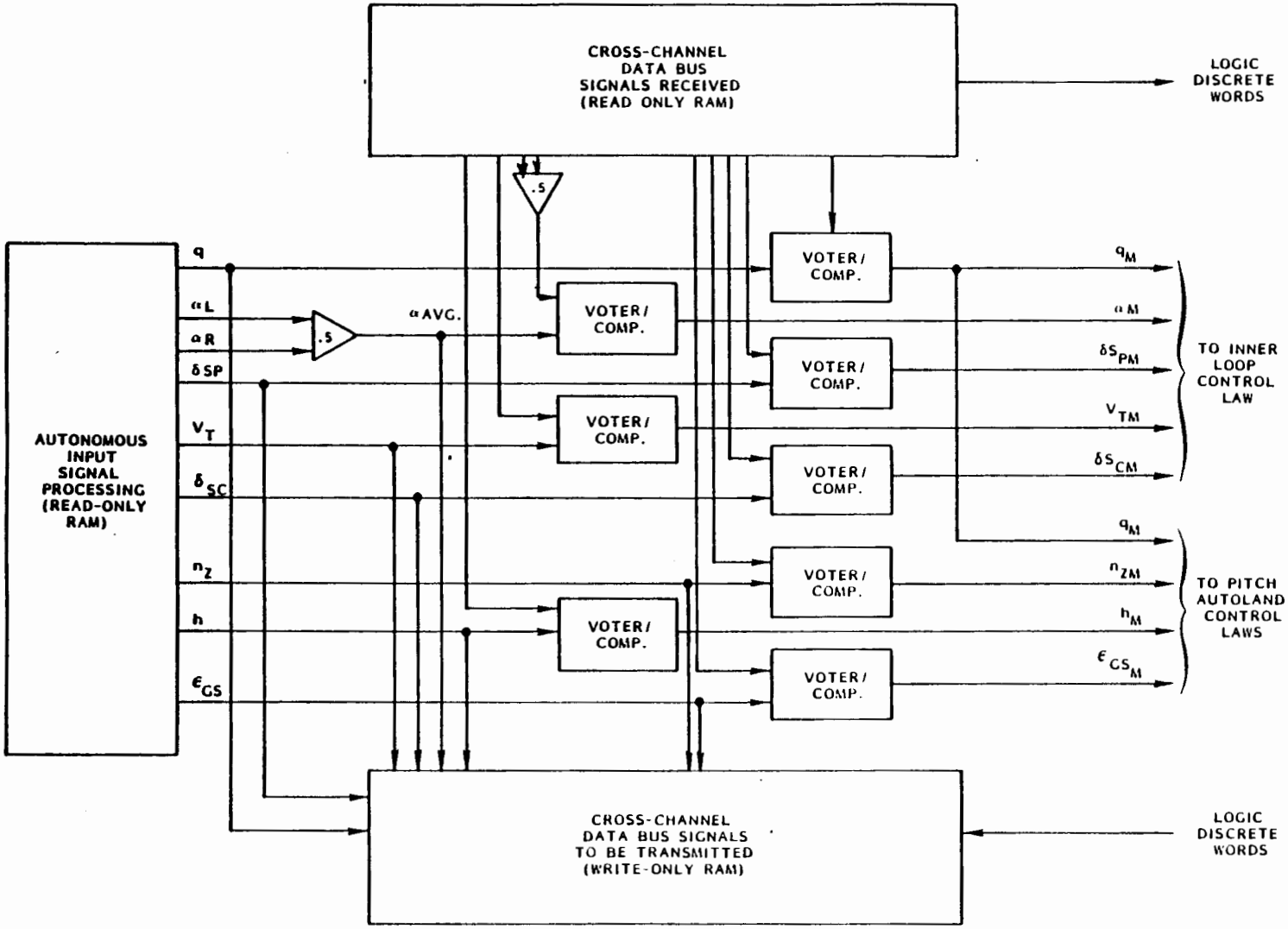


Figure 11 - Software Input Signal Flow

3.0 SOFTWARE DESIGN

Two aspects of software design were considered: the actual DFCS flight software for the four computational channels; and the test executive to manage N-version software execution and assessment on a single VAX 8650 host machine. This section develops the DFCS software design, while the test executive and the overall demonstration software organization are presented in Section 14.0.

For the test article, note that the lower-level DFCS software in each channel runs under an autonomous Ada task, DFCS_x_EXEC, in Figure 2. Here "x" denotes the DFCS channel number. All four of these tasks are active, although perhaps blocked, throughout testing. The DFCS_x_EXEC tasks serve to enable the non-real-time testing of parallel channels in a sequential, yet acceptable, manner involving coordinated task blocking at the cross-check points. The basic point, however, is that the same DFCS software can run in either parallel DFCS processors or a single test computer in effectively the same way.

The overall organization of each DFCS channel's flight software is depicted in Figure 12. Note that the two top-level procedures are not included in the DFCS test article, but partial contents of the top-level DFCS packages, CHANNEL_RESOURCES and SYSTEM_RESOURCES, are still used in the test set-up. The associated Ada source code listings are given in Figure 13, Parts a and b.

3.1 DFCS Software Design

The overall design of the software was intended to closely follow the prior design of a quadruplex DFCS that was implemented and demonstrated on the RDFCS (Reconfigurable DFCS) Simulator Facility at NASA Ames under the same contract as this task (Ref. 7). It is expected that the similarities will serve to strengthen the tutorial value of the contract reports by viewing essentially the same system from different perspectives.

The top-level DFCS software design is the same for each channel. With reference to Figure 12, the main program in each channel, RUN_DFCS_MAIN_x, is taken to be an austere operating system that establishes a given channel's readiness for operation upon start-up or following a temporary disruption. Normally then, the second-level procedure, RUN_DFCS_EXEC_x, directs ongoing system management during normal operation, e.g., major frame channel synchronization. In a complete flight software load module, it also calls Procedure RUN_FOREGROUND_x, which is included in the the N-version test article.

Figure 14 illustrates the looping, multirate structure of RUN_FOREGROUND_x, which in the test set-up is called by Task DFCS_x_EXEC of the test harness (which replaces Procedure RUN_DFCS_EXEC_x for demonstration purposes). After each top-to-bottom traversal of the flow diagram, control is re-assumed by DFCS_x_EXEC for a simulated elapsed time of 50 millisecond per computational cycle as defined in Figure 15). When appropriate, Procedure RUN_FOREGROUND_x is called again and the next one of the four path traversals is effected. Note in Figure 14 that the N-version cross-check points are shown following each of the affected applications procedures; at such points, the four

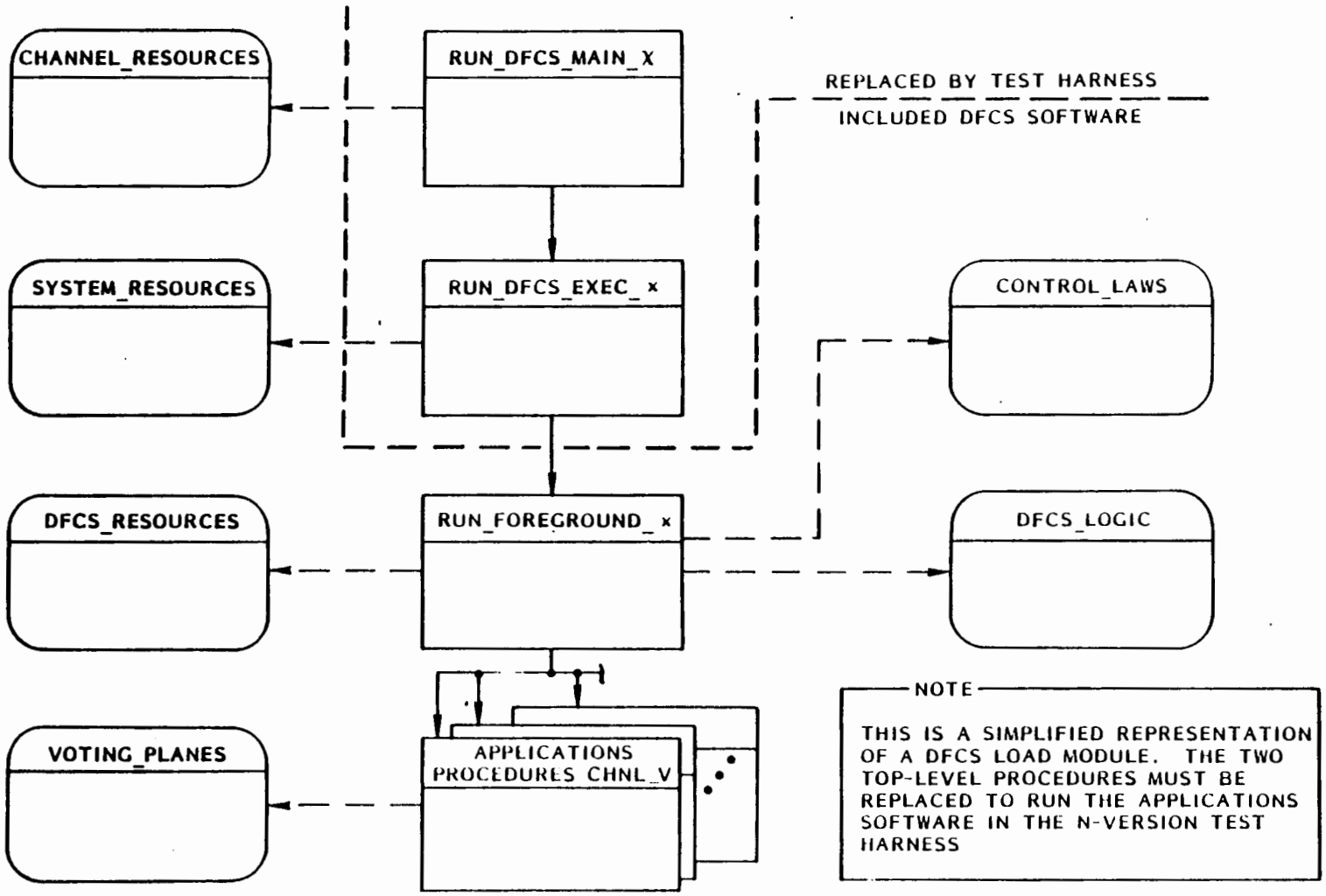


Figure 12 - Overall DFCS Flight Program Organization

```
package CHANNEL_RESOURCES is

--  Type Declarations
    -----------------

    --  Computer Channel Logic:

        type CMPTR_CHNL_STATUS is
            record
                CPU_CHK_OK        :        BOOLEAN ;
                IOP_BUS_OK        :        BOOLEAN ;
                MUX_BUS_OK        :        BOOLEAN ;
            end record ;

    --  Servo Status Logic:

        type SERVO_STATUS is
            record
                ACTUATOR_OK       :        BOOLEAN ;
                LVDT_VALID        :        BOOLEAN ;
                PLACR_AVAIL       :        BOOLEAN ;
            end record ;


--  Object Declarations
    -------------------

    --  Channel Status:

        CHNL_STATUS_V1, CHNL_STATUS_V2,
        CHNL_STATUS_V3, CHNL_STATUS_V4          :  BOOLEAN ;

        CMPTR_1, CMPTR_2, CMPTR_3, CMPTR_4      :  CMPTR_CHNL_STATUS ;

        SERVO_1, SERVO_2, SERVO_3, SERVO_4      :  SERVO_STATUS ;

end CHANNEL_RESOURCES ;
```

Figure 13a - Top-Level DFCS Package Listing
Package CHANNEL_RESOURCES

```
package SYSTEM_RESOURCES is

--  Procedure Declarations
    ----------------------

        procedure RUN_FOREGROUND_1 ;
        procedure RUN_FOREGROUND_2 ;
        procedure RUN_FOREGROUND_3 ;
        procedure RUN_FOREGROUND_4 ;

--  Type Declaration
    ----------------

        subtype PATH_RANGE       is INTEGER range 0..4 ;

--  Object Declarations
    -------------------

        PATH_1, PATH_2, PATH_3, PATH_4   : PATH_RANGE ;

end SYSTEM_RESOURCES ;

package body SYSTEM_RESOURCES is

        procedure RUN_FOREGROUND_1 is separate ;
        procedure RUN_FOREGROUND_2 is separate ;
        procedure RUN_FOREGROUND_3 is separate ;
        procedure RUN_FOREGROUND_4 is separate ;

end SYSTEM_RESOURCES ;
```

Figure 13b - Top-Level DFCS Package Listing
Package SYSTEM_RESOURCES

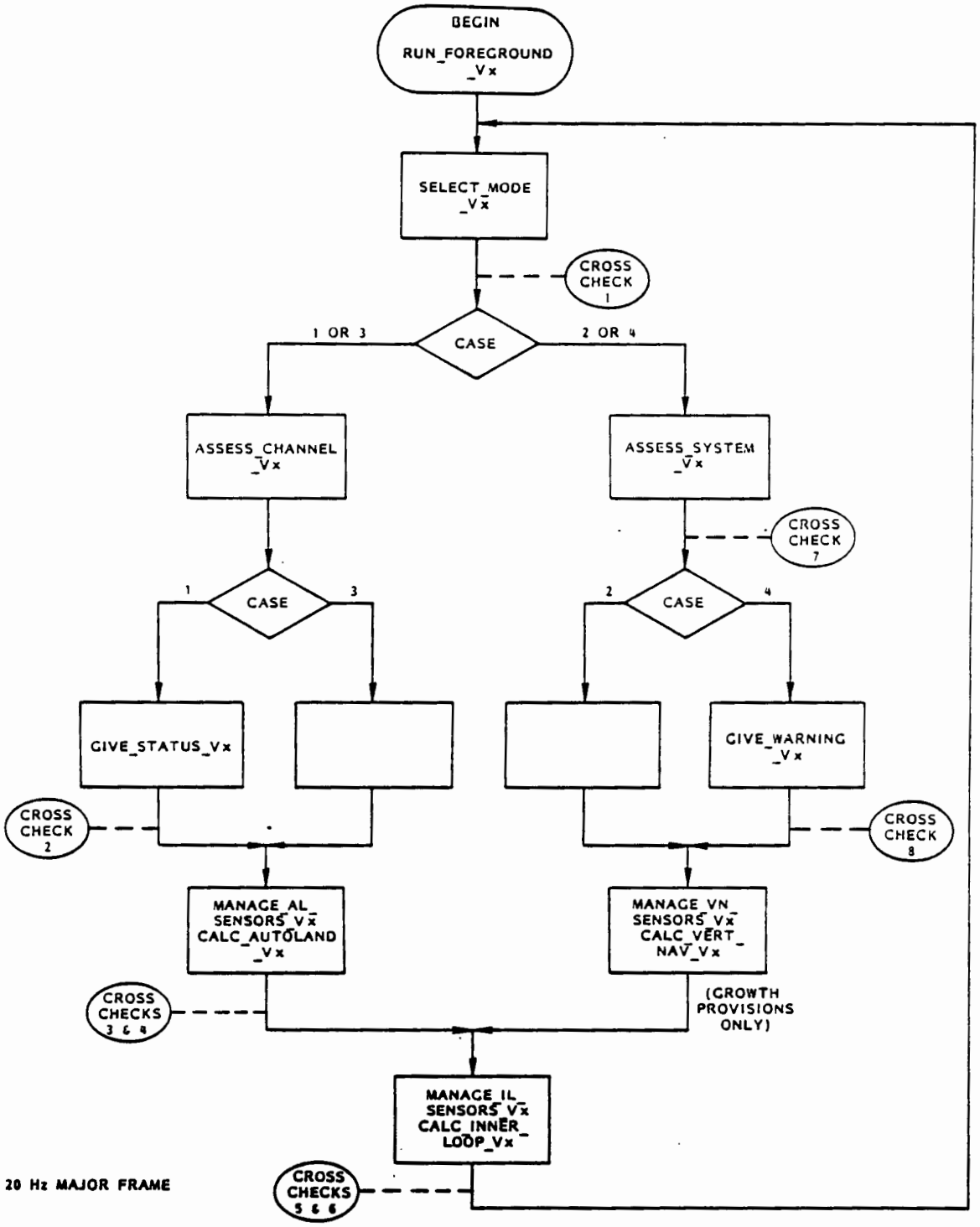


Figure 14 – Multirate Foreground Executive Flow Diagram

TIME SLICE NO.	SUB-FRAME TIME	PATH NUMBER				SUB FRAME TIME
		1	3	2	4	
T_1	≤ 2	SELECT_MODE_Vx				—
T_2	≤ 3	XCHK_SYNCH_x				—
T_3	≤ 2	ASSESS_CHANNEL_Vx		ASSESS_SYSTEM_Vx		≤ 3
T_4	= 0			XCHK_SYNCH_x		≤ 3
T_5	≤ 2	GIVE STATUS_x			GIVE WARNING_x	≤ 2
T_6	≤ 3	XCHK_SYNCH_x			XCHK_SYNCH_x	≤ 3
T_7	≤ 5	MANAGE_AL_SENSORS_Vx				= 0
T_8	≤ 3	XCHK_SYNCH_x				= 0
T_9	≤ 4	CALC_AUTOLAND_Vx				= 0
T_{10}	≤ 3	XCHK_SYNCH_Vx				= 0
T_{11}	≤ 5	MANAGE_IL_SENSORS_Vx				—
T_{12}	≤ 3	XCHK_SYNCH_Vx				—
T_{13}	≤ 6	CALC_INNER_LOOP_Vx				—
T_{14}	≤ 3	XCHK_SYNCH_Vx				—
T_{TOT}	≤	44	39	28	33	—

Figure 15 - Foreground Procedure Timing Diagram

channels must synchronize, exchange data, and vote before executing to the next applications module.

3.2 DFCS Applications Software

For years the authors have used a software design strategy that is based on control state decomposition (see Ref. 8), and this sufficed for the development of FAA's quadruplex DFCS at NASA Ames. The implementation language used there (AED), however, did not permit the extensive protection of data objects that Ada fosters through packages and strong typing. Hence, the the concern for minimization of the data objects' namespace over the total program could not be addressed systematically until the present implementation in Ada. The intent, of course, is to alleviate the vulnerability of data objects to inadvertant changes through reducing their respective scopes in the DFCS software.

To accomplish this, a data flow decomposition strategy (Ref. 9) has been introduced at the applications software level. Figure 16 depicts the intermediate step for this design stage. System and cross-channel input signals are introduced to a single channel on the left, and output signals emerge on the right of Figure 16. In between new data objects are identified that are internal to the DFCS applications software, along with their flow relative to the applications procedures in Figure 14. Finally, the three intermediate-level Ada packages of Figure 12 are shown, information additional to that normally contained in data flow diagrams.

This data flow representation was actually used to group associated data objects and procedures within these Ada packages and to determine the levels at which each data object is to be declared in the Ada-based design. The lower the levels are in general, the less is the namespace over most of the program execution. This type information is indicated to a certain extent statically in the call/usage graph presented in Figure 17. Definition of the actual Ada package specification parts is then initiated from information in these representations.

Note that Figure 17 portrays appreciable complexity and dispersed dependencies in the overall DFCS/test program. This complexity is due to software fault tolerance and to the composition of the test harness. Perhaps the biggest contributing factor is the non-interference requirement imposed on the test harness. Section 14.0 further describes the mechanization and rationale.

The associated Ada source code listings for the packages shown in Figures 16 are presented in Figure 18: (a) Package DFCS_LOGIC; (b) Package DFCS_RESOURCES; (c) Package CONTROL_LAWS; (d) Package N_VERSION_VOTERS; and (e) Package VOTING_PLANES. These package specifications, which capture much of the DFCS applications software design in non-executable Ada code, are referenced by the Ada N-version procedures pursuant to the namespace reduction strategy.

All but Packages N_VERSION_VOTER and VOTING_PLANES are referenced by Procedure RUN_FOREGROUND_x, as indicated in Figures 12 and 17. The affected applications procedures reference Package VOTING_PLANES, which effects cross-channel synchronization and calls the voter procedure contained in Package N_VERSION_VOTER. The voting requirements are summmarized in Figure 19.

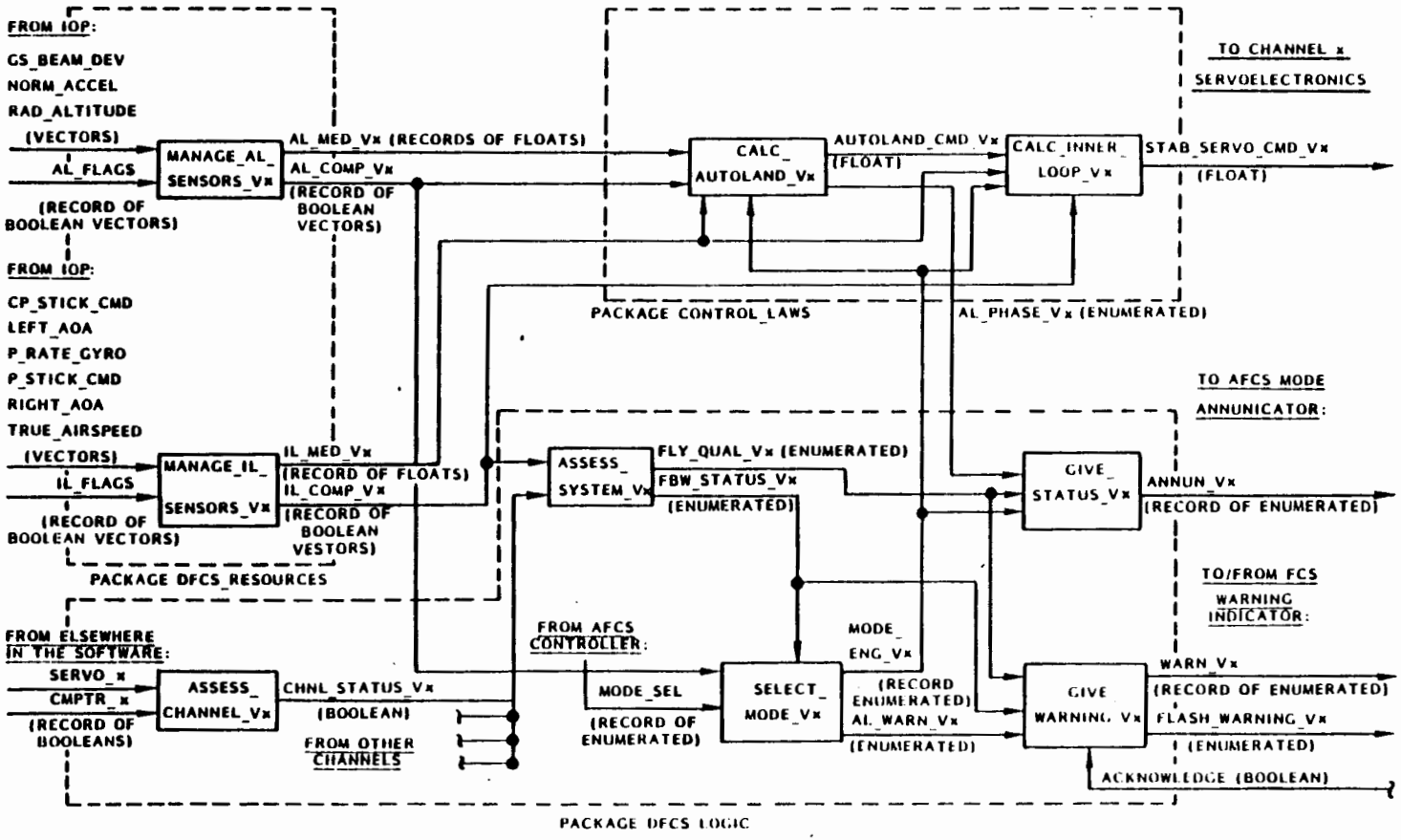


Figure 16 - DFCS Data Flow Diagram

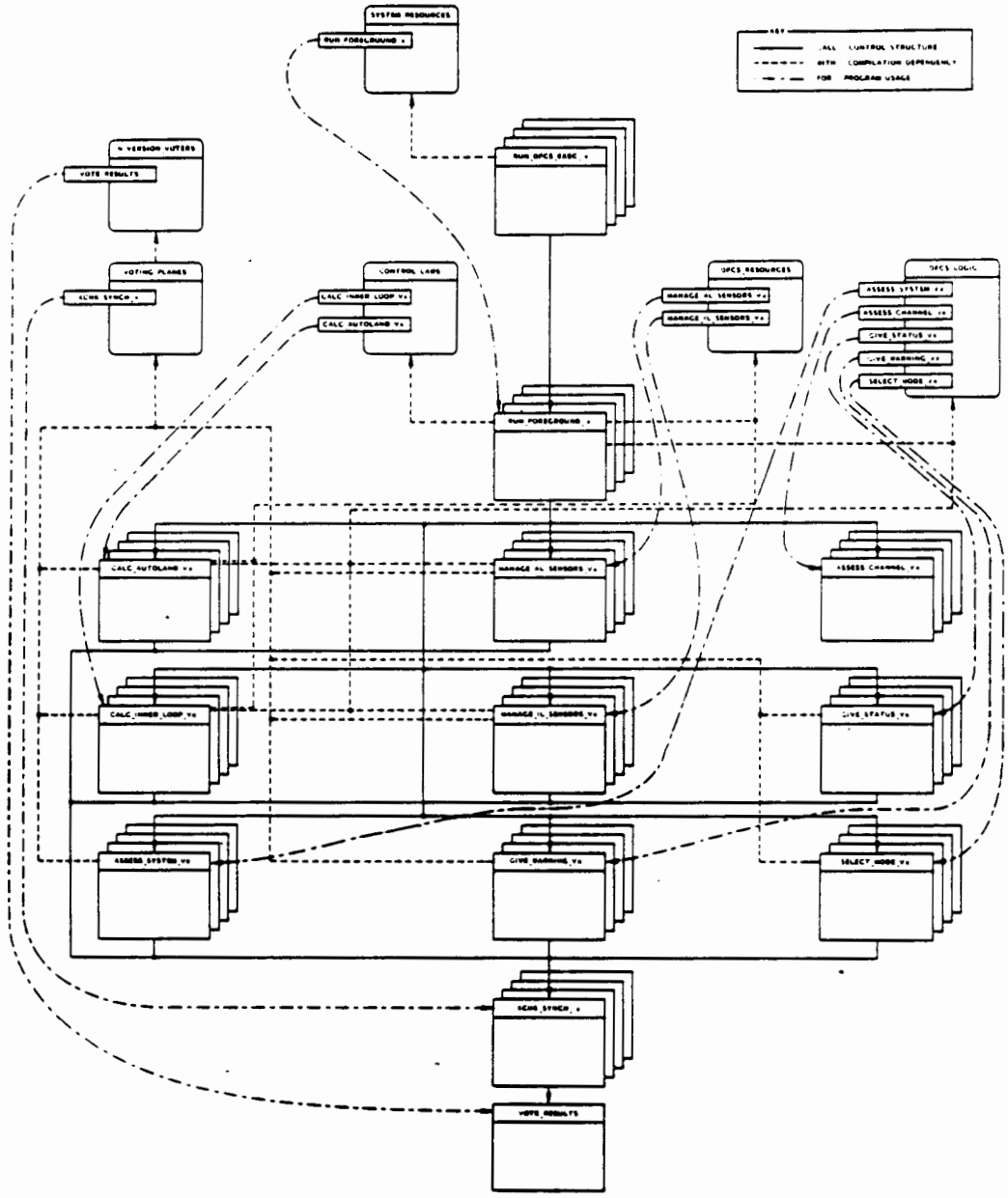


Figure 17 - Call/Usage Graph

```
package CONTROL_LAWS is

--  Procedure Declarations
    ----------------------

        procedure CALC_AUTOLAND_V1   ;
        procedure CALC_INNER_LOOP_V1 ;

        procedure CALC_AUTOLAND_V2   ;
        procedure CALC_INNER_LOOP_V2 ;

        procedure CALC_AUTOLAND_V3   ;
        procedure CALC_INNER_LOOP_V3 ;

        procedure CALC_AUTOLAND_V4   ;
        procedure CALC_INNER_LOOP_V4 ;

--  Type Declarations
    -----------------

        type PITCH_COMMAND      is new FLOAT range -5.0..10.0 ;

        type STAB_COMMAND       is new FLOAT range -11.0..2.0 ;

--  Object Declarations
    -------------------

        AUTOLAND_CMD_V1, AUTOLAND_CMD_V2, AUTOLAND_CMD_V3,
                         AUTOLAND_CMD_V4                  : PITCH_COMMAND ;

        STAB_SERVO_CMD_V1, STAB_SERVO_CMD_V2, STAB_SERVO_CMD_V3,
                           STAB_SERVO_CMD_V4               : STAB_COMMAND ;

end CONTROL_LAWS ;
```

```
package body CONTROL_LAWS is

        procedure CALC_AUTOLAND_V1   is separate ;
        procedure CALC_INNER_LOOP_V1 is separate ;

        procedure CALC_AUTOLAND_V2   is separate ;
        procedure CALC_INNER_LOOP_V2 is separate ;

        procedure CALC_AUTOLAND_V3   is separate ;
        procedure CALC_INNER_LOOP_V3 is separate ;

        procedure CALC_AUTOLAND_V4   is separate ;
        procedure CALC_INNER_LOOP_V4 is separate ;

end CONTROL_LAWS ;
```

Figure 18a - DFCS Applications Package Listing: Package CONTROL_LAWS

```
package DFCS_LOGIC is

--  Procedure Declarations
    ----------------------                          --  N-Version Modules

        procedure ASSESS_CHANNEL_V1 ;
        procedure ASSESS_SYSTEM_V1  ;
        procedure GIVE_STATUS_V1    ;
        procedure GIVE_WARNING_V1   ;
        procedure SELECT_MODE_V1    ;

        procedure ASSESS_CHANNEL_V2 ;
        procedure ASSESS_SYSTEM_V2  ;
        procedure GIVE_STATUS_V2    ;
        procedure GIVE_WARNING_V2   ;
        procedure SELECT_MODE_V2    ;

        procedure ASSESS_CHANNEL_V3 ;
        procedure ASSESS_SYSTEM_V3  ;
        procedure GIVE_STATUS_V3    ;
        procedure GIVE_WARNING_V3   ;
        procedure SELECT_MODE_V3    ;

        procedure ASSESS_CHANNEL_V4 ;
        procedure ASSESS_SYSTEM_V4  ;
        procedure GIVE_STATUS_V4    ;
        procedure GIVE_WARNING_V4   ;
        procedure SELECT_MODE_V4    ;

--  Type Declarations
    -----------------

    --  Sensor Validity Logic:

        type QUAD_VALIDITY    is array (1..4) of BOOLEAN ;
        type TRIAD_VALIDITY   is array (1..3) of BOOLEAN ;
        type PAIR_VALIDITY    is array (1..2) of BOOLEAN ;

    --  Autoland Sensor Logic:

        type AL_SENSOR_STATUS is
            record
                GS_BEAM_VAL      : QUAD_VALIDITY ;
                N_ACCEL_VAL      : TRIAD_VALIDITY ;
                RAD_ALT_VAL      : QUAD_VALIDITY ;
            end record ;
```

Figure 18b - DFCS Applications Package Listing
Package DFCS_LOGIC (1 of 4)

```
-- Inner Loop Sensor Logic:

   type IL_SENSOR_STATUS is
       record
           AFC_AOA_VAL        : QUAD_VALIDITY ;
           C_STR_VAL          : QUAD_VALIDITY ;
           LF_AOA_VAL         : QUAD_VALIDITY ;
           P_STR_VAL          : QUAD_VALIDITY ;
           P_RATE_VAL         : TRIAD_VALIDITY ;
           RL_AOA_VAL         : QUAD_VALIDITY ;
           TAS_VAL            : PAIR_VALIDITY ;
       end record ;

-- Autopilot Controller Logic:

   type AP_SELECTION          is (ALT_HOLD, AUTOLAND, BASIC,
                                  VERT_NAV, OFF) ;
   type AL_CATEGORY           is (CAT_1, CAT_2, CAT_3A, OFF) ;
   type AFCS_SELECTION is
       record
           AUTOPILOT          : AP_SELECTION ;
           AUTOLAND           : AL_CATEGORY ;
       end record ;

-- Annunciator Logic:

   type AL_PROGRESS           is (AUTOLAND_ARMED, GLIDE_CORE_TRACK,
                                  DECISION_ALTITUDE, ALERT_ALTITUDE,
                                  FLARE, AUTOLAND_INOP) ;
   type CUM_AL_PROGRESS       is array (1..5) OF BOOLEAN ;
   type ENGAGE_STATUS         is (AFCS_DISENGAGED, AUTOPILOT_ENGAGED,
                                  AUTOLAND_ENGAGED) ;
   type FLYING_QUALITIES      is (UNFLYABLE, MARGINAL, DEGRADED,
                                  NORMAL) ;
   type AN_FCS_STATUS         is
       record
           AFCS_STATUS            : ENGAGE_STATUS ;
           AL_PROG_DISP           : CUM_AL_PROGRESS ;
           AUTOPILOT_MODE         : AP_SELECTION ;
           FLY_QLTY               : FLYING_QUALITIES ;
       end record ;

-- Warning Logic:

   type MASTER_WARN           is (BLINKING, STEADY, OFF) ;
   type PRI_FCS_STATUS        is (OP_STATE_5, OP_STATE_3,
                                  OP_STATE_2, OP_STATE_1) ;
   type AL_STATUS             is (CAT_3_INOP, CAT_2_INOP, BLANK) ;
   type FBW_STATUS            is (OP_STATE_4, OP_STATE_3, OP_STATE_2,
                                  BLANK) ;
   type FQ_STATUS             is (IMPAIRED_FQ, BLANK) ;
   type WARNING_STATE         is
       record
           AUTOLAND           :       AL_STATUS ;
           FLY_BY_WIRE        :       FBW_STATUS ;
           FLYING_QUAL        :       FQ_STATUS ;
       end record ;
```

Figure 18b - DFCS Applications Package Listing Package DFCS_LOGIC (2 of 4)

```
Object declarations
-------------------

--  Sensor validities:

    AL_COMP_V1, AL_COMP_V2,
    AL_COMP_V3, AL_COMP_V4, AL_FLAGS          : AL_SENSOR_STATUS ;
    TL_COMP_V1, TL_COMP_V2,
    TL_COMP_V3, TL_COMP_V4, TL_FLAGS          : TL_SENSOR_STATUS ;

--  Annunciations:

    AL_PHASE_V1, AL_PHASE_V2,
    AL_PHASE_V3, AL_PHASE_V4                  : AL_PROGRESS ;
    ANNUN_V1, ANNUN_V2, ANNUN_V3, ANNUN_V4    : ANNUN_STATUS ;
    FLY_QUAL_V1, FLY_QUAL_V2,
    FLY_QUAL_V3, FLY_QUAL_V4                  : FLYING_QUALITIES ;

--  Warnings:

    AUDIO_ENABLE                              : BOOLEAN ;
    ALL_ARM_V1, ALL_ARM_V2, ALL_ARM_V3,
    ALL_ARM_V4                                : ALL_STATUS ;
    PRI_STATUS_V1, PRI_STATUS_V2,
    PRI_STATUS_V3, PRI_STATUS_V4              : PRI_FCS_STATUS ;
    FLASH_WARNING_V1, FLASH_WARNING_V2,
    FLASH_WARNING_V3, FLASH_WARNING_V4        : MASTER_WARN ;
    WARN_V1, WARN_V2, WARN_V3, WARN_V4        : WARNING_STATE ;

--  Mode engagements:

    MODE_ENG_V1, MODE_ENG_V2,
    MODE_ENG_V3, MODE_ENG_V4                  : AFCS_SELECTION ;

--  Mode Selections:

    MODE_SEL        :       AFCS_SELECTION  ;

end DFCS_LOGIC ;
```

Figure 18b - DFCS Applications Package Listing Package DFCS_LOGIC (3 of 4)

```
package body DFCS_LOGIC is

-- Procedures:

        procedure ASSESS_CHANNEL_V1  is separate ;
        procedure ASSESS_SYSTEM_V1   is separate ;
        procedure GIVE_STATUS_V1     is separate ;
        procedure GIVE_WARNING_V1    is separate ;
        procedure SELECT_MODE_V1     is separate ;

        procedure ASSESS_CHANNEL_V2  is separate ;
        procedure ASSESS_SYSTEM_V2   is separate ;
        procedure GIVE_STATUS_V2     is separate ;
        procedure GIVE_WARNING_V2    is separate ;
        procedure SELECT_MODE_V2     is separate ;

        procedure ASSESS_CHANNEL_V3  is separate ;
        procedure ASSESS_SYSTEM_V3   is separate ;
        procedure GIVE_STATUS_V3     is separate ;
        procedure GIVE_WARNING_V3    is separate ;
        procedure SELECT_MODE_V3     is separate ;

        procedure ASSESS_CHANNEL_V4  is separate ;
        procedure ASSESS_SYSTEM_V4   is separate ;
        procedure GIVE_STATUS_V4     is separate ;
        procedure GIVE_WARNING_V4    is separate ;
        procedure SELECT_MODE_V4     is separate ;

end DFCS_LOGIC ;
```

Figure 18b - DFCS Applications Package Listing
Package DFCS_LOGIC (4 of 4)

```
package DFCS_RESOURCES is

--  Procedure Declarations
    ----------------------                    --  N-Version Modules

        procedure MANAGE_AH_SENSORS_V1 ;
        procedure MANAGE_IO_SENSORS_V1 ;

        procedure MANAGE_AH_SENSORS_V2 ;
        procedure MANAGE_IO_SENSORS_V2 ;

        procedure MANAGE_AH_SENSORS_V3 ;
        procedure MANAGE_IO_SENSORS_V3 ;

        procedure MANAGE_AH_SENSORS_V4 ;
        procedure MANAGE_IO_SENSORS_V4 ;


--  Type Declarations
    -----------------

    --  Autoland:

        type ATCTL_SIGNAL     is new FLOAT range -1.0..3.0 ;       -- g's
        type BEAM_DEV_SIGNAL  is new FLOAT range -2.5..2.5 ;    -- degrees
        type RAD_ALT_SIGNAL   is new FLOAT range -20.0..2500.0 ;  -- feet

        type AH_SENSOR_SET is
        record
                GS_DEV          :       BEAM_DEV_SIGNAL ;
                N_ACCEL         :       ACCEL_SIGNAL ;
                RAD_ALT         :       RAD_ALT_SIGNAL ;
        end record ;


    --  Inner Loop:

        type ANG_RATE_SIGNAL  is new FLOAT range -25.0..25.0 ; -- deg/sec
        type AOA_SIGNAL       is new FLOAT range -10.0..50.0 ; -- degrees
        type STICK_CMD        is new FLOAT range -1.5..0.5 ;   -- degrees
        type TAS_SIGNAL       is new FLOAT range 100.0..800.0 ;  -- knots

        type IO_SENSOR_SET is
        record
                AOA_DISPL       :       AOA_SIGNAL ;
                CP_STICK        :       STICK_CMD ;
                P_RATE          :       ANG_RATE_SIGNAL ;
                P_STICK         :       STICK_CMD ;
                TR_AIRSPEED     :       TAS_SIGNAL ;
        end record ;
```

Figure 18c - DFCS Applications Package Listing
Package DFCS_RESOURCES (1 of 3)

```
-- Object Declarations
   -------------------

   -- Voter/Comparator Outputs

      AOA_FBLevl, AOA_MED_V2, AOA_FBD_V3, AOA_MED_V4  : AOA_INSUR_SET ;
      IAS_FBLevl, IAS_MED_V2, IAS_FBD_V3, IAS_MED_V4  : IAS_INSUR_SET ;


   -- Autoland Sensors

      type GS_DEV_QUAD          is array (1..4) of GEAR_DEV_SIGNAL ;
      type N_ACCEL_TRIAD        is array (1..3) of ACCEL_SIGNAL ;
      type RAD_ALT_QUAD         is array (1..4) of RAD_ALT_SIGNAL ;


   -- Inner Loop Sensors

      type AOA_QUAD             is array (1..4) of AOA_SIGNAL ;
      type RATE_GYRO_TRIAD      is array (1..3) of ANG_RATE_SIGNAL ;
      type STICK_CMD_QUAD       is array (1..4) of STICK_CMD ;
      type IAS_PAIR             is array (1..2) of IAS_SIGNAL ;


-- Object Declarations
   -------------------

   -- Autoland Sensors:        Voter/Comparator Inputs

      GS_DEV_QUAD      : GS_DEV_QUAD := ( 0.0, 0.0, 0.0, 0.0 ) ;
      NOR_ACCEL        : N_ACCEL_TRIAD := ( 0.0, 0.0, 0.0 ) ;
      RAD_ALTITUDE     : RAD_ALT_QUAD := ( 0.0, 0.0, 0.0, 0.0 ) ;


   -- Inner Loop Sensors:      Voter/Comparator Inputs

      CP_STICK_CMD     : STICK_CMD_QUAD := ( 0.0, 0.0, 0.0, 0.0 ) ;
      LEFT_AOA         : AOA_QUAD := ( 0.0, 0.0, 0.0, 0.0 ) ;
      P_RATE_GYRO      : RATE_GYRO_TRIAD := ( 0.0, 0.0, 0.0 ) ;
      P_STICK_CMD      : STICK_CMD_QUAD := ( 0.0, 0.0, 0.0, 0.0 ) ;
      RIGHT_AOA        : AOA_QUAD := ( 0.0, 0.0, 0.0, 0.0 ) ;
      TRUE_AIRSPEED    : IAS_PAIR := ( 100.0, 100.0 ) ;

end DFCS_RESOURCES ;
```

Figure 18c - DFCS Applications Package Listing
Package DFCS_RESOURCES (2 of 3)

```
package body DFCS_RESOURCES is

    --  Procedures:

        procedure MANAGE_AL_SENSORS_V1 is separate ,
        procedure MANAGE_IL_SENSORS_V1 is separate ;

        procedure MANAGE_AL_SENSORS_V2 is separate ;
        procedure MANAGE_IL_SENSORS_V2 is separate ;

        procedure MANAGE_AL_SENSORS_V3 is separate ;
        procedure MANAGE_IL_SENSORS_V3 is separate ;

        procedure MANAGE_AL_SENSORS_V4 is separate ;
        procedure MANAGE_IL_SENSORS_V4 is separate ;

end DFCS_RESOURCES ;
```

Figure 18c - DFCS Applications Package Listing
Package DFCS_RESOURCES (3 of 3)

```
with CONTROL_LAWS          : use CONTROL_LAWS ;
with DFCS_LOGIC            : use DFCS_LOGIC ;
with DFCS_RESOURCES        : use DFCS_RESOURCES ;
package N_VERSION_VOTERS is

--  Type Declarations
    -----------------

        subtype CC_POINT          is INTEGER range 0..8 ;

        type CC_DATA              is array (1..4,1..4) of BOOLEAN ;

--  Procedure Declaration
    ---------------------

        procedure VOTE_RESULTS(CORR_POW : in CC_POINT ;
                               OUTCOMES : out CC_DATA) ;

--  Type Declarations
    -----------------

        type AL_PROGRESS_VECT       is array (1..4) of AL_PROGRESS ;
        type AL_SENSOR_SET_VECT     is array (1..4) of AL_SENSOR_SET ;
        type AL_SENSOR_STATUS_VECT  is array (1..4) of AL_SENSOR_STATUS ;
        type ANNUN_STATUS_VECT      is array (1..4) of ANNUN_STATUS ;
        type AFCS_SELECTION_VECT    is array (1..4) of AFCS_SELECTION ;
        type FLYING_QUALITIES_VECT  is array (1..4) of FLYING_QUALITIES ;
        type IL_SENSOR_SET_VECT     is array (1..4) of IL_SENSOR_SET ;
        type IL_SENSOR_STATUS_VECT  is array (1..4) of IL_SENSOR_STATUS ;
        type MASTER_WARN_VECT       is array (1..4) of MASTER_WARN ;
        type PITCH_COMMAND_VECT     is array (1..4) of PITCH_COMMAND ;
        type PRI_FCS_STATUS_VECT    is array (1..4) of PRI_FCS_STATUS ;
        type STAB_COMMAND_VECT      is array (1..4) of STAB_COMMAND ;
        type WARNING_STATE_VECT     is array (1..4) of WARNING_STATE ;

--  Object Declarations
    -------------------

        VO_AL_COMP                  : AL_SENSOR_STATUS_VECT ;
        VO_AL_MED                   : AL_SENSOR_SET_VECT ;
        VO_AL_PHASE                 : AL_PROGRESS_VECT ;
        VO_ANNUN                    : ANNUN_STATUS_VECT ;
        VO_AFCS_OLAMP_CMD           : PITCH_COMMAND_VECT ;
        VO_PRI_STATUS               : PRI_FCS_STATUS_VECT ;
        VO_FLASH_WARNING            : MASTER_WARN_VECT ;
        VO_FLY_QUAL                 : FLYING_QUALITIES_VECT ;
        VO_IL_COMP                  : IL_SENSOR_STATUS_VECT ;
        VO_IL_MED                   : IL_SENSOR_SET_VECT ;
        VO_MODE_ENG                 : AFCS_SELECTION_VECT ;
        VO_STAB_SERVO_CMD           : STAB_COMMAND_VECT ;
        VO_WARN                     : WARNING_STATE_VECT ;

end N_VERSION_VOTERS ;
```

Figure 18d - DFCS Applications Package Listing
Package N_VERSION_VOTERS

```
with N_VERSION_VOTERS   ; use N_VERSION_VOTERS ;
package VOTING_PLANES is

--  Procedure Declarations
    ----------------------

        procedure XCHK_SYNCH_1 ;
        procedure XCHK_SYNCH_2 ;
        procedure XCHK_SYNCH_3 ;
        procedure XCHK_SYNCH_4 ;

--  Object Declarations
    -------------------

        CHNL_1_XCHK_NUM, CHNL_2_XCHK_NUM,
        CHNL_3_XCHK_NUM, CHNL_4_XCHK_NUM    : CC_POINT ;

end VOTING_PLANES ;

--------------------------------------------------------------------

package body VOTING_PLANES is

        procedure XCHK_SYNCH_1 is separate ;
        procedure XCHK_SYNCH_2 is separate ;
        procedure XCHK_SYNCH_3 is separate ;
        procedure XCHK_SYNCH_4 is separate ;

end VOTING_PLANES ;
```

Figure 18e - DFCS Applications Package Listing
Package VOTING_PLANES

XCHK PT	PROCEDURE	VOTED SIGNAL(S)	TYPE(S)	MATCH
1	SELECT_MODE_Vx	MODE_ENG_Vx	Record of Enumerated	Exact
2	GIVE_STATUS_Vx	ANNUN_Vx	Record of Enumerated	"
3	MANAGE_AL_SENSORS_Vx	AL_MED_Vx AL_COMP_Vx	Record of Floats Record of Boolean Vectors	TBD Exact
4	CALC_AUTOLAND_Vx	AUTOLAND_CMD_Vx AL_PHASE_Vx	Float Enumerated	TBD Exact
5	MANAGE_IL_SENSORS_Vx	IL_MED_Vx IL_COMP_Vx	Record of Floats Record of Boolean Vectors	TBD Exact
6	CALC_INNER_LOOP_Vx	STAB_SERVO_CMD_Vx	Float	TBD
7	ASSESS_SYSTEM_Vx	FLY_QUAL_Vx FBW_STATUS_Vx	Enumerated "	Exact "
8	GIVE_WARNING_Vx	WARN_Vx FLASH_WARNING_Vx	Record of Enumerated Enumerated	" "

Figure 19 - N-Version Voting Requirements

```
with CONTROL_LAWS        ; use CONTROL_LAWS ;
with DFCS_LOGIC          ; use DFCS_LOGIC ;
with DFCS_RESOURCES      ; use DFCS_RESOURCES ;

separate(SYSTEM_RESOURCES)
procedure RUN_FOREGROUND_1 is

begin

        SELECT_MODE_V1 ;
        case PATH_1 is
                when 0      => null ;
                when 1 | 3 =>
                        ASSESS_CHANNEL_V1 ;
                        if PATH_1 = 1 then
                                GIVE_STATUS_V1 ;
                        end if ;
                        MANAGE_AL_SENSORS_V1 ;
                        CALC_AUTOLAND_V1 ;
                when 2 | 4 =>
                        ASSESS_SYSTEM_V1 ;
                        if PATH_1 = 4 then
                                GIVE_WARNING_V1 ;
                        end if ;
        end case ;
        MANAGE_IL_SENSORS_V1 ;
        CALC_INNER_LOOP_V1 ;

end RUN_FOREGROUND_1 ;
```

Figure 20 - Procedure RUN_FOREGROUND_1 Listing

4.0 MULTIRATE EXECUTIVE DESCRIPTION

The control structure of Procedure RUN_FOREGROUND_x was presented in a multirate flow graph form in Figure 14, and the intent has been to implement it and its called procedures such that the applications software might run intact in either a (hypothetical) target flight computer or a host computer test harness. The foreground executive procedure was implemented for each of the four DFCS channels, and incorporated into the test harness (see Section 14.0). The Ada source code listing for RUN_FOREGROUND_1 is given in Figure 20.

With the exception of channel/version number designations, the software for RUN_FOREGROUND_1 is the same as for each of the other channels. In the names of program units such as RUN_FOREGROUND_1, note that the suffix "1" by itself denotes pre-existing Channel 1 software, whereas "V1" designates later-to-be-developed Version 1 software. Of course, all Version 1 software is used in Channel 1.

The listing in Figure 20 is mainly the executable code that calls the N-version control function applications procedures. The location of N-version cross-check points do not appear in the listing, as Figure 11 might suggest, because voter calls take place at a lower level in the program structure. This is done primarily because the same synchronization process is used by all voted procedures. Also, it facilitates changes to procedure outputs for fault correction purposes, which is more easily handled if the affected procedure has not been exited. The associated mechanization, moreover, seems to afford some reductions of the namespace.

Only one version of N-version voting is employed because multiple voting implementations would significantly and needlessly complicate the investigation. Also, voting of the foreground executive itself is avoided as an unwarranted addition of complexity. Besides, the limited scope and logic-oriented nature of DFCS executives render them amenable to formal verification. Hence, the need for software fault tolerance on this level may conceivably be alleviated. Such issues may possibly be addressed under NASA Langley auspices (see Ref. 10).

Timing intervals for the N-version code segments were stipulated in Figure 15, with a total of 50 milliseconds alloted for each top-down path traversal in Figure 14. Associated 20 Hertz hardware timer interrupts are in general assumed to be satisfied in all channels for most N-version testing purposes, but code segment timeouts at cross-check points are to be explored per Section 14.0.

5.0 SELECT_MODE PROCEDURE SPECIFICATION

The operational mode(s) of each channel shall be determined based on identical externally applied logic signals to each channel and on internally generated ones reflecting the availability conditions of the AFBW (augmented fly-by-wire) function and the autoland sensors. The internal logic shall confirm that the selected mode(s) is(are) engagable. The resultant mode selection(s) shall then be furnished for activation of the corresponding control functions and for indication of mode engagement to the autopilot controller.

5.1 Autopilot Modes

Autopilot mode engagement shall be determined by the externally applied logic signal MODE_SEL, which is available to all channels. The order of precedence of mode engagement in ascending order shall be: Basic, Altitude Hold, Vertical Navigation, Autoland, and Off. Due to external logic interlocks, when MODE_SEL.AUTOPILOT is set at Autoland, MODE_SEL.AUTOLAND will never be set at off. The output MODE_ENG_Vx.AUTOPILOT reflects the input selection in all but the Autoland Mode which shall be conditionally engageable.

5.2 Autoland Mode

The autoland selection, MODE_ENG_Vx.AUTOLAND, shall be determined by the internal logic signals, AL_COMP_Vx and FBW_STATUS_Vx, according to the following logic conditions:

```
Off  -->                Autoland Not Selected  OR  No Category Engagable.

Category 1  -->         Autoland Selected
                   AND  ({Category 1 Selected
                   AND  Minimum of 1 Each Autoland Sensors}

                   OR   {Category 2 or 3 Selected
                   AND  [Exactly 1 Sensor for at Least One Type of Sensor
                   AND  At  Least 1 of Other Types of Sensors
                   OR   Operational State Less than 2]}).

Category 2  -->         Autoland Selected
                   AND  ({Category 2 Selected
                   AND  Minimum of 2 Each Autoland Sensors
                   AND  Minimum Operational State 2}

                   OR   {Category 3 Selected
                   AND  [at Least 1 Autoland Sensor Fault
                   AND  Minimum of 2 Each Autoland Sensors
                   AND  Minimum of Operational State 2
                   OR   Operational State of 2]}).

Category 3A -->         Autoland Selected
                   AND  Category 3 Selected
                   AND  All Autoland Sensors
                   AND  Operational State 1.
```

5.2.1 Autoland Category Reversion

If failures occur during a higher category autoland, the autoland engagement shall revert to the next lower category whose engage logic is satisfied.

5.2.2 Autoland Select Warning

If Category 2 or 3 Autoland is selected, but cannot be engaged, this situation shall be reflected in the logic signal AL_WARN_Vx. If Category 3 is selected, but not engagable, AL_WARN_Vx shall be set to CAT_3_INOP. If Category 2 is selected, but not engageable, AL_WARN_Vx shall be set to CAT_2_INOP. In all other cases, AL_WARN_Vx shall be set to OFF.

5.3 Maximum Allowable Computational Time

The maximum allowable sub-frame time for this computation shall be 2 milliseconds.

5.4 Input/Output

```
                                  ---------
                                  |INPUTS |
                                  ---------

AL_COMP_Vx                  : AL_SENSOR_STATUS ;

type AL_SENSOR_STATUS is
     record
         GS_BEAM_VAL        : QUAD_VALIDITY ;
         N_ACCEL_VAL        : TRIAD_VALIDITY ;
         RAD_ALT_VAL        : QUAD_VALIDITY ;
     end record ;


FBW_STATUS_Vx               : PRI_FCS_STATUS ;

type PRI_FCS_STATUS is (OP_STATE_4, OP_STATE_3, OP_STATE_2,
                        OP_STATE_1) ;

MODE_SEL                    : AFCS_SELECTION ;

type AFCS_SELECTION is
   record
         AUTOPILOT                : AP_SELECTION ;
         AUTOLAND                 : AL_CATEGORY ;
     end record ;

type AP_SELECTION is (ALT_HOLD, AUTOLAND, BASIC, VERT_NAV, OFF) ;
type AL_CATEGORY  is (CAT_1, CAT_2, CAT_3A, OFF );
```

```
                                  ----------
                                  |OUTPUTS |
                                  ----------

AL_WARN_Vx                           : AL_STATUS ;

type AL_STATUS is (CAT_3_INOP, CAT_2_INOP, BLANK) ;

MODE_ENG_Vx                          : AFCS_SELECTION ;
```

Ada Procedure SELECT_MODE_Vx.ADA

```
with VOTING_PLANES ; use VOTING_PLANES ;
separate(DFCS_LOGIC)
procedure SELECT_MODE_Vx is

        --  Local Declarations (if any)
        --  Place Static Variables in Programmer-Defined Package(s).

-----------------------------------------------------------------------

-- Using the mode selection/enablement inputs as defined in Section
-- 5.4 (as declared in Package DFCS_LOGIC) determine the resultant
-- output signals: mode engagement (MODE_ENG_Vx) and autoland warning
-- (AL_WARN_Vx) per the English text specification requirements.

-----------------------------------------------------------------------

begin           -- Procedure SELECT_MODE_Vx


--
--
-- Add Demonstration Software Here
--
--


CHNL_x_XCHK_NUM  := 1 ;
XCHK_SYNCH_x ;                  -- Call for N-Version Vote

end SELECT_MODE_Vx ;
```

6.0 ASSESS_CHANNEL PROCEDURE SPECIFICATION

Once a channel is initialized and Procedure RUN_FOREGROUND_x is running, the fault logic states of channel components shall be monitored to ascertain the continued proper status of the channel, independent of the conditions of the other channels. Normally then, a channel will be activated and its servoactuator engaged before this procedure can be called. Once Procedure RUN_FOREGROUND_x is operating, this procedure oversees channel fault and recovery events until the maximum recoveries below are exceeded.

6.1 Channel Validity Logic

Channel x's status, CHNL_STATUS_Vx, shall be determined via an examination of the associated servo status, SERVO_x, and the computer channel states, CMPTR_x.

6.1.1 Servo Validity

Since SERVO_x is of Type Record, the various servo validities shall be examined. All must evaluate True under the limitations described below for the associated servo to be considered in acceptable condition.

6.1.2 Computer Validity

Similarly, CMPTR_x is a Record Type, so its elements must all evaluate True under the limitations prescribed below for acceptability.

6.2 Logic State Change

The state of CHNL_STATUS_Vx will have been set True prior to the initial calling of this procedure during any given execution. Having initially been set True, prescribed time delays, or iteration counts, shall be observed in declaring the channel validity False. Under certain conditions, a channel validity shall be restored if a faulted item remains healed sufficiently long.

6.2.1 Time Delays

The following delays shall be applied to the respective logic states on an independent basis. Specifically, there shall be no internal logic coupling of constituent validity states, so non-offending validity signals must be monitored while CHNL_STATUS_Vx is False for other reasons. CHNL_STATUS_Vx shall be set False if any of the input variables given below are False for the indicated number of times in a row:

CPU_CHK_OK	1 count
IO_PROC_OK	1 count
MUX_BUS_OK	3 counts

ACTUATOR_ON	3 counts
LVDT_VALID	4 counts
POWER_AVAIL	1 count

6.2.2 Channel Recovery Delays

A channel shall recover and operate in the foreground applications mode up to a specified number of times if all appropriate indications of channel recovery and acceptability are satisfied. Basically, recovery indications are particular durations of acceptable validity states following an associated validity trip:

CPU_CHK_OK or IO_PROC_OK	maximum of 5 recoveries, each following a 10-count duration of validity after a declared logic trip.
MUX_BUS_OK	maximum of 6 recoveries, each following a 50-count duration of validity after a declared logic trip.
ACTUATOR_ON or LVDT_VALID	maximum of 2 recoveries, each following a 50-count duration of validity after a declared logic trip.
POWER_AVAIL	no limit or delay on recoveries in software.

6.3 Maximum Allowable Computation Time

The maximum allowable sub-frame time for this computation shall be 2 milliseconds.

6.4 Input/Output

```
                                  ----------
                                  | INPUTS |
                                  ----------

CMPTR_x                : CMPTR_CHANNEL_STATUS ;

type CMPTR_CHANNEL_STATUS is
   record
        CPU_CHL_OK     : BOOLEAN ;
        IO_PROC_OK     : BOOLEAN ;
        MUX_BUS_OK     : BOOLEAN ;
    end record ;

SERVO_x                : SERVO_STATUS ;

type SERVO_STATUS is
     record
        ACTUATOR_ON    : BOOLEAN ;
        LVDT_VALID     : BOOLEAN ;
        POWER_AVAIL    : BOOLEAN ;
     end record ;


                                  -----------
                                  | OUTPUTS |
                                  -----------

CHNL_STATUS_Vx         : BOOLEAN ;
```

Procedure ASSESS_CHANNEL_Vx

```
with CHANNEL_RESOURCES ; use CHANNEL_RESOURCES ;
separate(DFCS_LOGIC)
procedure ASSESS_CHANNEL_Vx is

        -- Local Declarations (if any)
        -- Place Static Variables in User-Defined Package(s)

-----------------------------------------------------------------------

-- Using the computer channel status and servo status data as
-- defined in Package CHANNEL_RESOURCES, determine the overall
-- channel status, CHNL_STATUS_Vx, per the English text part of the
-- specification requirements.

-----------------------------------------------------------------------

begin                          -- Procedure ASSESS_CHANNEL_Vx

null;

--
--
-- Add Demonstration Software Here
--
--

-- No N-Version Vote Taken Because Status is Unique to each Channel

end ASSESS_CHANNEL_Vx;
```

7.0 GIVE_STATUS PROCEDURE SPECIFICATION

Mode annunciator outputs shall be generated based on internal logic computations. The outputs for pilot display shall include autopilot engage/select status, autoland progress, and stability augmentation performance. Each computer channel will generate an output, and the N-version voter will resolve contradictions.

7.1 Annunciator Display Outputs

Four functional state outputs shall be generated as ANNUN_Vx per the record type ANNUN_STATUS.

7.1.1 Automatic Flight Control System Status

The autopilot engage status shall be derived from the input MODE_ENG_VX, with the following rules for the output ANNUN_Vx.AFCS_STATUS:

```
MODE_ENG_Vx.AUTOPILOT = OFF          -->  AFCS_DISENGAGED

MODE_ENG_Vx.AUTOPILOT = ALT_HOLD + BASIC + VERT_NAV

                                     -->  AUTOPILOT_ENGAGED

MODE_ENG_Vx.AUTOPILOT = AUTOLAND     -->  AUTOLAND_ENGAGED
```

7.1.2 Autopilot Mode Engagement

The autopilot mode, ANNUN_Vx.AUTOPILOT_MODE, shall be set equal to the selected autopilot mode, MODE_ENG_Vx.AUTOPILOT, since they are both of Type AP_SELECTION.

7.1.3 Autoland Progress

If MODE_ENG_Vx.AUTOPILOT = AUTOLAND, the autoland progress display ANNUN_Vx.AL_PROG_DISP, a 1x5 Boolean vector, shall reflect the input state and input sequence furnished by AL_PHASE_Vx. Progress shall be indicated by setting corresponding output vector elements to True. Except for AUTOLAND_INOP, this Boolean vector has a one-to-one correspondence with the values of the enumerated type AL_PROGRESS : AUTOLAND_ARMED, GLIDESLOPE_TRACK, DECISION_ALTITUDE, ALERT_ALTITUDE, FLARE. Normal autoland progress is noted by stepping through these phases in the above order with the following externally controlled exceptions:

- **Category 1 progress proceeds only through DECISION_ALTITUDE**
- **Category 2 skips ALERT_ALTITUDE**
- **Category 3A skips DECISION_ALTITUDE.**

The output ANNUN_Vx.AL_PROG_DISP should reflect the progression observed by indicating the cumulative phases. Thus, once AL_PHASE_Vx is set to AUTOLAND_ARMED, it and the succeeding phases shall all be recorded in ANNUN_Vx.AL_PROG_DISP, until MODE_ENG_Vx.AUTOPILOT is no longer in AUTOLAND. If AL_PHASE_Vx = AUTOLAND_INOP or if MODE_ENG_Vx.AUTOPILOT = Not AUTOLAND, all components of ANNUN_Vx.AL_PROG_DISP shall be set to False.

7.1.4 Augmented Flying Qualities

The augmented flying qualities, as defined by FLY_QUAL_Vx, shall be displayed as output by ANNUN_Vx.FLY_QLTY exactly as furnished at the program unit input.

7.2 Update Conditions

Annunciator display updates shall be immediate, with a logic calculation iterations at a 20 Hz rate.

7.3 Maximum Allowable Time

The maximum allowable sub-frame for this computation shall be 2 milliseconds.

7.4 Input/Output

```
                              --------
                              |INPUTS|
                              --------

AL_PHASE_Vx          : AL_PROGRESS ;

type AL_PROGRESS  is  (AUTOLAND_ARMED, GLIDESLOPE_TRACK,
           DECISION_ALTITUDE, ALERT_ALTITUDE, FLARE, AUTOLAND_INOP) ;

FLY_QUAL_Vx          : FLYING_QUALITIES ;

type FLYING_QUALITIES  is (UNFLYABLE, MARGINAL, DEGRADED, NORMAL) ;

MODE_ENG_Vx          : AFCS_SELECTION ;

type AFCS_SELECTION is
     record
        AUTOPILOT     : AP_SELECTION ;
        AUTOLAND      : AL_CATEGORY ;
end record ;

type AP_SELECTION is (ALT_HOLD, AUTOLAND, BASIC, VERT_NAV, OFF) ;
type AL_CATEGORY is (CAT_1, CAT_2, CAT_3A, OFF );
```

```
                                  -----------
                                  | OUTPUTS |
                                  -----------


ANNUN_Vx            : ANNUN_STATUS ;

type ANNUN_STATUS is
    record
        AFCS_STATUS       : ENGAGE_STATUS ;
        AL_PROG_DISP      : CUM_AL_PROGRESS ;
        AUTOPILOT_MODE    : AP_SELECTION ;
        FLY_QLTY          : FLYING_QUALITIES ;
end record ;

type ENGAGE_STATUS is (AFCS_DISENGAGED, AUTOPLIOT_ENGAGED,
                       AUTOLAND_ENGAGED) ;

type CUM_AL_PROGRESS is array (1..5) of BOOLEAN ;

type AP_SELECTION is (ALT_HOLD, AUTOLAND, BASIC, VERT_NAV, OFF) ;

type FLYING_QUALITIES is (UNFLYABLE, MARGINAL, DEGRADED, NORMAL) ;
```

Ada Procedure GIVE_STATUS_Vx

```
with VOTING_PLANES ; use VOTING_PLANES ;
separate(DFCS_LOGIC)
procedure GIVE_STATUS_Vx is

        --  Local Declarations (if any)
        --  Declare Static Variables in User-Defined Package(s)

-----------------------------------------------------------------------

-- Using the inputs AL_PHASE_Vx, FLY_QUAL_Vx, and MODE_ENG_Vx,
-- compute the appropriate outputs to the Annunciator Displays,
-- ANNUN_Vx per the logic requirements in the English language
-- specification.

-----------------------------------------------------------------------

begin                           -- Procedure GIVE_STATUS_Vx

--
--
-- Add Demonstration Software Here
--
--

CHNL_x_XCHK_NUM  := 2 ;
XCHK_SYNCH_x ;

end GIVE_STATUS_Vx ;
```

8.0 MANAGE_AL_SENSOR PROCEDURE SPECIFICATION

Whenever the autopilot is engaged, the various sensors needed for automatic approach and landing shall be voted and compared to ensure the integrity of the signals used for autoland. Direct and cross-channel inputs shall be processed, and the results shall be placed in a record data structure. Logic states shall be maintained regarding both the internal and external status of the various sensor signals.

8.1 Sensor Signal Voting

Three separate autoland sensor signal votes shall be made on the input vectors each cycle: Glideslope Beam Deviation (GS_BEAM_DEV), Normal or Vertical Acceleration (NORM_ACCEL), and Radio Altitude (RAD_ALTITUDE). In each case a median output signal shall be generated and placed in a record, AL_MED_Vx. Where an even number of inputs is applied, the median shall be taken as the lesser of the two middle signal values.

8.1.1 Signal Ranges

The range of the respective input signals shall be defined in the derived type definitions in Package DFCS_RESOURCES, Figure 18c. Note that type conversions to Float may be necessary at some point.

8.1.2 Input Signal Validities

If the input validity signal associated with any input sensor signal, as reflected in the record AL_FLAGS, is False 5 consecutive iterations, the sensor signal shall be removed as an input to the corresponding voter. The associated signal comparator in AL_COMP_Vx shall then be set to False (tripped state) until 5 consecutive True values of the associated input validity flag signal are observed. The fault logic trip due to external signals flags shall be permitted to heal as many times as this logic is satisfied.

8.1.3 Signal Comparators

Each of the non-faulted input sensor signals shall be applied to a corresponding voter and shall be compared every iteration with the current median signal output of the voter. When the associated time and amplitude thresholds are simultaneously exceeded, the affected input signal shall be declared faulted in AL_COMP_Vx, and the signal shall be discontinued as an input to the voter. The associated fault logic shall latch, for no healing of comparison faulted sensor signals shall occur.

8.1.4 Amplitude Thresholds

The following absolute values of signal comparator differences (each voter input compared with the corresponding voter output) shall delineate out-of-tolerance input signals for the respective types of sensors:

Glideslope Beam Deviation	>=	0.05 degrees
Normal Acceleration	>=	0.025 g.
Radio Altitude	>=	2% of current median value of altitude

8.1.5 Time Thresholds

The following number of successive out-of-tolerance input sensor comparisons shall constitute the time thresholds for declaring a faulty input signal. Note that the comparisons, and hence each count, is only made every other call of this procedure.

Glideslope Beam Deviation	>=	5 counts
Normal Acceleration	>=	4 counts
Radio Altitude	>=	4 counts

8.2 Output Signals

The median output signals and the comparator state logic shall be available as data objects exported by Packages DFCS_RESOURCES and DFCS_LOGIC, respectively.

8.2.1 Median Output Signals

The median output signals, AL_MED_Vx, shall be a record of Type AL_SENSOR_SET.

8.2.2 Comparator State Output Signals

The comparator state output signals, AL_COMP_Vx, shall be a record of Type AL_SENSOR_STATUS. AL_COMP_Vx shall reflect the total effect of input sensor validity flags and the internal sensor comparator validities, i.e., the flag input for any sensor shall be OR-ed with the associated comparator validity to obtain the corresponding AL_COMP_Vx component value. The resultant states shall determine which input sensor signals are applied to the voters.

8.3 Program Structure Requirements

From a static standpoint, Procedure MANAGE_AL_SENSORS_Vx is incorporated into the program structure as shown in the call/usage graph in Figure 14. From a dynamic standpoint, the multirate executive control flow in Figure 11 depicts the invocation of MANAGE_AL_SENSORS_Vx.

8.3.1 Iteration Rate

As shown in Figure 14, the iteration rate for the autoland sensor processing is 10 Hz.

8.3.2 Maximum Allowable Computation Time

As indicated in Figure 15, the maximum allowable time for MANAGE_AL_SENSORS_Vx is 5 milliseconds.

8.4 Input/Output

```
                                  ----------
                                  | INPUTS |
                                  ----------

GS_BEAM_DEV       : GS_DEV_QUAD ;
type GS_DEV_QUAD     is array (1..4) of BEAM_DEV_SIGNAL ;
type BEAM_DEV_SIGNAL is new FLOAT range -2.5..2.5 ;

NORM_ACCEL       : N_ACCEL_TRIAD ;
type N_ACCEL_TRIAD  is array (1..3) of ACCEL_SIGNAL ;
type ACCEL_SIGNAL   is new FLOAT range -1.0..3.0 ;

RAD_ALTITUDE      : RAD_ALT_QUAD ;
type RAD_ALT_QUAD   is array (1..4) of RAD_ALT_SIGNAL ;
type RAD_ALT_SIGNAL is new FLOAT range -20.0..2500.0 ;

AL_FLAGS          : AL_SENSOR_STATUS ;

type AL_SENSOR_STATUS is
    record
        GS_BEAM_VAL      : QUAD_VALID ;
        N_ACCEL_VAL      : TRIAD_VALID ;
        RAD_ALT_VAL      : QUAD_VALID ;
    end record ;

                                  -----------
                                  | OUTPUTS |
                                  -----------

AL_COMP_Vx           : AL_SENSOR_STATUS ;

AL_MED_Vx            : AL_SENSOR_SET ;

type AL_SENSOR_SET is
    record
        GS_DEV        : BEAM_DEV_SIGNAL ;
        N_ACCEL       : ACCEL_SIGNAL ;
        RAD_ALT       : RAD_ALT_SIGNAL ;
    end record ;
```

Ada Procedure MANAGE_AL_SENSORS_Vx

```
with DFCS_LOGIC    ; use DFCS_LOGIC ;
with VOTING_PLANES ; use VOTING_PLANES ;
separate(DFCS_RESOURCES)
procedure MANAGE_AL_SENSORS_Vx is

        --  Local Declarations (if any)
        --  Place Static Variables in User-Defined Package(s)

-----------------------------------------------------------------------

-- Using the sets of autoland sensor inputs (GS_BEAM_DEV,
-- NORM_ACCEL, RAD_ALTITUDE), compute the respective median value
-- outputs for AL_MED_Vx, per the English text specification
-- requirements.

-- Do not vote an input signal if its associated validity flag,
-- AL_FLAGS(y), is False for a prescribed period. Then the indicated
-- fault should be reflected in the corresponding output comparator
-- logic, AL_COMP_Vx(y).

-- Compare each signal input with the associated median value, and
-- if out of specification tolerance, note a comparator trip in
-- AL_COMP_Vx(y).

-----------------------------------------------------------------------


begin                                   -- Procedure MANAGE_AL_SENSORS_Vx

--
--
-- Add Demonstration Software Here
--
--

CHNL_x_XCHK_NUM   := 3 ;
XCHK_SYNCH_x ;                                      -- Call for N-Version Vote

end MANAGE_AL_SENSORS_Vx ;
```

9.0 CALC_AUTOLAND PROCEDURE SPECIFICATION

Glideslope tracking and landing flare functions shall be provided as an orderly sequence of pitch axis sub-modes for automatic approach and landing under Category I, II, and IIIa weather conditions. Appropriate fault survivability capability will be provided based on fault logic external to this procedure. Depending on mode selection and component availability, autoland status annunciation outputs shall be generated for external display.

9.1 Control Laws

The glideslope and flare control laws shall be in accord with the analytical block diagram presented in Figure 21. Neither fixed point nor extended precision floating point arithmetic shall be used.

9.1.1 Signal Shaping

Digital filtering (as contrasted with numerical integration, for example) shall be used for the transfer functions. The sampling interval T shall be in accord with the iteration rate in paragraph 9.4.1. The Tustin transform may be used on the complex frequency operator, s, to obtain z, the complex delay operator as appears in digital filter equations:

$$s = \frac{2}{T}\frac{(z-1)}{(z+1)}$$

Since all of the filters are first-order, only the one previous input and output difference equation values must be saved. These saved values must be initialized, moreover, before mode engagement to preclude spurious transient steering commands. Specifically, high-pass filters (those with an s-operator in the numerator) must have their past input values set to input values present at engagement time, and their past output values set to zero. Low-pass filters (those with only a constant in the numerator) must have their outputs and saved values set to zero prior to engagement. The effect in both cases is to null filter outputs for the first computational cycle following mode engagement.

9.1.2 De-sensitization Schedule

The glideslope beam deviation signal shall be de-sensitized,or down-gained, as a function of decreasing radio altitude as shown in Figure 21 to offset the effects of beam convergence.

9.1.3 Glideslope Fader

Since some residual glideslope error signal may be present at flare engage, an exponential bleed-off signal fader shall be activated for Category II or IIIa autoland at flare engage, simultaneously with the switching in of the flare command signal per Figure 21. Category I approaches shall terminate at the Decision Altitude, and shall use this same fader to bleed off any residual command at this point.

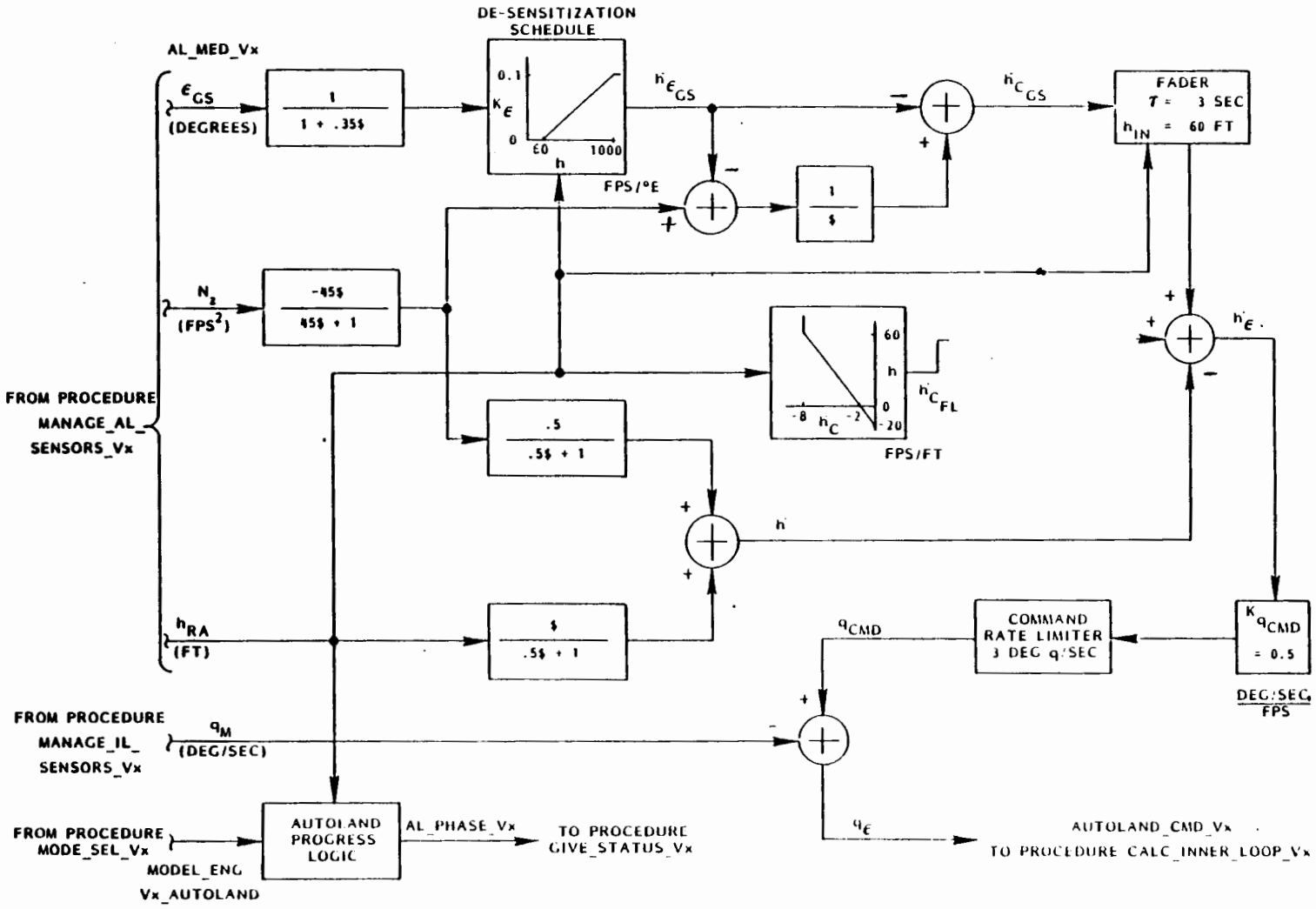


Figure 21 - Autoland Control Law Block Diagram

9.1.4 Flare Sink Rate Command

For Category II or IIIa autoland, an exponential flare path shall be generated upon descent to 60 feet of radio altitude in accordance with the altitude scheduling of the sink rate command as shown in Figure 21.

9.1.5 Altitude Rate Signal

An altitude rate signal shall be synthesized from normal acceleration and radio altitude as blended through complementary filtering as shown in Figure 21. This signal shall be summed with sink rate command to obtain sink rate error during both glideslope and flare modes.

9.1.6 Command Rate Limiting

Excursions of the sink rate error signal shall be limited by a command rate limiter per Figure 21 to preclude spurious or extreme flight path corrections.

9.1.7 Command Loop Closure

The autoland loop closure shall be effect through the summation of the sink rate command with pitch rate as shown in Figure 21. Pitch rate shall be obtained from IL_MED_Vx.P_RATE.

9.2 Mode Engagement Logic

Autoland mode engagement shall be effected via the logical signal MODE_ENG_Vx.AUTOPILOT = AUTOLAND, which reflects both pilot mode selections and component availability. The mode selection logic shall be used along with the radio altitude signal to activate control law sub-modes and to perform the autoland progress display logic computations.

9.2.1 Glideslope Mode Engagement

The glideslope mode shall be active in beam tracking mode for Categories II and IIIa autoland any time the radio altitude is above 60 feet. The radio altimeter level detector shall be included in Procedure CALC_AUTOLAND_Vx.

9.2.2 Flare Mode Engagement

The flare mode shall be engaged for either Category II or IIIa autoland when the radio altitude is below 60 feet.

9.2.3 Autoland Progress Display

The following logic conditions shall be observed in determining output logic states, AL_PHASE_Vx, for annunciation. Since it is an enumeration type data object, the assignments below are made upon satisfaction, and remain only until another condition is fulfilled, or until the Autoland Mode is reset.

AUTOLAND_ARMED --> Category II or IIIa Engaged

GLIDESLOPE_TRACK --> Glideslope Mode Engaged (presumed IC)

DECISION_ALTITUDE --> Category I Engaged * (h <= 200 ft.) or
Category II Engaged * (h <= 150 ft.)

ALERT_ALTITUDE --> Category IIIa * (h <= 100 ft.)

FLARE --> (Category II or Category IIIa) * (h <= 60 ft.)

AUTOLAND_INOP --> Category II and IIIa Engage Logic Lost During
Approach or Landing or when Autoland
De-Selected

9.3 Signal Interfaces

All sensor input signals will have been voted prior to receipt by Procedure CALC_AUTOLAND_Vx to eliminate discrepant inputs due to hardware faults.

9.3.1 Signal Inputs

All sensor inputs are derived types with constraints as follows. Type conversions to Float are therefore needed. Unit conversion from g's to feet per second squared for normal acceleration are also needed.

- o Glideslope Beam Deviation: +/- 2.5 degrees
- o Normal Acceleration: - 1.0/+3.0 g's
- o Radio Altimeter: - 20/+2500 feet
- o Pitch Rate: +/- 25 deg/sec

9.3.2 Logic Inputs

The logic inputs MODE_ENG_Vx is a record of enumeration types.

9.3.3 Pitch Command Output

The output steering command, AUTOLAND_CMD_Vx, is a derived type with a range constraint of -6.0/+3.0 degrees per second. A type conversion from Float is therefore needed for this output.

9.3.4 Logic Output

The logic output, AL_PHASE_Vx, is an enumeration type.

9.4 Program Structure

From a static standpoint, CALC_AUTOLAND_Vx is incorporated into the program structure as shown in the call/usage graph in Figure 17; from a dynamic standpoint, the multirate executive structure in Figure 14 depicts CALC_AUTOLAND_Vx's invocation.

9.4.1 Iteration Rate

As evident in Figure 14, the iteration rate for the autoland calculations is 10 Hz.

9.4.2 Maximum Allowable Computation Time

As indicated in Figure 15, the maximum allowable computation time for CALC_AUTOLAND_Vx is 4 milliseconds

9.5 Input/Output

```
                              | INPUTS |
                              ----------

MODE_ENG_Vx            : AFCS_SELECTION ;

type AFCS_SELECTION is
    record
        AUTOPILOT      : AP_SELECTION ;
        AUTOLAND       : AL_CATEGORY ;
    end record ;

type AP_SELECTION is (ALT_HOLD, AUTOLAND, BASIC, VERT_NAV, OFF) ;
type AL_CATEGORY  is (CAT_1, CAT_2, CAT_3A, OFF) ;

AL_MED_Vx              : AL_SENSOR_SET ;

type AL_SENSOR_SET is
    record
        GS_DEV         : BEAM_DEV_SIGNAL ;
        N_ACCEL        : ACCEL_SIGNAL ;
        RAD_ALT        : RAD_ALT_SIGNAL ;
    end record ;

type BEAM_DEV_SIGNAL is new FLOAT range -2.5..2.5 ;
type ACCEL_SIGNAL    is new FLOAT range -1.0..3.0 ;
type RAD_ALTSIGNAL   is new FLOAT range -20.0..2500.0 ;
```

```
IL_MED_Vx              : IL_SENSOR_SET ;

type IL_SENSOR_SET is
    record
                o
                o
        P_RATE                  : ANG_RATE_SIGNAL ;    (only component needed)
                o
    end record ;
```

OUTPUTS

```
AUTOLAND_CMD_Vx      : PITCH_COMMAND ;

type PITCH_COMMAND is new FLOAT -5.0..10.0 ;

AL_PHASE_Vx          : AL_PROGRESS ;

type        AL_PROGRESS        is        (AUTOLAND_ARMED,        GLIDESLOPE_TRACK,
DECISION_ALTITUDE, ALERT_ALTITUDE, FLARE, AUTOLAND_INOP) ;
```

Ada Procedure CALC_AUTOLAND_Vx

```
with DFCS_LOGIC ; use DFCS_LOGIC ;
with DFCS_RESOURCES ; use DFCS_RESOURCES ;
with VOTING_PLANES ; use VOTING_PLANES ;
separate(CONTROL_LAWS)
procedure CALC_AUTOLAND_Vx is

        --  Local Declarations (if any)
        --  Place Static Variables in User-Defined Package(s)

-----------------------------------------------------------------------

-- Conditional upon proper mode logic input, MODE_ENG_Vx, calculate
-- the pitch axis autoland command, AUTOLAND_COMMAND_Vx, using the
-- sensor inputs, AL_MED_Vx and IL_MED_Vx.P_RATE.

-- autoland is engaged, compute the progress display outputs,
-- AL_PHASE_Vx, as well.

-----------------------------------------------------------------------

begin

--
--
-- Add Demonstration Software Here
--
--

CHNL_x_XCHK_NUM  := 4 ;
XCHK_SYNCH_x ;

end CALC_AUTOLAND_Vx ;
```

10.0 MANAGE_IL_SENSORS PROCEDURE SPECIFICATION

During all foreground executive program execution, the inner loop sensor and command input signals shall be voted and compared to ensure the integrity of the signals used for DFCS functions. Direct and cross- channel inputs shall be processed, and the results placed in appropriate record data structures. Logic states shall be maintained regarding the status of the various input signal sources.

10.1 Sensor Signal Voting

Five separate inner loop sensor signal votes shall be made on the input vectors: Pilot's Stick Command (P_STICK_CMD), Copilot's Stick Command (CP_STICK_CMD), Average Angle-of-Attack (to be named), True Airspeed (TRUE_AIRSPEED), and Pitch Rate (P_RATE_GYRO). In each case a median output signal shall be generated and placed in a record, IL_MED_Vx. Where there are an even number of inputs applied, the median shall be taken as the lesser of the two middle value signals.

10.1.1 Signal Ranges

The range of the respective input signals are defined in Section 10.4. Since these signals are of derived types, type conversion to Float type may be necessary for calculation purposes.

10.1.2 Input Signal Validities

If the input validity flag signals furnished by the repsective sensors, per IL_FLAGS, is False 5 consecutive iterations, the sensor signal shall be removed as an input to the corresponding voter. The associated signal comparator output, IL_COMP_Vx, shall then be set to False (tripped state). Following a particular logic trip, 5 consecutive True inputs per IL_FLAGS shall reset the corresponding IL_COMP_Vx state.

10.1.3 Angle-of-Attack Inputs

Each corresponding left and right angle-of-attack signal pair shall be averaged prior to being voted, as illustrated in Figure 7.

10.1.4 Signal Comparators

Each of the input signals applied to a particular voter shall be compared each iteration with the current median signal output. When the associated time and amplitude thresholds are simultaneously exceeded, the affected input signal shall be declared faulted in IL_COMP_Vx, and it shall be permanently discontinued as an input to the voter.

10.1.5 Amplitude Thresholds

The following absolute values of signal comparator differences (between the median value and that of each voter input) shall delineate out-of-tolerance input signals for the respective types of signals:

Pilot's/Copilot's Stick Command	>= 0.2 degrees
Angle-of-Attack	>= 1.25 degrees
True Airspeed	>= 10 knots
Pitch Rate	>= 1.0 degrees/second.

10.1.6 Time Thresholds

The following number of consecutive out-of-tolerance amplitude comparisons shall constitute the time thresholds for declaring a faulty input signal:

Pilot's/Copilot's Stick Command	>= 6 counts
Angle-of-Attack & Pitch Rate	>= 8 counts
True Airspeed	>= 16 counts

10.2 Output Signals

The median output signals and the sensor status logic shall be available as data objects exported by Packages DFCS_RESOURCES and DFCS_LOGIC, respectively.

10.2.1 Median Output Signals

The median output signals, IL_MED_Vx, shall be a record of Type IL_SENSOR_SET.

10.2.2 Sensor Status Output Signals

The sensor status signals, IL_COMP_Vx, shall be a record of Type IL_SENSOR_STATUS.

10.3 Program Structure Requirements

From a static standpoint, Procedure MANAGE_IL_SENSORS_Vx is incorporated into the program structure as shown in the call/usage graph in Figure 17. From a dynamic standpoint, the multirate executive structure in Figure 14 depicts the invocation of MANAGE_IL_SENSORS_Vx.

10.3.1 Iteration Rate

As shown in Figure 14, the iteration rate for the inner loop sensor processing shall be 20 Hz.

10.3.2 Maximum Allowable Computation Time

As indicated in Figure 15, the maximum allowable time for MANAGE_IL_SENSORS_Vx is 5 milliseconds.

10.4 Input/Output

```
                                  ----------
                                  | INPUTS |
                                  ----------

CP_STICK_CMD              : STICK_CMD_QUAD ;
LEFT_AOA                  : AOA_QUAD ;
P_RATE_GYRO               : RATE_GYRO_TRIAD ;
P_STICK_CMD               : STICK_CMD_QUAD ;
RIGHT_AOA                 : AOA_QUAD ;
TRUE_AIRSPEED             : TAS_PAIR ;

IL_FLAGS                  : IL_SENSOR_STATUS ;

type IL_SENSOR_STATUS is
    record
         AVG_AOA_VAL      : QUAD_VALIDITY ;
         CP_STK_VAL       : QUAD_VALIDITY ;
         LF_AOA_VAL       : QUAD_VALIDITY ;
         P_STK_VAL        : QUAD_VALIDITY ;
         P_RATE_VAL       : TRIAD_VALIDITY ;
         RT_AOA_VAL       : QUAD_VALIDITY ;
         TAS_VAL          : PAIR_VALIDITY ;
    end record ;


                                  -----------
                                  | OUTPUTS |
                                  -----------

IL_MED_Vx                 : IL_SENSOR_SET ;

type IL_SENSOR_SET is
    record
        AOA_DISPL         : AOA_SIGNAL ;
        CP_STICK          : STICK_CMD ;
        P_RATE            : ANG_RATE_SIGNAL ;
        P_STICK           : STICK_CMD ;
        TR_AIRSPEED       : TAS_SIGNAL ;
    end record ;

type ANG_RATE_SIGNAL is new FLOAT range -25.0..25.0 ;  -- deg/sec
type AOA_SIGNAL      is new FLOAT range -10.0..50.0 ;  -- degrees
type STICK_CMD       is new FLOAT range -1.5..0.5 ;    -- degrees
type TAS_SIGNAL      is new FLOAT range 100.0..600.0 ; -- knots

IL_COMP_Vx                    : IL_SENSOR_STATUS ;
```

Ada Procedure MANAGE_IL_SENSORS_Vx

```
with DECS_LOGIC      ; use DFCS_LOGIC ;
with VOTING_PLANES   ; use VOTING_PLANES ;
separate(DFCS_RESOURCES)
procedure MANAGE_IL_SENSORS_Vx is

        --  Local Declarations (if any)
        --  Place Static Variables in User-Defined Package(s)

-----------------------------------------------------------------------

-- Using the Voter/Comparator Inputs (CP_STICK_CMD, LEFT_AOA,
-- P_RATE_GYRO, P_STICK_CMD, RIGHT_AOA, TRUE_AIRSPEED) compute
-- the median value outputs, IL_MED_Vx, per the English test
-- specification requirements.

-- Do not vote an input signal if its associated validity flag,
-- IL_FLAGS(y), is False.  Then record a corresponding comparator
-- trip, IL_COMP_Vx(y).

-- Compare each voted input signal with the associated median
-- value, and if out of specification tolerance, not a comparator
-- trip in IL_COMP_Vx(y).

-----------------------------------------------------------------------


begin                                   -- Procedure MANAGE_IL_SENSORS_Vx

--
--
-- Add Demonstration Software Here
--
--

CHNL_x_XCHK_NUM  := 5 ;
XCHK_SYNCH_x ;                      -- Call for N-Version Vote

end MANAGE_IL_SENSORS_Vx ;
```

11.0 CALC_INNER_LOOP PROCEDURE SPECIFICATION

A pitch inner loop stability augmentation control law shall be provided to improve the inherent flying qualities of the aircraft. Since a negative static stability margin is assumed for the aircraft, the pitch stability function shall be regarded as critical. Double fail-operational redundancy is therefore inherent in the design, with graceful degradation of performance under most multiple fault conditions.

11.1 Control Law

The pitch stability augmentation control law shall be in accord with the analytical block diagram presented in Figure 22. No extended precision arithmetic shall be used.

11.1.1 Signal Shaping

Digital filtering shall be used (as contrasted with numerical integration, for example) for dynamic signal shaping. The sampling interval T shall be in accord with the iteration rate in Paragraph 11.4.1. The Tustin transform may be used on the complex frequency operator, s, to obtain z, the complex delay operator as appears in digital filter equations:

$$s = \frac{2\,(z-1)}{T\,(z+1)}$$

11.1.2 Gain Scheduling

Sensor signal gains shall be scheduled as a function of true airspeed in accord with Figure 22. In the event that the true airspeed signal is questionable, i.e., if both components of IL_COMP_Vx.TAS_VAL are not valid, all gains shall revert to their lowest scheduled values.

11.1.3 Outer Loop Command Summation

When externally selected, via MODE_ENG_Vx.AUTOPILOT = AUTOLAND, an outer loop pitch servo command, AUTOLAND_CMD_Vx, shall be summed with the inner loop command as shown in Figure 22.

11.1.4 Command Limiting

The summation of the inner and outer loop servo commands shall be limited to -8.0, +1.0 degree of stabilizer displacement.

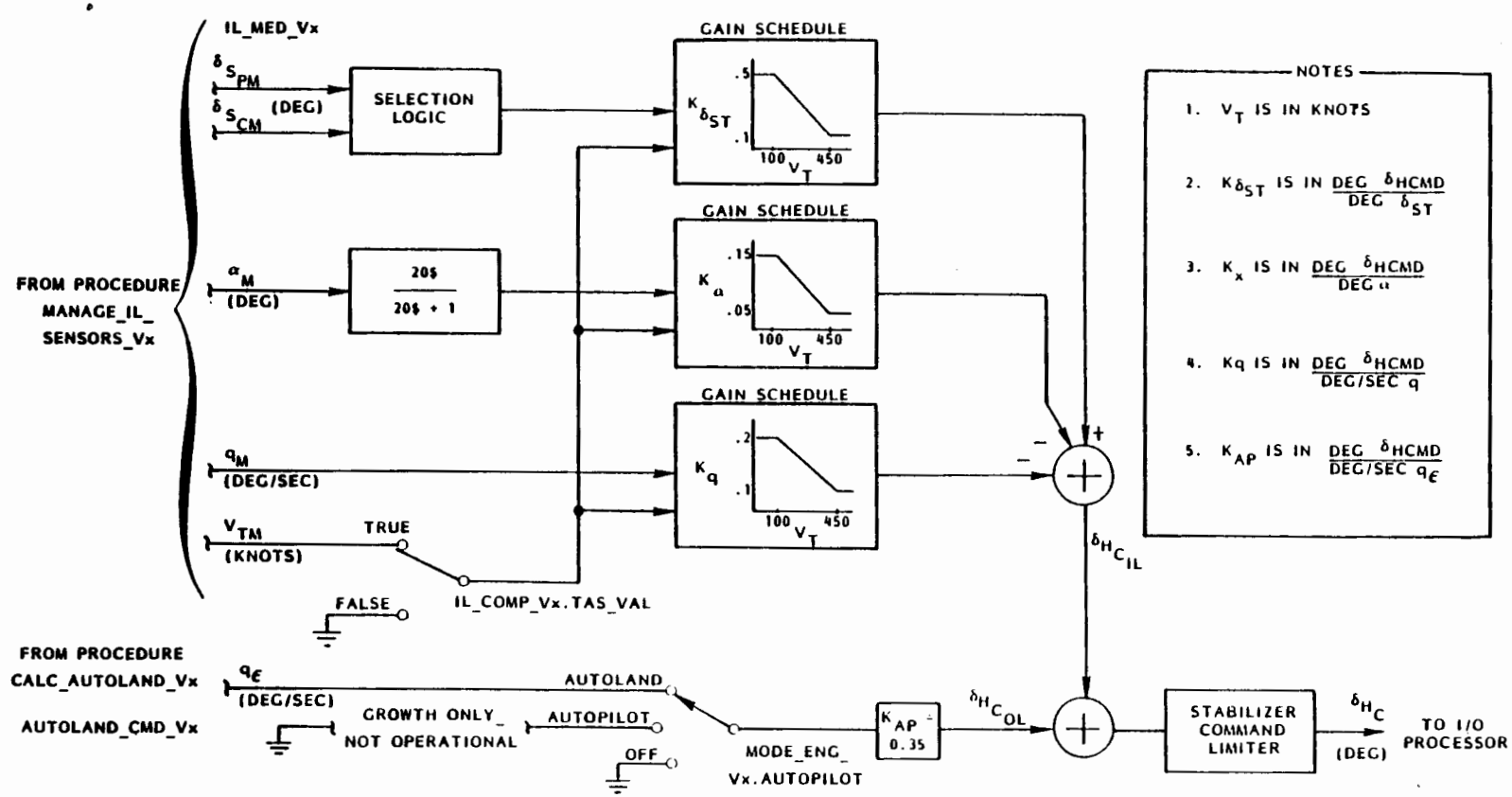


Figure 22 - Inner Loop Control Law Block Diagram

11.2 Activation Logic

The inner loop control law shall be engaged at all times, but it may be altered externally due to sensor resource depletion, which can cause a median sensor input(s) to clamp to zero.

11.2.1 Mode Engagement

The basic stability augmentation function shall be activated as a function of aircraft electrical power on, provided the respective DFCS channels are able to commence cycling in the foreground executive program (see Section 4.0). During the first pass through the control law following power application or resumption, the high-pass filter for angle-of-attack shall be initialized to set its output to zero (past difference equation output to zero, and past input value to present input value). This initialization precludes an engagement transient.

11.2.2 Stick Command Blending

Each of the pilots' stick command inputs shall be passed through a +/- 0.05 degrees of stabilizer command deadband; and then they shall be summed to obtain an averaged input value. The resultant command shall then be limited to an +/- 12.5 degrees of stick command.

11.2.3 True Airspeed Validity

The true airspeed validity signal, IL_COMP_Vx.TAS_VAL, shall be used to determine that the true airspeed signal is acceptable for use in gain scheduling. Both validity signals must be True.

11.3 Signal Interfaces

All input signals, with the possible exception of the outer loop command will have been voted prior to receipt by CALC_INNER_LOOP_Vx to eliminate discrepant inputs due to hardware faults.

11.3.1 Sensor Inputs

All sensor inputs are of derived types. Consequently, type conversion to Float shall be performed where necessary, e.g., prior to signal unit conversions.

11.3.2 Steering Command Input

The outer loop steering command input signal, AUTOLAND_CMD_Vx is incremental about the stabilizer trim position (which is irrelevant to the implementation of this procedure). Since it is a derived type, it shall be converted to Float type for control law computation.

11.3.3 Logic Inputs

The logic inputs, IL_COMP_Vx.TAS_VAL and MODE_ENG_Vx.AUTOPILOT, are a Boolean vector and a record enumeration types, respectively.

11.3.4 Servo Command Output

The stabilizer servo command output signal shall be converted from a Float type to the derived type, STAB_COMMAND, with a range constraint of -11.0/+2.0 degrees.

11.4 Program Structure Requirements

From a static standpoint, CALC_INNER_LOOP_Vx is incorporated into the program structure as shown in the call/usage graph in Figure 14; from a dynamic standpoint, the multirate executive structure in Figure 14 depicts CALC_INNER_LOOP_Vx's invocation.

11.4.1 Iteration Rate

As evident in Figure 14, the iteration rate for the inner loop control law is 20 Hz.

11.4.2 Maximum Computation Time

As indicated in Figure 15, the maximum allowable computation time for CALC_INNER_LOOP_Vx is 6 milliseconds.

11.5 Input/Output

```
                              ----------
                              | INPUTS |
                              ----------

IL_MED_Vx          : IL_SENSOR_SET ;

type IL_SENSOR_SET is
    record
        AOA_DISPL       : AOA_SIGNAL ;
        CP_STICK        : STICK_CMD ;
        P_RATE          : ANG_RATE_SIGNAL ;
        P_STICK         : STICK_CMD ;
        TR_AIRSPEED     : TAS_SIGNAL ;
end record ;

type ANG_RATE_SIGNAL is new FLOAT range -25.0..25.0 ;  -- deg/sec
type AOA_SIGNAL      is new FLOAT range -10.0..50.0 ;  -- degrees
type STICK_CMD       is new FLOAT range -1.5..0.5 ;    -- degrees
type TAS_SIGNAL      is new FLOAT range 100.0..600.0 ; -- knots
```

```
AUTOLAND_CMD_V3                 : PITCH_COMMAND ;

type PITCH_COMMAND is new FLOAT range -3..6 ;


MODE_ENG_Vx                     : AFCS_SELECTION ;

type AFCS_SELECTION is
    record
        AUTOPILOT               : AP_SELECTION ;
        AUTOLAND                : AL_CATEGORY ;
    end record ;

type AP_SELECTION is (ALT_HOLD, AUTOLAND, BASIC, VERT_NAV, OFF) ;

type AL_CATEGORY  is (CAT_1, CAT_2, CAT_3A, OFF) ;

IL_COMP_Vx.TAS_VAL                        : PAIR_VALIDITY ;

OUTPUTS :

STAB_SERVO_CMD_Vx               : STAB_COMMAND ;

type STAB_COMMAND is new FLOAT range -11.0..2.0 ;
```

Ada Procedure CALC_INNER_LOOP_Vx

```
with DFCS_LOGIC          ; use DFCS_LOGIC ;
with DFCS_RESOURCES      ; use DFCS_RESOURCES ;
with VOTING_PLANES       ; use VOTING_PLANES ;
separate(CONTROL_LAWS)
procedure CALC_INNER_LOOP_Vx is

        --  Local Declarations (if any)
        --  Place Static Variables in User-Defined Package(s)

-----------------------------------------------------------------------

-- the inner loop control law commands are generated from the
-- input signals, IL_MED_Vx.  If an autopilot mode is selected
-- via MODE_ENG_Vx, the autopilot input command is summed with
-- the inner loop command.  The output in either case is
-- STAB_SERVO_CMD_Vx.

-----------------------------------------------------------------------

begin

--
--
-- Add Demonstration Software Here
--
--

CHNL_x_XCHK_NUM  := 6 ;
XCHK_SYNCH_x ;

end CALC_INNER_LOOP_Vx ;
```

12.0 ASSESS_SYSTEM PROCEDURE SPECIFICATION

The fault logic states of all channels shall be evaluated to ascertain the status of the total system with respect to the augmented flying qualities and the operational state of the system. Note that the operational state implies a lower bound on flying qualtities level, which can be exceeded for non-normal operational states. The system status logic should be consistent with that given in Figure 6 of DOT/FAA/CT-86/33, but the following requirements shall govern. None of the following logic shall latch; any such effect would result from latching of input logic signals upstream in the data flow.

12.1 Flying Qualities Status

The fault status of the augmented fly-by-wire (AFBW) sensors, IL_COMP_Vx, shall be evaluated to determine flying qualities status, FLY_QUAL_Vx. The following logic shall be implemented, where AOA denotes angle-of-attack, and TAS denotes true airspeed:

```
Normal Flying Qualities  --> Minimum of 2 Rate Gyros Valid
                         AND Minimum of 2 AOA Pairs Valid
                         AND Both True Airspeeds Valid
                         AND Minimum of 2 Associated Stick
                             Commands Valid.

Degraded Flying Qualities--> (Maximum of 1 Rate Gyro Valid
                         OR  Maximum of 1 TAS Valid)
                         AND Minimum of 2 AOA Pairs Valid
                         AND Minimum of 2 Associated Stick
                             Commands Valid.

Marginal Flying Qualities--> Minimum of 2 Rate Gyros Valid
                         AND Maximum of 1 AOA Pairs Valid
                         AND Minimum of 2 Associated Stick
                             Commands Valid.

Unflyable                --> Anything Else
```

12.2 Redundancy Status

The fault status of the augmented fly-by-wire (AFBW) sensors, IL_COMP_Vx, and the status of all four computational channels, CHNL_STATUS_Vxs, shall be evaluated to determine component redundancy for residual component availability purposes. The following logic shall be implemented:

```
      Operational State 1  -->     All Rate Gyros Valid
 (Double Fail Operational)     AND All TAS Valid
                               AND All AOA Pairs Valid
                               AND [All of One Set of Stick
                                    Commands Valid
                               OR  Minimum of 3 of Both Sets of
                                    Stick Commands Valid]
                               AND All Computer Channels Valid.
```

```
    Operational State 2  -->     All Rate Gyros Valid
(Single Fail Operational)    AND  All TAS Valid
                             AND
                       {(Exactly 3 AOA Pairs Valid
              AND  Minimum of 3 Computer Channels Valid
              AND  [Minimum of 3 of One Set of Stick Commands
                   Valid
              OR   Minimum of 2 of Both Stick Commands Valid])

              OR   (Minimum of 3 AOA Pairs Valid
              AND  Exactly 3 Computer Channels Valid
              AND  [Minimum of 3 of One Set of Stick Commands
                   Valid
              OR   Minimum of 2 of Both Stick Commands Valid])

              OR   (Minimum of 3 AOA Pairs Valid
              AND  Minimum of 3 Computer Channels Valid
              AND  [Exactly 3 of One Set of Stick Commands
                   Valid
              AND  Maximum of 1 of Other Set of Stick Commands
                   Valid
              OR   Exactly 2 of Both Sets of Stick Commands
                   Valid]))}

    Operational State 3  -->
       (Fail Unsafe)

                   (Exactly 2 AOA Pairs Valid
              AND  Minimum of 2 Computer Channels Valid
              AND  Minimum of 2 of One Set of Stick Commands
                   Valid)

              OR   (Minimum of 2 AOA Pairs Valid
              AND  Exactly 2 Computer Channels Valid
              AND  Minimum of 2 of One Set of Stick Commands
                   Valid)

              OR   (Minimum of 2 AOA Pairs Valid
              AND  Minimum of 2 Computer Channels Valid
              AND  Exactly 2 of One Set of Stick Commands Valid
              AND  Maximum of 1 of Other Set of Stick Commands
                   Valid)

    Operational State 4  -->     Maximum of 1 AOA Pair Valid
    (Effectively Depleted)    OR Maximum of 1 Computer Channel Valid
                              OR Maximum of 1 Stick Command Valid
```

12.3 Maximum Allowable Computation Time

The maximum allowable sub-frame time for this computation shall be 3 milliseconds.

12.4 Input/Output

```
                                  ----------
                                  | INPUTS |
                                  ----------

CHNL_STATUS_V1, CHNL_STATUS_V2,
CHNL_STATUS_V3, CHNL_STATUS_V4        : BOOLEAN ;

IL_COMP_Vx                        : IL_SENSOR_STATUS ;

type IL_SENSOR_STATUS is
    record
        AVG_AOA_VAL       : QUAD_VALIDITY ;
        CP_STK_VAL        : QUAD_VALIDITY ;
        LF_AOA_VAL        : QUAD_VALIDITY ;
        P_STK_VAL         : QUAD_VALIDITY ;
        P_RATE_VAL        : TRIAD_VALIDITY ;
        RT_AOA_VAL        : QUAD_VALIDITY ;
        TAS_VAL           : PAIR_VALIDITY ;
end record ;

                                  -----------
                                  | OUTPUTS |
                                  -----------

FLY_QUAL_Vx       : FLYING_QUALITIES ;

type FLYING_QUALTITIES is (UNFLYABLE, MARGINAL, DEGRADED, NORMAL) ;

FBW_STATUS_Vx     : PRI_FCS_STATUS ;

type PRI_FCS_STATUS     is (OP_STATE_4, OP_STATE_3, OP_STATE_2,
                            OP_STATE_1) ;
```

Ada Procedure ASSESS_SYSTEM_Vx

```
with VOTING_PLANES  ; use VOTING_PLANES ;
separate(DFCS_LOGIC)
procedure ASSESS_SYSTEM_Vx is

        --  Local Declarations (if any)
        --  Place Static Variables in User-Defined Package(s)

-----------------------------------------------------------------------

-- Using fault logic inputs CHNL_V1, CHNL_V2, CHNL_V3, and CHNL_V4
-- along with IL_COMP_Vx, compute the system states, FLY_QUAL_Vx and
-- FBW_STATUS_Vx, per the logic requirements in the English
-- language part of the specification.

-----------------------------------------------------------------------


begin                                   -- Procedure ASSESS_SYSTEM_Vx

--
--
-- Add Demonstration Software Here
--
--

CHNL_x_XCHK_NUM  := 7 ;
XCHK_SYNCH_x  ;                         -- Call for N-Version Vote

end ASSESS_SYSTEM_Vx ;
```

13.0 GIVE_WARNING PROCEDURE SPECIFICATION

Warning display output signals shall be generated based on internal mode and fault logic variables to indicate control function status and availability. Information shall be displayed only when appropriate to inform the flight crew; this corresponds to warning logic conditions other than "BLANK."

13.1 Autoland Status

The autoland status output, WARN_Vx.AUTOLAND, shall directly reflect the logic input signal, AL_WARN_Vx, for both are of the same type.

13.2 Augmented Fly-By-Wire (AFBW) Status

The AFBW status output, WARN_Vx.FLY_BY_WIRE, shall reflect the logic input signal, FBW_STATUS_Vx, with the input state OP_STATE_1 mapping to BLANK.

13.3 Flying Qualities Status

Flying Qualities status, WARN_Vx.FLYING_QUAL, shall reflect the input logic signal, FLY_QUAL_Vx, with the following correspondences:

IMPAIRED_FQ	--> Degraded Flying Qualities OR Marginal Flying Qualtities OR Unflyable Flying Qualities.
BLANK	--> Normal Flying Qualities.

13.4 Master Warning Indicator

Each time a new warning state is first annunciated, a master warning signal, FLASH_WARNING_Vx, shall be set to BLINKING. When acknowledged by an externally applied Boolean variable ACKNOWLEDGE being momentarily set to True, FLASH_WARNING_Vx shall be set to STEADY, where it shall remain until a new warning is generated, or all prior warnings are terminated via the input logic to this procedure. When no warnings exist, FLASH_WARNING_Vx shall be set to OFF.

13.5 Maximum Allowable Computational Time

The maximum allowable sub-frame time for this computation shall be 2 milliseconds.

13.6 Input/Output

```
                              ----------
                              | INPUTS |
                              ----------

AL_WARN_Vx          : AL_STATUS ;

type AL_STATUS is (CAT_2_INOP, CAT_3_INOP, OFF) ;

FBW_STATUS_Vx       : PRI_FCS_STATUS ;

type PRI_FCS_STATUS is (OP_STATE_4, OP_STATE_3, OP_STATE_2, OP_STATE_1) ;

FLY_QUAL_Vx         : FLYING_QUALTITIES ;

type FLYING_QUALTITIES is (UNFLYABLE, MARGINAL, DEGRADED, NORMAL) ;

ACKNOWLEDGE         : BOOLEAN ;


                              -----------
                              | OUTPUTS |
                              -----------

WARN_Vx             : WARNING_STATE ;

type WARNING_STATE is
    record
        AUTOLAND        : AL_STATUS ;
        FLY_BY_WIRE     : FBW_STATUS ;
        FLYING_QUAL     : FQ_STATUS ;
end record ;

type FBW_STATUS is (OP_STATE_4, OP_STATE_3, OP_STATE_2, BLANK) ;
type FQ_STATUS  is (IMPAIRED_FQ, BLANK) ;

FLASH_WARNING_Vx              : MASTER_WARN ;

type MASTER_WARN is (BLINKING, STEADY, OFF) ;
```

Ada Procedure GIVE_WARNING_Vx

```
with VOTING_PLANES ; use VOTING_PLANES ;
separate(DFCS_LOGIC)
procedure GIVE_WARNING_Vx is

        --  Local Declarations (if any)
        --  Place Static Variables in User-Defined Package(s)

-----------------------------------------------------------------------

-- Using the inputs AL_WARN_Vx, FBW_STATUS,Vx, and FLY_QUAL_Vx,
-- compute the appropriate outputs to the Warning Display,
-- WARN_Vx, and in turn, the Master Warning, FLASH_WARNING_Vx, per
-- the logic given in the English text part of the specification.
-- The Boolean input ACKNOWLEDGE should cause the Master Warning to
-- glow steadily, rather than continue flashing as should occur
-- at the onset of a new warning.

-----------------------------------------------------------------------

begin                                   -- Procedure GIVE_WARNING_Vx

--
--
-- Add Demonstration Software Here
--
--

CHNL_x_XCHK_NUM := 8 ;
XCHK_SYNCH_x ;                          -- Call for N-Version Note

end GIVE_WARNING_Vx ;
```

14.0 TEST HARNESS SET-UP

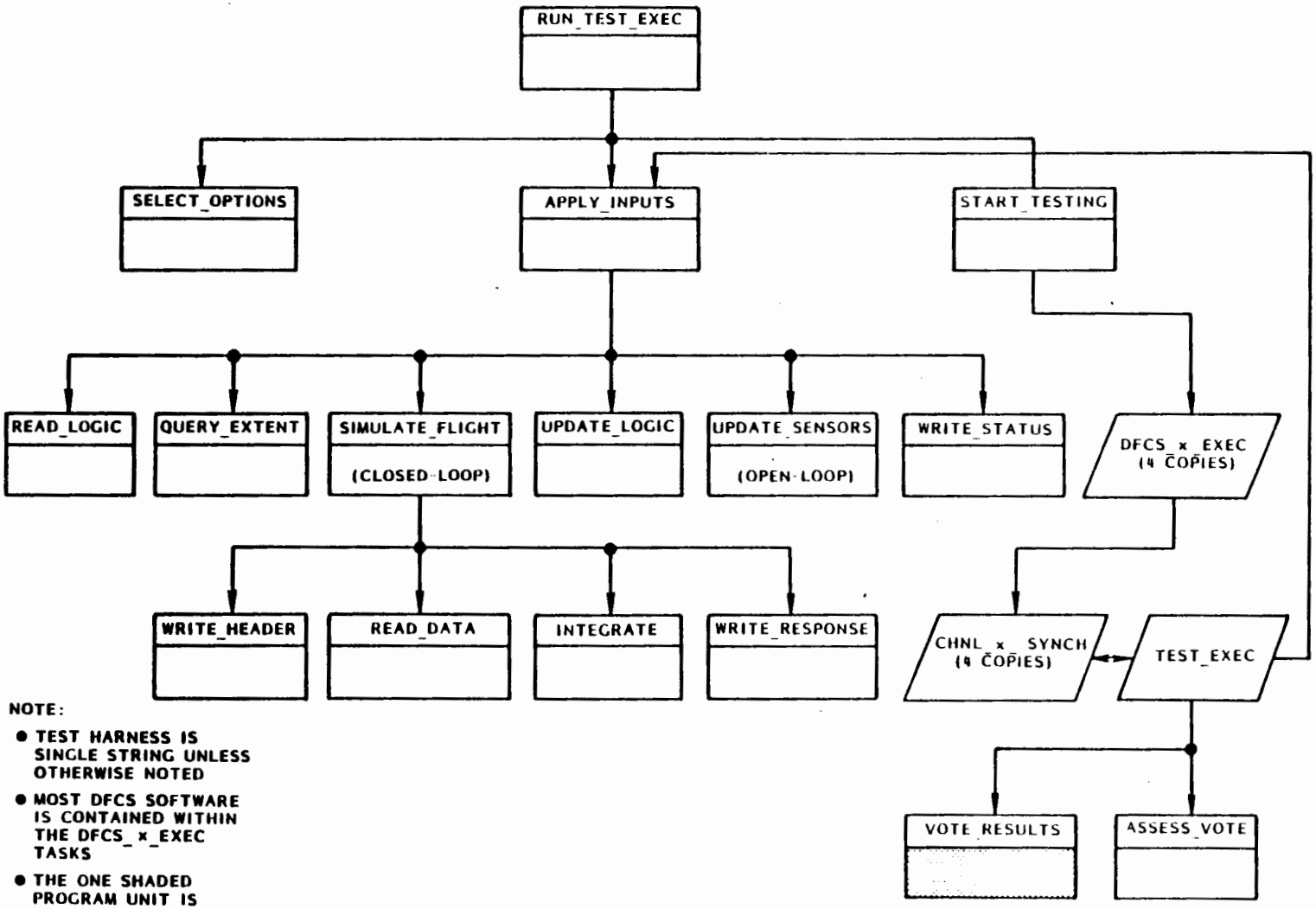
Although it was planned that the testing of the software fault-tolerant DFCS be done sequentially in non-realtime on a VAX computer, it was understood that the four versions of demonstration software would would normally reside in a quadruplex DFCS architecture. Hence, four parallel channels with double fail-operational capability were assumed, along with appropriate sensor/effector redundancy. Note, however, that the test harness software itself is mostly single string. The overall program organization to mechanize all this is shown in Figure 23; here only Tasks DFCS_x_EXEC and CHNL_x_SYNCH are replicated four times, because they interface with the four DFCS channels. All of the DFCS software, moreover, is effectively contained within Tasks DFCS_x_EXEC in the Figure 23 representation.

The test harness runs interactively on a non-realtime basis, with test cases applied through files readable by the test program. Considerable flexibility exists to expand the variety and extent of testing possible, but currently, the primary testing mode is customary airplane closed-loop simulation. The DFCS software is incorporated in the test harness as shown in Figure 24 for a typical channel. All of the program units shown belong to the DFCS except for the three shaded ones. As previously stated, the calling of Procedure RUN_FOREGROUND_x in the test harness is done by Task DFCS_x_EXEC in the test harness, rather than by Procedure RUN_DFCS_EXEC in the actual DFCS software load module. Also, Procedure VOTE_RESULTS is called by the test harness rather than by the DFCS software.

14.1 Test Harness Operation

At the outset of testing, the top-level program, Procedure RUN_TEST_EXEC, makes procedure calls to SELECT_OPTIONS and APPLY_INPUTS to initalize testing (see the listing in Figure 25a) based on prompted selections by the user. Following this Procedure START_TESTING (see the listing in Figure 25b) is invoked by RUN_TEST_EXEC, and actual testing ensues when entry is called to each of the four DFCS_x_EXEC tasks (see the body part listing for Package TEST_RESOURCES in Figure 25c). Normal testing then proceeds primarily under the control of Task TEST_EXEC (see the listing in Figure 25d). For each test cycle, it calls Procedure APPLY_INPUTS. As indicated in its source code in Figure 25e, this procedure can effect open or closed loop testing and faulted or fault free testing for a predefined number of cycles. Sensor and logic inputs can be altered independently.

Once a voted DFCS procedure called by Procedure RUN_FOREGROUND_x completes, it calls Procedure XCHK_SYNCH_x as listed in Figure 25f. These four DFCS procedures are the only ones modified whatsoever for test harness use. Basically, cross-channel voter synchronization would probably involve hardware oriented instruction that would be cumbersome to run on a general purpose computer. Furthermore, the effort would be difficult to justify for the type testing undertaken here. These procedures still perform the type conversions and voted value corrections as required in the DFCS application, but they make entry calls to test harness task, CHNL_x_SYNCH, as defined in Figure 25g.



NOTE:

- TEST HARNESS IS SINGLE STRING UNLESS OTHERWISE NOTED
- MOST DFCS SOFTWARE IS CONTAINED WITHIN THE DFCS_x_EXEC TASKS
- THE ONE SHADED PROGRAM UNIT IS PART OF THE DFCS SOFTWARE

Figure 23 - Overall Test Program Call Graph

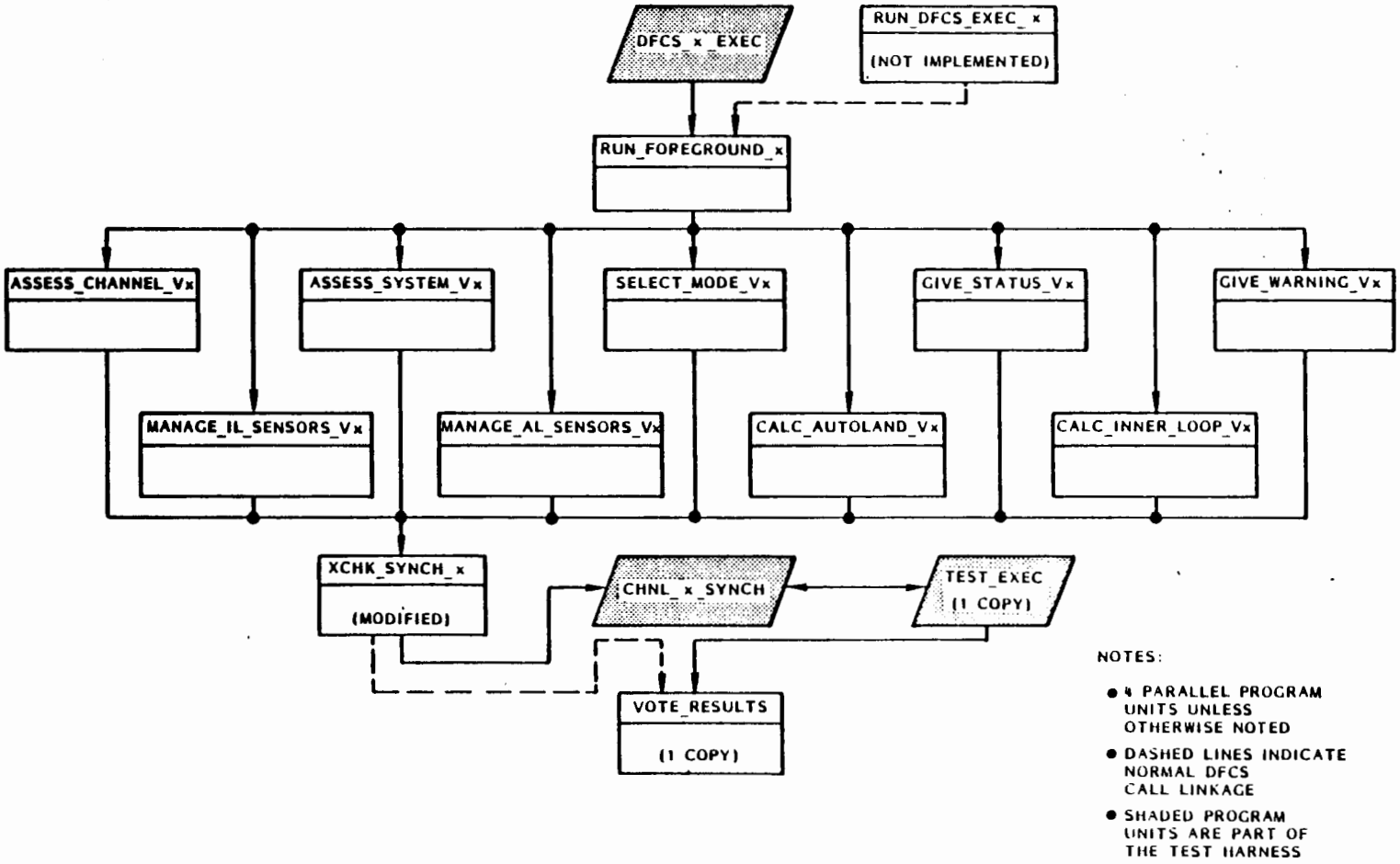


Figure 24 - DFCS Program Call Graph (In Test Harness)

```
with TEST_SET_UP          ; use TEST_SET_UP ;
procedure RUN_TEST_EXEC is

begin

SELECT_OPTIONS ;                    --  Prompts User for Desired Options
APPLY_INPUTS ;                      --  Called Here on Initial Setup Only
START_TESTING ;                     --  Activiation of DFCS Channel Tasks

end RUN_TEST_EXEC ;
```

Figure 25a - Test Harness Program Unit Listing
Procedure RUN_TEST_EXEC

```
with TEST_RESOURCES       ; use TEST_RESOURCES ;
separate (TEST_SET_UP)
procedure START_TESTING is
begin

        DFCS_1_EXECT.ENTARG ;                    --  Task Entry Calls to
        DFCS_2_EXECT.ENTARG ;                    --  the Respective DFCS
        DFCS_3_EXECT.ENTARG ;                    --  Channels to Initiate
        DFCS_4_EXECT.ENTARG ;                    --  their Execution

end START_TESTING ;
```

Figure 25b - Test Harness Program Unit Listing
Procedure START_TESTING

```
with TEST_SCENARIO        ; use TEST_SCENARIO ;
separate(TEST_SET_UP)
procedure APPLY_INPUTS is

begin
        if   BEGINNING
        then READ_LOGIC ;
             if   FLYING
             then SIMULATE_FLIGHT ;
             end if ;
             if   FAULTING
             then QUERY_EXTENT ;
             end if ;
             BEGINNING := FALSE ;
        elsif FLYING
        then SIMULATE_FLIGHT ;
        end if ;
        if   FAULTING and then CYCLES = SELECTION
        then QUERY_EXTENT ;
             UPDATE_LOGIC ;
             UPDATE_SENSORS ;
        end if ;
        WRITE_STATUS ;
        CYCLES := INTEGER'SUCC(CYCLES) ;
end APPLY_INPUTS ;
```

Figure 25c - Test Harness Program Unit Listing
Procedure APPLY_INPUTS

Use and/or disclosure is governed by the statement on the title page of this document.

```
with N_VERSION_VOTERS ;  use N_VERSION_VOTERS ;
with TEXT_IO ;           use TEXT_IO ;

package TEST_RESOURCES is

        RUNNING          : BOOLEAN := TRUE ;

                         -- Task Declarations --

--  As indicated in Figure 8, Procedure RUN_FOREGROUND_X is normally
--  called by Procedure RUN_DFCS_EXEC_X. For the subject test purposes,
--  Procedure RUN_FOREGROUND_X is called by Task DFCS_EXEC_X as shown in
--  Figure 20, because tasking permits all four channels to be active
--  simultaneously in a uniprocessor. Note that here the DFCS executive
--  calls only RUN_FOREGROUND_X because only the applications software
--  is to be executed.

--  Since it is an artifice to enable non-real-time testing, task
--  DFCS_EXEC_X is declared in this package. No entries are specified,
--  so all four DFCS channel tasks are queued to execute when the main
--  program, Procedure RUN_TEST_EXEC, begins to execute. In fact, one
--  of them will run immediately.

        task DFCS_1_EXEC is
                entry ENGAGE ;
        end DFCS_1_EXEC ;

        task DFCS_2_EXEC is
                entry ENGAGE ;
        end DFCS_2_EXEC ;

        task DFCS_3_EXEC is
                entry ENGAGE ;
        end DFCS_3_EXEC ;

        task DFCS_4_EXEC is
                entry ENGAGE ;
        end DFCS_4_EXEC ;
```

Figure 25d - Test Harness Program Unit Listing Package TEST_RESOURCES (1 of 3)

```
--  When any of the 4 DFCS channels reaches an n-version cross-check
--  point, it calls Procedure XCHK_SYNCH_x, which makes an entry call to
--  a corresponding secretary task and identifies its current voting
--  plane. As depicted in Figure 21, this entry is called to
--  CHNL_x_SYNCH, which is waiting to accept CHNL_x_READY. The
--  associated DFCS channel task is then suspended while all other DFCS
--  channels report. Each of the secretary tasks is suspended as well
--  after it calls entry to Task TEST_EXEC. This ordering ensures
--  that n-version voting does not occur until all 4 channels have
--  completed the associated code segment.

--  All 4 CHNL_x_SYNCH tasks remain suspended until TEST_EXEC has
--  completed DFCS segment timing, n-version voting, and test results
--  processing. TEST_EXEC then calls entry to all of the secretary
--  tasks, which in turn permit the DFCS channel tasks to execute the
--  next applications software segment.

        task CHNL_1_SYNCH is
                entry READY(CHNL_1_CC_PT : in CC_POINT) ;
                entry RESUME ;
        end  CHNL_1_SYNCH ;

        task CHNL_2_SYNCH is
                entry READY(CHNL_2_CC_PT : in CC_POINT) ;
                entry RESUME ;
        end  CHNL_2_SYNCH ;

        task CHNL_3_SYNCH is
                entry READY(CHNL_3_CC_PT : in CC_POINT) ;
                entry RESUME ;
        end  CHNL_3_SYNCH ;

        task CHNL_4_SYNCH is
                entry READY(CHNL_4_CC_PT : in CC_POINT) ;
                entry RESUME ;
        end  CHNL_4_SYNCH ;

--  In order to know which n-version voter(s) to call, task TEST_EXEC
--  has the voting plane designation, CHNL_x_CC_PT, passed to it from
--  each DFCS channel via the associated secretary task.

        task TEST_EXEC is
                entry CHNL_1_READY(CHNL_CC_PT : in CC_POINT) ;
                entry CHNL_2_READY(CHNL_CC_PT : in CC_POINT) ;
                entry CHNL_3_READY(CHNL_CC_PT : in CC_POINT) ;
                entry CHNL_4_READY(CHNL_CC_PT : in CC_POINT) ;
        end TEST_EXEC ;

end TEST_RESOURCES ;
```

Figure 25d - Test Harness Program Unit Listing Package TEST_RESOURCES (2 of 3)

```
with SYSTEM_RESOURCES ; use SYSTEM_RESOURCES ;
package body TEST_RESOURCES is
        task body CHNL_1_SYNCH is separate ;
        task body CHNL_2_SYNCH is separate ;
        task body CHNL_3_SYNCH is separate ;
        task body CHNL_4_SYNCH is separate ;
        task body TEST_EXEC is separate ;
        task body DECS_1_EXEC is
        begin
                accept ENGAGE
                do
                        PATH_1 := 1 ;
                        while RUNNING
                        loop
                                RUN_FOREGROUND_1 ;
                                PATH_1 := 1 + PATH_1 mod 4 ;
                        end loop ;
                end ENGAGE ;
        end DECS_1_EXEC ;
        task body DECS_2_EXEC is
        begin
                accept ENGAGE
                do
                        PATH_2 := 1 ;
                        while RUNNING
                        loop
                                RUN_FOREGROUND_2 ;
                                PATH_2 := 1 + PATH_2 mod 4 ;
                        end loop ;
                end ENGAGE ;
        end DECS_2_EXEC ;
        task body DECS_3_EXEC is
        begin
                accept ENGAGE
                do
                        PATH_3 := 1 ;
                        while RUNNING
                        loop
                                RUN_FOREGROUND_3 ;
                                PATH_3 := 1 + PATH_3 mod 4 ;
                        end loop ;
                end ENGAGE ;
        end DECS_1_EXEC ;
        task body DECS_4_EXEC is
        begin
                accept ENGAGE
                do
                        PATH_4 := 1 ;
                        while RUNNING
                        loop
                                RUN_FOREGROUND_4 ;
                                PATH_4 := 1 + PATH_4 mod 4 ;
                        end loop ;
                end ENGAGE ;
        end DECS_1_EXEC ;
end TEST_RESOURCES ;
```

Figure 25d - Test Harness Program Unit Listing Package TEST_RESOURCES (3 of 3)

```
with TEST_SCENARIO ; use TEST_SCENARIO ;
with TEST_SET_UP   ; use TEST_SET_UP ;
with TEST_IO       ; use TEST_IO ;
separate(TEST_RESOURCES)
task body TEST_EXEC is

        WRONG_PLANE                 : exception ;

        CC_SEQUENCE                 : constant array (1..20) of INTEGER
                          := (1,2,3,4,5,6,1,7,5,6,1,3,4,5,6,1,7,8,5,6) ;
        TOTAL_VOTE                  : constant CC_DATA
                          := ((TRUE, TRUE, TRUE, TRUE),
                              (TRUE, TRUE, TRUE, TRUE)) ;
        MAX_VOTES                   : constant INTEGER   := 20  ;

        CC_REF_PT                                   : CC_POINT ;
        CC_RESULTS                                  : CC_DATA ;
        CHNL_READY                  : array (1..4) of BOOLEAN ;
        NUM_LOOPS                                   : INTEGER := 0 ;
        NUM_VOTES                                   : INTEGER := 0 ;
        TIME_LIMIT                                  : INTEGER := 5 ;
        TIMEOUT_COUNTER                             : INTEGER ;
        VOTE_REF                                    : INTEGER := 0 ;
        VOTE_TIMEOUT                                : BOOLEAN ;
        XCHK_PT                     : array (1..4) of CC_POINT ;

begin

AUTO_TESTING ;
loop

        CHNL_READY        := (FALSE, FALSE, FALSE, FALSE) ;
        TIMEOUT_COUNTER   := 0 ;
        VOTE_TIMEOUT      := FALSE ;
        XCHK_PT           := (0, 0, 0, 0) ;

        VOTE_REF     := 1 + VOTE_REF mod 20 ;
        CC_REF_PT    := CC_SEQUENCE(VOTE_REF) ;
```

Figure 25e - Test Harness Program Unit Listings
Task Body TEST_EXEC (1 of 3)

```
SYNCHRONIZATION :
loop
        select
                accept  CHNL_1_READY(CHNL_CC_PT : in CC_POINT)
                        do
                        CHNL_READY(1) := TRUE ;
                        XCHK_PT(1)    := CHNL_CC_PT ;
                        if      XCHK_PT(1) /= CC_REF_PT then
                                raise   WRONG_PLANE ;
                        end if ;
                end     CHNL_1_READY ;
        or
                accept  CHNL_2_READY(CHNL_CC_PT : in CC_POINT)
                        do
                        CHNL_READY(2) := TRUE ;
                        XCHK_PT(2)    := CHNL_CC_PT ;
                        if      XCHK_PT(2) /= CC_REF_PT then
                                raise   WRONG_PLANE ;
                        end if ;
                end     CHNL_2_READY ;
        or
                accept  CHNL_3_READY(CHNL_CC_PT : in CC_POINT)
                        do
                        CHNL_READY(3) := TRUE ;
                        XCHK_PT(3)    := CHNL_CC_PT ;
                        if      XCHK_PT(3) /= CC_REF_PT then
                                raise   WRONG_PLANE ;
                        end if ;
                end     CHNL_3_READY ;
        or
                accept  CHNL_4_READY(CHNL_CC_PT : in CC_POINT)
                        do
                        CHNL_READY(4) := TRUE ;
                        XCHK_PT(4)    := CHNL_CC_PT ;
                        if      XCHK_PT(4) /= CC_REF_PT then
                                raise   WRONG_PLANE ;
                        end if ;
                end     CHNL_4_READY ;
        or
                delay   1.0 ;
                        TIMEOUT_COUNTER := TIMEOUT_COUNTER + 1 ;
        end select ;
        if TIMEOUT_COUNTER = TIME_LIMIT then
                VOTE_TIMEOUT := TRUE ;
                put_line("VOTE TIMEOUT") ;
        end if ;
        exit when CHNL_READY(1) and CHNL_READY(2) and
                CHNL_READY(3) and CHNL_READY(4) or VOTE_TIMEOUT ;
end loop SYNCHRONIZATION ;
```

Figure 25e - Test Harness Program Unit Listings
Task Body TEST_EXEC (2 of 3)

```
        if      VOTE_TIMEOUT then
                put_line("Abnormal Vote Forthcoming") ;
        else    put_line("Normal Vote Forthcoming") ;
        end if ;
        VOTE_RESULTS(CC_REF_PT, CC_RESULTS) ;
        if      CC_RESULTS /= IDEAL_VOTE then
                ASSESS_VOTE(CC_REF_PT, CC_RESULTS) ;
        end if ;
        NUM_VOTES := NUM_VOTES + 1 ;
        if      NUM_VOTES >= MAX_VOTES
        then    NUM_LOOPS := NUM_LOOPS + 1 ;
                NUM_VOTES := 0 ;
                if      NUM_LOOPS < MAX_LOOPS
                then    APPLY_INPUTS ;
                else    RUNNING := FALSE ;
                        CHNL_1_SYNCH.RESUME ;
                        CHNL_2_SYNCH.RESUME ;
                        CHNL_3_SYNCH.RESUME ;
                        CHNL_4_SYNCH.RESUME ;
                        exit AUTO_TESTING ;
                end if ;
        end if ;
                -- Moving to Next Voting Plane
        if CHNL_READY(1) then
                CHNL_1_SYNCH.RESUME ;
        end if ;
        if CHNL_READY(2) then
                CHNL_2_SYNCH.RESUME ;
        end if ;
        if CHNL_READY(3) then
                CHNL_3_SYNCH.RESUME ;
        end if ;
        if CHNL_READY(4) then
                CHNL_4_SYNCH.RESUME ;
        end if ;
end loop AUTO_TESTING ;
                -- Test of Foreground Complete
exception
        when WRONG_PLANE =>
                put_line("bad Synchronization") ;
end TEST_EXEC ;
```

Figure 25e - Test Harness Program Unit Listings
Task Body TEST_EXEC (3 of 3)

```
with CONTROL_LAWS          ; use CONTROL_LAWS ;
with DECS_LOGIC            ; use DECS_LOGIC ;
with DECS_RESOURCES        ; use DECS_RESOURCES ;
with TEST_RESOURCES        ; use TEST_RESOURCES ;
with TEXT_IO               ; use TEXT_IO ;
separate(VDITAG_PLANES)
procedure XCHK_SYNCH_1 is

begin

case CHNL_1_XCHK_NUM is
        when    0 =>    put_line("Illegal Voting Plane") ;
        when    1 =>    VD_MODE_ENG(1) := MODE_ENG_V1 ;
                        CHNL_1_SYNCH.READY(CHNL_1_XCHK_NUM) ;
                        MODE_ENG_V1 := VD_MODE_ENG(1) ;
        when    2 =>    VD_ANGUN(1) := ANGUN_V1 ;
                        CHNL_1_SYNCH.READY(CHNL_1_XCHK_NUM) ;
                        ANGUN_V1 := VD_ANGUN(1) ;
        when    3 =>    VD_AL_COMP(1) := AL_COMP_V1 ;
                        VD_AL_MED(1)  := AL_MED_V1 ;
                        CHNL_1_SYNCH.READY(CHNL_1_XCHK_NUM) ;
                        AL_COMP_V1 := VD_AL_COMP(1) ;
                        AL_MED_V1  := VD_AL_MED(1) ;
        when    4 =>    VD_AL_PHASE(1)      := AL_PHASE_V1 ;
                        VD_AUTOLAND_CMD(1) := AUTOLAND_CMD_V1 ;
                        CHNL_1_SYNCH.READY(CHNL_1_XCHK_NUM) ;
                        AL_PHASE_V1      := VD_AL_PHASE(1) ;
                        AUTOLAND_CMD_V1 := VD_AUTOLAND_CMD(1) ;
        when    5 =>    VD_IL_COMP(1) := IL_COMP_V1 ;
                        VD_IL_MED(1)  := IL_MED_V1 ;
                        CHNL_1_SYNCH.READY(CHNL_1_XCHK_NUM) ;
                        IL_COMP_V1 := VD_IL_COMP(1) ;
                        IL_MED_V1  := VD_IL_MED(1) ;
        when    6 =>    VD_STAB_SERVO_CMD(1) := STAB_SERVO_CMD_V1 ;
                        CHNL_1_SYNCH.READY(CHNL_1_XCHK_NUM) ;
                        STAB_SERVO_CMD_V1 := VD_STAB_SERVO_CMD(1) ;
        when    7 =>    VD_FBW_STATUS(1) := FBW_STATUS_V1 ;
                        VD_FLY_QUAL(1)   := FLY_QUAL_V1 ;
                        CHNL_1_SYNCH.READY(CHNL_1_XCHK_NUM) ;
                        FBW_STATUS_V1 := VD_FBW_STATUS(1) ;
                        FLY_QUAL_V1   := VD_FLY_QUAL(1) ;
        when    8 =>    VD_FLASH_WARNING(1) := FLASH_WARNING_V1 ;
                        VD_WARN(1)          := WARN_V1 ;
                        CHNL_1_SYNCH.READY(CHNL_1_XCHK_NUM) ;
                        FLASH_WARNING_V1 := VD_FLASH_WARNING(1) ;
                        WARN_V1          := VD_WARN(1) ;
end case ;

end XCHK_SYNCH_1 ;
```

Figure 25f - Test Harness Program Unit Listings
Procedure XCHK_SYNCH_1

```
separate(TEST_RESOURCES)
task body CHNL_1_SYNCH is

VOTE_PLANE        : CC_POINT ;

begin
while TEST_EXEC
loop

        VOTE_PLANE    := 0 ;

        accept  READY(CHNL_1_CC_PT : in CC_POINT) do
                VOTE_PLANE := CHNL_1_CC_PT ;
                TEST_EXEC.CHNL_1_READY(VOTE_PLANE) ;
                accept RESUME ;
        end     READY ;

end loop ;

end CHNL_1_SYNCH ;
```

Figure 25g - Test Harness Program Unit Listings
Task Body CHNL_1_SYNCH

14.2 N-Version Voter Synchronization

Including the top-level program for the N-version demonstration, Procedure RUN_TEST_EXEC, ten Ada tasks are active from the outset of program execution. These include the four test harness DFCS executive programs, Task DFCS_x_EXEC, four secretary tasks that regulate voting plane synchronization, Task CHNL_x_SYNCH, and the test coordinator, Task TEST_EXEC. The secretary tasks needed to effect a four-way synchronization using Ada inherently two-way rendezvous. These tasks are declared in Package TEST_RESOURCES per Figure 25c. Intertask communication as depicted in Figure 26 continues so long as the master control Boolean, RUNNING, is True.

Initially, entries to Tasks DFCS_x_EXEC are called from Procedure START_TESTING, namely, DFCS_x_EXEC.ENGAGE for each of the four channels. As each voted DFCS applications procedure completes, the associated Procedure XCHK_SYNCH_x calls entry to the corresponding secretary tasks with a CHNL_x_SYNCH.READY statement. When the CHNL_x_SYNCH accepts the entry call and relays it to Task TEST_EXEC, both DFCS_x_EXEC and CHNL_x_SYNCH are suspended. Then the other channel tasks are activated one by one until all have reported in to TEST_EXEC's timed select loop that accepts TEST_EXEC.CHNL_x_READY entry calls. After checking to ensure that all DFCS channels are at the correct voting plane, Task TEST_EXEC calls Procedure VOTE_RESULTS and analyzes and records the results.

TEST_EXEC then checks for additional test case selections. If so, it calls applies them and one by one releases DFCS channels for the next test cycle. This is done by a CHNL_x_SYNCH.RESUME entry call that completes two rendezvous and permits DFCS_x_EXEC to become active again. The next DFCS applications module in RUN_FOREGROUND_x is then executed, and the next voting plane is sought via a repeat of the four-way synchronization process. If Task TEST_EXEC determines that all test has been completed, it sets RUNNING to False and terminates. The rest of the tasks then terminate as well.

14.3 Closed-Loop Simulation

The closed-loop simulation set-up is depicted in Figure 27 in a state variable form that coincides with the external DFCS sensor/effector signal interfaces. The source code for the simulation is presented in Figure 28. Basically, it reads in flight case data from an interactively named file, trims the airplane under selected conditions, and commences to generate the array of inner and outer loop sensor signals based on the input STAB_SERVO_CMD_x. The output signals undergo data type and scaling changes as appropriate. Signal fan-out for multiple sensors and fault insertion faculties reside in Procedure UPDATE_SENSORS, which is also called by Procedure APPLY_INPUTS per Figure 23.

14.4 Software Development

During DFCS software versions, the test harness was modified for single channel use. Basically, this involved disabling all but one particular channels tasks, and tailoring input test data for limited scope or unit testing. Some data object visibility problems were encountered that necessitated selective raising of the variable namespace so that the test harness could import and access certain variables. Basically, the test

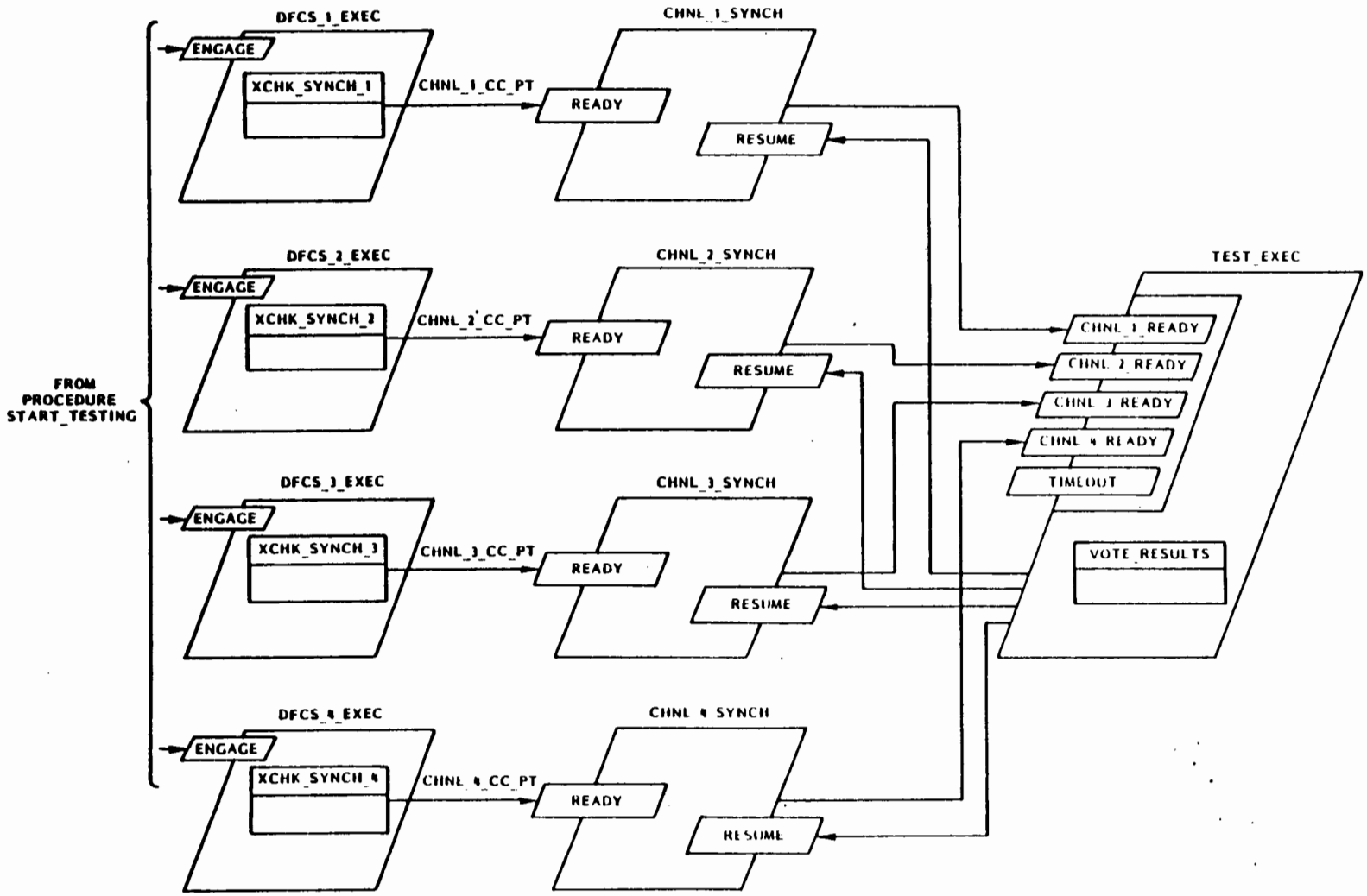


Figure 26 - Ada Multitasking Communication

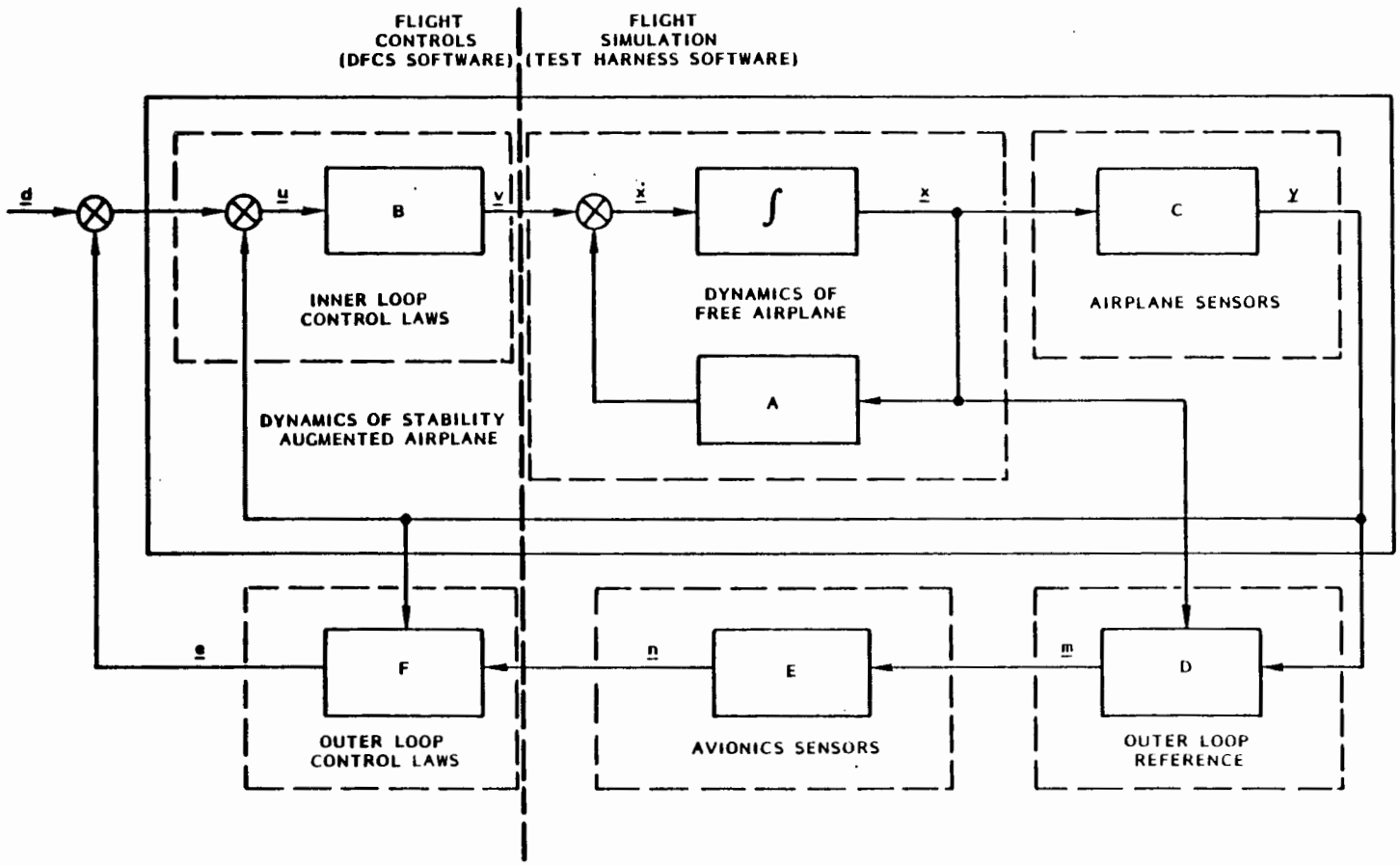


Figure 27 - Closed-Loop Simulation Block Diagram

```
with TEXT_IO              ; use TEXT_IO ;
with INTEGRATION          ; use INTEGRATION ;
with SIMULATION_DATA      ; use SIMULATION_DATA ;
separate(TEXT_IO.AMIO)
procedure SIMULATE_FLIGHT is

package SIMULATION_IO is new FLOAT_IO( NUM => FLOAT ) ;
use SIMULATION_IO ;

    RAD_CONV                : constant := 57.296 ;
    FPS_CONV                : constant := 1.6878 ;
    FPSS_CONV               : constant := 32.174 ;
    G_CONV                  : constant := 0.03108 ;
    FPSFPS_CONV             : constant := 0.59249 ;
    FPSG_CONV               : constant := 0.01745 ;

begin

-- Initialize simulation

if   FIRST
then READ_DATA ;
     GAMMA        := FLT_PATH*RAD_CONV ;
     X_DOT        := TAS + WIND*FPS_CONV ;
     OLD_X_DOT    := X_DOT ;
     OLD_DIST     := DISTANCE ;
     H_DOT        := X_DOT*GAMMA ;
     OLD_H_DOT    := H_DOT ;
     OLD_HEIGHT   := HEIGHT ;
     T_ACCEL      := 1.0 ;
     MASS         := WT*G_CONV ;
     QS           := 0.5*RHO*TAS*TAS*S ;
     CL           := WT/QS ;
     AOA          := (CL - CLA0)/CLA ;
     for TRIMMING in 1..10
          loop
               STAB_TRIM := -(CMA0 + CMA*AOA)/CMDH ;
               CL        := CLA0 + CLA*AOA + CLDH*STAB_TRIM ;
               CD        := CDA0 + CDA*AOA ;
               PITCH_ATT := AOA + GAMMA - WIND*GAMMA/TAS ;
               THRUST    := QS*CD - QS*CL*AOA + WT*PITCH_ATT ;
               CL        := (WT - QS*CD*AOA)/QS ;
               AOA       := (CL - CLA0 - CLDH*STAB_TRIM)/CLA ;
          end loop ;
     OLD_PITCH_ATT  := PITCH_ATT ;
     OLD_U_BODY_GRD := X_DOT ;
     OLD_V_BODY_GRD := TAS*AOA + WIND*FPS_CONV ;
     STAB_POS       := STAB_TRIM ;
     FIRST          := FALSE ;
     WRITE_HEADER ;
     if   P_DOT /= 0.0
     then PITCH_DOT       := P_DOT*RAD_CONV ;
          OLD_PITCH_DOT1 := PITCH_DOT ;
          OLD_PITCH_DOT2 := PITCH_DOT ;
     end if ;
```

Figure 28 - Procedure SIMULATE_FLIGHT Listing (1 of 3)

```
loop

-- Fix, airplane

    STAB_POS          := (FLOAT(STAB_SERVO_CMD_V3))*RAD_CONV + STAB_TRIM ;
    ALPHA_DOT         := W_DOT_BODY/TAS ;
    CM                := CMAO + CMA*AOA + CMDH*STAB_POS +
                         0.5*CBAR*(CMQ*PITCH_DOT + CMAD*ALPHA_DOT)/TAS ;
    CL                := CLAO + CLA*AOA + CLDH*STAB_POS ;
    CD                := CDAO + CDA*AOA ;
    QS                := 0.5*RHO*TAS*TAS*S ;
    Q_RATE_DOT        := (QS*CBAR*CM)/IYY ;
    XS                := - QS*CD ;
    ZS                := - QS*CL ;
    U_DOT_BODY        := (XS - ZS*AOA - WT*PITCH_ATT + THRUST)/MASS -
                         PITCH_DOT*W_BODY_GRD ;
    W_DOT_BODY        := (XS*AOA + ZS + WT)/MASS +
                         PITCH_DOT*U_BODY_GRD ;
    INTEGRATE(P_RATE_DOT, OLD_P_RATE_DOT, PITCH_DOT,  OLD_PITCH_DOT1) ;
    INTEGRATE(PITCH_DOT,  OLD_PITCH_DOT2, PITCH_ATT,  OLD_PITCH_ATT1)  ;
    INTEGRATE(U_DOT_BODY, OLD_U_DOT_BODY, U_BODY_GRD, OLD_U_BODY_GRD) ;
    INTEGRATE(W_DOT_BODY, OLD_W_DOT_BODY, W_BODY_GRD, OLD_W_BODY_GRD) ;
    TIME              := TIME + 0.05 ;

-- Compute Airplane Sensors

    AOA               := W_BODY_GRD/(U_BODY_GRD - WIND) ;
    TAS               := U_BODY_GRD - WIND + W_BODY_GRD*AOA ;

-- Generate Outer Loop References

    X_DOT             := (U_BODY_GRD + W_BODY_GRD*PITCH_ATT) ;
    H_DOT             := U_BODY_GRD*PITCH_ATT - W_BODY_GRD ;
    INTEGRATE(-X_DOT, OLD_X_DOT, DISTANCE, OLD_DIST) ;
    INTEGRATE(H_DOT, OLD_H_DOT, HEIGHT, OLD_HEIGHT) ;

end if ;

PITCH             := PITCH_ATT*DEG_CONV ;
Q_RATE            := PITCH_DOT*DEG_CONV ;
D_AOA             := AOA*DEG_CONV ;
D_ALPHA           := D_AOA ;
V_TOTAL           := TAS*KNOTS_CONV ;
D_ACCEL           := 1.0 - G_DOT_BODY*G_CONV ;
FPM_RATE          := (H_DOT/X_DOT)*DEG_CONV ;
GS_ERROR          := (HEIGHT/DISTANCE)*DEG_CONV - 2.5 ;
STABILIZER        := STAB_POS*DEG_CONV ;

WRITE_RESPONSE ;
```

Figure 28 - Procedure SIMULATE_FLIGHT Listing (2 of 3)

```
-- Furnish Computed Output for DFCS Use

for INDEX in 1..4
loop
    GS_ERROR_LEV(INDEX)            := MEAN_DEV_SIGNAL(GS_ERROR) ;
    RAD_ALTITUDE(INDEX)            := RAD_ALT_SIGNAL(HEIGHT) ;
    LEFT_ACA(INDEX)                := ACA_SIGNAL(L_ACA) ;
    RIGHT_ACA(INDEX)               := ACA_SIGNAL(R_ACA) ;
    if   INDEX <= 3
    then BODY_ACCEL(INDEX)         := ACCEL_SIGNAL(R_ACCEL) ;
         P_RATE_GYRO(INDEX)        := ANG_RATE_SIGNAL(P_OUT) ;
         if   INDEX <= 2
         then TRUE_AIRSPEED(INDEX) := TAS_SIGNAL(V_TRUE) ;
         end if ;
    end if ;
end loop ;

end SIMULATE_FLIGHT ;
```

Figure 28 - Procedure SIMULATE_FLIGHT Listing (3 of 3)

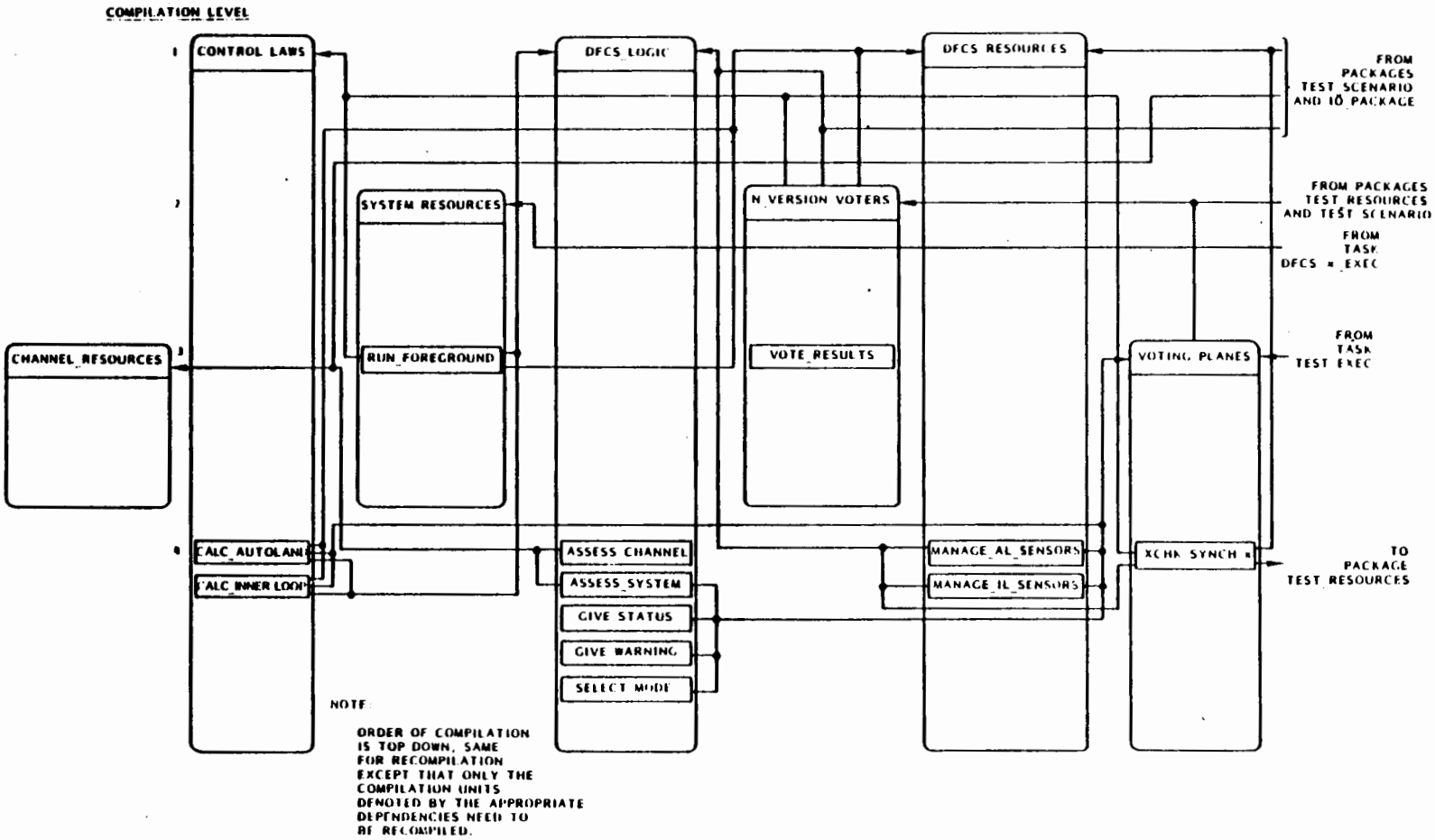


Figure 29 - Overall Compilation Dependencies (1 of 2)

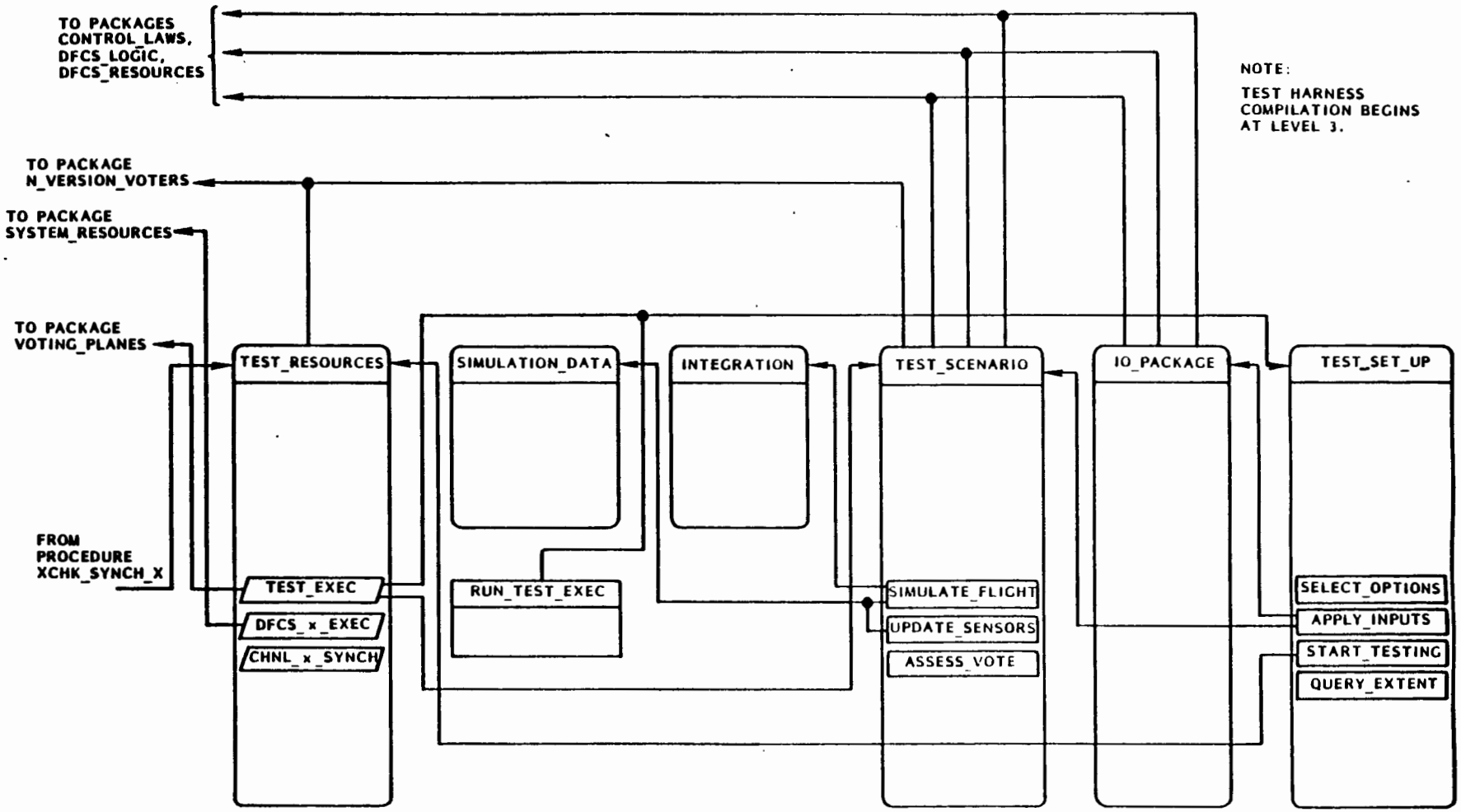


Figure 29 - Overall Compilation Dependencies (2 of 2)

harness could import and access certain variables. Basically, the test harness was readily usable, and naturally provided all the Ada package objects needed for testing. Test case definition was problematical because of the data dependencies among applications program units, but test case application via the harness was quite convenient.

14.5 Compilation Dependencies

The total DFCS/test program is exceptionally complex for its lines of source code because of the N-version voting requirements and the test observability requirements. While the procedure/task calling structure in Figure 23 is rather straightforward, the compilation dependencies are quite tortuous, as Figure 29 reveals. They can complicate recompilation following essentially minor code changes. These dependencies are inherent in Ada, and they are the price of global consistency checks among program units. This figure, however, makes it clear what recompilation sequences are required, and hence facilitates orderly software development.

15.0 RESULTS AND CONCLUSIONS

The all-up test harness was run with a modest amount of further development. Despite prior awareness of the criticality of specifications (e.g., see Reference 11), several iterations of specification de-bugging were necessary to eliminate associated software faults. The Ada program structuring techniques seemed to work well, with the exception of raising the visibility level of many variables for test observability or N-version voting. The software fault tolerance seemed to work well, but further study of the voter mechanisms is indicated.

Since some of the programmers had no prior Ada experience, the incidence of software faults was somewhat high. But all considered, programmer usage of Ada was really quite good. Variations among versions was very substantial, alleviating concern that Ada restrictions would hamper independence of software versions. The richness of Ada admits diverse ways of implementing the same functionality, provided the encompassing design does not encroach beyond program unit interfaces. This means that N-version programmers must have freedom to define and control all data objects at the level they are developing, a rule that was learned by early and unsuccessful initiatives to the contrary.

A summary critique of the effort is presented in Figure 30, and expanded in the following sub-sections.

15.1 N-Version Software Demonstration

Basically, the N-version demonstration was satisfactory. Ample faults indigenous to the four versions permitted affirmation of the fundamental adequacy of the N-version approach, but some questions remain due to the limited scale evaluation possible. Still, the degree of complexity of the N-version software was surprisingly high, largely due to mode and fault logic. The problems with the specifications resided mostly in this area as well. The preparation of adequate specifications was found to be especially problemsome. Hence, our continuing interest in formal specification has been intensified. Larger-scale logic definition problems may dictate some new type verification tools with respect to correctness and completeness.

In the course of N-version development, it was also discovered that the top-level design had been too encompassing. For example, the definition of data types and objects for the applications programs units was found to be best left to the individual programmer's discretion. This enabled greater independence among versions and better overall program structure. At the same time, the low-level N-version programmer defined packages were found to be very useful in a variety of ways, such as containing saved variables and text for newly defined procedures. The ultimate variation among versions was appreciable, alleviating concerns that Ada would be too restrictive.

15.2 Methodology Extensions

Basically, the Ada package partitioning technique produced qualitatively good results in limited use. Certain benefits accrue to source code compactness and comprehensibility. For example, the way in which data objects were declared obviated the need for the N-version program units to have parameters

APPROACH	RESULTS	CONCLUSIONS	FOLLOW-UP
N-Version Software Fault Tolerance	Implementation Demonstration	Promising but Uncalibrated Benefits	Rigorous Validation
		Specification Difficulties	Specification Verification
Program Structuring Technique	Extensions to Methodology	General Value of Structuring Method	Complexity Metrics
		Recompilation Acknowledgement	Dependancy Structuring
Ada Multitasking Test Harness	Automated Test Harness	Instrumentation Degrades Structure	Ada Testability
		Excessive Test Cases	Long-Term Testing

Figure 30 - Project Critique

passed to them, thereby making the source code for Procedure RUN_FOREGROUND_x far less cluttered than it otherwise would be. Had it not been for the raising of the namespace for voting or testability, this absence of parameter passing would have been accompanied by reductions in the data object namespace. Specifically, some of the objects would have been declared in package bodies, rather than in specification parts.

The same package definition approach lent itself to "detached" test harness observability of the voted DFCS data objects in that the DFCS code was unaffected by the test harness except for certain object visibility elevations. This passive observation capacity is of course inherent in the Ada language. For unit checkout/de-bugging, the test harness was set up to run for just one DFCS channel task. This worked well, but it prompted concern over unit testing in Ada in general. Basically, access to the entities required of all interfacing program units seems to complicate unit testing. Since the single channel test harness alleviated such problems, perhaps this type tool may prove widely useful.

Despite the relatively modest size of the overall program, a significant effort was involved in coping with compilation dependencies among Ada units. Such dependancies are complicated in the combined DFCS/test software. More generally, they are the price of Ada's global syntax checks, so the only alternative is the purposeful improvement of program structuring relative to compilation dependancies. This was accomplished using graphical representations of the kind illustrated in Ref. 17. This technique yielded the perspective to lower the levels of some dependencies. It also made recompilation demands more apparent. Based on this experience, it would seem appropriate to include compilation dependencies in the characterization of Ada program structuredness.

15.3 Test Harness Flexibility

The test harness was surprisingly compact and extensible, as well as very serviceable. Although the harness met essentially all of its requirements, it was necessary to modify the test article software at the lowest, hardware-oriented level. This was considered reasonable in the absence of target computers, for the tradeoffs for simulating synchronization hardware was very unfavorable. Note that testing the software in flight computers would normally enable visibility of any address location, independent of program structuring of the namespace. This suggests that the raising of the namespace for test observability purposes might not be necessary under a different testing senario. This issue, together with the Ada unit testing question, prompts further investigation into Ada testing techniques.

To date the test harness usage has been somewhat limited compared with its potential. The test driver and test instrumentation/monitor are inherently adaptable and are being augmented for protracted, multiple test cases. The aforementioned DFCS logic complexity, in part, motivates this, along with the prospect of probing for persistent software faults. These are of major concern because they are the kind that software fault tolerance must cope with. Another pending use of the harness is a proposed investigation of N-version voters.

15.4 Conclusions

The following conclusions have been formulated as a result of this project:

- o Calibration of benefits of N-version software are needed that quantitatively validate its favorable impact on system reliability
- o Complexity metrics are needed to quantitatively delineate design techniques or alternatives relative to program structure
- o Means to characterize the overall structure of Ada programs are desirable that acknowledge compilation dependancies
- o Ada testability needs to be explored in terms of data object visibility versus preferred program structuring alternatives
- o Specification technology needs to be improved to facilitate orderly N-version software development and preclude specification oriented faults.

Despite the extent of these follow-on recommendations, the investigation results were quite favorable with regard to improved structuring techniques, high-fidelity multitasking testing, and N-version software implementation. The identification of further needs are actually an indication of progress.

References

1. Slivinski, T. et Al.: "Study of Fault-Tolerant Software Technology," NASA CR 172385.

2. Avizienis, A.: "The N-Version Approach to Fault-Tolerant Software," IEEE Transactions on Software Engineering, December 1985.

3. ANSI/MIL-STD 1815A, "Military Standard Ada Programming Language," U.S. Department of Defense Ada Joint Program Office, 22 January 1983.

4. Balzer, R. et el.: "Informality in Program Specifications," IEEE Transactions on Software Engineering, March 1978.

5. Luckham, D.C. and F.W. von Henke: "An Overview of Anna, a Specification Language for Ada," IEEE Software, March 1985.

6. MIL-F-9490D, "Flight Control Systems - Design, Installation, and Test of Piloted Aircraft, General Specification for," U.S. Air Force 6 June 1975.

7. Mulcare, D.B. et al.: "Quadruplex Digital Flight Control System Assessment," DOT/FAA/CT-86/30, October 1986.

8. Mulcare, D.B. et al.: "Analytical Design and Assurance of Digital Flight Control System Structure," AIAA Journal of Guidance, Dynamics, and Control, May-June 1984.

9. Yourdan, E. and L.L. Constantine: "Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design," Prentice-Hall 1979.

10. Mulcare, D.B. et al.: "Description and Rationale of AIRLAB Experiments for Digital Flight Controls Software Fault Tolerance and Reliability," Lockheed-Georgia Report LG86ER0051 for NAS1-16199, Task C-372, September 1986.

11. Avizienis, A., and J.P.J. Kelly: "Fault Tolerance by Design Diversity: Concepts and Experiments," IEEE Computer, August 1984.

Appendix A - Version 3 Applications Software

Altogether, six versions of DFCS applications software were generated. Ultimately, two were required for specification de-bugging. The following version is provided as an example of the Ada source code produced. Note that the programmer defined Ada packages were a key to approaching version independence in that any desired data types or objects could be declared there. Also, the packages permitted the definition of saved variables as needed for digital filters or logic latches, and the shortening of procedure bodies by distributing source code. The sequence of program unit listings in this appendix is:

Figure No.	Title	Page
A-1	Procedure SELECT_MODE_V3	A-2
A-2	Procedure ASSESS_CHANNEL_V3	A-4
A-3	Package CHNL_3_ASSESSMENT	A-8
A-4	Procedure GIVE_STATUS_V3	A-9
A-5	Procedure MANAGE_AL_SENSORS_V3	A-10
A-6	Package CHNL_3_AL_VOTER	A-12
A-7	Procedure CALC_AUTOLAND_V3	A-17
A-8	Package AL_RESOURCES	A-18
A-9	Procedure MANAGE_IL_SENSORS_V3	A-24
A-10	Package CHNL_3_IL_VOTER	A-26
A-11	Procedure CALC_INNER_LOOP_V3	A-30
A-12	Package IL_RESOURCES	A-33
A-13	Procedure ASSESS_SYSTEM_V3	A-34
A-14	Procedure GIVE_WARNING_V3	A-37
A-15	Package WARNING_CHECKS	A-38

Ada Procedure SELECT_MODE_ V3.ADA

```
with VOTING_PLANES ; use VOTING_PLANES ;
separate(DFCS_LOGIC)
procedure SELECT_MODE_V3 is

begin
AL_WARN_V3 := BLANK ;
case MODE_SEL.AUTOPILOT is
when  ALT_HOLD  =>      MODE_ENG_V3.AUTOPILOT := ALT_HOLD ;
                        MODE_ENG_V3.AUTOLAND  := OFF ;
when  BASIC     =>      MODE_ENG_V3.AUTOPILOT := BASIC ;
                        MODE_ENG_V3.AUTOLAND  := OFF ;
when  OFF       =>      MODE_ENG_V3.AUTOPILOT := OFF ;
                        MODE_ENG_V3.AUTOLAND  := OFF ;
when  VERT_NAV  =>      MODE_ENG_V3.AUTOPILOT := VERT_NAV ;
                        MODE_ENG_V3.AUTOLAND  := OFF ;
when  AUTOLAND  =>
    AUTOLAND_ENGAGE_LOGIC :
    declare
        type VALIDITY_CNT is
            record
                GS : INTEGER range 0..4 := 0 ;
                NA : INTEGER range 0..3 := 0 ;
                RA : INTEGER range 0..4 := 0 ;
            end record ;
        NUM_VAL     : VALIDITY_CNT ;
    begin
        MODE_SEL.AUTOPILOT := AUTOLAND ;
        for INDEX in 1..4
            loop
                if AL_COMP_V3.GS_BEAM_VAL(INDEX) = TRUE
                then NUM_VAL.GS := NUM_VAL.GS + 1 ;
                end if ;
                if AL_COMP_V3.RAD_ALT_VAL(INDEX) = TRUE
                then NUM_VAL.RA := NUM_VAL.RA + 1 ;
                end if ;
                if INDEX /= 4
                then if AL_COMP_V3.N_ACCEL_VAL(INDEX) = TRUE
                     then NUM_VAL.NA := NUM_VAL.NA + 1 ;
                     end if ;
                end if;
            end loop ;
```

Figure A-1 Procedure SELECT_MODE_V3 (Sheet 1 of 2)

```
        case  MODE_SEL.AUTOLAND is
        when CAT_3A =>
            if.     FBW_STATUS_V3 = OP_STATE_1  and  NUM_VAL.NA = 3
                    and  NUM_VAL.GS = 4  and  NUM_VAL.RA = 4
            then    MODE_ENG_V3.AUTOLAND := CAT_3A ;
            elsif   FBW_STATUS_V3 >= OP_STATE_2  and  NUM_VAL.NA >= 2
                    and  NUM_VAL.GS >= 2  and NUM_VAL.RA >= 2
            then    MODE_ENG_V3.AUTOLAND := CAT_2 ;
                    AL_WARN_V3 := CAT_3_INOP ;
            elsif   FBW_STATUS_V3 < OP_STATE_2  or  NUM_VAL.NA = 1
                    or  NUM_VAL.GS = 1  or  NUM_VAL.RA = 1
            then    MODE_ENG_V3.AUTOLAND := CAT_1 ;
                    AL_WARN_V3 := CAT_3_INOP ;
            else    MODE_ENG_V3.AUTOLAND := OFF ;
                    AL_WARN_V3 := CAT_3_INOP ;
            end if ;
        when CAT_2 =>
            if      FBW_STATUS_V3 >= OP_STATE_2  and NUM_VAL.NA >= 2
                    and  NUM_VAL.GS >= 2  and  NUM_VAL.RA >= 2
            then    MODE_ENG_V3.AUTOLAND := CAT_2 ;
            elsif   FBW_STATUS_V3 < OP_STATE_2  or  NUM_VAL.NA = 1
                    or  NUM_VAL.GS = 1  or  NUM_VAL.RA = 1
            then    MODE_ENG_V3.AUTOLAND := CAT_1 ;
                    AL_WARN_V3 := CAT_2_INOP ;
            else    MODE_ENG_V3.AUTOLAND := OFF ;
                    AL_WARN_V3 := CAT_2_INOP ;
            end if ;
        when CAT_1 =>
            if      NUM_VAL.NA >= 1  and  NUM_VAL.GS >= 1
                    and NUM_VAL.RA >= 1
            then    MODE_ENG_V3.AUTOLAND := CAT_1 ;
            else    MODE_ENG_V3.AUTOLAND := OFF ;
            end if ;
        when OFF =>
            null ;
        end case ;
    end AUTOLAND_ENGAGE_LOGIC ;

end case ;
CHNL_3_XCHK_NUM := 1 ;
XCHK_SYNCH_3 ;                          -- Call for N-Version Vote
end SELECT_MODE_V3 ;
```

Figure A-1 Procedure SELECT_MODE_V3 (Sheet 2 of 2)

Ada Procedure ASSESS_CHANNEL_V3

```
with CHNL_3_ASSESSMENT ; use CHNL_3_ASSESSMENT ;
with CHANNEL_RESOURCES ; use CHANNEL_RESOURCES ;
separate(DFCS_LOGIC)
procedure ASSESS_CHANNEL_V3 is

begin

case CPU_COUNT is                                   --  Computer Channel
when 0  =>                                                  --  Normal
    if   CMPTR_3.CPU_CHK_OK = FALSE
    then CPU_CHK   := FALSE ;
         CPU_COUNT := 1 ;
    end if ;
when 1  =>                                                  --  Faulted
    if   CMPTR_3.CPU_CHK_OK = TRUE
    then CPU_COUNT := -1 ;
    end if ;
when -10..-1 =>                                             --  Healing
    if   CMPTR_3.CPU_CHK_OK = TRUE
    then CPU_COUNT := CPU_COUNT - 1 ;
         if   CPU_COUNT <= -10
         then CPU_HEAL := CPU_HEAL + 1 ;
              if   CPU_HEAL > 5
              then CPU_COUNT := 2 ;
              else CPU_CHK   := TRUE ;
                   CPU_COUNT := 0 ;
              end if ;
         end if ;
    else CPU_COUNT := -1 ;
    end if ;
when 2 =>                                                   --  Failed
    CPU_CHK := FALSE ;
end case ;

case IOP_COUNT is                                   --  I/O Processor
when 0  =>                                                  --  Normal
    if   CMPTR_3.IO_PROC_OK = FALSE
    then IOP_CHK   := FALSE ;
         IOP_COUNT := 1 ;
    end if ;
when 1  =>                                                  --  Faulted
    if   CMPTR_3.IO_PROC_OK = TRUE
    then IOP_COUNT := -1 ;
    end if ;
```

Figure A-2 Procedure ASSESS_CHANNEL_V3 (Sheet 1 of 4)

```
when -10..-1 =>                                          --  Healing
    if   CMPTR_3.IO_PROC_OK = TRUE
    then IOP_COUNT := IOP_COUNT - 1 ;
         if   IOP_COUNT <= -10
         then IOP_HEAL := IOP_HEAL + 1 ;
              if   IOP_HEAL > 5
              then IOP_COUNT := 2 ;
              else IOP_CHK   := TRUE ;
                   IOP_COUNT := 0 ;
              end if ;
         end if ;
    else IOP_COUNT := -1 ;
    end if ;
when 2 =>                                                --  Failed
    IOP_CHK := FALSE ;
end case ;

case MUX_COUNT is                                   --  MUX Bus Checks --
when 0..2 =>                                             --  Normal
    if   MUX_COUNT = 0
    then if   CMPTR_3.MUX_BUS_OK = FALSE
         then MUX_COUNT := 1 ;
         end if ;
    else if   CMPTR_3.MUX_BUS_OK = FALSE
         then MUX_COUNT := MUX_COUNT + 1 ;
              if   MUX_COUNT >= 3
              then MUX_CHK := FALSE ;
              end if ;
         else MUX_COUNT := 0 ;
         end if ;
    end if ;
when 3 =>                                                --  Faulted
    if   CMPTR_3.MUX_BUS_OK = TRUE
    then MUX_COUNT := -1 ;
    end if ;
when -50..-1 =>                                          --  Healing
    if   CMPTR_3.MUX_BUS_OK = TRUE
    then MUX_COUNT := MUX_COUNT - 1 ;
         if   MUX_COUNT <= -50
         then MUX_HEAL := MUX_HEAL + 1 ;
              if   MUX_HEAL > 6
              then MUX_COUNT := 4 ;
              else MUX_CHK   := TRUE ;
                   MUX_COUNT := 0 ;
              end if ;
         end if ;
    else MUX_COUNT := -1 ;
    end if ;
when 4 =>                                                --  Failed
    CPU_CHK := FALSE ;
end case ;
```

Figure A-2 Procedure ASSESS_CHANNEL_V3 (Sheet 2 of 4)

```
case ACTR_COUNT is                                  --  Actuator Checks
when 0..2 =>                                            --  Normal
    if   ACTR_COUNT = 0
    then if   SERVO_3.ACTUATOR_ON = FALSE
         then ACTR_COUNT := 1 ;
         end if ;
    else if   SERVO_3.ACTUATOR_ON = FALSE
         then ACTR_COUNT := ACTR_COUNT + 1 ;
              if   ACTR_COUNT >= 3
              then ACTR_CHK := FALSE ;
              end if ;
         else ACTR_COUNT := 0 ;
         end if ;
    end if ;
when 3 =>                                               --  Faulted
    if   SERVO_3.ACTUATOR_ON = TRUE
    then ACTR_COUNT := -1 ;
    end if ;
when -50..-1 =>                                         --  Healing
    if   SERVO_3.ACTUATOR_ON = TRUE
    then ACTR_COUNT := ACTR_COUNT - 1 ;
         if   ACTR_COUNT <= -50
         then ACTR_HEAL := ACTR_HEAL + 1 ;
              if   ACTR_HEAL > 2
              then ACTR_COUNT := 4 ;
              else ACTR_CHK   := TRUE ;
                   ACTR_COUNT := 0 ;
              end if ;
         end if ;
    else ACTR_COUNT := -1 ;
    end if ;
when 4 =>                                               --  Failed
    ACTR_CHK := FALSE ;
end case ;
```

Figure A-2 Procedure ASSESS_CHANNEL_V3 (Sheet 3 of 4)

```
case LVDT_COUNT is                                  --  LVDT Sensor Checks
when 0..3 =>                                            --  Normal
    if   LVDT_COUNT = 0
    then if   SERVO_3.LVDT_VALID = FALSE
         then LVDT_COUNT := 1 ;
         end if ;
    else if   SERVO_3.LVDT_VALID = FALSE
         then LVDT_COUNT := LVDT_COUNT + 1 ;
              if   LVDT_COUNT >= 4
              then LVDT_CHK := FALSE ;
              end if ;
         else LVDT_COUNT := 0 ;
         end if ;
    end if ;
when 4 =>                                               --  Faulted
    if   SERVO_3.LVDT_VALID = TRUE
    then LVDT_COUNT := -1 ;
    end if ;
when -50..-1 =>                                         --  Healing
    if   SERVO_3.LVDT_VALID = TRUE
    then LVDT_COUNT := LVDT_COUNT - 1 ;
         if   LVDT_COUNT <= -50
         then LVDT_HEAL := LVDT_HEAL + 1 ;
              if   LVDT_HEAL > 2
              then LVDT_COUNT := 5 ;
              else LVDT_CHK   := TRUE ;
                   LVDT_COUNT := 0 ;
              end if ;
         end if ;
    else LVDT_COUNT := -1 ;
    end if ;
when 5 =>                                               --  Failed
    LVDT_CHK := FALSE ;
end case ;

CHNL_STATUS_V3 := CPU_CHK  and  IOP_CHK  and  MUX_CHK  and  ACTR_CHK
                           and  LVDT_CHK  and  SERVO_3.POWER_AVAIL ;

--  No N-Version Vote Taken Because Status is Unique to each Channel

end ASSESS_CHANNEL_V3 ;
```

Figure A-2 Procedure ASSESS_CHANNEL_V3 (Sheet 4 of 4)

Ada Package CHNL_3_ASSESSMENT_V3

```
package CHNL_3_ASSESSMENT is

        CPU_COUNT          : INTEGER range -10..2 := 0 ;
        IOP_COUNT          : INTEGER range -10..2 := 0 ;
        MUX_COUNT          : INTEGER range -50..4 := 0 ;
        ACTR_COUNT         : INTEGER range -50..4 := 0 ;
        LVDT_COUNT         : INTEGER range -50..5 := 0 ;

        CPU_CHK, IOP_CHK, MUX_CHK, ACTR_CHK, LVDT_CHK
                           : BOOLEAN := TRUE ;

        CPU_HEAL           : INTEGER range   0..5 := 0 ;
        IOP_HEAL           : INTEGER range   0..5 := 0 ;
        MUX_HEAL           : INTEGER range   0..6 := 0 ;
        ACTR_HEAL          : INTEGER range   0..2 := 0 ;
        LVDT_HEAL          : INTEGER range   0..2 := 0 ;

end CHNL_3_ASSESSMENT ;

package body CHNL_3_ASSESSMENT is

begin
null ;
end CHNL_3_ASSESSMENT ;
```

Figure A-3 Package CHNL_3_ASSESSMENT

Ada Procedure GIVE_STATUS_V3

```
with VOTING_PLANES ; use VOTING_PLANES ;
separate(DFCS_LOGIC)
procedure GIVE_STATUS_V3 is

begin

ANNUN_V3.FLY_QLTY := FLY_QUAL_V3 ;
case  MODE_ENG_V3.AUTOPILOT is
    when OFF =>
        ANNUN_V3.AFCS_STATUS     := AFCS_DSIENGAGED ;
        ANNUN_V3.AL_PROG_DISP    := (1..5 => FALSE) ;
        ANNUN_V3.AUTOPILOT_MODE  := OFF ;
    when AUTOLAND =>
        if   AL_PHASE_V3 = AUTOLAND_INOP
        then ANNUN_V3.AFCS_STATUS        := AUTOPILOT_ENGAGED ;
             ANNUN_V3.AUTOPILOT_MODE     := BASIC ;
             ANNUN_V3.AL_PROG_DISP       := AUTOLAND_ENGAGED ;
        else ANNUN_V3.AFCS_STATUS        := AUTOPILOT_ENGAGED ;
             ANNUN_V3.AUTOPILOT_MODE     := AUTOLAND ;
             case AL_PHASE_V3 is
                 when AUTOLAND_INOP =>
                     ANNUN_V3.AL_PROG_DISP := (1..5 => FALSE) ;
                 when AUTOLAND_ARMED =>
                     ANNUN_V3.AL_PROG_DISP := (1 => TRUE,
                                               2..5 => FALSE) ;
                 when GLIDESLOPE_TRACK =>
                     ANNUN_V3.AL_PROG_DISP(2) := TRUE ;
                 when DECISION_ALTITUDE =>
                     ANNUN_V3.AL_PROG_DISP(3) := TRUE ;
                 when ALERT_ALTITUDE =>
                     ANNUN_V3.AL_PROG_DISP(4) := TRUE ;
                 when FLARE =>
                     ANNUN_V3.AL_PROG_DISP(5) := TRUE ;
             end case ;
        end if ;
    when others =>
        ANNUN_V3.AFCS_STATUS     := AUTOPILOT_ENGAGED ;
        ANNUN_V3.AUTOPILOT_MODE  := MODE_ENG_V3.AUTOPILOT ;
        ANNUN_V3.AL_PROG_DISP    := (1..5 => FALSE ) ;
end case ;
CHNL_3_XCHK_NUM := 2;
XCHK_SYNCH_3 ;

end GIVE_STATUS_V3 ;
```

Figure A-4 Procedure GIVE_STATUS_V3

```
                    Ada Procedure MANAGE_AL_SENSORS_V3
                    ----------------------------------

with CHNL_3_AL_VOTER ; use CHNL_3_AL_VOTER ;
with DFCS_LOGIC      ; use DFCS_LOGIC ;
with VOTING_PLANES   ; use VOTING_PLANES ;
separate(DFCS_RESOURCES)
procedure MANAGE_AL_SENSORS_V3 is

        GS_FLAGS_IN, GS_COMP_IN, GS_COMP_OUT, RA_FLAGS_IN, RA_COMP_IN,
                RA_COMP_OUT                 : BOOL_VECTOR(1..4) ;
        NA_FLAGS_IN, NA_COMP_IN, NA_COMP_OUT
                                            : BOOL_VECTOR(1..3) ;
        GS_SIGNALS, RA_SIGNALS              : REAL_VECTOR(1..4) ;
        NA_SIGNALS                          : REAL_VECTOR(1..3) ;
        GS_MED, NA_MED, RA_MED              : FLOAT ;

begin

for INDEX in 1..4
    loop
        GS_FLAGS_IN(INDEX) := AL_FLAGS.GS_BEAM_VAL(INDEX) ;
        RA_FLAGS_IN(INDEX) := AL_FLAGS.RAD_ALT_VAL(INDEX) ;
    end loop ;
for INDEX in 1..3
    loop
        NA_FLAGS_IN(INDEX) := AL_FLAGS.N_ACCEL_VAL(INDEX) ;
    end loop ;

CHK_AL_FLAGS_IN(GS_FLAGS_IN, 1, 4) ;                    --  Check Sensor
CHK_AL_FLAGS_IN(NA_FLAGS_IN, 2, 3) ;                    --  Flag Input
CHK_AL_FLAGS_IN(RA_FLAGS_IN, 3, 4) ;                    --  Validities

for INDEX in 1..4
   loop
       GS_SIGNALS(INDEX) := FLOAT(GS_BEAM_DEV(INDEX)) ;
       RA_SIGNALS(INDEX) := FLOAT(RAD_ALTITUDE(INDEX)) ;
   end loop ;

for INDEX in 1..3
   loop
        NA_SIGNALS(INDEX) := FLOAT(NORM_ACCEL(INDEX)) ;
   end loop ;

VOTE_AL_SENSORS(GS_SIGNALS, 1, 4, GS_MED) ;
        AL_MED_V3.GS_DEV := BEAM_DEV_SIGNAL(GS_MED) ;          --  Select
VOTE_AL_SENSORS(NA_SIGNALS, 2, 3, NA_MED) ;                    --  Median
        AL_MED_V3.N_ACCEL := ACCEL_SIGNAL(NA_MED) ;            --  Sensor
VOTE_AL_SENSORS(RA_SIGNALS, 3, 4, RA_MED) ;                    --  Signals
        AL_MED_V3.RAD_ALT := RAD_ALT_SIGNAL(RA_MED) ;
```

Figure A-5 Procedure MANAGE_AL_SENSORS_V3 (Sheet 1 of 2)

```
if   MY_TURN   then
CHK_FAULT_LOGIC(GS_SIGNALS, GS_MED, 1, 4, GS_COMP_OUT) ;     --  Compare
CHK_FAULT_LOGIC(NA_SIGNALS, NA_MED, 2, 3, NA_COMP_OUT) ;     --  Inputs &
CHK_FAULT_LOGIC(RA_SIGNALS, RA_MED, 3, 4, RA_COMP_OUT) ;     --  Check
MY_TURN := FALSE ;
else MY_TURN := TRUE ;
end if ;
                                                             --  Comparators
for INDEX in 1..4
    loop
        AL_COMP_V3.GS_BEAM_VAL(INDEX) := GS_COMP_OUT(INDEX) ;
        AL_COMP_V3.N_ACCEL_VAL(INDEX) := NA_COMP_OUT(INDEX) ;
    end loop ;
for INDEX in 1..3
    loop
        AL_COMP_V3.RAD_ALT_VAL(INDEX) := RA_COMP_OUT(INDEX) ;
    end loop ;

CHNL_3_XCHK_NUM := 3 ;
XCHK_SYNCH_3 ;                                  -- Call for N-version vote

end MANAGE_AL_SENSORS_V3 ;
```

Figure A-5 Procedure MANAGE_AL_SENSORS_V3 (Sheet 2 of 2)

Package CHNL_3_AL_VOTER

```
package CHNL_3_AL_VOTER is

        AL_COMP_COUNT :  array (1..3, 1..4) of INTEGER range  0..6
                            := (1..3 => (1..4 => 0)) ;
        AL_FLAG_COUNT :  array (1..3, 1..4) of INTEGER range -5..5
                            := (1..3 => (1..4 => 0)) ;
        AL_FLAG_IN    :  array (1..3, 1..4) of BOOLEAN
                            := (others => (others => TRUE)) ;
        AL_COMP_OUT   :  array (1..3, 1..4) of BOOLEAN
                            := (others => (others => TRUE)) ;

        MY_TURN                  :  BOOLEAN ;
        NUM_SENSORS              :  INTEGER range 3..4 := 4 ;
        NUM_VOTES                :  INTEGER range 0..4 := 0 ;
        SET_NUM                  :  INTEGER range 1..3 := 1 ;

        type BOOL_VECTOR is array (INTEGER range <>) of BOOLEAN ;
        type REAL_VECTOR is array (INTEGER range <>) of FLOAT ;

        procedure CHK_AL_FLAGS_IN(AL_FLAG : in BOOL_VECTOR ; SET_NUM,
                                  NUM_SENSORS : in INTEGER) ;
        procedure VOTE_AL_SENSORS(AL_SENSORS : in REAL_VECTOR ;
                                  SET_NUM, NUM_SENSORS : in INTEGER ;
                                  AL_SENSOR_MED : out FLOAT) ;
        procedure CHK_FAULT_LOGIC(AL_SENSORS: in REAL_VECTOR ;
                AL_SENSOR_MED : in FLOAT ; SET_NUM, NUM_SENSORS :
                in INTEGER ; AL_COMP_VAL : out BOOL_VECTOR) ;

end CHNL_3_AL_VOTER ;
```

Figure A-6 Package CHNL_3_AL_VOTER (Sheet 1 of 4)

```
package body CHNL_3_AL_VOTER is

procedure CHK_AL_FLAGS_IN(AL_FLAG : in BOOL_VECTOR ; SET_NUM,
                          NUM_SENSORS : in INTEGER) is
begin
    for INDEX in 1..NUM_SENSORS
    loop
        case AL_FLAG_COUNT(SET_NUM, INDEX) is
        when 0 =>                                          -- Normal
             if   AL_FLAG(INDEX) = FALSE
             then AL_FLAG_COUNT(SET_NUM, INDEX) := 1 ;
             end if ;
        when 1..5 =>                                       -- Faulted
             if   AL_FLAG(INDEX) = FALSE
             then AL_FLAG_COUNT(SET_NUM, INDEX) :=
                  AL_FLAG_COUNT(SET_NUM, INDEX) + 1;
                  if   AL_FLAG_COUNT(SET_NUM, INDEX) >= 5
                  then AL_FLAG_IN(SET_NUM, INDEX)    := FALSE ;
                       AL_FLAG_COUNT(SET_NUM, INDEX) := -1 ;
                  end if ;
             else AL_FLAG_COUNT(SET_NUM, INDEX)      := 0 ;
             end if ;
        when -5..-1 =>                                     -- Healing
             if   AL_FLAG(INDEX) = TRUE
             then AL_FLAG_COUNT(SET_NUM, INDEX) :=
                  AL_FLAG_COUNT(SET_NUM, INDEX) - 1 ;
                  if   AL_FLAG_COUNT(SET_NUM, INDEX) <= -5
                  then AL_FLAG_IN(SET_NUM, INDEX)    := TRUE ;
                       AL_FLAG_COUNT(SET_NUM, INDEX) := 0 ;
                  end if ;
             else AL_FLAG_COUNT(SET_NUM, INDEX)      := - 1 ;
             end if ;
        end case ;
    end loop ;
end CHK_AL_FLAGS_IN ;
```

Figure A-6 Package CHNL_3_AL_VOTER (Sheet 2 of 4)

```
procedure VOTE_AL_SENSORS(AL_SENSORS : in REAL_VECTOR ; SET_NUM,
                NUM_SENSORS : in INTEGER ; AL_SENSOR_MED : out FLOAT) is

     SET_RANKING  : array (1..4) of INTEGER range 0..4 := (0, 0, 0, 0) ;
     V            : array (1..4) of FLOAT := (0.0, 0.0, 0.0, 0.0) ;
     TEMP         : FLOAT := 0.0 ;
begin
     NUM_VOTES    := NUM_SENSORS ;
     for INDEX in 1..NUM_SENSORS
     loop
         if   AL_COMP_OUT(SET_NUM, INDEX) = FALSE
         then NUM_VOTES          := NUM_VOTES - 1 ;
         else SET_RANKING(INDEX) := INDEX ;
         end if ;
     end loop ;
     for INDEX in 1..NUM_VOTES
     loop
         for  CHNL_NUM in INDEX..4
         loop
              if   CHNL_NUM =  SET_RANKING(CHNL_NUM)
              then V(INDEX) := AL_SENSORS(CHNL_NUM) ;
              exit ;
              end if ;
         end loop ;
     end loop ;
     case NUM_VOTES is
     when 0 =>
         null ;
     when 1 =>
         AL_SENSOR_MED       := V(1) ;
     when 2 =>
         if     V(1) <= V(2)
         then AL_SENSOR_MED := V(1) ;
         else AL_SENSOR_MED := V(2) ;
         end if ;
     when 3 =>
         if     (V(2) <= V(1)  and  V(1) <= V(3))  or
                (V(3) <= V(1)  and  V(1) <= V(1))
         then AL_SENSOR_MED := V(1) ;
         elsif (V(1) <= V(2)  and  V(2) <= V(3))  or
                (V(3) <= V(2)  and  V(2) <= V(1))
         then AL_SENSOR_MED := V(2) ;
         else AL_SENSOR_MED := V(3) ;
         end if ;
     when 4 =>
         for  I in 1..NUM_VOTES-1
         loop
              for J in I+1..NUM_VOTES
              loop
                   if   V(I) >= V(J)
                   then TEMP := V(I) ;
                        V(I) := V(J) ;
                        V(J) := TEMP ;
                   end if ;
              end loop ;
         end loop ;
         AL_SENSOR_MED := V(2) ;
     end case ;
end VOTE_AL_SENSORS ;
```

Figure A-6 Package CHNL_3_AL_VOTER (Sheet 3 of 4)

```
procedure CHK_FAULT_LOGIC(AL_SENSORS: in REAL_VECTOR ;
                AL_SENSOR_MED : in FLOAT ; SET_NUM, NUM_SENSORS :
                in INTEGER ; AL_COMP_VAL : out BOOL_VECTOR) is

    AMPL_LIMIT  :  array (1..3) of FLOAT
                := (1 => 0.050, 2 => 0.025) ;
    MAX_CT      :  constant array (1..3) of INTEGER := (5, 4, 4) ;

begin
    for INDEX in 1..NUM_SENSORS
    loop
        if   SET_NUM = 3
        then AMPL_LIMIT(3) := 0.02* AL_SENSOR_MED ;
        end if ;
        case AL_COMP_COUNT(SET_NUM, INDEX) is
        when 0 =>                                          --  Normal
             if   abs(AL_SENSOR_MED - AL_SENSORS(INDEX)) >=
                  AMPL_LIMIT(SET_NUM)
             then AL_COMP_COUNT(SET_NUM, INDEX) := 1 ;
             end if ;
        when 1..5 =>                                       --  Faulted
             if   abs(AL_SENSOR_MED - AL_SENSORS(INDEX)) >=
                  AMPL_LIMIT(SET_NUM)
             then AL_COMP_COUNT(SET_NUM, INDEX) :=
                  AL_COMP_COUNT(SET_NUM, INDEX) + 1 ;
                  if   AL_COMP_COUNT(SET_NUM, INDEX) >= MAX_CT(SET_NUM)
                  then AL_COMP_OUT(SET_NUM, INDEX)   := FALSE ;
                       AL_COMP_COUNT(SET_NUM, INDEX) := 6 ;
                  end if ;
             else AL_COMP_COUNT(SET_NUM, INDEX) := 0 ;     -- Recovering
             end if ;
        when 6 =>                                          --  Failed
             null ;
        end case ;
        AL_COMP_VAL(INDEX) := AL_COMP_OUT(SET_NUM, INDEX)  or
                              AL_FLAG_IN(SET_NUM, INDEX) ;
    end loop ;
end CHK_FAULT_LOGIC ;

end CHNL_3_AL_VOTER ;
```

Figure A-6 Package CHNL_3_AL_VOTER (Sheet 4 of 4)

Ada Procedure CALC_AUTOLAND_V3

```
with AL_RESOURCES          ; use AL_RESOURCES ;
with DFCS_LOGIC            ; use DFCS_LOGIC ;
with DFCS_RESOURCES        ; use DFCS_RESOURCES ;
with VOTING_PLANES         ; use VOTING_PLANES ;

separate(CONTROL_LAWS)
procedure CALC_AUTOLAND_V3 is

begin

    case MODE_ENG_V3.AUTOLAND is
    when CAT_1 | CAT_2 | CAT_3A =>
         if   INITIALIZE = FALSE
         then AL_PHASE_V3 := AUTOLAND_INOP ;
              AL_SENS_MEDIAN(1) := FLOAT(AL_MED_V3.GS_DEV) ;
              AL_SENS_MEDIAN(2) := FLOAT(AL_MED_V3.N_ACCEL) ;
              AL_SENS_MEDIAN(3) := FLOAT(AL_MED_V3.RAD_ALT) ;
              AL_SENS_MEDIAN(4) := FLOAT(IL_MED_V3.P_RATE) ;
              CALC_AL_STEERING(AL_SENS_MEDIAN, MODE_ENG_V3.AUTOLAND,
                               AL_PHASE_V3, AL_STEERING_CMD);
              INITIALIZE := TRUE ;
         end if ;
         RAD_ALT := FLOAT(AL_MED_V3.RAD_ALT) ;
         CHECK_SUB_MODE(MODE_ENG_V3.AUTOLAND, RAD_ALT, AL_PHASE_V3) ;
         AL_SENS_MEDIAN(1) := FLOAT(AL_MED_V3.GS_DEV) ;
         AL_SENS_MEDIAN(2) := FLOAT(AL_MED_V3.N_ACCEL) ;
         AL_SENS_MEDIAN(3) := FLOAT(AL_MED_V3.RAD_ALT) ;
         AL_SENS_MEDIAN(4) := FLOAT(IL_MED_V3.P_RATE) ;
         CALC_AL_STEERING(AL_SENS_MEDIAN, MODE_ENG_V3.AUTOLAND,
                          AL_PHASE_V3, AL_STEERING_CMD) ;
         AUTOLAND_CMD_V3 := PITCH_COMMAND(AL_STEERING_CMD) ;
    when OFF =>
         if   INITIALIZE = TRUE
         then AL_PHASE_V3 := AUTOLAND_INOP ;
              AL_SENS_MEDIAN(1) := FLOAT(AL_MED_V3.GS_DEV) ;
              AL_SENS_MEDIAN(2) := FLOAT(AL_MED_V3.N_ACCEL) ;
              AL_SENS_MEDIAN(3) := FLOAT(AL_MED_V3.RAD_ALT) ;
              AL_SENS_MEDIAN(4) := FLOAT(IL_MED_V3.P_RATE) ;
              CALC_AL_STEERING(AL_SENS_MEDIAN, MODE_ENG_V3.AUTOLAND,
                               AL_PHASE_V3, AL_STEERING_CMD);
              INITIALIZE := FALSE ;
         end if ;
    end case ;
    CHKL_3_XCHK_NUM := 4 ;
    XCHK_SYNCH_3 ;
end CALC_AUTOLAND_V3 ;
```

Figure A-7 Procedure CALC_AUTOLAND_V3

Package AL_RESOURCES

```
with DFCS_LOGIC      ; use DFCS_LOGIC ;
with DFCS_RESOURCES ; use DFCS_RESOURCES ;
package AL_RESOURCES is

--  Signal Shaping Filter Coefficients

        --  Filter 1 : Glideslope Deviation Low-Pass

                GSI_K, GSI_KM1          : constant :=  1.0/8.0 ;
                GSO_KM1                 : constant :=  3.0/4.0 ;

        --  Filter 2 : Normal Acceleration High-Pass

                NZI_K                   : constant :=  900.0/901.0 ;
                NZI_KM1                 : constant := -900.0/901.0 ;
                NZO_KM1                 : constant :=  899.0/901.0 ;

        --  Filter 3 : Altitude Acceleration Low-Pass

                H2DI_K, H2DI_KM1        : constant :=  1.0/22.0 ;
                H2DO_KM1                : constant :=  9.0/11.0 ;

        --  Filter 4 : Radio Altitude High-Pass

                HI_K                    : constant :=  20.0/11.0 ;
                HI_KM1                  : constant := -20.0/11.0 ;
                HO_KM1                  : constant :=   9.0/11.0 ;

        --  Filter 5 : Glideslope Command Fader

                AGSI_K, AGSI_KM1        : constant :=   1.0/61.0 ;
                AGSO_KM1                : constant :=  59.0/61.0 ;

        --  Filter 6 : Command Rate Limiter

                RATE_LIMIT              : constant :=  3.0 ;

        --  Filter 7 : Pitch Rate Error Fader

                PREI_K, PREI_KM1        : constant :=   1.0/31.0 ;
                PREO_KM1                : constant :=  29.0/31.0 ;

        --  Filter 8 : Altitude Acceleration Integrator

                H2DAI_K, H2DAI_KM1      : constant :=  1.0/20.0 ;
                H2DAO_KM1               : constant :=        1.0 ;
```

Figure A-8 Package AL_RESOURCES (Sheet 1 of 6)

```
-- Gain Schedules

        -- Glideslope Desentization Gain (60 to 1000 FT.)

                KGS                     : constant := 1.0/9400.0 ;

        -- Flare Command Gain (-20 to 60 FT.)

                KFL                     : constant := -8.0/60.0 ;

-- Control Law Variables

        GS_ERR, GS_ERR_LP, GS_ERR_DS,
        DEL_NZ, H_2DOT, H_2DOT_AUG, H_2DOT_LP,
        HRA, HRA_HP, H_DOT,
        H_DOT_GS, H_DOT_REF, H_DOT_AGS, H_DOT_CMD1,
        H_DOT_CMD2, H_DOT_ERR,
        PR_CMD, PR_CMD_LIM, PR_ERR                          : FLOAT ;

-- Filter Memory Variables

        OLD_GS_ERR, OLD_GS_ERR_LP, OLD_DEL_NZ, OLD_H_2DOT,
        OLD_H_2DOT_LP, OLD_H_2DOT_AUG, OLD_H_DOT_REF, OLD_HRA,
        OLD_HRA_HP, OLD_H_DOT_AGS, OLD_PR_ERR,
        OLD_H_DOT_CMD1                                      : FLOAT ;

-- Glideslope/Autoland Progress Trip Points

                ALT_REF_1       : constant := 200.0 ;
                ALT_REF_2       : constant := 150.0 ;
                ALT_REF_3       : constant := 100.0 ;
                ALT_REF_4       : constant :=  60.0 ;

        type SENSOR_VECTOR     is array (1..4) of FLOAT ;
        AL_SENS_MEDIAN           : SENSOR_VECTOR ;

        AL_STEERING_CMD, PITCH_RATE, RAD_ALT     : FLOAT ;

        INITIALIZE               : BOOLEAN := FALSE ;

        procedure CALC_AL_STEERING(AL_SENSOR_MEDS : in SENSOR_VECTOR ;
                  SEL_AL_MODE : in AL_CATEGORY ; MODE_STATUS :
                  in AL_PROGRESS ; PITCH_AL_CMD : out FLOAT) ;

        procedure CHECK_SUB_MODE(SEL_AL_MODE : in AL_CATEGORY ;
                        RAD_ALT : in FLOAT ; MODE_STATUS :
                        in out AL_PROGRESS) ;

        procedure INITIALIZE_FILTERS ;
        procedure CALCULATE_GLIDESLOPE ;
        procedure CALCULATE_FLARE ;
        procedure FADER_LIMITER ;
        procedure RESET_FILTERS ;

end AL_RESOURCES ;
```

Figure A-8 Package AL_RESOURCES (Sheet 2 of 6)

```
package body AL_RESOURCES is

procedure CALC_AL_STEERING(AL_SENSOR_MEDS : in SENSOR_VECTOR ;
          SEL_AL_MODE : in AL_CATEGORY ; MODE_STATUS : in AL_PROGRESS ;
          PITCH_AL_CMD : out FLOAT) is
begin
         GS_ERR          := AL_SENSOR_MEDS(1) ;
         DEL_NZ          := AL_SENSOR_MEDS(2) ;
         HRA             := AL_SENSOR_MEDS(3) ;
         PITCH_RATE      := AL_SENSOR_MEDS(4) ;
case SEL_AL_MODE is
when CAT_2 | CAT_3A =>
    if   MODE_STATUS = AUTOLAND_INOP
    then INITIALIZE_FILTERS ;
    elsif MODE_STATUS = FLARE
    then CALCULATE_FLARE ;
    else CALCULATE_GLIDESLOPE ;
    end if ;
when CAT_1 =>
    if   MODE_STATUS = AUTOLAND_INOP
    then INITIALIZE_FILTERS ;
    elsif MODE_STATUS = GLIDESLOPE_TRACK
    then CALCULATE_GLIDESLOPE ;
    elsif MODE_STATUS = DECISION_ALTITUDE
    then FADER_LIMITER ;
    end if ;
when OFF =>
    RESET_FILTERS ;
end case ;
PITCH_AL_CMD := PR_ERR ;
end CALC_AL_STEERING ;
```

Figure A-8 Package AL_RESOURCES (Sheet 3 of 6)

```
procedure CHECK_SUB_MODE(SEL_AL_MODE : in AL_CATEGORY ; RAD_ALT :
        in FLOAT ; MODE_STATUS : in out AL_PROGRESS) is
begin
    case SEL_AL_MODE is
    when CAT_2 | CAT_3A =>
         case MODE_STATUS is
         when AUTOLAND_INOP =>
              MODE_STATUS := AUTOLAND_ARMED ;
         when AUTOLAND_ARMED =>
              MODE_STATUS := GLIDESLOPE_TRACK ;
         when GLIDESLOPE_TRACK =>
              if   SEL_AL_MODE = CAT_2 and then RAD_ALT <= ALT_REF_2
              then  MODE_STATUS := DECISION_ALTITUDE ;
              elsif RAD_ALT <= ALT_REF_3
                   then MODE_STATUS := ALERT_ALTITUDE ;
              end if ;
         when DECISION_ALTITUDE | ALERT_ALTITUDE =>
              if   RAD_ALT <= ALT_REF_4
              then MODE_STATUS := FLARE ;
              end if ;
         when FLARE =>
              null ;
         end case ;
    when CAT_1 =>
         case MODE_STATUS is
         when AUTOLAND_INOP =>
              MODE_STATUS := GLIDESLOPE_TRACK ;
         when GLIDESLOPE_TRACK =>
              if   RAD_ALT <= ALT_REF_4
              then MODE_STATUS := DECISION_ALTITUDE ;
              end if ;
         when others =>
              null ;
         end case ;
    when OFF =>
         null ;
    end case ;
end CHECK_SUB_MODE ;
```

Figure A-8 Package AL_RESOURCES (Sheet 4 of 6)

```
procedure INITIALIZE_FILTERS is
begin
        OLD_DEL_NZ := DEL_NZ ;
        H_2DOT     := 0.0 ;
        OLD_H_2DOT := 0.0 ;
        OLD_HRA    := HRA ;
        HRA_HP     := 0.0 ;
        OLD_HRA_HP := 0.0 ;
end INITIALIZE_FILTERS ;

procedure CALCULATE_GLIDESLOPE is
begin
        GS_ERR_LP        := GSO_KM1*OLD_GS_ERR_LP +
                            GSI_KM1*OLD_GS_ERR + GSI_K*GS_ERR ;
        GS_ERR_DS        := 0.1*GS_ERR_LP ;
        if   HRA < 1000.0
        then GS_ERR_DS   := (HRA-60.0)*KGS*GS_ERR_DS ;
        end if ;
        H_2DOT           := NZO_KM1*OLD_H_2DOT +
                            NZI_KM1*OLD_DEL_NZ + NZI_K*DEL_NZ ;
        OLD_DEL_NZ       := DEL_NZ ;
        H_2DOT_LP        := H2DO_KM1*OLD_H_2DOT_LP +
                            H2DI_KM1*OLD_H_2DOT + H2DI_K*H_2DOT ;
        OLD_H_2DOT       := H_2DOT ;
        OLD_H_2DOT_LP    := H_2DOT_LP ;
        HRA_HP           := HO_KM1*OLD_HRA_HP +
                            HI_KM1*OLD_HRA + HI_K*HRA ;
        OLD_HRA          := HRA ;
        OLD_HRA_HP       := HRA_HP ;
        H_DOT            := H_2DOT_LP + HRA_HP ;
        H_2DOT_AUG       := H_2DOT - GS_ERR_DS ;
        H_DOT_REF        := H2DAO_KM1*OLD_H_DOT_REF +
                           H2DAI_KM1*OLD_H_2DOT_AUG + H2DAI_K*H_2DOT_AUG;
        OLD_H_DOT_REF    := H_DOT_REF ;
        OLD_H_2DOT_AUG   := H_2DOT_AUG ;
        H_DOT_AGS        := H_DOT_REF - GS_ERR_DS ;
        H_DOT_CMD1       := H_DOT_REF ;
        OLD_H_DOT_CMD1   := H_DOT_CMD1 ;
        H_DOT_CMD2       := -8.0 ;
        H_DOT_ERR        := H_DOT_CMD1 + H_DOT_CMD2 - H_DOT ;
        PR_CMD           := 0.5*H_DOT_ERR ;
        if   abs(PR_CMD_LIM - PR_CMD) >= RATE_LIMIT
        then if   PR_CMD > 0.0
             then PR_CMD_LIM := PR_CMD_LIM + 0.3 ;
             else PR_CMD_LIM := PR_CMD_LIM - 0.3 ;
             end if ;
        else PR_CMD_LIM := PR_CMD ;
        end if ;
        PR_ERR           := PR_CMD_LIM - PITCH_RATE ;
end CALCULATE_GLIDESLOPE ;
```

Figure A-8 Package AL_RESOURCES (Sheet 5 of 6)

```
procedure CALCULATE_FLARE is
begin
        H_2DOT              := NZO_KM1*OLD_H_2DOT +
                               NZI_KM1*OLD_DEL_NZ + NZI_K*DEL_NZ ;
        OLD_DEL_NZ          := DEL_NZ ;
        H_2DOT_LP           := H2DO_KM1*OLD_H_2DOT_LP +
                               H2DI_KM1*OLD_H_2DOT + H2DI_K*H_2DOT ;
        OLD_H_2DOT          := H_2DOT ;
        OLD_H_2DOT_LP       := H_2DOT_LP ;
        HRA_HP              := HO_KM1*OLD_HRA_HP +
                               HI_KM1*OLD_HRA + HI_K*HRA ;
        OLD_HRA             := HRA ;
        OLD_HRA_HP          := HRA_HP ;
        H_DOT               := H_2DOT_LP + HRA_HP ;
        H_DOT_CMD1          := PREO_KM1*OLD_H_DOT_CMD1 ; --  Fader
        OLD_H_DOT_CMD1      := H_DOT_CMD1 ;
        H_DOT_CMD2          := (HRA + 20.0)*KFL ;
        H_DOT_ERR           := H_DOT_CMD1 + H_DOT_CMD2 - H_DOT ;
        PR_CMD              := 0.5*H_DOT_ERR ;
        if   abs(PR_CMD_LIM - PR_CMD) >= RATE_LIMIT
        then if   PR_CMD > 0.0
             then PR_CMD_LIM := PR_CMD_LIM + 0.3 ;
             else PR_CMD_LIM := PR_CMD_LIM - 0.3 ;
             end if ;
        else PR_CMD_LIM := PR_CMD ;
        end if ;
        PR_ERR              := PR_CMD_LIM - PITCH_RATE ;
end CALCULATE_FLARE ;

procedure FADER_LIMITER is
begin
        H_DOT_CMD1          := PREO_KM1*OLD_H_DOT_CMD1 ;
        OLD_H_DOT_CMD1      := H_DOT_CMD1 ;
        PR_CMD              := H_DOT_CMD1 ;
        if   abs(PR_CMD_LIM - PR_CMD) >= RATE_LIMIT
        then if   PR_CMD > 0.0
        then PR_CMD_LIM := PR_CMD_LIM + 0.3 ;
             else PR_CMD_LIM := PR_CMD_LIM - 0.3 ;
             end if ;
        else PR_CMD_LIM := PR_CMD ;
        end if ;
        PR_ERR              := PR_CMD_LIM - PITCH_RATE ;
end FADER_LIMITER ;

procedure RESET_FILTERS is
begin
        OLD_GS_ERR          := 0.0 ;
        OLD_GS_ERR_LP       := 0.0 ;
        OLD_DEL_NZ          := 0.0 ;
        OLD_H_2DOT          := 0.0 ;
        OLD_H_2DOT_LP       := 0.0 ;
        OLD_H_2DOT_AUG      := 0.0 ;
        OLD_H_DOT_REF       := 0.0 ;
        OLD_HRA             := 0.0 ;
        OLD_HRA_HP          := 0.0 ;
        OLD_H_DOT_CMD1      := 0.0 ;
        OLD_PR_ERR          := 0.0 ;
end RESET_FILTERS ;

end AL_RESOURCES ;
```

Figure A-8 Package AL_RESOURCES (Sheet 6 of 6)

```
                Ada Procedure MANAGE_IL_SENSORS_V3
                ----------------------------------

with CHNL_3_IL_VOTER ; use CHNL_3_IL_VOTER ;
with DFCS_LOGIC      ; use DFCS_LOGIC ;
with VOTING_PLANES   ; use VOTING_PLANES ;
separate(DFCS_RESOURCES)
procedure MANAGE_IL_SENSORS_V3 is

        CP_STK_FLAGS, P_STK_FLAGS, AOA_FLAGS,
        CP_STK_COMP,  P_STK_COMP,  AVG_AOA_COMP : BOOL_VECTOR(1..4) ;
        P_RATE_FLAGS, P_RATE_COMP               : BOOL_VECTOR(1..3) ;
        TAS_FLAGS,    TAS_COMP                  : BOOL_VECTOR(1..2) ;

        CP_STK_SIGNALS, P_STK_SIGNALS,
        AVG_AOA_SIGNALS                         : REAL_VECTOR(1..4) ;
        P_RATE_SIGNALS                          : REAL_VECTOR(1..3) ;
        TAS_SIGNALS                             : REAL_VECTOR(1..2) ;

        CP_STK_MED, P_STK_MED, AVG_AOA_MED,
        P_RATE_MED, TAS_MED                     : FLOAT ;

begin

for INDEX in 1..4                          -- Input Flag Type Conversion
    loop
        CP_STK_FLAGS(INDEX)  := IL_FLAGS.CP_STK_VAL(INDEX) ;
        P_STK_FLAGS(INDEX)   := IL_FLAGS.P_STK_VAL(INDEX)  ;
        AOA_FLAGS(INDEX)     := IL_FLAGS.LF_AOA_VAL(INDEX)  and
                                IL_FLAGS.RT_AOA_VAL(INDEX) ;
        if   INDEX <= 3 then
        P_RATE_FLAGS(INDEX)  := IL_FLAGS.P_RATE_VAL(INDEX) ;
        if   INDEX <= 2 then
        TAS_FLAGS(INDEX)     := IL_FLAGS.TAS_VAL(INDEX);
        end if ;
        end if ;
    end loop ;
                                           -- Input Validity Flag Checks
CHECK_IL_FLAGS(CP_STK_FLAGS, 1, 4) ;
CHECK_IL_FLAGS(P_STK_FLAGS,  2, 4) ;
CHECK_IL_FLAGS(AOA_FLAGS,    3, 4) ;
CHECK_IL_FLAGS(P_RATE_FLAGS, 4, 3) ;
CHECK_IL_FLAGS(TAS_FLAGS,    5, 2) ;

for INDEX in 1..4                        -- Sensor Signal Type Conversion
    loop
        CP_STK_SIGNALS(INDEX)  := FLOAT(CP_STICK_CMD(INDEX)) ;
        P_STK_SIGNALS(INDEX)   := FLOAT(P_STICK_CMD(INDEX)) ;
        AVG_AOA_SIGNALS(INDEX) := (FLOAT(LEFT_AOA(INDEX)) +
                                  FLOAT(RIGHT_AOA(INDEX)))/2.0 ;
        if   INDEX <= 3 then
        P_RATE_SIGNALS(INDEX)  := FLOAT(P_RATE_GYRO(INDEX)) ;
        if   INDEX <= 2 then
        TAS_SIGNALS(INDEX)     := FLOAT(TRUE_AIRSPEED(INDEX)) ;
        end if ;
        end if ;
    end loop ;
```

Figure A-9 Procedure MANAGE_IL_SENSORS_V3 (Sheet 1 of 2)

```
                                                    -- Median Signal Selection

VOTE_IL_SENSORS(CP_STK_SIGNALS,  1, 4, CP_STK_MED) ;
VOTE_IL_SENSORS(P_STK_SIGNALS,   2, 4, P_STK_MED) ;
VOTE_IL_SENSORS(AVG_AOA_SIGNALS, 3, 4, AVG_AOA_MED)   ;
VOTE_IL_SENSORS(P_RATE_SIGNALS,  4, 3, P_RATE_MED) ;
VOTE_IL_SENSORS(TAS_SIGNALS,     5, 2, TAS_MED) ;

                                                      -- Median Select Outputs

IL_MED_V3.CP_STICK     := STICK_CMD(CP_STK_MED) ;
IL_MED_V3.P_STICK      := STICK_CMD(P_STK_MED)  ;
IL_MED_V3.AOA_DISPL    := AOA_SIGNAL(AVG_AOA_MED)  ;
IL_MED_V3.P_RATE       := ANG_RATE_SIGNAL(P_RATE_MED) ;
IL_MED_V3.TR_AIRSPEED  := TAS_SIGNAL(TAS_MED) ;

                                                    -- Comparator Logic Checks

CHK_FAULT_LOGIC(CP_STK_SIGNALS,  CP_STK_MED,  1, 4, CP_STK_COMP) ;
CHK_FAULT_LOGIC(P_STK_SIGNALS,   P_STK_MED,   2, 4, P_STK_COMP) ;
CHK_FAULT_LOGIC(AVG_AOA_SIGNALS, AVG_AOA_MED, 3, 4, AVG_AOA_COMP) ;
CHK_FAULT_LOGIC(P_RATE_SIGNALS,  P_RATE_MED,  4, 3, P_RATE_COMP) ;
CHK_FAULT_LOGIC(TAS_SIGNALS,     TAS_MED,     5, 2, TAS_COMP) ;

for INDEX in 1..4                             -- Comparator Validity Output
    loop
        IL_COMP_V3.CP_STK_VAL(INDEX)  := CP_STK_COMP(INDEX) ;
        IL_COMP_V3.P_STK_VAL(INDEX)   := P_STK_COMP(INDEX) ;
        IL_COMP_V3.AVG_AOA_VAL(INDEX) := AVG_AOA_COMP(INDEX) ;
        if   INDEX <= 3 then
        IL_COMP_V3.P_RATE_VAL(INDEX)  := P_RATE_COMP(INDEX) ;
        if   INDEX <= 2 then
        IL_COMP_V3.TAS_VAL(INDEX)     := TAS_COMP(INDEX) ;
        end if ;
        end if ;
    end loop ;

CHNL_3_XCHK_NUM := 5 ;                                -- N-Version Voting
XCHK_SYNCH_3 ;

end MANAGE_IL_SENSORS_V3 ;
```

Figure A-9 Procedure MANAGE_IL_SENSORS_V3 (Sheet 2 of 2)

Package CHNL_3_IL_VOTER

```
package CHNL_3_IL_VOTER is

        NUM_SENSORS                 : INTEGER range 2..4 := 2 ;
        NUM_VOTES                   : INTEGER range 0..4 := 0 ;
        SET_NUM                     : INTEGER range 1..5 := 1 ;

        type BOOL_VECTOR            is array (INTEGER range <>) of BOOLEAN ;
        type REAL_VECTOR            is array (INTEGER range <>) of FLOAT ;

        IL_COMP_COUNT    : array (1..5, 1..4) of INTEGER range 0..17
                                    := (others => (others => 0)) ;
        IL_FLAG_COUNT    : array (1..5, 1..4) of INTEGER range -5..5
                                    := (others => (others => 0)) ;
        IL_FLAG_IN       : array (1..5, 1..4) of BOOLEAN
                                    := (others => (others => TRUE)) ;
        IL_COMP_OUT      : array (1..5, 1..4) of BOOLEAN
                                    := (others => (others => TRUE)) ;

procedure CHECK_IL_FLAGS(IL_FLAGS : in BOOL_VECTOR ; SET_NUM,
                         NUM_SENSORS : in INTEGER) ;
procedure VOTE_IL_SENSORS(IL_SENSORS : in REAL_VECTOR ; SET_NUM,
            NUM_SENSORS : in INTEGER ; IL_SENSOR_MED : out FLOAT) ;
procedure CHK_FAULT_LOGIC(IL_SENSORS : in REAL_VECTOR ; IL_SENSOR_MED :
            in FLOAT ; SET_NUM, NUM_SENSORS : in INTEGER ;
                                    IL_COMP_VAL : out BOOL_VECTOR) ;

end CHNL_3_IL_VOTER ;
```

Figure A-10 Package CHNL_3_IL_VOTER (Sheet 1 of 4)

```
package body CHNL_3_IL_VOTER is

procedure CHECK_IL_FLAGS(IL_FLAGS : in BOOL_VECTOR ; SET_NUM,
                         NUM_SENSORS : in INTEGER) is

begin
    for INDEX in 1..NUM_SENSORS
    loop
        case IL_FLAG_COUNT(SET_NUM, INDEX) is
        when 0 =>                                                  -- Normal
             if   IL_FLAGS(INDEX) = FALSE
             then IL_FLAG_COUNT(SET_NUM, INDEX) := 1 ;
             end if ;
        when 1..5 =>                                               -- Faulted
             if   IL_FLAGS(INDEX) = FALSE
             then IL_FLAG_COUNT(SET_NUM, INDEX) :=
                  IL_FLAG_COUNT(SET_NUM, INDEX) + 1 ;
                  if   IL_FLAG_COUNT(SET_NUM, INDEX) >= 5
                  then IL_FLAG_IN(SET_NUM, INDEX) := FALSE ;
                       IL_FLAG_COUNT(SET_NUM, INDEX) := -1 ;
                  end if ;
             else IL_FLAG_COUNT(SET_NUM, INDEX) := 0 ;
             end if ;
        when -5..-1 =>                                             -- Healing
             if   IL_FLAGS(INDEX) = TRUE
             then IL_FLAG_COUNT(SET_NUM, INDEX) :=
                  IL_FLAG_COUNT(SET_NUM, INDEX) -1 ;
                  if   IL_FLAG_COUNT(SET_NUM, INDEX) <= -5
                  then IL_FLAG_IN(SET_NUM, INDEX) := TRUE ;
                       IL_FLAG_COUNT(SET_NUM, INDEX) := 0 ;
                  end if ;
             else IL_FLAG_COUNT(SET_NUM, INDEX) := - 1 ;
             end if ;
        end case ;
    end loop ;

end CHECK_IL_FLAGS ;
```

Figure A-10 Package CHNL_3_IL_VOTER (Sheet 2 of 4)

```
procedure VOTE_IL_SENSORS(IL_SENSORS : in REAL_VECTOR ; SET_NUM,
            NUM_SENSORS : in INTEGER ; IL_SENSOR_MED : out FLOAT) is

    SET_RANKING : array (1..4) of INTEGER range 0..4 := (0, 0, 0, 0) ;
    V           : array (1..4) of FLOAT := (0.0, 0.0, 0.0, 0.0) ;
    TEMP        : FLOAT := 0.0 ;

begin

    NUM_VOTES := NUM_SENSORS ;
    for INDEX in 1..NUM_SENSORS                        -- which Sensors Voting
        loop
            if   IL_COMP_OUT(SF1_NUM, INDEX) = FALSE
            then NUM_VOTES           := NUM_VOTES - 1 ;
            else SET_RANKING(INDEX) := INDEX ;
            end if ;
        end loop ;
    for INDEX in 1..NUM_VOTES                          -- which Values Voting
        loop
            for CHNL_NUM in INDEX..4
                loop
                    if   CHNL_NUM = SET_RANKING(CHNL_NUM)
                    then V(INDEX) := IL_SENSORS(CHNL_NUM) ;
                    exit ;
                    end if ;
                end loop ;
        end loop ;
    case NUM_VOTES is                                  -- Sensor Signal Voting
        when 0 =>
            null ;
        when 1 =>
            IL_SENSOR_MED := V(1) ;
        when 2 =>
            if   V(1) <= V(2)
            then IL_SENSOR_MED := V(1) ;
            else IL_SENSOR_MED := V(2) ;
            end if ;
        when 3 | 4 =>
            for I in 1..NUM_VOTES-1
                loop
                    for J in I+1..NUM_VOTES
                        loop
                            if   V(I) >= V(J)
                            then TEMP := V(I) ;
                                 V(I) := V(J) ;
                                 V(J) := TEMP ;
                            end if ;
                        end loop ;
                end loop ;
            IL_SENSOR_MED := V(2) ;
        end case ;

end VOTE_IL_SENSORS ;
```

Figure A-10 Package CHNL_3_IL_VOTER (Sheet 3 of 4)

```
procedure CHK_FAULT_LOGIC(IL_SENSORS : in REAL_VECTOR ; IL_SENSOR_MED :
            in FLOAT ; SET_NUM, NUM_SENSORS : in INTEGER ;
                                IL_COMP_VAL : out BOOL_VECTOR) is

        AMPL_LIMIT        :  constant array (1..5) of FLOAT
                          := ( 0.2, 0.2, 1.25, 1.0, 10.0) ;
        MAX_CT            :  constant array (1..5) of INTEGER
                          := (6, 6, 8, 8, 16) ;

begin

    for INDEX in 1..SET_NUM
    loop
        case IL_COMP_COUNT(SET_NUM, INDEX) is
        when 0 =>                                                  -- Normal
            if   abs(IL_SENSOR_MED - IL_SENSORS(INDEX)) >=
                 AMPL_LIMIT(SET_NUM)
            then IL_COMP_COUNT(SET_NUM, INDEX) := 1 ;
            end if ;
        when 1..16 =>                                             -- Faulted
            if   abs(IL_SENSOR_MED - IL_SENSORS(INDEX)) >=
                 AMPL_LIMIT(SET_NUM)
            then IL_COMP_COUNT(SET_NUM, INDEX) :=
                 IL_COMP_COUNT(SET_NUM, INDEX) + 1 ;
                 if   IL_COMP_COUNT(SET_NUM, INDEX) >= MAX_CT(SET_NUM)
                 then IL_COMP_OUT(SET_NUM, INDEX)   := FALSE ;
                      IL_COMP_COUNT(SET_NUM, INDEX) := 17 ;
                 end if ;
            else IL_COMP_COUNT(SET_NUM, INDEX) := 0 ;       -- Recovering
            end if ;
        when 17 =>                                                -- Failed
            null ;
        end case ;
        IL_COMP_VAL(INDEX) := IL_COMP_OUT(SET_NUM, INDEX)  or
                              IL_FLAG_IN(SET_NUM, INDEX) ;
    end loop ;

end CHK_FAULT_LOGIC ;

end CHNL_3_IL_VOTER ;
```

Figure A-10 Package CHNL_3_IL_VOTER (Sheet 4 of 4)

```
                Ada Procedure CALC_INNER_LOOP_V3
                --------------------------------

with DFCS_LOGIC          ; use DFCS_LOGIC ;
with DFCS_RESOURCES      ; use DFCS_RESOURCES ;
with IL_RESOURCES        ; use IL_RESOURCES ;
with VOTING_PLANES       ; use VOTING_PLANES ;
separate(CONTROL_LAWS)
procedure CALC_INNER_LOOP_V3 is

        type SCHEDULE   is (HIGH, MID, LOW) ;

        P_STICK_MED, CP_STICK_MED, TOT_STICK,
        P_RATE_MED,  AVG_AOA_MED,  AOA_HP,
        TAS_MED,     DEL_TAS,
        K_STICK,     K_ALPHA,      K_P_RATE,
        IL_STAB_CMD, OL_STAB_CMD,  TOT_STAB_CMD : FLOAT ;

        SPEED                                   : SCHEDULE ;

        TAS_VALIDITY                            : BOOLEAN ;

begin
                                                          -- Conversions
        P_STICK_MED  := FLOAT(IL_MED_V3.P_STICK) ;
        CP_STICK_MED := FLOAT(IL_MED_V3.CP_STICK) ;
        P_RATE_MED   := FLOAT(IL_MED_V3.P_RATE) ;
        AVG_AOA_MED  := FLOAT(IL_MED_V3.AOA_DISPL) ;
        TAS_MED      := FLOAT(IL_MED_V3.TR_AIRSPEED) ;
        TAS_VALIDITY := IL_COMP_V3.TAS_VAL(1) and
                        IL_COMP_V3.TAS_VAL(2) ;

        if   TAS_VALIDITY = TRUE                          -- Gain Scheduling
        then if    TAS_MED >= 450.0
             then  SPEED   := HIGH ;
             elsif TAS_MED <= 100.0
             then  SPEED   := LOW ;
             else  SPEED   := MID ;
             end if ;
        else SPEED := HIGH ;
        end if ;
        case SPEED is
             when LOW   =>
                 K_STICK  := 0.5 ;
                 K_ALPHA  := 0.15 ;
                 K_P_RATE := 0.2 ;
             when  MID  =>
                 DEL_TAS  := TAS_MED - 100.0 ;
                 K_STICK  := 0.50 - DEL_TAS * DEL_K_STICK ;
                 K_ALPHA  := 0.05 - DEL_TAS * DEL_K_ALPHA ;
                 K_P_RATE := 0.10 - DEL_TAS * DEL_K_P_RATE ;
             when HIGH  =>
                 K_STICK  := 0.1 ;
                 K_ALPHA  := 0.05 ;
                 K_P_RATE := 0.1 ;
        end case ;
```

Figure A-11 Procedure CALC_INNER_LOOP_V3 (Sheet 1 of 2)

```
        if   abs(P_STICK_MED)  <= 0.05                    -- Stick Blending
        then P_STICK_MED       := 0.0 ;
        end if ;
        if   abs(CP_STICK_MED) <= 0.05
        then CP_STICK_MED      := 0.0 ;
        end if ;
        TOT_STICK := P_STICK_MED + CP_STICK_MED ;
        if   abs(TOT_STICK) >   12.5
        then if   TOT_STICK >    1.0
             then TOT_STICK :=  12.5 ;
             else TOT_STICK := -12.5 ;
             end if ;
        end if ;

        if   not INITIALIZED                            --  Alpha High-Pass
        then OLD_AVG_AOA_MED := AVG_AOA_MED ;
             OLD_AOA_HP      := 0.0 ;
        end if ;
        AOA_HP := AOAI_K*AVG_AOA_MED + AOAI_KM1*OLD_AVG_AOA_MED
                                     - AOAO_KM1*OLD_AOA_HP ;

                                                     -- Inner Loop Control

        IL_STAB_CMD := P_RATE_MED * K_P_RATE  +  AOA_HP * K_ALPHA -
                       TOT_STICK  * K_STICK ;

        if   MODE_ENG_V3.AUTOPILOT = AUTOLAND       -- Outer Loop Summing
        then OL_STAB_CMD   := - 0.35 * FLOAT(AUTOLAND_CMD_V3) ;
        else OL_STAB_CMD   :=   0.0 ;
        end if ;
        TOT_STAB_CMD := IL_STAB_CMD + OL_STAB_CMD ;
        if    TOT_STAB_CMD >   1.0
        then  TOT_STAB_CMD :=  1.0 ;
        elsif TOT_STAB_CMD <  -8.0
        then  TOT_STAB_CMD := -8.0 ;
        end if ;
        STAB_SERVO_CMD_V3  :=  STAB_COMMAND(TOT_STAB_CMD) ;     -- Output

CHNL_3_XCHK_NUM := 6 ;
XCHK_SYNCH_3 ;

end CALC_INNER_LOOP_V3 ;
```

Figure A-11 Procedure CALC_INNER_LOOP_V3 (Sheet 2 of 2)

Package IL_RESOURCES

```
package IL_RESOURCES is

        DEL_K_STICK      : constant FLOAT := 0.4/350.0 ;
        DEL_K_ALPHA      : constant FLOAT := 0.1/350.0 ;
        DEL_K_P_RATE     : constant FLOAT := 0.1/350.0 ;

        --  Angle-of-Attack High-Pass Filter Coefficients

                AOAI_K           : constant   :=  800.0/801.0 ;
                AOAI_KM1         : constant   := -800.0/801.0 ;
                AOAO_KM1         : constant   :=  799.0/801.0 ;

        OLD_AVG_AOA_MED, OLD_AOA_HP      : FLOAT := 0.0 ;

        INITIALIZED                      : BOOLEAN := FALSE ;

end IL_RESOURCES ;
```

Figure A-12 Package IL_RESOURCES

```
                        ASSESS_SYSTEM_V3
                        ----------------

with CHANNEL_RESOURCES ; use CHANNEL_RESOURCES ;
with VOTING_PLANES     ; use VOTING_PLANES ;
separate(DFCS_LOGIC)
procedure ASSESS_SYSTEM_V3 is

        CP_STK_CT,P_STK_CT,AOA_CT,
        CMPTR_CT                          : INTEGER range 0..1 := 0 ;
        P_RATE_CT                         : INTEGER range 0..3 := 0 ;
        TAS_CT                            : INTEGER range 0..2 := 0 ;

        GO_ON                             : BOOLEAN := FALSE ;

begin
        if   CHNL_STATUS_V1 = TRUE                        -- Check LRU Status
        then CMPTR_CT := 1 ;
        end if ;
        if   CHNL_STATUS_V2 = TRUE
        then CMPTR_CT := CMPTR_CT + 1 ;
        end if ;
        if   CHNL_STATUS_V3 = TRUE
        then CMPTR_CT := CMPTR_CT + 1 ;
        end if ;
        if   CHNL_STATUS_V4 = TRUE
        then CMPTR_CT := CMPTR_CT + 1 ;
        end if ;

        for  INDEX in 1..4
             loop
                 if   IL_COMP_V3.CP_STK_VAL(INDEX) = TRUE
                 then CP_STK_CT := CP_STK_CT + 1 ;
                 end if ;
                 if   IL_COMP_V3.P_STK_VAL(INDEX)  = TRUE
                 then P_STK_CT  := P_STK_CT  + 1 ;
                 end if ;
                 if   IL_COMP_V3.LF_AOA_VAL(INDEX) = TRUE  and
                      IL_COMP_V3.RT_AOA_VAL(INDEX) = TRUE
                 then AOA_CT    := AOA_CT    + 1 ;
                 end if ;
                 if   INDEX <= 3 and then
                      IL_COMP_V3.P_RATE_VAL(INDEX) = TRUE
                 then P_RATE_CT := P_RATE_CT + 1 ;
                 end if ;
                 if   INDEX <= 2 and then
                      IL_COMP_V3.TAS_VAL(INDEX)    = TRUE
                 then TAS_CT    := TAS_CT    + 1 ;
                 end if ;
             end loop ;
```

Figure A-13 Procedure ASSESS_SYSTEM_V3 (Sheet 1 of 2)

```
        if   P_RATE_CT = 3  and  IAS_CT = 2            --Check FBW Status
        then GO_ON := FALSE ;
             if    AOA_CT = 4  and  CMPTR_CT = 4  and
                   (CP_STK_CT = 4  or else  P_STK_CT = 4 or else
                   (CP_STK_CT >= 3  and  P_STK_CT >= 3))
             then  FBW_STATUS_V3 := OP_STATE_1 ;
             elsif AOA_CT = 3  and  CMPTR_CT >= 3  and
                   (P_STK_CT >= 3  or else  CP_STK_CT >= 3  or else
                   (P_STK_CT >= 2  and  CP_STK_CT >=2))
             then  FBW_STATUS_V3 := OP_STATE_2 ;
             elsif AOA_CT >= 3  and  CMPTR_CT = 3  and
                   (P_STK_CT >= 3  or else  CP_STK_CT >= 3  or else
                   (P_STK_CT >= 2  and  CP_STK_CT >=2))
             then  FBW_STATUS_V3 := OP_STATE_2 ;
             elsif AOA_CT >= 3  and  CMPTR_CT >= 3  and
                   ((P_STK_CT = 3  and  CP_STK_CT <= 1) or else
                   (CP_STK_CT = 3  and  P_STK_CT <= 1)  or else
                   (P_STK_CT  = 2  and  CP_STK_CT =2))
             then  FBW_STATUS_V3 := OP_STATE_2 ;
             else  GO_ON := TRUE ;
             end if ;
        else GO_ON := TRUE ;
        end if ;
        if   GO_ON = TRUE
        then if    (AOA_CT = 2  and  CMPTR_CT >= 2  and
                   (P_STK_CT >= 2  or CP_STK_CT >= 2)) or
                   (AOA_CT >= 2  and  CMPTR_CT = 2  and
                   (P_STK_CT >= 2  or CP_STK_CT >= 2)) or
                   (AOA_CT >= 2  and  CMPTR_CT >= 2 and
                   ((P_STK_CT = 2  and then  CP_STK_CT <= 1) or
                   (P_STK_CT <= 1  and then  CP_STK_CT = 2)))
             then  FBW_STATUS_V3 := OP_STATE_3 ;
             else  FBW_STATUS_V3 := OP_STATE_4 ;
             end if ;
        end if ;

        case FBW_STATUS_V3 is                    --  Check Flying Quality
        when OP_STATE_1 | OP_STATE_2 =>
             FLY_QUAL_V3 := NORMAL ;
        when OP_STATE_3 | OP_STATE_4 =>
             if    P_RATE_CT >= 2  and  AOA_CT >= 2  and
                   IAS_CT = 2  and  (P_STK_CT >= 2  or  CP_STK_CT >= 2)
             then  FLY_QUAL_V3 := NORMAL ;
             elsif (P_RATE_CT <= 1  or  IAS_CT <= 1) and
                   AOA_CT >= 2  and  (P_STK_CT >= 2  or  CP_STK_CT >= 2)
             then  FLY_QUAL_V3 := DEGRADED ;
             elsif P_RATE_CT >= 2  and  AOA_CT <= 1  and
                   (P_STK_CT >= 2  or  CP_STK_CT >= 2)
             then  FLY_QUAL_V3 := MARGINAL ;
             else  FLY_QUAL_V3 := UNFLYABLE ;
             end if ;
        end case ;

        CHNL_3_XCHK_NUM := 7 ;
        XCHK_SYNCH_3 ;

end ASSESS_SYSTEM_V3 ;
```

Figure A-13 Procedure_ASSESS_SYSTEM_V3 (Sheet 2 of 2)

```
                    Ada Procedure GIVE_WARNING_V3
                    -----------------------------

with VOTING_PLANES  ; use VOTING_PLANES ;
with WARNING_CHECKS ; use WARNING_CHECKS ;
separate(UFCS_LOGIC)
procedure GIVE_WARNING_V3 is

begin

   case FLASH_WARNING_V3 is
        when OFF =>
             if   AL_WARN_V3            <= CAT_2_INOP
             then WARN_V3.AUTOLAND      := AL_WARN_V3 ;        -- New Fault
                  NUM_FAULTS            := 1 ;
                  FLASH_WARNING_V3      := BLINKING ;
             end if ;
             case FBW_STATUS_V3  is
                  when OP_STATE_1 =>
                       null ;
                  when OP_STATE_2 =>
                       WARN_V3.FLY_BY_WIRE := OP_STATE_2 ;    -- New Fault
                       FLASH_WARNING_V3    := BLINKING ;
                       NUM_FAULTS          := INTEGER'SUCC(NUM_FAULTS) ;
                  when OP_STATE_3 =>
                       WARN_V3.FLY_BY_WIRE := OP_STATE_3 ;    -- New Fault
                       FLASH_WARNING_V3    := BLINKING ;
                       NUM_FAULTS          := INTEGER'SUCC(NUM_FAULTS) ;
                  when OP_STATE_4 =>
                       WARN_V3.FLY_BY_WIRE := OP_STATE_4 ;    -- New Fault
                       FLASH_WARNING_V3    := BLINKING ;
                       NUM_FAULTS          := INTEGER'SUCC(NUM_FAULTS) ;
             end case ;
             if   FLY_QUAL_V3           <= DEGRADED
             then WARN_V3.FLYING_QUAL   := IMPAIRED_FQ ;       -- New Fault
                  NUM_FAULTS            := INTEGER'SUCC(NUM_FAULTS) ;
                  FLASH_WARNING_V3      := BLINKING ;
             end if ;
        when BLINKING =>
             if   ACKNOWLEDGE
             then FLASH_WARNING_V3      := STEADY ;        -- Fault(s) Noted
             end if ;
             UPDATE_STATUS(AL_WARN_V3, FBW_STATUS_V3, FLY_QUAL_V3,
                           WARN_V3, FLASH_WARNING_V3) ;
        when STEADY =>
             UPDATE_STATUS(AL_WARN_V3, FBW_STATUS_V3, FLY_QUAL_V3,
                           WARN_V3, FLASH_WARNING_V3) ;
    end case ;

CHNL_3_XCHK_NUM := 6 ;
XCHK_SYNCH_3 ;                                  -- Call for N-Version Vote

end GIVE_WARNING_V3 ;
```

Figure A-14 Procedure GIVE_WARNING_V3

Package WARNING_CHECKS

```
with DFCS_LOGIC ;  use DFCS_LOGIC ;
package WARNING_CHECKS is

        NUM_FAULTS       : INTEGER range 0..3 := 0 ;

        procedure UPDATE_STATUS(AL_WARN_V3: in AL_STATUS; FBW_STATUS_V3:
                    in PRI_FCS_STATUS; FLY_QUAL_V3: in FLYING_QUALITIES;
                    WARN_V3: in out WARNING_STATE ; FLASH_WARNING_V3:
                                                    out MASTER_WARN) ;
end WARNING_CHECKS ;

package body WARNING_CHECKS is

procedure UPDATE_STATUS(AL_WARN_V3: in AL_STATUS; FBW_STATUS_V3: in
          PRI_FCS_STATUS; FLY_QUAL_V3: in FLYING_QUALITIES; WARN_V3:
          in out WARNING_STATE ; FLASH_WARNING_V3: out MASTER_WARN)  is
begin
   if    WARN_V3.AUTOLAND        =  BLANK  and then
         AL_WARN_V3              <= CAT_2_INOP
   then  FLASH_WARNING_V3        := BLINKING ;                  -- New Fault
         NUM_FAULTS              := INTEGER'SUCC(NUM_FAULTS) ;
   elsif WARN_V3.AUTOLAND        <= CAT_2_INOP  and then
         AL_WARN_V3              =  BLANK
   then  WARN_V3.AUTOLAND        := BLANK ;              -- One Fault Healed
         NUM_FAULTS              := INTEGER'PRED(NUM_FAULTS) ;
         if    NUM_FAULTS        =  0
         then  FLASH_WARNING_V3  := OFF ;                -- All Faults Healed
         end if ;
   end if ;
   if    WARN_V3.FLY_BY_WIRE     =  BLANK  and then
         FBW_STATUS_V3           =  OP_STATE_2
   then  WARN_V3.FLY_BY_WIRE     := OP_STATE_2 ;                -- New Fault
         FLASH_WARNING_V3        := BLINKING ;
         NUM_FAULTS              := INTEGER'SUCC(NUM_FAULTS) ;
   elsif WARN_V3.FLY_BY_WIRE     >= OP_STATE_2  and then
         FBW_STATUS_V3           =  OP_STATE_3
   then  WARN_V3.FLY_BY_WIRE     := OP_STATE_3 ;
         FLASH_WARNING_V3        := BLINKING ;
         NUM_FAULTS              := INTEGER'SUCC(NUM_FAULTS) ;
   elsif WARN_V3.FLY_BY_WIRE     >= OP_STATE_3  and then
         FBW_STATUS_V3           =  OP_STATE_4
   then  WARN_V3.FLY_BY_WIRE     := OP_STATE_4 ;
         FLASH_WARNING_V3        := BLINKING ;
         NUM_FAULTS              := INTEGER'SUCC(NUM_FAULTS) ;
```

Figure A-15 Package WARNING_CHECKS (Sheet 1 of 2)

```
    elsif WARN_V3.FLY_BY_WIRE  <= OP_STATE_2
    then case FBW_STATUS_V3 is
         when OP_STATE_1 =>
              WARN_V3.FLY_BY_WIRE    := BLANK ;          -- One Fault Healed
              NUM_FAULTS             := INTEGER'PRED(NUM_FAULTS) ;
              if   NUM_FAULTS        =  0
              then FLASH_WARNING_V3  := OFF ;            -- All Faults Healed
              end if ;
         when OP_STATE_2 =>
              if   WARN_V3.FLY_BY_WIRE =  OP_STATE_3
              then WARN_V3.FLY_BY_WIRE := OP_STATE_2 ;
              end if ;
              if   WARN_V3.FLY_BY_WIRE =  OP_STATE_4
              then WARN_V3.FLY_BY_WIRE := OP_STATE_2 ;
              end if ;
         when OP_STATE_3 =>
              if   WARN_V3.FLY_BY_WIRE =  OP_STATE_3
              then WARN_V3.FLY_BY_WIRE := OP_STATE_3 ;
              end if ;
         when OP_STATE_4 =>
              null ;
         end case ;
    end if ;
    if   WARN_V3.FLYING_QUAL    =  BLANK  and then
         FLY_QUAL_V3            <= DEGRADED
    then FLASH_WARNING_V3       := BLINKING ;                -- New Fault
         NUM_FAULTS             := INTEGER'SUCC(NUM_FAULTS) ;
    elsif WARN_V3.FLYING_QUAL   <= IMPAIRED_FQ and then
         FLY_QUAL_V3            =  NORMAL
    then WARN_V3.FLYING_QUAL    := BLANK ;                  -- One Fault Healed
         NUM_FAULTS             := INTEGER'PRED(NUM_FAULTS) ;
         if   NUM_FAULTS        =  0
         then FLASH_WARNING_V3  := OFF ;                    -- All Faults Healed
         end if ;
    end if ;
end UPDATE_STATUS ;

end WARNING_CHECKS ;
```

Figure A-15 Package WARNING_CHECKS (Sheet 2 of 2)