EMBEDDED COMPUTER PERFORMANCE MEASUREMENT (ECPM)

Advanced Avionics Subsystems and Technology Multiprocessor ECPM Software Documentation

NAWC
NAVAL AIR WARFARE CENTER

INDIANAPOLIS

Best
Available
Copy

PHASE II, Version 2.1

Advanced Avionics Subsystems & Technology (AAS&T) Program

Embedded Computer Performance Measurement (ECPM)

MIL-STD-1553 Interface Definition

10 June 1993

prepared by:

Naval Air Warfare Center, Indianapolis
6000 E. 21st. Street
Indianapolis IN 46219-2189

June 93 FINAL (Oct 90 to Jun 93)

Embedded Computer Performance Measurement (ECPM) Advanced Avionics Subsystems and Technology Multiprocessor ECPM Software Documentation C-N00163-90-C-0165

Diane Kohalmi, John Newport, Chuck Roark, Diane Paul, Dave Struble

Naval Air Warfare Center, Aircraft Division, Indpls
Indianapolis, Indiana

NONE

Naval Air Systems Command (AIR-546-TD)
Washington, DC 20361-0001 NONE

Unlimited Distribution

The report consists of software documentation for a new computer performance measurement tool written in Ada. The tool is designed for easy portability between computer systems. Included are a MIL-STD-1553B Interface Definition, A DoD-STD-2167 Systems Requirements Specification, and a DoD-STD-2167 Interface Requirements Specification.

ECPM

Unclassified Unclassified Unclassified

CONTENTS

A. Message and Word Counts Definitions 8
B. Details of Word Formats 14
C. Bit Packing Algorithms 35
D. Word Packing Procedure Calls 38

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

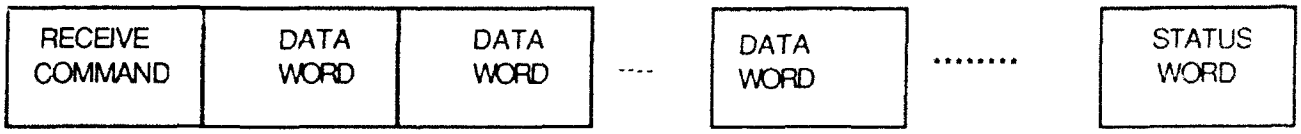
The Naval Air Warfare Center (NAWC), Indianapolis, Advanced Avionics Subsystems and Technology (AAS&T) Computers and Software element has developed a software tool to be used in the measurement of embedded computer system reserve requirements. This tool, the Embedded Computer Performance Measurement (ECPM), is written in Ada and is designed to provide input/output and scheduling requirements similar to those of operational software.

This document defines the details of the AAS&T ECPM software interface for MIL-STD-1553 data transfers used for the multiprocessor Engineering Change Proposal (ECP). This interface permits communication between the Digital Avionic System Laboratory (DASL) VAX computers and the computer under test. These communications involve transfer of simulated sensor data to the unit under test, and transfer of navigation solutions and performance information from the unit under test.

Military standard data conventions are assumed. For example, the Most Significant Bit (MSB) of all digital quantities must be transmitted first, as required by MIL-STD-1553. Also, the MSB is assumed to be the sign bit, in accordance with MIL-STD-1750 (VAX data conventions are consistent with this assumption).

All transactions are Bus Controller (BC) to Remote Terminal (RT) or RT to BC, as defined in MIL-STD-1553. Each message is composed of command, status, and data words, as specified by the MIL-STD-1553 (see Figure 1). Figure 2 illustrates command, status, and data word formats. The actual transmissions are controlled by the BC, which in the case of the ECPM is the DASL VAX. Optional MIL-STD-1553 features, such as dynamic bus control, mode codes, or broadcast, are not used for this interface (appropriate bits set to zero in command and status words). RT 31 is not to be used as this is reserved for broadcast. Similarly, subaddresses 0 and 31 are not to be used since they are reserved for mode codes.

The next section provides detailed descriptions of all message formats and data words to be used in this interface.



BC TO RT TRANSFER

RT TO BC TRANSFER

* RESPONSE TIME

Figure 1. MIL-STD-1553 Message Formats

```
BITS:                0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15

COMMAND WORD:        |-----------|  |  |-----------|  |-----------|
                         RT #            Subaddress     Word Count
                        (0-4)            (6-10)         (11-15)

COMMAND BIT  5: Transmit/Receive


DATA WORD(s):        |-------------------------------------------|
                        (0-15)


STATUS WORD:         |-----------|  |  |  |  |-----|  |  |  |  |  |
                         RT #                reserved
                        (0-4)                (8-10)
```

STATUS BIT 5: Message Error
STATUS BIT 6: Instrumentation
STATUS BIT 7: Service Request
STATUS BIT 11: Broadcast Command Received
STATUS BIT 12: Busy
STATUS BIT 13: Subsystem Flag
STATUS BIT 14: Dynamic Bus Control
STATUS BIT 15: Terminal Flag

NOTE: Each word is actually 20 bits in length, but the initial 3 synchronization bits and final parity bit have been deleted for clarity.

Figure 2. MIL-STD-1553 Word Formats.

II. MIL-STD-1553 MESSAGE MIX

Table I provides an overview of the message structures and word counts of all messages used in this interface. The Unit Under Test (UUT) is designated RT5. The first column indicates the message number, which corresponds with a subaddress number. The second column indicates the number of words sent in that message. The "T/R" column indicates whether RT5 transmits or receives the message. The update rate in Hertz is indicated for each message. The last column gives a high level description of the contents of the message.

TABLE I. MESSAGE SUMMARY

MESSAGE/SA	WORD COUNT	T/R	RATE (Hz.)	DESCRIPTION
1,7,14,25	5*	T	20	Angular solution
3,9,21,17	16*	T	20	Translational Solution
4,11,22,18	31*	T	20	Lat./Long./Alt.
5R	15	R	20	Sensor Inputs
5T	17	T	20	Benchmark Command Echo
6	17	T	20	Results Output
7	7	R	20	Benchmark Commands
2,8,20,16	2*	T	20	Constant
10,12,23,29	5*	T	20	Constant
15,13,24,30	11*	T	20	Constant

* Word Count For Each Message.

A. Message and Word Counts Definitions

Table II contains a description of each word for all the messages. The RT subaddress is programmed as the message number. Therefore, according to the MIL-STD-1553 protocol, the message number must be between 1 and 30. The word count must be between 1 and 32. The "CONTENTS" column indicates the benchmark software data base variable name for RT5 and RT7 outputs, or value of constants. RT5 and RT7 inputs must be declared in the input/output package specification. Many of these inputs are assumed to be 32 bit precision (two sixteen bit words). As usual with MIL-STD-1553 message structures, the most significant word is transmitted first. The final column indicates whether the word is a variable or a constant. Finally, message numbers or words within a message which are zero are not presented.

All messages are outputs from the navigation benchmark except for messages 5 and 7. Therefore, the "contents" column of Table II contains the name of the local variable in the navigation benchmark which contains the value of this output. These names must not be confused with the simulation data base names, many of which are identical. Message 5 contains the sensor inputs to the navigation benchmark which are needed to execute the update equations. Message 7 contains the benchmark control and performance information.

The details of the word formats are provided in Table III. The number in parentheses after the "CONTENTS" in Table II refers to the detailed word format entry in Table III for words with variable content.

TABLE II. MESSAGE SHEETS

MESSAGE	WORD	CONTENTS	TYPE
1	1	PSI (18)	VARIABLE
1	2	THETA (19)	VARIABLE
1	3	PHI (20)	VARIABLE
1	4	Message 1 Status (35)	VARIABLE
1	5	Module Identification (38)	VARIABLE
2	1	45056.0	CONSTANT
2	2	Module Identification (35)	VARIABLE
3	1	32768.0	CONSTANT
3	2	32768.0	CONSTANT
3	3	4576.0	CONSTANT
3	4	PSI (18)	VARIABLE
3	5	NAV_VEL_X (15)	VARIABLE
3	6	NAV_VEL_Y (16)	VARIABLE
3	7	NAV_VEL_Z (17)	VARIABLE
3	8	PLATFORM_X_ACCELERATION (1)	VARIABLE
3	9	PLATFORM_Y_ACCELERATION (2)	VARIABLE
3	10	VERTICAL_ACCELERATION (3)	VARIABLE
3	11	RATE_X (7)	VARIABLE
3	12	RATE_Y (8)	VARIABLE
3	13	RATE_Z (9)	VARIABLE
3	14	NAV_BAROMETRIC_RATE (14)	VARIABLE
3	15	Message 3 Status (36)	VARIABLE
3	16	Module Identification (38)	VARIABLE

TABLE II. MESSAGE SHEETS (continued)

MESSAGE	WORD	CONTENTS	TYPE
4	1	32768.0	CONSTANT
4	2	32768.0	CONSTANT
4	3	49139.0	CONSTANT
4	4	PSI (18)	VARIABLE
4	5	PSI (18)	VARIABLE
4	6	THETA (19)	VARIABLE
4	7	PHI (20)	VARIABLE
4	8	PHI (20)	VARIABLE
4	9	NAV_VEL_Y (16)	VARIABLE
4	10	NAV_VEL_X (15)	VARIABLE
4	11	NAV_VEL_Z (17)	VARIABLE
4	12	NAV_ALTITUDE_1 (23)	VARIABLE
4	13	NAV_ALTITUDE_2 (23)	VARIABLE
4	14	NAV_LATITUDE_DEG (21)	VARIABLE
4	15	NAV_LONGITUDE_DEG (22)	VARIABLE
4	16	2050.0	CONSTANT
4	17	PLATFORM_Y_ACCELERATION (2)	VARIABLE
4	18	PLATFORM_X_ACCELERATION (1)	VARIABLE
4	19	VERTICAL_ACCELERATION (3)	VARIABLE
4	20	1.0	CONSTANT
4	21	1.0	CONSTANT
4	22	0.0	CONSTANT
4	23	0.0	CONSTANT
4	24	0.0	CONSTANT
4	25	RATE_X (7)	VARIABLE
4	26	RATE_Y (8)	VARIABLE
4	27	RATE_Z (9)	VARIABLE
4	28	20.0	CONSTANT
4	29, 30	Message 4 Status (37)	VARIABLE
4	31	Module Identification (38)	VARIABLE

TABLE II. MESSAGE SHEETS (continued)

MESSAGE	WORD	CONTENTS	TYPE
5R	1	PLATFORM_X_ACCELERATION_1 (4)	VARIABLE, MSW
5R	2	PLATFORM_X_ACCELERATION_2 (4)	VARIABLE, LSW
5R	3	PLATFORM_Y_ACCELERATION_1 (5)	VARIABLE, MSW
5R	4	PLATFORM_Y_ACCELERATION_2 (5)	VARIABLE, LSW
5R	5	VERTICAL_ACCELERATION_1 (6)	VARIABLE, MSW
5R	6	VERTICAL_ACCELERATION_2 (6)	VARIABLE, LSW
5R	7	RATE_X_1 (10)	VARIABLE, MSW
5R	8	RATE_X_2 (10)	VARIABLE, LSW
5R	9	RATE_Y_1 (11)	VARIABLE, MSW
5R	10	RATE_Y_2 (11)	VARIABLE, LSW
5R	11	RATE_Z_1 (12)	VARIABLE, MSW
5R	12	RATE_Z_2 (12)	VARIABLE, LSW
5R	13	BAROMETRIC_ALTITUDE_1 (13)	VARIABLE, MSW
5R	14	BAROMETRIC_ALTITUDE_2 (13)	VARIABLE, LSW
5R	15	Module Identification (38)	CONSTANT (-1)

TABLE II. MESSAGE SHEETS (continued)

5T, 6	1	ECPM Control Word (24)	VARIABLE
5T, 6	2	Benchmark Duration Counter (25)	VARIABLE
5T, 6	3	Input/Output Max. Iterations (26) Per Frame	VARIABLE
5T,6	4	Status Word (27)	VARIABLE
5T,6	5	Spare Processing Time, Integer (28)	VARIABLE, MSW
5T,6	6	Spare Processing Time, Fraction (29)	VARIABLE, LSW
5T,6	7	Maximum Throughput DASL Loop Counts, Most Significant Half (31)	VARIABLE
5T,6	8	Maximum Throughput DASL Loop Counts, Least Significant Half (31)	VARIABLE
5T,6	9	Maximum Throughput Time in Seconds, Integer Part (32)	VARIABLE
5T,6	10	Maximum Throughput Time in Seconds, Fractional Part (33)	VARIABLE
5T,6	11	Additional IO DASL Loop Count, Most Significant Half (34)	VARIABLE
5T,6	12	Additional IO DASL Loop Count, Least Significant Half (34)	VARIABLE
5T,6	13	Maximum Input/Output Count (30)	VARIABLE
5T,6	14	ECPM Mode (40)	VARIABLE
5T,6	15	Module Identification for IO Mix Slave (38)	VARIABLE
5T,6	16	Navigation Results Output Set (39)	VARIABLE
5T,6	17	Module Identification (38)	VARIABLE

TABLE II. MESSAGE SHEETS (continued)

7	1	ECPM Control Word (24)	VARIABLE
7	2	Benchmark Duration Counter (25)	VARIABLE
7	3	Input/Output Max Iterations Per Frame (26)	VARIABLE
7	4	ECPM Mode (40)	VARIABLE
7	5	Module Identification to receive Additional IO Mix Slave (38)	VARIABLE
7	6	Navigation Results Output Set (39)	VARIABLE
7	7	Module Identification (selects module which performs calculations) (38)	VARIABLE
10	1	892.0	CONSTANT
10	2	13184.0	CONSTANT
10	3	540.0	CONSTANT
10	4	37840.0	CONSTANT
10	5	Module Identification (38)	VARIABLE
15	1	1.0	CONSTANT
15	2	30.0	CONSTANT
15	3	0.0	CONSTANT
15	4	0.0	CONSTANT
15	5	0.0	CONSTANT
15	6	65397.0	CONSTANT
15	7	2214.0	CONSTANT
15	8	8160.0	CONSTANT
15	9	0.0	CONSTANT
15	10	2114.0	CONSTANT
15	11	Module Identification (38)	VARIABLE

B. Details of Word Formats

Table III provides a detailed explanation of each word which is a variable in the message mix. Constant values are given in Table II for the words which are constants. Constant values in both decimal, hex, and binary are given in Table IV.

The "SOURCE" entry indicates whether the UUT or the DASL VAX transmits this word. Since the DASL VAX acts as the Mission Computer (MC), the source may be indicated as "MC". Specific source software applications may be indicated when the source is the UUT. Several words generated by the UUT merely echo back the sensor inputs, therefore mux_io is indicated as the procedure of origin.

The "DEST" indicates the computer which receives the word. If the MC is the receiver and the word is used as input to the display software, this fact is so indicated in the table entry.

The "TYPE" entry indicates whether the variable is symmetric, non-symmetric, or discrete. In the case of non-symmetric variables, the offset is part of the entry. This information indicates the encoding and decoding algorithms used to convert the real variable into a 16-bit integer. These algorithms, together with variables "OFFSET", "MAX", "MIN", and "RESOLUTION" will be discussed in the last section of this document.

"MAX" and "MIN" refer to the maximum and minimum values of the MIL-STD-1553 word. The minimum is offset by the resolution ("RES") because of the bit packing method discussed in the next section of this document. The value of the resolution is the same as the "scaling factor" which is also discussed in the next section.

All outputs are 16 bit (two byte) packed integers. Entries identified as 32 bit values consist of two sixteen bit words. The most significant word is first.

"UNITS" identifies the unit of mmeasure for the variable.

"RESOLUTION" identifies the numerical difference of adjacent bit values of the packed word due to the algorithm used. This is also the value of the Least Significant Bit (LSB). Calculation of this value is discussed in the next section.

TABLE III. Word Definitions.

1. PLATFORM_X_ACCELERATION

SOURCE: UUT (NAV, mux_io)
DEST: MC (VAX), used by displays
TYPE: SYMMETRIC

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+720.0	-720.0 - RES	16	ft/sec**2	720/32767

2. PLATFORM_Y_ACCELERATION

SOURCE: UUT (NAV, mux_io)
DEST: MC (VAX), used by displays
TYPE: SYMMETRIC

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+720.0	-720.0 - RES	16	ft/sec**2	720/32767

3. VERTICAL ACCELERATION

SOURCE: UUT (NAV, mux_io)
DEST: MC (VAX), used by displays
TYPE: SYMMETRIC

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+720.0	-720.0 - RES	16	ft/sec**2	720/32767

TABLE III. Word Definitions (continued).

4. PLATFORM_X_ACCELERATION_1, PLATFORM_X_ACCELERATION_2

SOURCE: MC (VAX acceleration sensor model)
DEST: UUT (NAV)
TYPE: SYMMETRIC, two sixteen bit words

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+720.0	-720.0 - RES	32	ft/sec**2	720.0/2147483647.0

5. PLATFORM_Y_ACCELERATION_1, PLATFORM_Y_ACCELERATION_2

SOURCE: MC (VAX acceleration sensor model)
DEST: UUT (NAV)
TYPE: SYMMETRIC, two sixteen bit words

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+720.0	-720.0 - RES	32	ft/sec**2	720.0/2147483647.0

6. VERTICAL ACCELERATION_1, VERTICAL_ACCELERATION_2

SOURCE: MC (VAX acceleration sensor model)
DEST: UUT (NAV)
TYPE: SYMMETRIC, two sixteen bit words

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+720.0	-720.0 - RES	32	ft/sec**2	720.0/2147483647.0

TABLE III. Word Definitions (continued).

7. RATE_X

SOURCE: UUT (NAV, mux_io)
DEST: MC (VAX)
TYPE: SYMMETRIC

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+20.0	-20.0 - RES	16	rads/sec	20/32767

8. RATE_Y

SOURCE: UUT (NAV, mux_io)
DEST: MC (VAX)
TYPE: SYMMETRIC

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+20.0	-20.0 - RES	16	rads/sec	20/32767

9. RATE_Z

SOURCE: UUT (NAV, mux_io)
DEST: MC (VAX)
TYPE: SYMMETRIC

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+20.0	-20.0 - RES	16	rads/sec	20/32767

TABLE III. Word Definitions (continued).

10. RATE_X_1, RATE_X_2

SOURCE: MC (VAX rate sensor model)
DEST: UUT (NAV)
TYPE: SYMMETRIC, two sixteen bit words

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+20.0	-20.0 - RES	32	rads/sec	20.0/2147483647.0

11. RATE_Y_1, RATE_Y_2

SOURCE: MC (VAX rate sensor model)
DEST: UUT (NAV)
TYPE: SYMMETRIC, two sixteen bit words

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+20.0	-20.0 - RES	32	rads/sec	20.0/2147483647.0

12. RATE_Z_1, RATE_Z_2

SOURCE: MC (VAX rate sensor model)
DEST: UUT (NAV)
TYPE: SYMMETRIC, two sixteen bit words

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+20.0	-20.0 - RES	32	rads/sec	20.0/2147483647.0

TABLE III. Word Definitions (continued).

13. BAROMETRIC_ALTITUDE_1, BAROMETRIC_ALTITUDE_2

SOURCE: MC (VAX Air Data Computer model)
DEST: UUT (NAV)
TYPE: NONSYMMETRIC, OFFSET = 39,250, two sixteen bit words

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+80,000.0	-1500.0 - RES	32	feet	40750.0/2147483647.0

14. NAV_BAROMETRIC RATE

SOURCE: UUT(NAV, v_barom)
DEST: MC (VAX)
TYPE: SYMMETRIC

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+32767	-32767 - RES	16	feet/sec	1

15. NAV_VEL_X

SOURCE: UUT (NAV, horiz_nav)
DEST: MC (VAX used by displays)
TYPE: SYMMETRIC

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+2500.0	-2500.0 - RES	16	feet/sec	2500/32767

TABLE III. Word Definitions (continued).

16. NAV_VEL_Y

SOURCE: UUT (NAV, horiz_nav)
DEST: MC (VAX used by displays)
TYPE: SYMMETRIC

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+2500.0	-2500.0 - RES	16	feet/sec	2500/32767

17. NAV_VEL_Z

SOURCE: UUT (NAV, vert_nav)
DEST: MC (VAX used by displays)
TYPE: SYMMETRIC

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+2500.0	-2500.0 - RES	16	feet/sec	2500/32767

18. PSI

SOURCE: UUT (NAV, nav_att)
DEST: MC (VAX used by displays)
TYPE: SYMMETRIC

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+180.0	-180.0 - RES	16	degrees	180/32767

19. THETA

SOURCE: UUT (NAV, nav_att)
DEST: MC (VAX used by displays)
TYPE: SYMMETRIC

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+90.0	-90.0 - RES	16	degrees	90/32767

TABLE III. Word Definitions (continued).

20. PHI

SOURCE: UUT (NAV, nav_att)
DEST: MC (VAX used by displays)
TYPE: SYMMETRIC

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+180.0	-180.0 - RES	16	degrees	180/32767

21. NAV_LATITUDE_DEG

SOURCE: UUT (NAV, nav_horiz)
DEST: MC (VAX used by displays)
TYPE: SYMMETRIC

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+90.0	-90.0 - RES	16	degrees	90/32767

22. NAV_LONGITUDE_DEG

SOURCE: UUT (NAV, nav_horiz)
DEST: MC (VAX used by displays)
TYPE: SYMMETRIC

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+180.0	-180.0 - RES	16	degrees	180/32767

TABLE III. Word Definitions (continued).

23. NAV_ALTITUDE_1, NAV_ALTITUDE_2

SOURCE: UUT (NAV, nav_vert)
DEST: MC (VAX used by displays)
TYPE: NONSYMMETRIC, OFFSET = 39,250, two sixteen bit words

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
+80,000.0	-1500.0 - RES	32	feet	40750.0/2147483647.0

TABLE III. Word Definitions (continued).

24. ECPM Control Word (discrete word)

SOURCE: UUT [Message 6]
MC [Message 7]
DEST: MC [Message 6]
UUT [Message 7]

TYPE: DISCRETE

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
-	-	16	(none)	-

Bit 13	Bit 14	Bit 15	
0	0	0	Command to Configure System
0	0	1	Start ECPM for configuration selected
0	1	0	Stop navigate only mode
0	1	1	Measure Maximum IO
1	0	0	Measure Maximum Throughput
1	0	1	Transmit Benchmark Results
1	1	0	Reserved
1	1	1	Reserved

Bits 12-0: Reserved, ignored by UUT, error for MC

NOTE: Set by MC, read by UUT

TABLE III. Word Definitions (continued).

25. Benchmark Duration Counter

SOURCE: UUT [Message 6]
MC [Message 7]
DEST: MC [Message 6]
UUT [Message 7]

TYPE: UNSIGNED INTEGER (0 to 65535)

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
65535	0	16	SECONDS	1

NOTE: Set by MC, read by UUT

26. Input/ Output Maximum Iterations Per Frame

SOURCE: UUT [Message 6]
MC [Message 7]
DEST: MC [Message 6]
UUT [Message 7]

TYPE: UNSIGNED INTEGER (0 to 65535)

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
65535	0	16	NONE	1

NOTE: Set by MC, read by UUT;
This word is the number of iterations per 50 msec. minor frame commanded.

TABLE III. Word Definitions (continued).

27. Status Word

SOURCE: UUT
DEST: MC

TYPE: DISCRETE

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
-	-	16	-	-

Bit 15	: 0/1 Indicates OK/Not OK ECPM Control Word see word 24, Table III)
Bit 14	: 0/1 Indicates Can/Cannot Run This Input/Output Mix With The Benchmark (This is a Timeout)
Bit 13	: 0/1 indicates results valid/invalid due to Stop Command during recording (see word 24, Table III)
Bit 12	: 0/1 indicates that valid/invalid navigation data subaddress was set (see word 39, Table III)
Bit 11	: 0/1 indicates valid/invalid slave module identifier (see words 24 and 38, Table III)
Bit 10	: 0/1 indicates valid/invalid ECPM mode (see word 40, Table III)
Bit 9	: 0/1 indicates valid/invalid master module identifier (see words 24 and 38, Table III)
Bits 0-8	: Reserved, ignored by UUT, error for MC if non-zero

TABLE III. Word Definitions (continued).

28. Spare Processing Time, Integer Part

SOURCE: UUT
DEST: MC

TYPE: UNSIGNED INTEGER (0 to 65535)

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
65535	0	16	SECONDS	1

29. Spare Processing Time, Fractional Part

SOURCE: UUT
DEST: MC

TYPE: UNSIGNED INTEGER (0 to 65535)

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
65534/65535	0	16	Seconds	1/65535

30. Maximum Input/Output Count

SOURCE: UUT
DEST: MC

TYPE: UNSIGNED INTEGER (0 to 65535)

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
65535	0	16	NONE (ITERATIONS)	1

TABLE III. Word Definitions (continued).

31. Maximum Throughput DASL Loop Counts

SOURCE: UUT
DEST: MC

TYPE: UNSIGNED INTEGER (0 to 2**32-1)

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
2**32-1	0	32	NONE (ITERATIONS)	1

32. Maximum Throughput Time in Seconds, Integer Part

SOURCE: UUT
DEST: MC

TYPE: UNSIGNED INTEGER (0 to 65535)

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
65535	0	16	SECONDS	1

33. Maximum Throughput Time in Seconds, Fractional Part

SOURCE: UUT
DEST: MC

TYPE: UNSIGNED INTEGER (0 to 65535)

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
65534/65535	0	16	Seconds	1/65535

TABLE III. Word Definitions (continued).

34. Additional IO DASL Loop Count

SOURCE: UUT
DEST: MC

TYPE: UNSIGNED INTEGER (0 to 2**32-1)

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
2**32-1	0	32	NONE (ITERATIONS)	1

35. Message 1 Status

SOURCE: UUT
DEST: MC

TYPE: DISCRETE WORD

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
65535	0	16	NONE	1

Bit	Meaning
0	Word 1 of Message 1 Truncated
1	Word 2 of Message 1 Truncated
2	Word 3 of Message 1 Truncated
3-15	Not Used

TABLE III. Word Definitions (continued).

36. Message 3 Status

SOURCE: UUT
DEST: MC

TYPE: DISCRETE WORD

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
65535	0	16	NONE	1

Bit	Meaning
0	Word 1 of Message 3 Truncated
1	Word 2 of Message 3 Truncated
2	Word 3 of Message 3 Truncated
3	Word 4 of Message 3 Truncated
4	Word 5 of Message 3 Truncated
5	Word 6 of Message 3 Truncated
6	Word 7 of Message 3 Truncated
7	Word 8 of Message 3 Truncated
8	Word 9 of Message 3 Truncated
9	Word 10 of Message 3 Truncated
10	Word 11 of Message 3 Truncated
11	Word 12 of Message 3 Truncated
12	Word 13 of Message 3 Truncated
13	Word 14 of Message 3 Truncated
14-15	Not Used

TABLE III. Word Definitions (continued).

37. Message 4 Status

SOURCE: UUT
DEST: MC

TYPE: DISCRETE WORD

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
2**32-1	0	32	NONE	1

Bit	Meaning
0	Word 1 of Message 4 Truncated
1	Word 2 of Message 4 Truncated
2	Word 3 of Message 4 Truncated
3	Word 4 of Message 4 Truncated
4	Word 5 of Message 4 Truncated
5	Word 6 of Message 4 Truncated
6	Word 7 of Message 4 Truncated
7	Word 8 of Message 4 Truncated
8	Word 9 of Message 4 Truncated
9	Word 10 of Message 4 Truncated
10	Word 11 of Message 4 Truncated
11	Word 12 of Message 4 Truncated
12	Word 13 of Message 4 Truncated
13	Word 14 of Message 4 Truncated
14	Word 15 of Message 4 Truncated
15	Word 16 of Message 4 Truncated

TABLE III. Word Definitions (continued).

37. Message 4 Status (continued)

16	Word 17 of Message 4 Truncated
17	Word 18 of Message 4 Truncated
18	Word 19 of Message 4 Truncated
19	Word 20 of Message 4 Truncated
20	Word 21 of Message 4 Truncated
21	Word 22 of Message 4 Truncated
22	Word 23 of Message 4 Truncated
23	Word 24 of Message 4 Truncated
24	Word 25 of Message 4 Truncated
25	Word 26 of Message 4 Truncated
26	Word 27 of Message 4 Truncated
27	Word 28 of Message 4 Truncated
28-31	Not Used

TABLE III. Word Definitions (continued).

38. Module Identification

SOURCE: UUT (Messages 1,2,3,4,6,10,15)
MC (Message 7)
DEST: MC (Messages 1,2,3,4,6,10,15)
UUT (Message 7)

TYPE: Integer

MAX	MIN	WORD SIZE	UNITS	RESOLUTION
32767	-1	16	None	1

NOTES:
[1] The value -1 indicates that the message applies to all modules.
[2] Normal values are 0 to 32767
[3] Negative values (other than -1) are illegal
[4] Values 10, 11, 12 are typical for the TI MDP

39. Navigation Solution Output Select

SOURCE: UUT (Messages 5T, 6)
MC (Message 7)
DEST: MC (Message 5T, 6)
UUT (Message 7)
TYPE: DISCRETE

Bit 14	Bit 15	Meaning
0	0	Set 0 (outputs to subaddresses 1,2,3,4,10,15)
0	1	Set 1 (outputs to subaddresses 7,8,9,11,12,13)
1	0	Set 2 (outputs to subaddresses 14,20,21,22,23,24)
1	1	Set 3 (outputs to subaddresses 25,26,27,28,29,30)

Bits 0-13: Not Used

TABLE III. Word Definitions (continued).

40. ECPM Mode

SOURCE:	UUT (Message 6)
	MC (Message 7)
DEST:	MC (Message 6)
	UUT (Message 7)
TYPE:	DISCRETE

Bit 15	Meaning
0	Navigate Only Mode
1	Record Results Mode

Bits 0-14: Not Used

TABLE IV. Constant Values.

MESSAGE/WORD	VALUE (DEC)	VALUE (BINARY)	VALUE (HEX)
		MSB LSB	
2/1	45056 * [-20480]	1 011 000 000 000 000	B000
3/1	32768 * [-32768]	1 000 000 000 000 000	8000
3/2	32768 * [-32768]	1 000 000 000 000 000	8000
3/3	4576	0 001 000 111 100 000	11E0
4/1	32768 * [-32768]	1 000 000 000 000 000	8000
4/2	32768 * [-32768]	1 000 000 000 000 000	8000
4/3	49139 * [-16397]	1 011 111 111 110 011	BFF3
4/16	2050	0 000 100 000 000 010	0802
4/20	1	0 000 000 000 000 001	0001
4/21	1	0 000 000 000 000 001	0001
4/22	0	0 000 000 000 000 000	0000
4/23	0	0 000 000 000 000 000	0000
4/24	0	0 000 000 000 000 000	0000
4/28	20 (LEVER_ARM)	0 000 000 000 010 100	0014
10/1	892	0 000 001 101 111 100	037C
10/2	13184	0 011 001 110 000 000	3380
10/3	540	0 000 001 000 011 100	021C
10/4	37840 *[-27696]	1 001 001 111 010 000	93D0
15/1	1	0 000 000 000 000 001	0001
15/2	30	0 000 000 000 011 110	001E
15/3	0	0 000 000 000 000 000	0000
15/4	0	0 000 000 000 000 000	0000
15/5	0	0 000 000 000 000 000	0000
15/6	65397 * [-139]	1 111 111 101 110 101	FF75
15/7	2214	0 000 100 010 110 010	08A6
15/8	8160	0 001 111 111 100 000	1FE0
15/9	0	0 000 000 000 000 000	0000
15/10	2114	0 000 100 001 000 010	0842

* MIL-STD-1750 does not provide an unsigned integer word format. The values in brackets are equivalent MIL-STD-1750 values for the defined bit settings.

C. Bit Packing Algorithms

Two types of packing algorithms are required to generate the entries of Table III. The first type is "Symmetric", which is defined as the case when the entry for "MAX" is equal to the absolute value of "MIN" (MAX = |MIN| - RESOLUTION). The second type is "Nonsymmetric",which is defined as the case when "MAX" is not equal to the absolute value of "MIN". Each type of algorithm is needed for 16-bit words (single precision) and for 32-bit words (double precision).

The single precision symmetric case assumes that bit 0 (leftmost) is the sign bit and that bit 15 (rightmost) is LSB. This gives a range to the resulting packed integer (1553_word) of -32768 to +32767. For the sixteen bit signed integer format, the bit pattern "1 000 000 000" is defined as -32768. Other examples are shown in Table V.

This permits the resulting real number to be unpacked easily with a scaling factor:

[1] real := scaling_factor * FLOAT(1553_word)

where

real	-> 4 byte real variable,
scaling_factor	-> four byte real, value = MAX/32767.0,
1553_word	-> decoded two byte integer input,
	+32767 $\geq$ value $\geq$ -32768.

In a similar fashion, a symmetric, single precision real variable can be packed into an integer output (1553_word):

[2] 1553_word := real/scaling_factor.

The double precision symmetric case is very similar, except that the scaling factor is 31 bits. The two sixteen bit words are unpacked into a 32 bit real variable as below:

[3] real := scaling_factor * (65536.0 * word_1 + word_2);

where:

real -> four byte real variable,
scaling_factor -> scaling factor, MAX/(2**31 -1),
word_1 -> most significant word (transmitted first on the MIL-STD-1553 bus), two byte integer,
word_2 -> least significant word (transmitted second on the MIL-STD-1553 bus), two byte integer.

The inverse of the double precision representations can be computed as follows.

[4] scaled_real = real/scaling_factor

ms_word = INTEGER (scaled_real/65536.0),
ls_word = INTEGER (scaled_real - FLOAT (65536 * ms_word)),

where:

scaled_real -> four byte real;
ms_word -> two byte integer (packed value), most significant word;
ls_word -> four byte integer (packed value, needed to prevent overflow), least significant word;
INTEGER : Integer function of real (truncates, not rounds).

Two points of caution must be observed regarding this algorithm. First, if ls_word is larger than or equal to 32768, then the two byte integer MIL-STD-1553 least significant word will be equal to ls_word - 65536. In a similar fashion, if ms_word is negative, then the value of the output two byte integer corresponding to ls_word will be the same as calculated above, except that the MIL-STD-1553 output will equal ls_word + 65535. The second point is that the function "INTEGER" must truncate, not round. This is especially critical in the calculation of ms_word.

The nonsymmetric case is also quite similar, except that a midpoint offset is required:

[5] OFFSET := (MAX + MIN)/2;
real := real_input - offset.

At this point, the shifted value (real) is a symmetric real variable and the single precision symmetric packing algorithm can be used.

TABLE V. Binary, Hex, and Decimal Values For Packed Words.

BINARY	HEX	DECIMAL
0 111 111 111 111 111	EFFF	scaling_factor * 32767
0 100 000 000 000 000	4000	scaling_factor * 16384
0 001 000 000 000 000	2000	scaling_factor * 4096
0 000 000 000 000 001	0001	scaling_factor * 1
0 000 000 000 000 000	0000	0.0
1 111 111 111 111 111	FFFF	scaling_factor * -1
1 111 000 000 000 000	F000	scaling_factor * -4096
1 100 000 000 000 000	C000	scaling_factor * -16384
1 000 000 000 000 001	8001	scaling_factor * -32767
1 000 000 000 000 000	8000	scaling_factor * -32768

scaling_factor = Scaling Factor

D. Word Packing Procedure Calls

Procedures will be needed to pack and unpack the 16-bit integers from the MIL-STD-1553 interface into benchmark software data base variables and VAX variables. It is assumed that the driver routine automatically transmits the word to the RT output device, with no further intervention from the application. This transmission may involve a backplane bus transaction. The MIL-STD-1553 interface scheduling is controlled by the bus controller (VAX). In a similar fashion, the application is assumed to receive the most recent input value. Therefore, bit packing and unpacking routines may be machine dependent since some machines potentially use the LSB for the sign bit. However, the interfaces must be standardized to assure code portability.

The calling procedure for the single precision routines is as follows:

```
UNPACK (local_real , 1553_word , scaling_factor, offset) ;
.
.
.
PACK ( local_real, 1553_word , scaling_factor, offset);

local_real      => benchmark software variable,
1553_word       => MIL-STD-1553 16-bit integer output,
scaling_factor, offset [see section D., these are real numbers].
```

The offset for a symmetric variable will be 0.0.

A symmetric variable can also be easily encoded or decoded in the application by dividing or multiplying by the scaling factor. However, MIL-STD-1750 bit conventions must be observed (MSB is sign bit).

III. CONCLUSIONS

This paper has presented the data conventions, message formats, and word formats for the AAS&T ECPM for multiprocessors.

V. NOTES

A. ACRONYMS

AAS&T	Advanced Avionics Subsystems & Technology
DASL	Digital Avionic System Laboratory
ECPM	Embedded Computer Performance Measurement
ECP	Engineering Change Proposal
LSB	Least Significant Bit
LSW	Least Significant Word
MSB	Most Significant Bit
MSW	Most Significant Word
NAWC	Naval Air Warfare Center
RT	Remote Terminal
scaling_factor	Scaling Factor
TI	Texas Instruments
UUT	Unit Under Test

B. OTHER AAS&T ECPM DOCUMENTS

Further information on the ECPM benchmark code can be found in other documents produced as part of the Phase I effort. A complete listing of such documents is provided below.

- Software Requirements Specification for the AAS&T ECPM CSCI
- Interface Requirements Specification for the AAS&T ECPM

- AAS&T ECPM MIL-STD-1553 Interface Definition (this document)
- AAS&T ECPM Ada Source Code Listing
- AAS&T ECPM MIL-STD-1750A Assembly Language Code Listing
- AAS&T Real-Time Simulation Overview, NAC Technical Report 2458

C. AAS&T PATCH PANEL CONFIGURATION

The MIL-STD-1553 hardware connection in DASL to the simulation computers is accomplished with two patch panels. One patch panel is located in the computer room and the second is near the cockpit. Refer to the Software User's Manual for cable connections.

SOFTWARE REQUIREMENTS SPECIFICATION

FOR THE

ADVANCED AVIONICS TECHNOLOGY DEMONSTRATION (AATD) CSCI

OF

AATD SYSTEM

09/19/90

PREPARED BY:

SOFTWARE TECHNOLOGY DEPARTMENT
DEFENSE SYSTEMS & ELECTRONICS GROUP
TEXAS INSTRUMENTS INCORPORATED
6550 CHASE OAKS DRIVE
PLANO, TEXAS 75086

PREPARED BY: _____
CHUCK ROARK
AATD SYSTEMS ENGINEER

APPROVED BY: _____ 21Sept90
RICK OZMENT
SOFTWARE QUALITY ASSURANCE

APPROVED BY: _____
DAVE STRUBLE
AATD PROGRAM MANAGER

TABLE of CONTENTS

LIST of FIGURES

APPENDIXES

1. SCOPE

1.1 IDENTIFICATION

This Software Requirements Specification (SRS) establishes the requirements for the Computer Software Configuration Item (CSCI) identified as the Advanced Avionics Technology Demonstration (AATD), CSCI#1, of the AATD program under the terms of Contract Number N00163-09-C-0165 and in accordance with the AATD Statement of Work (SOW).

1.2 CSCI Overview

The AATD Program objective is to measure embedded computer system reserve requirements. It also has an objective of being able to compare different embedded computers with regards to processing and input/output (I/O) throughput. The Naval Avionics Center (NAC) has developed a VAX-hosted navigation benchmark. TI is to port this benchmark to its 1750A-based Mission Data Processor (MDP) and demonstrate the benchmark at NAC. TI is also to enhance the benchmark by adding a mechanism to measure reserve I/O and reserve processor throughput. A goal of the enhanced benchmark is that it be "easily" portable to other vendor computers. The benchmark will be controlled by NAC software hosted on their Digital Avionic System Laboratory (DASL) VAX computers. This software will interface to the TI MDP via a 1553B interface. The application level software protocol is described in the accompanying AATD Interface Requirements Specification (IRS).

This CSCI consists of the NAC navigation benchmark ported to the TI MDP, together with enhancements to measure spare I/O and spare processor throughput.

1.3 DOCUMENT OVERVIEW

This SRS documents the requirements for programming design, adaptation, quality factors, and traceability of the AATD Software CSCI. This SRS specifies the requirements allocated to the AATD Software CSCI and enables AATD Systems Engineering to assess whether or not the completed CSCI complies with those requirements. Upon AATD Systems Engineering approval and authentication, this SRS becomes the allocated baseline for the AATD Software CSCI. This SRS is used by the AATD software development staff as the basis for development and formal

2. REFERENCED DOCUMENTS

The following documents, of the exact issue shown, form a part of this specification to the extent specified herein.

2.1 GOVERNMENT DOCUMENTS

DOD-STD-2167A	Defense System Software Development - 29 February 1988
MIL-STD-1815A	Ada Programming Language - 17 February 1983
AATD SOW	Statement of Work for Embedded Computer Performance Measurement - 26 February 1990
AATD ECPM	AATD Program Embedded Computer Performance Measurement (ECPM) MIL-STD-1553 Interface Definition - 15 Aug 1990

2.2 NON-GOVERNMENT DOCUMENTS

SP15-25	TI Software Engineering Standards - 19 November 1989
AATD SQPP V1.0	TI AATD Software Quality Assurance Plan
AATD IRS	TI AATD Interface Requirements Specification
AATD DN	TI AATD Design Note: Benchmark Measurement - 7 August 1990

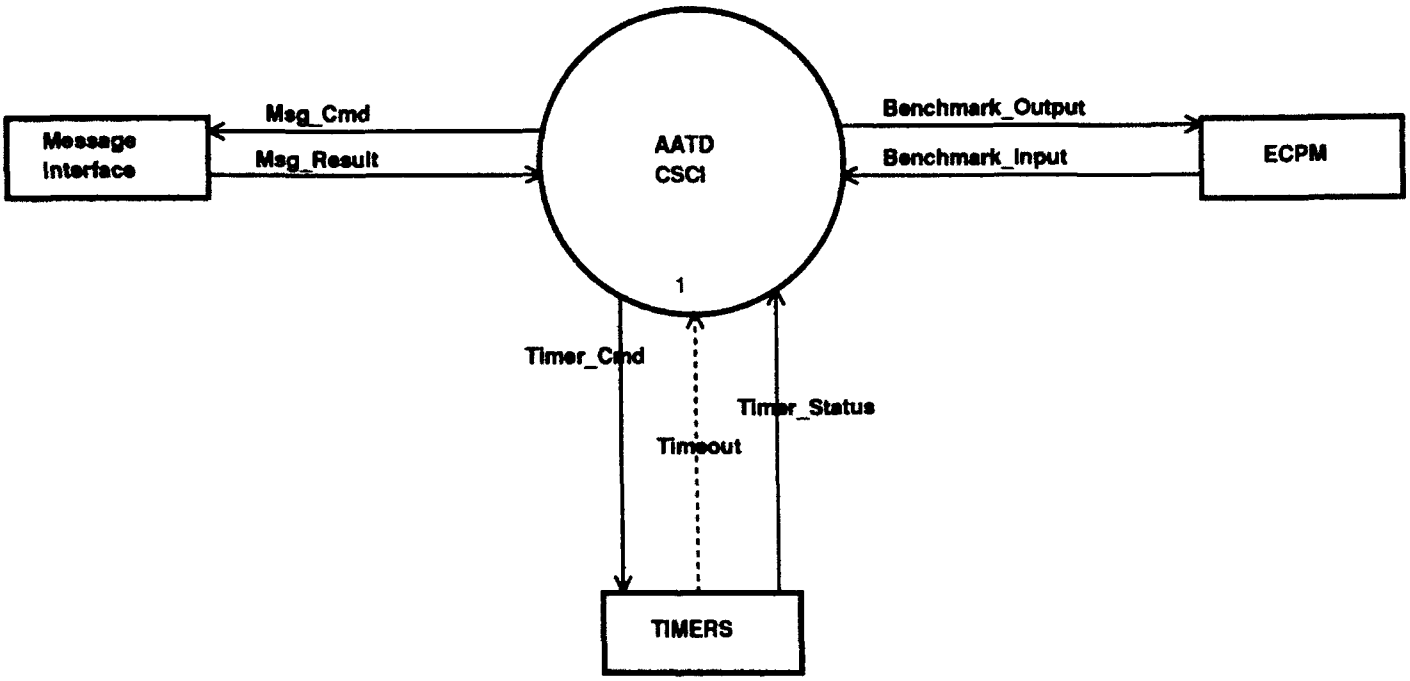


Figure 3-1 AATD CSCI Context Diagram

* Calculating (Benchmark Results) - The AATD CSCI is not executing the navigation benchmark but is in the process of calculating the benchmark results.

The AATD CSCI capabilities map into the AATD CSCI modes as shown in Table I.

Table I AATD CSCI Capabilities vs. CSCI Modes

	Mode				
Capability	Init	Idle	Execution	Calculating	Nav Only
Control AATD	x	x	x	x	x
Execute Nav			x		x
Execute I/O			x		
Measure Spare			x		
Determine Results				x	

The AATD CSCI executes entirely within the 1750A processing module(s) contained in the TI Mission Display Processor (MDP).

The following paragraphs define the capability requirements of the AATD CSCI.

3.2.1 Control AATD Capability - AATD.SRS.CTL

The purpose of the Control AATD capability is to control the sequencing of execution of the benchmark. In particular, this capability is responsible for initializing the AATD CSCI, interfacing with the ECPM for invocation of the benchmark and returning results, causing the Determine Results capability to execute upon completion of the Execute Nav capability, and awaiting reinvocation of the benchmark while idle. Figure 3-3 shows the DFD for the Control AATD capability. The following subparagraphs define the requirements for the Control AATD capability.

3.2.1.1 Seq Control AATD - AATD.SRS.CTL.01

The purpose of the Seq Control AATD subcapability is to control execution of the benchmark. The State Transition Diagram for Seq Control AATD is shown in Figure 3-4.

The requirements for Sequence Control AATD are:

* [AATD.SRS.CTL.01-1] Sequence Control AATD shall place itself in the initialization mode and invoke the Init AATD capability.

* [AATD.SRS.CTL.01-2] Upon completion of Init AATD (as signified by Init_Complete), Sequence Control AATD shall place itself in the idle mode and await benchmark commands from the ECPM.

* [AATD.SRS.CTL.01-3] While in the idle mode, upon reception of a Nav_Only benchmark command, Sequence Control AATD shall place itself in the Nav Only mode and invoke Execute Nav (via Nav_Start).

* [AATD.SRS.CTL.01-4] While in the idle mode, upon reception of a Benchmark benchmark command, Sequence Control AATD shall place itself in the Execution mode and set up the benchmark timeout by invoking Control Timer (via Start_Timer). Upon completion of the benchmark timer setup (as signified by Timer_Started), Sequence Control AATD shall invoke Execute Nav (via Nav_Start), Execute IO (via IO_Start), and Measure Spare (via Spare_Start).

* [AATD.SRS.CTL.01-5] While in the idle mode, upon reception of a Stop benchmark command, Sequence Control AATD shall remain in the idle mode and ignore the command.

* [AATD.SRS.CTL.01-6] While in the idle mode, upon reception of a Measure_IO benchmark command, Sequence Control AATD shall start the measurement of total I/O by invoking the Measure Total IO capability (via MeasureIO_Start). Upon completion of the measurement of total I/O (as signified via MeasureIO_Complete), Sequence Control AATD shall return to the idle mode.

* [AATD.SRS.CTL.01-7] While in the Nav Only mode, upon reception of a Stop benchmark command, Sequence Control AATD shall halt the execution of Execute Nav (via Nav_Stop) and return to the idle mode, awaiting a benchmark command.

* [AATD.SRS.CTL.01-8] While in the Execution mode, upon reception of the benchmark timeout (as indicated via Timeout), Sequence Control AATD shall halt the execution of Execute Nav (via Nav_Stop), Execute IO (via IO_Stop), and Measure Spare (via Spare_Stop), invoke Determine Results (via Determine_Results), and enter the Calculating mode.

* [AATD.SRS.CTL.01-9] While in the Execution mode, upon reception of an error indication (via Nav_Error) from Execute Nav, Sequence Control AATD shall halt the execution of Execute Nav (via Nav_Stop), Execute IO (via IO_Stop), and Measure Spare (via Spare_Stop), signal the occurrence of the error (via Bench_Error) to the ECPM, and return to the idle mode, awaiting

Table II Control AATD Inputs/Outputs

NAME	I/O	DESCRIPTION
Nav_Error	IN	Navigation error indication
Bench_Cmd	IN	Benchmark command
MeasureIO_Complete	IN	Signal indicating Total I/O has been measured
Results_Complete	IN	Signal indicating results have been calculated and transferred
Timeout	IN	Benchmark timeout indication
Timer_Started	IN	Signal signifying Benchmark timeout setup
Init_Complete	IN	Initialization complete indication
Start_Init	OUT	Signal for Init AATD to begin execution
MeasureIO_Start	OUT	Signal to start measuring Total I/O
Spare_Start	OUT	Command for Measure Spare to begin executing its spare execution loop
Spare_Stop	OUT	Command for Measure Spare to stop executing its spare execution loop
Nav_Start	OUT	Signal for Execute Nav to begin executing navigation algorithm
Nav_Stop	OUT	Command for Execute Nav to stop executing navigation algorithm
Bench_Error	OUT	Benchmark error indication
Start_Timer	OUT	Signal to setup benchmark timer
IO_Start	OUT	Signal for Execute I/O to begin executing I/O mix
IO_Stop	OUT	Command for Execute I/O to stop executing I/O mix
Determine_Results	OUT	Command for Determine Results to calculate Measure Spare execution time

3.2.1.2 Init AATD - AATD.SRS.CTL.02

The purpose of the AATD subcapability is to initialize the AATD CSCI.

The requirements for Init AATD are:

* [AATD.SRS.CTL.02-1] Init AATD shall initialize the AATD CSCI.

The inputs and outputs for Execute Nav are shown in Table V.

Table V Execute Nav Inputs/Outputs

NAME	I/O	DESCRIPTION
Nav_Start	IN	Signal for Execute Nav to begin executing navigation algorithm
Nav_Input	IN	Navigation algorithm input
Nav_Stop	IN	Command for Execute Nav to stop executing navigation algorithm
Nav_Output	OUT	Navigation algorithm output
Nav_Error	OUT	Nav error indication

3.2.3 Execute I/O Capability - AATD.SRS.I/O

The purpose of the Execute I/O capability is to execute the I/O mix specified for the current benchmark execution.

The requirements for Execute I/O are:

* [AATD.SRS.I/O-1] Execute I/O shall cause the requested I/O mix to execute during (Benchmark) Execution mode. Refer to Appendix III for a description of the base I/O mix.

The inputs and outputs for Execute I/O are shown in Table VI.

Table VII Measure Spare Inputs/Outputs

NAME	I/O	DESCRIPTION
Spare_Start	IN	Command for Measure Spare to begin executing its spare execution loop
Spare_Stop	IN	Command for Measure Spare to stop executing its spare execution loop
<Updated>Loops_Executed	OUT	Number of loops executed in Measure Spare

3.2.5 Determine Results Capability - AATD.SRS.RES

The purpose of the Determine Results capability is to determine the spare processing results for the benchmark executed with the given I/O mix.

The requirements for Determine Results are:

* [AATD.SRS.RES-1] Determine Results shall determine the processing time that Measure Spare executed during the execution of the benchmark based on the information stored in Loops_Executed by Measure Spare.

* [AATD.SRS.RES-2] Determine Results shall initiate a transfer to ECPM of the calculated time that Measure Spare Executed.

The inputs and outputs for Determine Results are shown in Table VIII.

Table IX Determine Results Inputs/Outputs

NAME	I/O	DESCRIPTION
Timer_Status	IN	Timer related status
Msg_Results	IN	Results from a message command
Msg_Cmd	OUT	Message command
Timer_Cmd	OUT	Command to invoke timer function

3.2.7 Measure Total IO - AATD.SRS.TOTIO

The purpose of the Measure Total IO capability is to measure the total I/O available for the computer system under test.

The requirements for Measure Total IO are:

* [AATD.SRS.TOTIO-1] Measure Total IO shall measure the total IO available for the system under test. The results shall be returned in terms of the base I/O mix. Refer to Appendix III for a description of the base I/O mix.

The inputs and outputs for Measure Total IO are shown in Table X.

Table X Measure Total IO Inputs/Outputs

NAME	I/O	DESCRIPTION
MeasureIO_Start	IN	Signal to start measuring Total I/O
MeasureIO_Complete	OUT	Signal indicating Total I/O measured
Total_IO	OUT	Total I/O available for system under test

3.3 CSCI INTERNAL INTERFACES

The AATD CSCI is shown with its logical internal interfaces in Figure 3-5. These logical interfaces are identified and described below. Detailed information concerning the data elements transmitted across each interface is contained in paragraph 3.4 CSCI Data Element Requirements.

* Control AATD/Execute Nav interface (IF_CTL_NAV). This interface is used to pass control commands between Control AATD and Execute Nav. The summary information transmitted over this interface consists of the following:

Data Element	Source	Destination
Nav_Start	CTL01	NAV02
Nav_Stop	CTL01	NAV02
Nav_Error	NAV02	CTL01

* Control AATD/Execute IO interface (IF_CTL_IO). This interface is used to pass control commands between Control AATD and Execute IO. The summary information transmitted over this interface consists of the following:

Data Element	Source	Destination
IO_Start	CTL01	IO03
IO_Stop	CTL01	IO03

* Control AATD/Measure Spare interface (IF_CTL_MSP). This interface is used to pass control commands between Control AATD and Execute Nav. The summary information transmitted over this interface consists of the following:

Data Element	Source	Destination
Spare_Start	CTL01	MSP04
Spare_Stop	CTL01	MSP04

* Control AATD/Determine Results interface (IF_CTL_RES). This interface is used to pass control commands between Control AATD and Determine Results. The summary information transmitted over this interface consists of the following:

Data Element	Source	Destination
Determine_Results	CTL01	RES05
Results_Complete	RES05	CTL01

* Control AATD/Measure Total IO Interface (IF_CTL_MEASIO). This interface is used to pass control commands between Control AATD and Measure Total IO. The summary information transmitted over this interface consists of the following:

Table XI AATD CSCI Data Element Requirements

Identifier	Description	Units	Range	Res
Bench_Cmd	Benchmark command. See IRS.	N/A	Nav_Only, Benchmark, Measure_IO, Stop	N/A
Bench_Error	Benchmark error signal. See IRS.	[tbd]	[tbd]	N/A
Benchmark_Input	Input from ECPM to benchmark: [Bench_Cmd|Nav_Input| IO_Mix|Benchmark_ Time]			
Benchmark_ Output	Output to ECPM from Benchmark: [Bench_Error| Bench_ Results| Nav_Output| Total_IO]			
Bench_Results	Results from Benchmark See IRS.	[tbd]	[tbd]	[tbd]
Benchmark_Time	Time benchmark is to execute. See IRS.	50msec	[tbd]	[tbd]
Determine Results	Signal for Determine Results to begin Execution.	N/A	N/A	N/A
Loops_Executed	Number of loops exec- uted by Measure Spare.	32-bit Integer	(2**32)-1	1
IO_Mix	IO_Mix to execute during benchmark. See Appendix III.	IO mix/sec	[tbd]	1
IO_Start	Signal for Execute IO to begin execution.	N/A	N/A	N/A
IO_Stop	Signal for Execute IO to stop execution.	N/A	N/A	N/A

Table XI AATD CSCI Data Element Requirements

Identifier	Description	Units	Range	Res
Timer_Started	Signal indicating timer setup.	N/A	N/A	N/A
Timer_Status	Timer status indication. See IRS.	N/A	N/A	N/A
Total_IO	Total IO available for target. See IRS.	IO mix/sec	[tbd]	[tbd]

3.5 ADAPTATION REQUIREMENTS

This paragraph specifies the requirements for adapting the CSCI to site-unique conditions and to changes in system environments.

3.5.1 Installation-Dependent Data

None.

3.5.2 Operational Parameters

None.

3.6 SIZING AND TIMING REQUIREMENTS

The AATD CSCI has no sizing and timing requirements, except those which can be inferred from executing the AATD demonstration. In particular there are no reserve sizing and timing requirements.

3.7 SAFETY REQUIREMENTS

None.

3.8 SECURITY REQUIREMENTS

None.

3.12 REQUIREMENTS TRACEABILITY

Table XII Requirements Traceability Table

Requirement Name	SRS Para	Ref. Doc.	Ref. Para
AATD.SRS.CTL.01-1	3.2.1.1	SOW	3.1.4
AATD.SRS.CTL.01-2	3.2.1.1	AATD DN	1.0
AATD.SRS.CTL.01-3	3.2.1.1	SOW	3.1.4
AATD.SRS.CTL.01-4	3.2.1.1	AATD DN	2,3,4
AATD.SRS.CTL.01-5	3.2.1.1	Customer	N/A
AATD.SRS.CTL.01-6	3.2.1.1	Customer	N/A
AATD.SRS.CTL.01-7	3.2.1.1	Customer	N/A
AATD.SRS.CTL.01-8	3.2.1.1	AATD DN	2.0, 3.0, 4.0
AATD.SRS.CTL.01-9	3.2.1.1	AATD DN	2.0, 3.0, 4.0
AATD.SRS.CTL.01-10	3.2.1.1	Customer	N/A
AATD.SRS.CTL.01-11	3.2.1.1	AATD DN	2.0, 3.0, 4.0
AATD.SRS.CTL.02-1	3.2.1.2	AATD DN	1.0
AATD.SRS.CTL.03-1	3.2.1.3	Customer	N/A
AATD.SRS.NAV-1	3.2.2	SOW	3.1.4
AATD.SRS.NAV-2	3.2.2	AATD DN	4.0
AATD.SRS.I/O-1	3.2.3	AATD DN	3.0
AATD.SRS.MSP-1	3.2.4	AATD DN	2.0
AATD.SRS.MSP-2	3.2.4	AATD DN	2.0
AATD.SRS.MSP-3	3.2.4	AATD DN	2.0
AATD.SRS.RES-1	3.2.5	AATD DN	2.0
AATD.SRS.RES-2	3.2.5	AATD DN	2.0
AATD.SRS.OS-1	3.2.6	SOW	3.1.2
AATD.SRS.OS-2	3.2.6	AATD DN	2.0
AATD.SRS.TOTIO-1	3.2.7	Customer	N/A

4.1.1.4 Qualification Methods

The methods for validating the identified requirements are listed in the Test Phases fields, and are defined below:

a. Demonstration (D). Demonstration is a method whereby the performance of the software product is tested by visual observation. Demonstration shall be used when detailed qualitative measurement is not required, or a requirement allocation is not meaningful below the system level.

b. Inspection (I). Inspection of a software product shall use physical examination of the product to verify conformance to the requirements of the product.

c. Analysis (A). Analysis is the use of recognized techniques to explain or illustrate the performance of the software product. Analysis shall include the use of test drivers to emulate input and output activities and the interpretation or extrapolation of test data.

d. Additional Qualification Methods - None.

NOTE

For some of the test phases, the qualification methods are marked as A|D (analysis (A) or demonstration (D)) or I|D (inspection (I) or demonstration (D)). In such cases, the method will be demonstration if the testing is performed on AATD hardware and no software stubbing is necessary. Otherwise, in such cases, the method will be analysis or inspection, as appropriate. The "+" operator means "and". N/A means not applicable.

5. PREPARATION FOR DELIVERY

Reference the AATD SDP for a discussion of preparations for delivery.

of our system, and where possible, we have decided to think logically of where we get our data and control, and not physically. That is why we have information coming into our system from the "OPERATOR" and "VIDEO INTERFACE", rather than the physical hardware that we directly interface with. We want to concentrate on what data and control our system has to handle, and not how we physically receive that data and control. That way if some piece of hardware is redesigned, our requirements model should not have to change, if the interface remains constant.

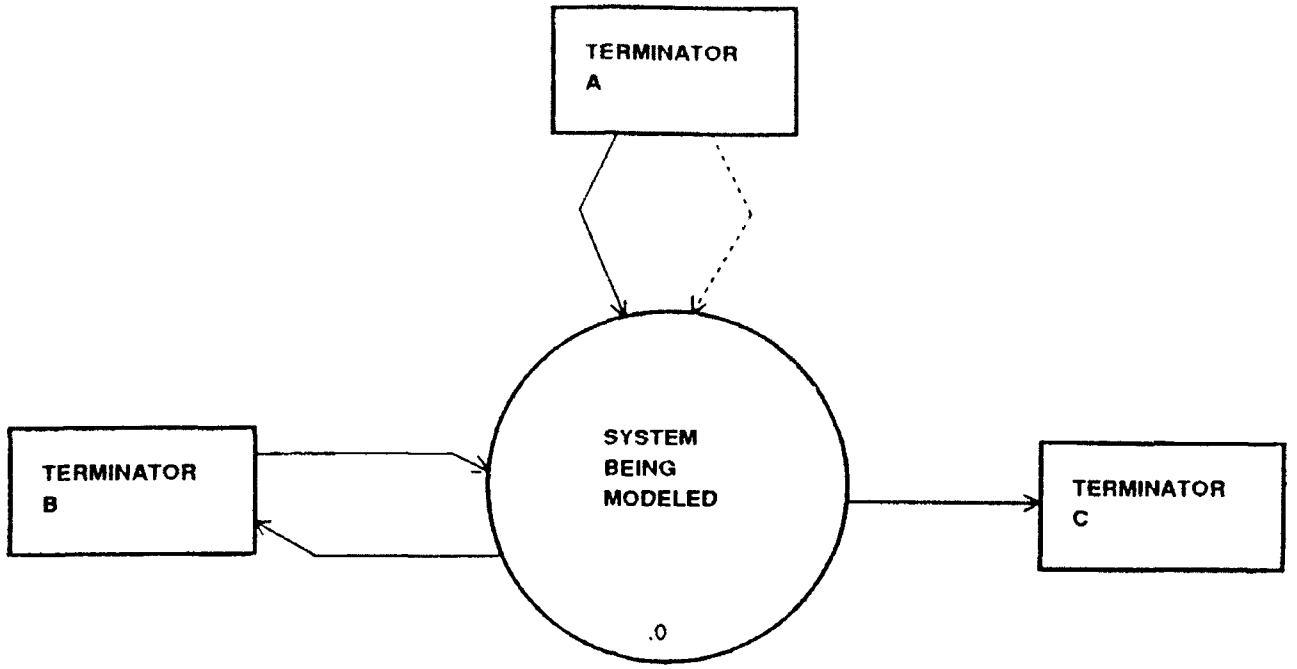
The next step in our modelling process is to define the major capabilities of our system. The major capabilities are at a very high level and will need more detail at a lower level. As we define the major capabilities, we need to define what types of data and control need to be passed between these capabilities. We also need to make sure that the data and control that we defined in the context diagram is present. If we need to add or take away any of these, we should consider it now.

Again, this is an abstract model of our system, not a physical design. It is possible for our physical design to change many times without affecting the requirements. One of the main assumptions of the philosophy that encourages us to step back from the physical is that each process is assumed to be instantaneous, that is, as soon as each process gets its required inputs, it produces the required outputs. This is obviously not characteristic of any physical design.

From here, we define deeper and deeper levels in the model, we keep checking all our inputs and outputs, and interactively work all levels of our model until we decide that we have a "complete" set of requirements. Throughout this process, we are careful to keep design decisions out of our model to avoid placing unnecessary constraints on the designer.

The following pages give a brief description of the major visual pieces of the modelling philosophy, a sample Context Diagram, and Data Flow Diagram (DFD).

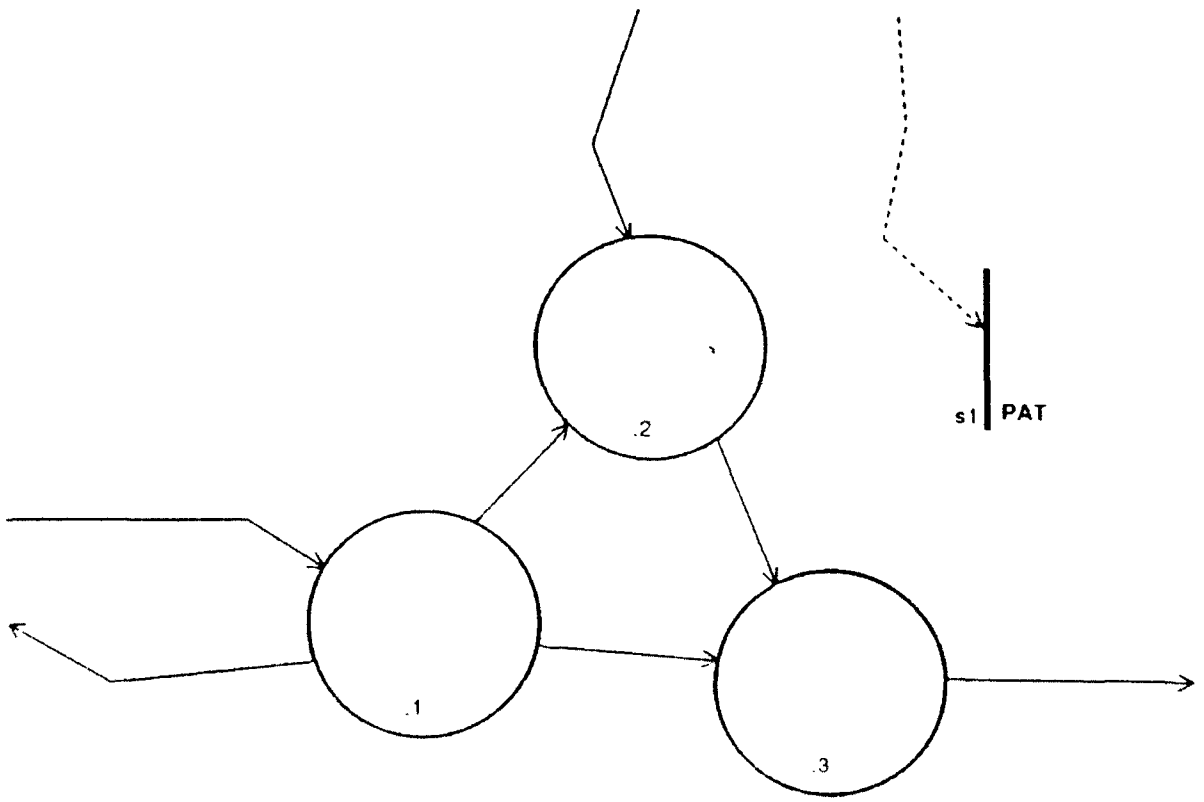
sample Context



Context Diagram

The Context Diagram shows the boundary of the system being modeled. It also shows the objects that the system interacts with. These objects are external to the system, and are referred to as its environment.

The Context Diagram contains only one process, numbered "0", representing the highest level of data flow and control flow for the system. (Note that Hatley and Pirbhai separate the data and control context diagrams.) Inside Process 0 is the top-level Data Flow Diagram (DFD), also referred to as DFD 0. The diagram below shows the relationship between the Context Diagram and DFD 0.



APPENDIX I

AATD Benchmark Source

```
        with FLOAT_MATH_LIB ; use FLOAT_MATH_LIB ;

-- Declare the main procedure.

separate (nav_exec)

-- Beginning of this procedure.

PROCEDURE air_data is

        ratio   ,
        tmp1    ,
        tmp2    ,
        tmp3    : FLOAT ;

begin


---
        ratio   := pressure_static / 29.92 ;

--- Barometric altitude.

        barometric_altitude := -32500.0 * FLOAT_MATH_LIB.LOG(ratio) ;

--- Air speed.

        tmp1        := pressure_dynamic - pressure_static ;
        tmp2        := 0.5 * 0.002378 * ratio  ;
        tmp3        := tmp1 / tmp2             ;

        air_speed := FLOAT_MATH_LIB.SQRT(tmp3) ;
end air_data;


with FLOAT_MATH_LIB     ;

separate (nav_exec)-- Declare the main procedure.


-- Beginning of this procedure.
PROCEDURE horiz_nav is
        nav_latitude_tmp ,
        nav_longitude_tmp ,
        DEL_LAT          , DEL_LONG
        C_LAT            , R_TMP                  : FLOAT ;
        package real_io is new float_io(float);

begin
---     1.0 Velocity calculations.


        nav_vel_x        := nav_vel_x + PLATFORM_X_ACCELERATION * dt ;


        nav_vel_y        := nav_vel_y + PLATFORM_Y_ACCELERATION * dt ;


---     2.0 Earth gravity and radius model.

        nav_radius       := 3600.0*5280.0          ;
        nav_gravity      := 32.2                   ;

---     3.0 Initialize some temporary variables

        C_LAT   :=  FLOAT_MATH_LIB.COSD(nav_latitude_RAD)       ;
```



```
        R_TMP   :=  FLOAT(nav_radius) + nav_altitude    ;

---     4.0 Rotation rates

        DEL_LAT          :=   nav_vel_x * dt / R_TMP              ;
        DEL_LONG         := - nav_vel_y * dt / (R_TMP*C_LAT)      ;

---     5.0 Latitude and longitude update in degrees and radians.

        nav_latitude_tmp        := nav_latitude_RAD  + DEL_LAT  ;
        nav_longitude_tmp       := nav_longitude_RAD + DEL_LONG ;

        nav_latitude_RAD        := nav_latitude_tmp     ;
        NAV_LATITUDE_DEG        := NAV_LATITUDE_RAD * 57.29578 ;
        IF NAV_LATITUDE_DEG < - 90.0 THEN
           NAV_LATITUDE_DEG := - 90.0 ;
        ELSIF NAV_LATITUDE_DEG > 90.0 THEN
           NAV_LATITUDE_DEG := 90.0 ;
        END IF ;
        nav_longitude_RAD       := nav_longitude_tmp    ;
        NAV_LONGITUDE_DEG       := NAV_LONGITUDE_RAD * 57.29578 ;
        IF NAV_LONGITUDE_DEG < -180.0 THEN
           NAV_LONGITUDE_DEG := -180.0 ;
        ELSIF NAV_LONGITUDE_DEG > 180.0 THEN
           NAV_LONGITUDE_DEG := 180.0 ;
        END IF ;


end horiz_nav;


--      Nav_exec.ada is the main calling program. This is used for debugging
--      only since the individual units will be used by an ADAS simulation.
--
--      modified by DIANE KOHALMI 8/89
--
-- The procedures emulate straight and level flight, due north.
-- The navigation calculations are general, but the sensor model is
-- set for PLATFORM_X_ACCELERATION = 0.0 , PLATFORM_Y_ACCELERATION = 0.0 ,
-- VERTICAL_ACCELERATION = 32.2 and rate_x = 0.0, rate_y = 0.0,
-- rate_z = 0.0.

        with TEXT_IO, CALENDAR, MUX_IO, MUX_IO_INPUT    ;
        use TEXT_IO, CALENDAR, MUX_IO, MUX_IO_INPUT     ;

procedure NAV_EXEC is

-- Declare the real variables.

        PLATFORM_X_ACCELERATION ,
        PLATFORM_Y_ACCELERATION ,
        VERTICAL_ACCELERATION   ,
        RATE_X   ,
        RATE_Y   ,
        RATE_Z   ,
        PRESSURE_STATIC  ,
        PRESSURE_DYNAMIC  : FLOAT := 0.0 ;                  -- sensor models

        NAV_LATITUDE_RAD ,
        NAV_LATITUDE_DEG ,
        NAV_LONGITUDE_RAD ,
        NAV_LONGITUDE_DEG ,
        NAV_ALTITUDE     ,
        NAV_VEL_X        ,
        NAV_VEL_Y        ,
        NAV_VEL_Z        ,
```

```
        NAV_BAROMETRIC_RATE
        NAV_GRAVITY      : FLOAT := 0.0 ;          -- navigation solutions
        NAV_RADIUS       : LONG_FLOAT := 0.0;

        PSI : FLOAT := 45.0 ; -- initialize PSI to 45 degrees

        THETA , PHI : FLOAT := 0.0 ; -- initialize PHI, THETA to 0 degrees

        A11  , A12 , A13 ,
        A21  , A22 , A23 ,
        A31  , A32 , A33 ,

        C11  , C12 , C13 ,
        C21  , C22 , C23 ,
        C31  , C32 , C33 ,

        DC11  , DC12 , DC13 ,
        DC21  , DC22 , DC23 ,
        DC31  , DC32 , DC33 : FLOAT := 0.0 ;     -- attitude solutions

        BAROMETRIC_ALTITUDE ,
        AIR_SPEED            : FLOAT := 0.0 ;     -- air data

        Z_LEVER_ARM : FLOAT := 1.0;

        CORIOLIS_X ,
        CORIOLIS_Y ,
        CORIOLIS_Z : FLOAT := 0.0 ;               -- coriolis

        DT                        : FLOAT := 0.0   ;       -- update time
        ELAPSED_TIME              : FLOAT := 0.0   ;       -- total run time
        EXECUTIVE_TIME_1          : FLOAT := 0.0   ;       -- executive clock
        EXECUTIVE_TIME_20         : FLOAT := 0.0   ;       -- executive clock
        NOW                       : TIME  := CLOCK ;       -- timer values
        OLD                       : TIME  := CLOCK ;       -- timer values

        LOOP_COUNTER              : INTEGER ;      -- loop variables

-- Declare output buffers

        type OUTPUT_BUFFER is array (1 .. 32) of FLOAT ;

        RT5_SA01_OUT : OUTPUT_BUFFER ;
        RT5_SA02_OUT : OUTPUT_BUFFER ;
        RT5_SA03_OUT : OUTPUT_BUFFER ;
        RT5_SA04_OUT : OUTPUT_BUFFER ;
        RT5_SA05_OUT : OUTPUT_BUFFER ;
        RT5_SA10_OUT : OUTPUT_BUFFER ;
        RT5_SA11_OUT : OUTPUT_BUFFER ;
        RT5_SA17_OUT : OUTPUT_BUFFER ;
        RT5_SA18_OUT : OUTPUT_BUFFER ;
        RT5_SA19_OUT : OUTPUT_BUFFER ;
        RT5_SA21_OUT : OUTPUT_BUFFER ;
        RT5_SA22_OUT : OUTPUT_BUFFER ;
        RT5_SA23_OUT : OUTPUT_BUFFER ;
        RT5_SA28_OUT : OUTPUT_BUFFER ;
        RT5_SA29_OUT : OUTPUT_BUFFER ;
        RT5_SA31_OUT : OUTPUT_BUFFER ;
        RT5_SA32_OUT : OUTPUT_BUFFER ;
        RT5_SA33_OUT : OUTPUT_BUFFER ;

    type OUTPUT_RECORD is record
        RT5_SA01_OUT : OUTPUT_BUFFER ;
        RT5_SA02_OUT : OUTPUT_BUFFER ;
```



```
        RT5_SA03_OUT : OUTPUT_BUFFER ;
        RT5_SA04_OUT : OUTPUT_BUFFER ;
        RT5_SA05_OUT : OUTPUT_BUFFER ;
        RT5_SA10_OUT : OUTPUT_BUFFER ;
        RT5_SA17_OUT : OUTPUT_BUFFER ;
        RT5_SA18_OUT : OUTPUT_BUFFER ;
        RT5_SA29_OUT : OUTPUT_BUFFER ;
        RT5_SA31_OUT : OUTPUT_BUFFER ;

    end record ;

    RESULT_RECORD : OUTPUT_RECORD ;
-- Declare input buffers

        RT5_SA01_IN : SMALL_INPUT_BUFFER := (others => 0.0) ;
        RT5_SA06_IN : INPUT_BUFFER := (others => 0.0) ;
        RT5_SA07_IN : INPUT_BUFFER := (others => 0.0) ;
        RT5_SA08_IN : INPUT_BUFFER := (others => 0.0) ;
        RT5_SA09_IN : INPUT_BUFFER := (others => 0.0) ;
        RT5_SA28_IN : INPUT_BUFFER := (others => 0.0) ;
        RT5_SA30_IN : INPUT_BUFFER := (others => 0.0) ;


       RAW_DATA : INPUT_DATA ;
       SENSOR_DATA : EXTRACTED_DATA ;
       SIMULATION : MUX_IO.CALCULATED_DATA ;
       RESULTS : MUX_IO.FINAL_DATA ;

       TEXT_FILE : TEXT_IO.FILE_TYPE ;
       RESULT_FILE : TEXT_IO.FILE_TYPE ;

-- The bodies of the other procedure are in separate units.

        procedure INITIALIZE is separate;
        procedure HORIZ_NAV is separate ;
        procedure VERT_NAV  is separate ;
        procedure SENSORS   is separate ;
        procedure CORIOLIS  is separate ;
        procedure AIR_DATA  is separate ;
        procedure DC_MATRIX is separate ;
        procedure NAV_ATT   is separate ;
        procedure V_BAROM   is separate ;
        procedure VCD       is separate ;
        procedure TIMER (dt : in out FLOAT)    is separate ;
        procedure GET_NAV_DATA(RT5_SA01_IN : in out SMALL_INPUT_BUFFER)
                          is separate ;
        procedure WRITE_NAV_DATA(result_record : in out FINAL_DATA ;
                                 raw_data : in out INPUT_DATA)
                          is separate ;

        PACKAGE REAL_IO is new  TEXT_IO.FLOAT_IO(FLOAT) ;
        PACKAGE FLOAT_IO is new  TEXT_IO.FLOAT_IO(FLOAT)        ;
        PACKAGE INTEGER_IO is new TEXT_IO.INTEGER_IO(INTEGER) ;

begin


        timer (dt)              ; -- initialize the timer
        delay 0.5        ; -- delay needed to prevent clock underflow


        initialize       ; -- initialization routine

        TEXT_IO.OPEN(TEXT_FILE, TEXT_IO.IN_FILE,
          "[8825_USERS.KORALMI.AATD.TESTNAV]NAVSIM.DAT");
```

```
TEXT_IO.CREATE(RESULT_FILE, TEXT_IO.OUT_FILE,
  "[8825_USERS.KOHALMI.AATD.TESTNAV]RESULTS.DAT");
TEXT_IO.SET_INPUT(TEXT_FILE) ;
TEXT_IO.SET_OUTPUT(RESULT_FILE) ;

READ_DATA_FILE:
WHILE NOT TEXT_IO.END_OF_FILE(TEXT_FILE) LOOP

RUN_SIMULATION:
FOR LOOP_COUNTER in 1 .. 20  LOOP

IF TEXT_IO.END_OF_FILE(TEXT_FILE) THEN
        EXIT RUN_SIMULATION ;
END IF ;

executive_time_20        := 0.0 ;

LOOP
        executive_time_20        := executive_time_20 + dt ;
exit when executive_time_20  > 0.05 ; -- 20 Hz.
end LOOP ;

executive_time_1 := executive_time_1 + executive_time_20 ; -- 1 Hz.


RAW_DATA.RT5_SA01_IN := RT5_SA01_IN ;
RAW_DATA.RT5_SA06_IN := RT5_SA06_IN ;
RAW_DATA.RT5_SA07_IN := RT5_SA07_IN ;
RAW_DATA.RT5_SA08_IN := RT5_SA08_IN ;
RAW_DATA.RT5_SA09_IN := RT5_SA09_IN ;
RAW_DATA.RT5_SA28_IN := RT5_SA28_IN ;
RAW_DATA.RT5_SA30_IN := RT5_SA30_IN ;

timer (dt)               ; -- real time clock
elapsed_time             := elapsed_time + dt ;

TEXT_IO.PUT("ELAPSED TIME: ") ;
FLOAT_IO.PUT(ELAPSED_TIME) ;
TEXT_IO.NEW_LINE ;

TEXT_IO.PUT("DT: ") ;
FLOAT_IO.PUT(DT) ;
TEXT_IO.NEW_LINE ;

TEXT_IO.PUT("PHI: ") ;
FLOAT_IO.PUT(PHI) ;
TEXT_IO.NEW_LINE ;

TEXT_IO.PUT("THETA: ") ;
FLOAT_IO.PUT(THETA) ;
TEXT_IO.NEW_LINE ;

TEXT_IO.PUT("PSI: ") ;
FLOAT_IO.PUT(PSI) ;
TEXT_IO.NEW_LINE ;


GET_NAV_DATA(RT5_SA01_IN) ;
RAW_DATA.RT5_SA01_IN := RT5_SA01_IN ;

MUX_IO_INPUTS(RAW_DATA, sensor_data) ;

  PLATFORM_X_ACCELERATION := SENSOR_DATA.PLATFORM_X_ACCELERATION ;
  PLATFORM_Y_ACCELERATION := SENSOR_DATA.PLATFORM_Y_ACCELERATION ;
  VERTICAL_ACCELERATION := SENSOR_DATA.VERTICAL_ACCELERATION ;
  RATE_X := SENSOR_DATA.RATE_X ;
```



```
  RATE_Y := SENSOR_DATA.RATE_Y ;
  RATE_Z := SENSOR_DATA.RATE_Z ;
  BAROMETRIC_ALTITUDE := SENSOR_DATA.BAROMETRIC_ALTITUDE ;
  NAV_VEL_X := SENSOR_DATA.NAV_VEL_X ;
  NAV_VEL_Y := SENSOR_DATA.NAV_VEL_Y ;
  NAV_VEL_Z := SENSOR_DATA.NAV_VEL_Z ;
  NAV_LATITUDE_DEG := SENSOR_DATA.NAV_LATITUDE_DEG ;
  NAV_LONGITUDE_DEG := SENSOR_DATA.NAV_LONGITUDE_DEG ;

air_data        ; -- air data solutions
sensors         ; -- inertial and pressure sensors
horiz_nav       ; -- horizontal navigation solutions
vert_nav        ; -- vertical navigation solutions
dc_matrix       ; -- direction cosine matrix update
nav_att         ; -- navigation attitude solutions
coriolis        ; -- Coriolis corrections
v_barom         ; -- navigation differentiated barometric altitude

SIMULATION.PLATFORM_X_ACCELERATION := PLATFORM_X_ACCELERATION ;
SIMULATION.PLATFORM_Y_ACCELERATION := PLATFORM_Y_ACCELERATION ;
SIMULATION.VERTICAL_ACCELERATION := VERTICAL_ACCELERATION ;
SIMULATION.RATE_X := RATE_X ;
SIMULATION.RATE_Y := RATE_Y ;
SIMULATION.RATE_Z := RATE_Z ;
SIMULATION.PRESSURE_STATIC := PRESSURE_STATIC ;
SIMULATION.PRESSURE_DYNAMIC := PRESSURE_DYNAMIC ;
SIMULATION.PSI := PSI ;
SIMULATION.PHI := PHI ;
SIMULATION.THETA := THETA ;
SIMULATION.NAV_LATITUDE_DEG  := NAV_LATITUDE_DEG ;
SIMULATION.NAV_LATITUDE_RAD  := NAV_LATITUDE_RAD ;
SIMULATION.NAV_LONGITUDE_DEG  := NAV_LONGITUDE_DEG ;
SIMULATION.NAV_LONGITUDE_RAD  := NAV_LONGITUDE_RAD ;
SIMULATION.NAV_ALTITUDE := NAV_ALTITUDE ;
SIMULATION.NAV_VEL_X := NAV_VEL_X ;
SIMULATION.NAV_VEL_Y := NAV_VEL_Y ;
SIMULATION.NAV_VEL_Z := NAV_VEL_Z ;
SIMULATION.Z_LEVER_ARM := Z_LEVER_ARM ;
SIMULATION.NAV_BAROMETRIC_RATE := NAV_BAROMETRIC_RATE ;
SIMULATION.NAV_GRAVITY := NAV_GRAVITY ;
SIMULATION.NAV_RADIUS :=NAV_RADIUS ;
SIMULATION.BAROMETRIC_ALTITUDE := BAROMETRIC_ALTITUDE ;
SIMULATION.AIR_SPEED := AIR_SPEED ;

mux_io_OUTPUTS(SIMULATION, RESULTS)     ; -- 1553 input-output


IF executive_time_1 > 1.0 THEN
        vcd             ;               -- 1 Hz. model;
                                -- vertical channel divergence control
        executive_time_1 := 0.0 ;
end IF;



END LOOP RUN_SIMULATION ;

END LOOP READ_DATA_FILE ;

WRITE_NAV_DATA(RESULTS, RAW_DATA) ;


TEXT_IO.CLOSE(TEXT_FILE) ;
TEXT_IO.CLOSE(RESULT_FILE) ;
```

```
end NAV_EXEC ;
----- Vcd is used to control the vertical channel divergence phenomenon.
----- This routine operates at 1/20 times the fundamental rate (20 Hz.),
----- or 1 Hz. However, the differentiation algorithm operates at 20 Hz.
-- --
----- Inputs:
----- dt
----- barometric_altitude
----- barometric_rate

-- Declare the main procedure.

separate (nav_exec)

-- Beginning of this procedure.

PROCEDURE vcd is

        altitude_error  ,
        U1              ,
        U2              ,
        v_2             ,
        p               ,
        q               ,
        r               ,
        w               ,
        C1              ,
        C2              ,
        C3              ,
        vertical_acceleration   ,
        a_z_tmp         ,
        nav_barometric_rate_tmp ,
        local_time      : FLOAT ;
begin

--- 1. Construct the control signals.

        local_time := 20.0 * dt ;       -- 1 Hz.

--  1.1 Check limits.

        IF
                nav_barometric_rate >  117.0 then
                nav_barometric_rate :=  117.0 ;
        END IF ;
        IF
                nav_barometric_rate < -117.0 then
                nav_barometric_rate := -117.0 ;
        END IF ;

        altitude_error := barometric_altitude - nav_altitude ;

        IF
                altitude_error >  1000.0 then
                altitude_error :=  1000.0 ;
        END IF ;
        IF
                altitude_error < -1000.0 then
                altitude_error := -1000.0 ;
        END IF ;

-- 1.2 Calculate the coefficients.

        v_2     := nav_barometric_rate * nav_barometric_rate ;
```



```
        q       := v_2 / ( v_2 + 2500.0)        ;
        p       := (1.0 - q) / 100.0            ;
        w       := 2.0 * nav_gravity / FLOAT(nav_radius) ;

        C1      := 2.0 * p * p * p              ;
        C2      := w * w + 4.0 * p * p          ;
        C3      := 3.0 * p                      ;

-- 1.3 Calculate the control signals.

        U1      := C1 * altitude_error * local_time +
                       C2 * altitude_error ;

        U2      := C3 * altitude_error ;

-- 1.4 Reset the control variables.

        a_z_tmp := vertical_acceleration + U1   ;
        vertical_acceleration   := a_z_tmp      ;

        nav_barometric_rate_tmp := nav_barometric_rate + U2 ;
        nav_barometric_rate     := nav_barometric_rate_tmp  ;

end vcd;
-- Declare the main procedure.

separate (nav_exec)

-- Beginning of this procedure.
PROCEDURE coriolis is

begin

        coriolis_x := 0.0 ;
        coriolis_y := 0.0 ;
        coriolis_z := 0.0 ;

end coriolis;

        with FLOAT_MATH_LIB ; use FLOAT_MATH_LIB ;

-- Declare the main procedure.

separate (nav_exec)

-- Beginning of this procedure.

PROCEDURE initialize is

        ratio : FLOAT ;
        COS_THETA, SIN_THETA,
        COS_PSI, SIN_PSI,
        COS_PHI, SIN_PHI : FLOAT ;

begin

        coriolis_x := 0.0       ;
        coriolis_y := 0.0       ;
        coriolis_z := 0.0       ;

        nav_latitude_RAD := 0.7853981 ; -- 45 degrees
        NAV_LATITUDE_DEG := 45.0 ;
        R15_SA09_IN(01) := NAV_LATITUDE_DEG ;
        nav_longitude_RAD := 0.1474801 ; -- 39 degrees
        NAV_LONGITUDE_DEG := 39.0 ;
```

```
RT5_SA09_IN(03) := NAV_LONGITUDE_DEG ;
nav_altitude    := 10000.0;
RT5_SA07_IN(01) := NAV_ALTITUDE ;

nav_vel_x       := 1000.0 ;
RT5_SA07_IN(04) := NAV_VEL_X ;
nav_vel_y       := 0.0    ;
RT5_SA07_IN(05) := NAV_VEL_Y ;
nav_vel_z       := 0.0    ;
RT5_SA07_IN(06) := NAV_VEL_Z ;

COS_THETA := FLOAT_MATH_LIB.COSD(THETA) ;
SIN_THETA := FLOAT_MATH_LIB.SIND(THETA) ;
COS_PSI := FLOAT_MATH_LIB.COSD(PSI) ;
SIN_PSI := FLOAT_MATH_LIB.SIND(PSI) ;
COS_PHI := FLOAT_MATH_LIB.COSD(PHI) ;
SIN_PHI := FLOAT_MATH_LIB.SIND(PHI) ;

A11     := COS_THETA * COS_PSI ;
A12     := SIN_PSI * COS_THETA ;
A13     := - SIN_THETA ;

A21     := (-SIN_PSI*COS_PHI) + (COS_PSI*SIN_THETA*SIN_PHI) ;
A22     := (COS_PSI*COS_PHI) + (SIN_PSI*SIN_THETA*SIN_PHI) ;
A23     := COS_THETA * SIN_PHI ;

A31     := (SIN_PHI*SIN_PSI) + (COS_PSI*SIN_THETA*COS_PHI) ;
A32     := (-COS_PSI*SIN_PHI) + (SIN_PSI*SIN_THETA*COS_PHI) ;
A33     := COS_THETA * COS_PHI ;

C11     := 1.0 ;
C12     := 0.0 ;
C13     := 0.0 ;

C21     := 0.0 ;
C22     := 1.0 ;
C23     := 0.0 ;

C31     := 0.0 ;
C32     := 0.0 ;
C33     := 1.0 ;

DC11    := 0.0 ;
DC12    := 0.0 ;
DC13    := 0.0 ;

DC21    := 0.0 ;
DC22    := 0.0 ;
DC23    := 0.0 ;

DC31    := 0.0 ;
DC32    := 0.0 ;
DC33    := 0.0 ;


barometric_altitude :=  nav_altitude ;
air_speed           :=
        FLOAT_MATH_LIB.SQRT(nav_vel_x * nav_vel_x  +
                            nav_vel_y * nav_vel_y) ;

ratio := FLOAT_MATH_LIB.EXP(-barometric_altitude / 32500.0) ;

pressure_static   := 29.92 * ratio ;

pressure_dynamic  :=
        pressure_static +
```



```
                0.5 * 0.002378 * air_speed * air_speed * ratio ;

end initialize;

-- MUX_IO_OUTPUTS  is used to set the MIL-STD-1553B output buffers.
-- Diane Kohalmi 6/89

WITH TEXT_IO;
USE TEXT_IO;

PACKAGE MUX_IO IS

  type OUTPUT_BUFFER is array (1 .. 32) of float ;

        RT5_SA01_OUT : OUTPUT_BUFFER ;
        RT5_SA02_OUT : OUTPUT_BUFFER ;
        RT5_SA03_OUT : OUTPUT_BUFFER ;
        RT5_SA04_OUT : OUTPUT_BUFFER ;
        RT5_SA05_OUT : OUTPUT_BUFFER ;
        RT5_SA10_OUT : OUTPUT_BUFFER ;
        RT5_SA11_OUT : OUTPUT_BUFFER ;
        RT5_SA17_OUT : OUTPUT_BUFFER ;
        RT5_SA18_OUT : OUTPUT_BUFFER ;
        RT5_SA19_OUT : OUTPUT_BUFFER ;
        RT5_SA21_OUT : OUTPUT_BUFFER ;
        RT5_SA22_OUT : OUTPUT_BUFFER ;
        RT5_SA23_OUT : OUTPUT_BUFFER ;
        RT5_SA28_OUT : OUTPUT_BUFFER ;
        RT5_SA29_OUT : OUTPUT_BUFFER ;
        RT5_SA31_OUT : OUTPUT_BUFFER ;
        RT5_SA32_OUT : OUTPUT_BUFFER ;
        RT5_SA33_OUT : OUTPUT_BUFFER ;


  type FINAL_DATA is record
        RT5_SA01_OUT : OUTPUT_BUFFER ;
        RT5_SA02_OUT : OUTPUT_BUFFER ;
        RT5_SA03_OUT : OUTPUT_BUFFER ;
        RT5_SA04_OUT : OUTPUT_BUFFER ;
        RT5_SA05_OUT : OUTPUT_BUFFER ;
        RT5_SA10_OUT : OUTPUT_BUFFER ;
        RT5_SA11_OUT : OUTPUT_BUFFER ;
        RT5_SA17_OUT : OUTPUT_BUFFER ;
        RT5_SA18_OUT : OUTPUT_BUFFER ;
        RT5_SA19_OUT : OUTPUT_BUFFER ;
        RT5_SA21_OUT : OUTPUT_BUFFER ;
        RT5_SA22_OUT : OUTPUT_BUFFER ;
        RT5_SA23_OUT : OUTPUT_BUFFER ;
        RT5_SA28_OUT : OUTPUT_BUFFER ;
        RT5_SA29_OUT : OUTPUT_BUFFER ;
        RT5_SA31_OUT : OUTPUT_BUFFER ;
        RT5_SA32_OUT : OUTPUT_BUFFER ;
        RT5_SA33_OUT : OUTPUT_BUFFER ;
  end record;

  type CALCULATED_DATA is record
        PLATFORM_X_ACCELERATION : float ;
        PLATFORM_Y_ACCELERATION : float ;
        VERTICAL_ACCELERATION   : float ;
        rate_x  : float ;
        rate_y  : float ;
        rate_z  : float ;
        pressure_static         : float ;
        pressure_dynamic        : float ;
        PSI                     : float ;
        THETA                   : float ;
```

```
        PHI : FLOAT ;            -- sensor models

        nav_latitude_RAD : float ;
        NAV_LATITUDE_DEG : float ;
        nav_longitude_RAD : float ;
        NAV_LONGITUDE_DEG : float ;
        nav_altitude     : float ;
        nav_vel_x        : float ;
        nav_vel_y        : float ;
        nav_vel_z        : float ;
        Z_LEVER_ARM      : float ;
        nav_barometric_rate  : float ;
        nav_gravity      : FLOAT ;        -- navigation solutions
        nav_radius       : LONG_FLOAT ;



        barometric_altitude : float ;
        air_speed           : FLOAT ;    -- air data
    end record ;

    SIM_DATA : CALCULATED_DATA ;
    RESULTS : FINAL_DATA ;

        PROCEDURE MUX_IO_OUTPUTS(SIM_DATA : in out CALCULATED_DATA ;
                                 RESULTS : in out FINAL_DATA) ;

END MUX_IO ;

PACKAGE BODY MUX_IO IS

PROCEDURE MUX_IO_OUTPUTS (SIM_DATA : in out CALCULATED_DATA ;
                          RESULTS : in out FINAL_DATA) IS

        OUTPUT_FILE : TEXT_IO.FILE_TYPE ;
        ARRAY_COUNTER : INTEGER := 1 ;
        PACKAGE RESULT_IO IS NEW FLOAT_IO(FLOAT) ;


BEGIN

-- MESSAGE 1: 4 DATA WORDS

        RT5_SA01_OUT := (SIM_DATA.PSI,
                         SIM_DATA.THETA,
                         SIM_DATA.PHI,
                         OTHERS => 0.0) ;
        RESULTS.RT5_SA01_OUT := RT5_SA01_OUT ;

-- MESSAGE 2: AHRS STATUS WORD

        RT5_SA02_OUT := (45056.0 ,
                         OTHERS => 0.0) ;
        RESULTS.RT5_SA02_OUT := RT5_SA02_OUT ;

-- MESSAGE 3: 15 DATA WORDS AND 1 STATUS WORD

        RT5_SA03_OUT := (32768.0  ,
                         32768.0  ,
                         4576.0   ,
                         SIM_DATA.PSI ,
                         SIM_DATA.NAV_VEL_X ,
                         SIM_DATA.NAV_VEL_Y ,
                         SIM_DATA.NAV_VEL_Z ,
                         SIM_DATA.PLATFORM_X_ACCELERATION ,
                         SIM_DATA.PLATFORM_Y_ACCELERATION,
```



```
                         SIM_DATA.VERTICAL_ACCELERATION ,
                         SIM_DATA.RATE_X ,
                         SIM_DATA.RATE_Y ,
                         SIM_DATA.RATE_Z ,
                         OTHERS => 0.0) ;
        RESULTS.RT5_SA03_OUT := RT5_SA03_OUT ;


-- MESSAGE 4: 30 DATA WORDS AND 1 STATUS WORD

        RT5_SA04_OUT := (32768.0 ,
                         32768.0 ,
                         49139.0 ,
                         SIM_DATA.PSI   ,
                         SIM_DATA.PSI ,
                         SIM_DATA.THETA ,
                         SIM_DATA.PHI ,
                         SIM_DATA.PHI ,
                         SIM_DATA.NAV_VEL_Y ,
                         SIM_DATA.NAV_VEL_X ,
                         SIM_DATA.NAV_VEL_Z ,
                         SIM_DATA.BAROMETRIC_ALTITUDE ,
                         SIM_DATA.NAV_LATITUDE_DEG ,
                         SIM_DATA.NAV_LONGITUDE_DEG ,
                         2050.0 ,
                         SIM_DATA.platform_y_ACCELERATION ,
                         SIM_DATA.platform_x_ACCELERATION ,
                         SIM_DATA.VERTICAL_ACCELERATION ,
                         1.0    ,
                         1.0    ,
                         0.0 ,
                         0.0 ,
                         0.0 ,
                         SIM_DATA.RATE_X ,
                         SIM_DATA.RATE_Y ,
                         SIM_DATA.RATE_Z ,
                         SIM_DATA.Z_LEVER_ARM ,
                         OTHERS => 0.0 ) ;

        RESULTS.RT5_SA04_OUT := RT5_SA04_OUT ;


-- MESSAGE FIVE: SAHRS VALUES WILL BE UPDATED WHEN THAT SECTION IS
-- COMPLETED; THIS MESSAGE CONTAINS 22 DATA WORDS AND 2 STATUS WORDS

        RT5_SA05_OUT := (OTHERS => 0.0) ;
        RESULTS.RT5_SA05_OUT := RT5_SA05_OUT ;


-- MESSAGE 10: CODE IDENTIFICATION WORDS, SET IN INITIALIZATION ROUTINE

--      RT5_SA10_OUT(01) :=                    ; -- := 00892 '84'
--      RT5_SA10_OUT(02) :=                    ; -- := 13184 '80'
--      RT5_SA10_OUT(03) :=                    ; -- := 00540 '2 '
--      RT5_SA10_OUT(04) :=                    ; -- := 37840 'L0'

        RT5_SA10_OUT := (OTHERS => 0.0) ;
        RESULTS.RT5_SA10_OUT := RT5_SA10_OUT ;

-- MESSAGE 11: REFERENCE AND CORRECTION DATA WORDS

--      RT5_SA11_OUT(01) := 0.0 ; -- TIME OF TRANSMISSION
--      RT5_SA11_OUT(02) := 0.0 ; -- TIME VALID
--      RT5_SA11_OUT(03) := 0.0 ; -- MODE BIT DEFINITION
--      RT5_SA11_OUT(04) := 0.0 ; -- REFERENCE LATITUDE
```

```
--      RT5_SA11_OUT(06) := 0.0 ; -- REFERENCE LONGITUDE
--      RT5_SA11_OUT(08) := 0.0 ; -- REFERENCE ALTITUDE
--      RT5_SA11_OUT(09) := 0.0 ; -- REFERENCE HEADING
--      RT5_SA11_OUT(10) := 0.0 ; -- X VELOCITY CORRECTION
--      RT5_SA11_OUT(11) := 0.0 ; -- Y VELOCITY CORRECTION
--      RT5_SA11_OUT(12) := 0.0 ; -- Z VELOCITY CORRECTION
--      RT5_SA11_OUT(13) := 0.0 ; -- HEADING CORRECTION
--      RT5_SA11_OUT(14) := 0.0 ; -- LOCAL LEVEL X TILT CORRECTION
--      RT5_SA11_OUT(15) := 0.0 ; -- LOCAL LEVEL Y TILT CORRECTION

        RT5_SA11_OUT := (OTHERS => 0.0) ;
        RESULTS.RT5_SA11_OUT := RT5_SA11_OUT ;

-- MESSAGE 17: KALMAN FILTER VALUES TBD

        RT5_SA17_OUT := (OTHERS => 0.0) ;
        RESULTS.RT5_SA17_OUT := RT5_SA17_OUT ;

-- MESSAGE 18: KALMAN FILTER VALUES TBD

        RT5_SA18_OUT := (OTHERS => 0.0) ;
        RESULTS.RT5_SA18_OUT := RT5_SA18_OUT ;

-- MESSAGE 19: VELOCITY ERRORS AND POSITION CORRECTIONS - ALL ZERO AT THIS TIME

        RT5_SA19_OUT := (OTHERS => 0.0) ;
        RESULTS.RT5_SA19_OUT := RT5_SA19_OUT ;

-- MESSAGE 21: DATA DISPLAY CONTROL - ALL ZERO AT THIS TIME

        RT5_SA21_OUT := (OTHERS => 0.0) ;
        RESULTS.RT5_SA21_OUT := RT5_SA21_OUT ;

-- MESSAGE 22: LINE 1 AND 2 OF SCREEN DISPLAY - ALL ZERO AT THIS TIME

        RT5_SA22_OUT := (OTHERS => 0.0) ;
        RESULTS.RT5_SA22_OUT := RT5_SA22_OUT ;

-- MESSAGE 23: LINE 3 AND 4 OF SCREEN DISPLAY - ALL ZERO AT THIS TIME

        RT5_SA23_OUT := (OTHERS => 0.0) ;
        RESULTS.RT5_SA23_OUT := RT5_SA23_OUT ;

-- MESSAGE 29: MEMORY DATA WORDS

        RT5_SA29_OUT := (OTHERS => 0.0 ) ;
        RESULTS.RT5_SA29_OUT := RT5_SA29_OUT ;

-- MESSAGE 31: STATUS SENT TO MC ON REQUEST

        RT5_SA31_OUT := (1.0     ,
                         30.0    ,
                         0.0     ,
                         0.0     ,
                         0.0     ,
                         65397.0 ,
                         2214.0  ,
                         8160.0  ,
                         0.0     ,
                         2114.0  ,
                         OTHERS => 0.0 ) ;
        RESULTS.RT5_SA31_OUT := RT5_SA31_OUT ;


-- REMAINDER OF MESSAGE 31 IS STATUS WORDS SET TO ZERO
```



```
END MUX_IO_OUTPUTS ;

END MUX_IO;

-- Declare the main procedure.

separate (nav_exec)

-- Beginning of this procedure.
-- Variables declared (global) in nav_exec are:
--      vertical_acceleration
--      coriolis_z
--      nav_gravity
--      dt                  [FLOAT, 0.05 seconds (20 HZ.)]

PROCEDURE vert_nav is

        accel_z          ,
        nav_vel_z_tmp    ,
        nav_altitude_tmp: FLOAT ;
begin

-----
-----   1. Normalize the sensor input by subtracting off 1 g,
-----      then integrate.
-----

--- 1.1 Net vertical acceleration.

        accel_z := vertical_acceleration  + coriolis_z - nav_gravity ;

--- 1.2 Velocity.

        nav_vel_z_tmp   := nav_vel_z + accel_z * dt  ;
        nav_vel_z       := nav_vel_z_tmp ;

--- 1.3 Altitude.

        nav_altitude_tmp := nav_altitude + nav_vel_z * dt ;
        nav_altitude     := nav_altitude_tmp ;


end vert_nav;
---
---     Dc_matrix.ada updates the direction cosine (DC) matrix
---     used by the attitude calculations (nav_att.ada).
---
---     John Newport 5/2/88
---

-- Declare the main procedure.

separate (nav_exec)

-- Beginning of this procedure.

PROCEDURE dc_matrix is

        PX              , PY            , PZ            ,
        DC11_dot        , DC12_dot      , DC13_dot      ,
        DC21_dot        , DC22_dot      , DC23_dot      ,
        DC31_dot        , DC32_dot      , DC33_dot      : FLOAT ;

begin
```

```
--- Transport rates

        PX      :=      0.0 ;
        PY      :=      0.0 ;
        PZ      :=      0.0 ;

--- Construct the derivatives

        DC11_dot        := DC12 * PZ - DC13 * PY  ;
        DC12_dot        := DC13 * PX - DC11 * PZ  ;
        DC13_dot        := DC11 * PY - DC12 * PX  ;

        DC21_dot        := DC22 * PZ - DC23 * PY  ;
        DC22_dot        := DC23 * PX - DC21 * PZ  ;
        DC23_dot        := DC21 * PY - DC22 * PX  ;

        DC31_dot        := DC32 * PZ - DC33 * PY  ;
        DC32_dot        := DC33 * PX - DC31 * PZ  ;
        DC33_dot        := DC31 * PY - DC32 * PX  ;

--- Integrate

        DC11            := DC11_dot * dt + DC11 ;
        DC12            := DC12_dot * dt + DC12 ;
        DC13            := DC13_dot * dt + DC13 ;

        DC21            := DC21_dot * dt + DC21 ;
        DC22            := DC22_dot * dt + DC22 ;
        DC23            := DC23_dot * dt + DC23 ;

        DC31            := DC31_dot * dt + DC31 ;
        DC32            := DC32_dot * dt + DC32 ;
        DC33            := DC33_dot * dt + DC33 ;


end dc_matrix ;
        with FLOAT_MATH_LIB ; use FLOAT_MATH_LIB ;

-- Declare the main procedure.

separate (nav_exec)

-- Beginning of this procedure.

PROCEDURE sensors is

        ratio, platform_x_acceleration, platform_y_acceleration, *
        vertical_acceleration : FLOAT ;

begin

--- Acceleration

--      platform_x_acceleration :=   0.0 ;
--      platform_y_acceleration :=   0.0 ;
--      vertical_acceleration   :=  32.2 ;

--- Angular rates

--      rate_x  :=   0.0 ;
--      rate_y  :=   0.0 ;
--      rate_z  :=   0.0 ;

--- Air data
```



```
        ratio := FLOAT_MATH_LIB.EXP(-barometric_altitude / 32500.0) ;

        pressure_static   := 29.92 * ratio ;

        pressure_dynamic  :=
                pressure_static +
                0.5 * 0.002378 * air_speed * air_speed * ratio ;

end sensors;

-- MUX_IO_INPUTS  is used to extract data from the MIL-STD-1553B input buffers.
--
-- Diane Kohalmi 8/89


Package MUX_IO_INPUT IS

     type INPUT_BUFFER is array (1 .. 32) of float ;
     type SMALL_INPUT_BUFFER is array (1 .. 6) of float ;

     type INPUT_DATA is record
        RT5_SA01_IN : SMALL_INPUT_BUFFER ;
        RT5_SA06_IN : INPUT_BUFFER ;
        RT5_SA07_IN : INPUT_BUFFER ;
        RT5_SA08_IN : INPUT_BUFFER ;
        RT5_SA09_IN : INPUT_BUFFER ;
        RT5_SA28_IN : INPUT_BUFFER ;
        RT5_SA30_IN : INPUT_BUFFER ;
     end record;

     RAW_DATA : INPUT_DATA ;

     type EXTRACTED_DATA is record
        PLATFORM_X_ACCELERATION : float := 0.0 ;
        PLATFORM_Y_ACCELERATION : float := 0.0 ;
        VERTICAL_ACCELERATION : float := 0.0 ;
        RATE_X : float := 0.0 ;
        RATE_Y : float := 0.0 ;
        RATE_Z : float := 0.0 ;
        BAROMETRIC_ALTITUDE : float := 0.0 ;
        NAV_VEL_X : float := 0.0 ;
        NAV_VEL_Y : float := 0.0 ;
        NAV_VEL_Z : float := 0.0 ;
        PSI : float := 45.0 ; -- PSI is initialized tp 45 degrees
        PHI : float := 0.0 ;
        THETA : float := 0.0 ;
        NAV_LATITUDE_DEG : float := 0.0 ;
        NAV_LONGITUDE_DEG : float := 0.0 ;
        Z_LEVER_ARM : FLOAT:= 0.0 ;
     end record ;

     SENSOR_DATA : EXTRACTED_DATA ;

PROCEDURE MUX_IO_INPUTS (RAW_DATA : IN OUT INPUT_DATA;
                         SENSOR_DATA : in OUT EXTRACTED_DATA );

END MUX_IO_INPUT;


PACKAGE BODY MUX_IO_INPUT IS

PROCEDURE MUX_IO_INPUTS (RAW_DATA : IN OUT INPUT_DATA;
                         SENSOR_DATA : in OUT EXTRACTED_DATA) IS
BEGIN
```

```
-- MESSAGE 1: 6 INPUT WORDS FROM MASTER COMPUTER

SENSOR_DATA.PLATFORM_X_ACCELERATION := RAW_DATA.RT5_SA01_IN(01) ;
SENSOR_DATA.PLATFORM_Y_ACCELERATION := RAW_DATA.RT5_SA01_IN(02) ;
SENSOR_DATA.VERTICAL_ACCELERATION := RAW_DATA.RT5_SA01_IN(03) ;
SENSOR_DATA.RATE_X := RAW_DATA.RT5_SA01_IN(04) ;
SENSOR_DATA.RATE_Y := RAW_DATA.RT5_SA01_IN(05) ;
SENSOR_DATA.RATE_Z := RAW_DATA.RT5_SA01_IN(06) ;


-- MESSAGE 7: 8 DATA WORDS

-- SENSOR_DATA.BAROMETRIC_ALTITUDE := RAW_DATA.RT5_SA07_IN(01) ;
-- SENSOR_DATA.NAV_VEL_X := RAW_DATA.RT5_SA07_IN(04) ; -- NORTH_SOUTH_VELOCITY ;
-- SENSOR_DATA.NAV_VEL_Y := RAW_DATA.RT5_SA07_IN(05) ; -- EAST_WEST_VELOCITY ;
-- SENSOR_DATA.NAV_VEL_Z := RAW_DATA.RT5_SA07_IN(06) ; -- VERTICAL_VELOCITY ;


-- MESSAGE 9: 12 DATA WORDS

-- SENSOR_DATA.NAV_LATITUDE_DEG := RAW_DATA.RT5_SA09_IN(01) ; -- PRESENT_LATITUD
E
-- SENSOR_DATA.NAV_LONGITUDE_DEG := RAW_DATA.RT5_SA09_IN(03) ; -- PRESENT_LONGIT
UDE
-- SENSOR_DATA.PSI := RAW_DATA.RT5_SA09_IN(10) ; -- CARRIER_HEADING
-- SENSOR_DATA.Z_LEVER_ARM := RAW_DATA.RT5_SA09_IN(12) ;


END MUX_IO_INPUTS;

END MUX_IO_INPUT;
----- V_barometric differentiates the barometric altitude. This
----- routine operates at 20 Hz.
-----
----- Inputs:
-----
----- barometric_altitude

-- Declare the main procedure.

separate (nav_exec)

-- Beginning of this procedure.

PROCEDURE v_barom is

        alt_1   ,
        alt_2   ,
        alt_3   ,
        alt_4   ,
        alt_5   ,
        v_tmp   : FLOAT ;

begin

--- 1. Reset the stack.

        alt_5 := alt_4  ;
        alt_4 := alt_3  ;
        alt_3 := alt_2  ;
        alt_2 := alt_1  ;
        alt_1 := barometric_altitude ;
```



```
--- 2. Differentiate (Five point algorithm.)

        IF dt /= 0.0 THEN

        v_tmp :=  (alt_1 - 8.0 * alt_2 +
                8.0 * alt_4 - alt_5) / ( 12.0 * dt ) ;

        END IF;

--- 3. Set the output.

        nav_barometric_rate := v_tmp ;

end v_barom;
separate (NAV_EXEC)
procedure GET_NAV_DATA(RT5_SA01_IN : in out SMALL_INPUT_BUFFER)  is
  COUNTER : integer := 1 ;
  package FLOAT_IO is new TEXT_IO.FLOAT_IO(FLOAT) ;
begin
  while (COUNTER < 7) loop
    FLOAT_IO.GET(TEXT_FILE, RT5_SA01_IN(COUNTER)) ;
    COUNTER := COUNTER + 1 ;
  end loop ;
end GET_NAV_DATA ;
--- This procedure performs the rotational calculations

        with FLOAT_MATH_LIB ; use FLOAT_MATH_LIB ;

separate (nav_exec)     -- Declare the main procedure.

PROCEDURE nav_att is


        P_INT   , Q_INT   ,  R_INT        ,
        kappa_m , cosc_kappa_m , sinc_kappa_m : FLOAT ;

        ARGS_PHI          , ARGC_PHI        ,
        ARGS_PSI          , ARGC_PSI        ,
        ABS_ARGS_PHI      , ABS_ARGC_PHI    ,
        ABS_ARGS_PSI      , ABS_ARGC_PSI    ,
        PHIO              , PSIO            ,
        CF        , SF    , CP      , SP    ,
        cosine_theta      , abs_A13         : FLOAT ;

        M11  , M12  , M13  ,
        M21  , M22  , M23  ,
        M31  , M32  , M33  : FLOAT ;

        MSQ11  , MSQ12  , MSQ13  ,
        MSQ21  , MSQ22  , MSQ23  ,
        MSQ31  , MSQ32  , MSQ33  : FLOAT ;

        RF11  , RF12  , RF13  ,
        RF21  , RF22  , RF23  ,
        RF31  , RF32  , RF33  : FLOAT ;


        AR11  , AR12  , AR13  ,
        AR21  , AR22  , AR23  ,
        AR31  , AR32  , AR33  : FLOAT ;

        A11_TMP  , A12_TMP  , A13_TMP  ,
        A21_TMP  , A22_TMP  , A23_TMP  ,
        A31_TMP  , A32_TMP  , A33_TMP  : FLOAT ;

begin
```

```
---     Calculate the integrals of P,Q,R

        P_INT :=  rate_x * dt   ;
        Q_INT :=  rate_y * dt   ;
        R_INT :=  rate_z * dt   ;

---     Calculate the sinc and cosc

        kappa_m := FLOAT_MATH_LIB.SQRT(P_INT*P_INT
                + Q_INT*Q_INT + R_INT*R_INT) ;

        cosc_kappa_m    := 0.5  ;
        sinc_kappa_m    := 1.0  ;

        IF kappa_m /= 0.0 THEN
                sinc_kappa_m := FLOAT_MATH_LIB.SIN(kappa_m) / kappa_m ;
                cosc_kappa_m := (1.0 -
                        FLOAT_MATH_LIB.COS(kappa_m))/(kappa_m * kappa_m) ;
        END IF;

---     Calculate the M matrix


        M11 :=  0.0     ;
        M12 :=  R_INT   ;
        M13 := -Q_INT   ;

        M21 := -M12     ;
        M22 :=  0.0     ;
        M23 :=  P_INT   ;

        M31 := -M13     ;
        M32 := -M23     ;
        M33 :=  0.0     ;

---     II.H. Calculate M**2

        MSQ11   := -(M12*M12 + M13*M13) ;
        MSQ12   := -M13*M23             ;
        MSQ13   :=  M12*M23             ;

        MSQ21   :=  MSQ12               ;
        MSQ22   := -(M12*M12 + M23*M23) ;
        MSQ23   := -M12*M13             ;

        MSQ31   :=  MSQ13               ;
        MSQ32   :=  MSQ23               ;
        MSQ33   := -(M13*M13 + M23*M23) ;

---     Compute the RF matrix

        RF11    := 1.0 + cosc_kappa_m * MSQ11 + sinc_kappa_m * M11  ;
        RF12    :=       cosc_kappa_m * MSQ12 + sinc_kappa_m * M12  ;
        RF13    :=       cosc_kappa_m * MSQ13 + sinc_kappa_m * M13  ;


        RF21    :=       cosc_kappa_m * MSQ21 + sinc_kappa_m * M21  ;
        RF22    := 1.0 + cosc_kappa_m * MSQ22 + sinc_kappa_m * M22  ;
        RF23    :=       cosc_kappa_m * MSQ23 + sinc_kappa_m * M23  ;


        RF31    :=       cosc_kappa_m * MSQ31 + sinc_kappa_m * M31  ;
        RF32    :=       cosc_kappa_m * MSQ32 + sinc_kappa_m * M32  ;
        RF33    := 1.0 + cosc_kappa_m * MSQ33 + sinc_kappa_m * M33  ;
```



```
---     Intermediate matrix calculations

        A11_tmp := RF11*A11 + RF12*A21 + RF13*A31  ;
        A12_tmp := RF11*A12 + RF12*A22 + RF13*A32  ;
        A13_tmp := RF11*A13 + RF12*A23 + RF13*A33  ;


        A21_tmp := RF21*A11 + RF22*A21 + RF23*A31  ;
        A22_tmp := RF21*A12 + RF22*A22 + RF23*A32  ;
        A23_tmp := RF21*A13 + RF22*A23 + RF23*A33  ;


        A31_tmp := RF31*A11 + RF32*A21 + RF33*A31  ;
        A32_tmp := RF31*A12 + RF32*A22 + RF33*A32  ;
        A33_tmp := RF31*A13 + RF32*A23 + RF33*A33  ;

---     Update the Euler matrix

        A11     := A11_TMP + DC11       ;
        A12     := A12_TMP + DC12       ;
        A13     := A13_TMP + DC13       ;

        A21     := A21_TMP + DC21       ;
        A22     := A22_TMP + DC22       ;
        A23     := A23_TMP + DC23       ;

        A31     := A31_TMP + DC31       ;
        A32     := A32_TMP + DC32       ;
        A33     := A33_TMP + DC33       ;

---     Calculate theta


        abs_A13 := A13;
        IF A13 < 0.0 then abs_A13 := -A13 ; END IF;

        IF abs_A13 < 1.0 then
                THETA := -FLOAT_MATH_LIB.ASIND(A13) ;
        END IF ;

        cosine_theta := FLOAT_MATH_LIB.COSD(THETA) ;

---     Calculate sine and cosine of phi and psi

        IF cosine_theta /= 0.0 then

                ARGC_PHI :=  A33/cosine_theta   ;
                ARGS_PHI :=  A23/cosine_theta   ;
                ARGC_PSI :=  A11/cosine_theta   ;
                ARGS_PSI :=  A12/cosine_theta   ;

        END IF ;

---
                SP              := 1.0          ;
                ABS_ARGS_PSI    := ARGS_PSI     ;
                IF ARGS_PSI < 0.0 then
                        SP              := -1.0         ;
                        ABS_ARGS_PSI    := - ARGS_PSI ;
                END IF;
---
                CP              := 1.0          ;
                IF ARGC_PSI < 0.0 then
                        CP              := -1.0         ;
                END IF;
---
```

```
                ABS_ARGS_PHI := ARGS_PHI        ;
                SF              := 1.0          ;
                IF ARGS_PHI < 0.0 then
                        SF              := -1.0         ;
                        ABS_ARGS_PHI := - ARGS_PHI ;
                END IF;
...

                CF              := 1.0          ;
                IF ARGC_PHI < 0.0 then
                        CF              := -1.0         ;
                END IF;

---     Provide reference angles phio and psio

        IF ABS_ARGS_PHI < 1.0 then
                PHIO := FLOAT_MATH_LIB.ASIND(ARGS_PHI) ;
        END IF;

        IF ABS_ARGS_PSI < 1.0 then
                PSIO := FLOAT_MATH_LIB.ASIND(ARGS_PSI) ;
        END IF;

---     Set up phi and psi for the correct quadrants

        PSI := PSIO*0.5*(1.0 + CP) + 0.5*(1.0-CP)*(SP*180.0 - PSIO) ;

        PHI := PHIO*0.5*(1.0 + CF) + 0.5*(1.0-CF)*(SF*180.0 - PHIO) ;

--- EXIT.
end nav_att;
----
----    20 Hertz timer routine
----

separate (nav_exec)

PROCEDURE timer (dt : in out float) is

        delta_t                 : DAY_DURATION  := 0.0  ;
        time_left_over          : FLOAT                 ;
        fl_delta_t              : FLOAT                 ;
        package real_io is new float_io(float);
begin
        dt                      := 0.0                  ;
        time_left_over          := 0.05                 ;
        old                     := CLOCK                ;
        fl_delta_t              := 0.0                  ;

WHILE time_left_over > 0.0  loop

        now             := CLOCK                                ;
        delta_t         := SECONDS (now)  -  SECONDS(old)       ;
        fl_delta_t      := FLOAT(delta_t)                       ;
        dt              := dt + fl_delta_t                      ;
        time_left_over  := time_left_over - fl_delta_t          ;

end loop ;

end timer ;
-- Results of navigation calculations are written to a file called results.DAT.
separate (nav_exec)
procedure WRITE_NAV_DATA(result_record : in out FINAL_DATA ;
                         RAW_DATA : in out INPUT_DATA) is
  COUNTER : integer := 1 ;
  package FLOAT_IO is new TEXT_IO.FLOAT_IO(FLOAT) ;
```



```
  package INTEGER_IO is new TEXT_IO.INTEGER_IO(INTEGER) ;
begin

  COUNTER := 1 ;
  while (COUNTER < 7) loop
    TEXT_IO.PUT("RT5_SA01_IN(");
    INTEGER_IO.PUT(RESULT_FILE, COUNTER) ;
    TEXT_IO.PUT(") := ") ;
    FLOAT_IO.PUT(RESULT_FILE, RAW_DATA.RT5_SA01_IN(COUNTER)) ;
    TEXT_IO.NEW_LINE ;
    COUNTER := COUNTER + 1 ;
  end loop ;
-- output buffers range from 1 to 31, but not all numbers are used
  COUNTER := 1 ;
  while (COUNTER < 33) loop
    TEXT_IO.PUT("RT5_SA01_OUT(");
    INTEGER_IO.PUT(RESULT_FILE, COUNTER) ;
    TEXT_IO.PUT(") := ") ;
    FLOAT_IO.PUT(RESULT_FILE, result_record.RT5_SA01_OUT(COUNTER)) ;
    TEXT_IO.NEW_LINE ;
    COUNTER := COUNTER + 1 ;
  end loop ;
  COUNTER := 1 ;
  while (COUNTER < 33) loop
    TEXT_IO.PUT("RT5_SA02_OUT(");
    INTEGER_IO.PUT(RESULT_FILE, COUNTER) ;
    TEXT_IO.PUT(") := ") ;
    FLOAT_IO.PUT(RESULT_FILE, result_record.RT5_SA02_OUT(COUNTER)) ;
    TEXT_IO.NEW_LINE ;
    COUNTER := COUNTER + 1 ;
  end loop ;
  COUNTER := 1 ;
  while (COUNTER < 33) loop
    TEXT_IO.PUT("RT5_SA03_OUT(");
    INTEGER_IO.PUT(RESULT_FILE, COUNTER) ;
    TEXT_IO.PUT(") := ") ;
    FLOAT_IO.PUT(RESULT_FILE, result_record.RT5_SA03_OUT(COUNTER)) ;
    TEXT_IO.NEW_LINE ;
    COUNTER := COUNTER + 1 ;
  end loop ;
  COUNTER := 1 ;
  while (COUNTER < 33) loop
    TEXT_IO.PUT("RT5_SA04_OUT(");
    INTEGER_IO.PUT(RESULT_FILE, COUNTER) ;
    TEXT_IO.PUT(") := ") ;
    FLOAT_IO.PUT(RESULT_FILE, result_record.RT5_SA04_OUT(COUNTER)) ;
    TEXT_IO.NEW_LINE ;
    COUNTER := COUNTER + 1 ;
  end loop ;
  COUNTER := 1 ;
  while (COUNTER < 33) loop
    TEXT_IO.PUT("RT5_SA05_OUT(");
    INTEGER_IO.PUT(RESULT_FILE, COUNTER) ;
    TEXT_IO.PUT(") := ") ;
    FLOAT_IO.PUT(RESULT_FILE, result_record.RT5_SA05_OUT(COUNTER)) ;
    TEXT_IO.NEW_LINE ;
    COUNTER := COUNTER + 1 ;
  end loop ;
  COUNTER := 1 ;
  while (COUNTER < 33) loop
    TEXT_IO.PUT("RT5_SA10_OUT(");
    INTEGER_IO.PUT(RESULT_FILE, COUNTER) ;
    TEXT_IO.PUT(") := ") ;
    FLOAT_IO.PUT(RESULT_FILE, result_record.RT5_SA10_OUT(COUNTER)) ;
    TEXT_IO.NEW_LINE ;
    COUNTER := COUNTER + 1 ;
```

```
  end loop ;
  COUNTER := 1 ;
  while (COUNTER < 33) loop
    TEXT_IO.PUT("RT5_SA11_OUT(");
    INTEGER_IO.PUT(RESULT_FILE, COUNTER) ;
    TEXT_IO.PUT(") := ") ;
    FLOAT_IO.PUT(RESULT_FILE, result_record.RT5_SA11_OUT(COUNTER)) ;
    TEXT_IO.NEW_LINE ;
    COUNTER := COUNTER + 1 ;
  end loop ;
  COUNTER := 1 ;
  while (COUNTER < 33) loop
    TEXT_IO.PUT("RT5_SA17_OUT(");
    INTEGER_IO.PUT(RESULT_FILE, COUNTER) ;
    TEXT_IO.PUT(") := ") ;
    FLOAT_IO.PUT(RESULT_FILE, result_record.RT5_SA17_OUT(COUNTER)) ;
    TEXT_IO.NEW_LINE ;
    COUNTER := COUNTER + 1 ;
  end loop ;
  COUNTER := 1 ;
  while (COUNTER < 33) loop
    TEXT_IO.PUT("RT5_SA18_OUT(");
    INTEGER_IO.PUT(RESULT_FILE, COUNTER) ;
    TEXT_IO.PUT(") := ") ;
    FLOAT_IO.PUT(RESULT_FILE, result_record.RT5_SA18_OUT(COUNTER)) ;
    TEXT_IO.NEW_LINE ;
    COUNTER := COUNTER + 1 ;
  end loop ;
  COUNTER := 1 ;
  while (COUNTER < 33) loop
    TEXT_IO.PUT("RT5_SA19_OUT(");
    INTEGER_IO.PUT(RESULT_FILE, COUNTER) ;
    TEXT_IO.PUT(") := ") ;
    FLOAT_IO.PUT(RESULT_FILE, result_record.RT5_SA19_OUT(COUNTER)) ;
    TEXT_IO.NEW_LINE ;
    COUNTER := COUNTER + 1 ;
  end loop ;
  COUNTER := 1 ;
  while (COUNTER < 33) loop
    TEXT_IO.PUT("RT5_SA29_OUT(");
    INTEGER_IO.PUT(RESULT_FILE, COUNTER) ;
    TEXT_IO.PUT(") := ") ;
    FLOAT_IO.PUT(RESULT_FILE, result_record.RT5_SA29_OUT(COUNTER)) ;
    TEXT_IO.NEW_LINE ;
    COUNTER := COUNTER + 1 ;
  end loop ;
  COUNTER := 1 ;
  while (COUNTER < 33) loop
    TEXT_IO.PUT("RT5_SA31_OUT(");
    INTEGER_IO.PUT(RESULT_FILE, COUNTER) ;
    TEXT_IO.PUT(") := ") ;
    FLOAT_IO.PUT(RESULT_FILE, result_record.RT5_SA31_OUT(COUNTER)) ;
    TEXT_IO.NEW_LINE ;
    COUNTER := COUNTER + 1 ;
  end loop ;
end WRITE_NAV_DATA
```

APPENDIX II

AATD Benchmark Spare Processing Mix

APPENDIX III

AATD Benchmark I/O Mix

NAC-ECPM-P1-IRS-0003 12 January 1991

INTERFACE REQUIREMENTS SPECIFICATION FOR THE ADVANCED AVIONICS TECHNOLOGY DEMONSTRATION EMBEDDED COMPUTER PERFORMANCE MEASUREMENT PROGRAM (AATD ECPM)

CONTRACT NO. N00163-09-C-0165
CDRL SEQUENCE NO. A002

PREPARED FOR:

Naval Avionics Center
Code 826

PREPARED BY:

Software Technology Department
Defense Systems & Electronics Group
Texas Instruments Incorporated
6550 Chase Oaks Drive
Plano, Texas 75086

Authenticated by: _____
(Naval Avionics Center)

Date: _____

Approved by: _____
(Texas Instruments Inc.)

Date: _____

Table of Contents

List of Figures

List of Tables

1. Scope

1.1. Identification

This document is the Interface Requirements Specification (IRS) for the Advanced Avionics Technology Demonstration (AATD) Embedded Computer Performance Measurement (ECPM) Software and applies to the interface designated as AATD CSCI (Computer Software Configuration Item). This document was prepared for the Naval Avionics Center (NAC) under Contract No. N00163-09-C-0165.

1.2. System Overview

The purpose of the ECPM software is to provide a highly portable, advanced performance measurement facility for future avionic systems. Motivation for the ECPM comes from realizing that existing evaluation methods do not support direct performance comparisons using reserve processor and I/O throughput. The ECPM is the first known attempt to provide this capability.

The core of the ECPM is a six Degree-of-Freedom (6-DOF) simulation that accepts data from a MIL-STD-1553B interface as input and provides latitude and longitude, as well as other navigational data, as output. The nature and complexity of the ECPM is such that it cannot be delivered as a standalone product for immediate retargeting to a new architecture. Rather, it is partitioned in a way to facilitate its movement to novel architectures and backplanes. This generality places some additional requirements on the end user to provide a mechanism that allows the modular components of the ECPM to communicate with the AATD CSCI.

Hooks have been added to the ECPM to allow it to calculate reserve processor and I/O throughput in addition to positional data. The calculation of *reserve* throughput data allows the performance of different processors to be compared. To facilitate collection of this performance data, a set of calls to an underlying Network Operating System (NOS) have been defined to allow inter-task communication. The semantics and protocol of these supporting NOS procedures is one of the principal subjects of this document.

1.3. Document Overview

This purpose of this document is to define the interfaces between the AATD CSCI and other major components of the ECPM. These components have been developed to allow transportability of the software to new operating environments. The style of this document is intended to serve as much as a tutorial on how to port the ECPM as it is to document its technical details.

This document describes the Ada specifications and supporting data structures needed to implement a set of primitive communications procedures. These procedures are needed to implement an interface between the ECPM software and a given avionic processor testbed. The first release of this demonstration software was implemented for MIL-STD-1750A targets communicating via a Pi bus backplane. Input to the ECPM is in the form of messages received over a MIL-STD-1553B interface. By convention, each message type carries a unique numerical designation that equates to the 1553B subaddress through which the message is routed. For example, message type 5 is always routed through subaddress 5. Output from the ECPM is delivered to an external device via this same 1553B interface. The low-level details associated with the reading and writing of these messages is handled by the NOS that underlies the ECPM software. To execute this program on a given processor, the user must develop a set of Ada package bodies. These bodies implement the semantics and protocol assumed by the NOS primitives that will implement the desired interprocessor communications.

This section of the IRS provides an overview of the system and of this IRS.

Section 2 of this document lists all other applicable documents.

Section 3 of this document defines the interface requirements for the AATD CSCI.

Section 4 of this document describes the quality assurance provisions and is not applicable.

Section 5 of this document describes the preparations for delivery and is not applicable.

Section 6 of this document contains general information pertaining to the requirements defined in this IRS and a list of acronyms.

Section 7 of this document contains the appendices.

This document has been produced in the format explicitly required by Data Item Description (DID) DI-MCCR-80026A.

1.4. Conventions

References to the "ICD" shall be interpreted as referring to the AATD ECPM MIL-STD-1553 Interface Control Document, Reference [1] of Section 2.1.

References to the "SRS" shall be interpreted as referring to the Software Requirements Specification for the AATD CSCI, Reference [1] of Section 2.2.

References to the "IRS" shall be interpreted as referring to the Interface Requirements Specification for the AATD ECPM, which is this document.

2. Applicable Documents

2.1. Government Documents

The following documents of the exact issue shown form a part of this specification to the extent specified herein. In the event of conflict between the documents referenced herein and the contents of this specification, the contents of this specification shall be considered a superseding requirement.

1. *Advanced Avionics Technology Demonstration (AATD) Program, Embedded Computer Performance Measurement (ECPM) MIL-STD-1553 Interface Definition, Version 3.6*; 10 October 1990, Naval Avionics Center, Branch 826.

2. *Ada Language Reference Manual*, Department of Defense, ANSI/MIL-STD-1815A-1983; February 17, 1983.

Copies of specifications, standards, drawings, and publications required by suppliers in connection with specified procurement functions should be obtained from the contracting agency or as directed by the contracting officer.

2.2. Non-Government Documents

The following document of the exact issue shown forms a part of this specification to the extent specified herein. In the event of conflict between the document referenced herein and the contents of this specification, the contents of this specification shall be considered a superseding requirement.

> *Software Requirements Specification for the Advanced Avionics Technology Demonstration (AATD) CSCI of AATD System*; 9/19/90, Software Technology Department (STD), Defense Systems and Electronics Group (DSEG), Texas Instruments Incorporated. *This document can be obtained by contacting Software Configuration Management, Texas Instruments Incorporated, P.O. Box 869305, MS 8435, Plano, Texas, 75086.*

3. Interface Specification

The following paragraphs describe the three principal interfaces to the AATD CSCI. Recall that the main objective of the ECPM is to measure *reserve* processor and I/O throughput, although raw throughput and I/O bandwidth can be deduced as well. The former is accomplished by calculating the number of iterations of a contrived (whetstone variant) instruction mix that can be executed concurrently, along with normal 6-DOF simulation processing, in a specified period of time. The latter is accomplished by calculating the number of iterations of the Input/Output Built-In-Test Interface Description Specification (IOBIDS) message mix that can be executed in the same period. The state diagram describing the overall flow of control in this process is shown in Figure 1.

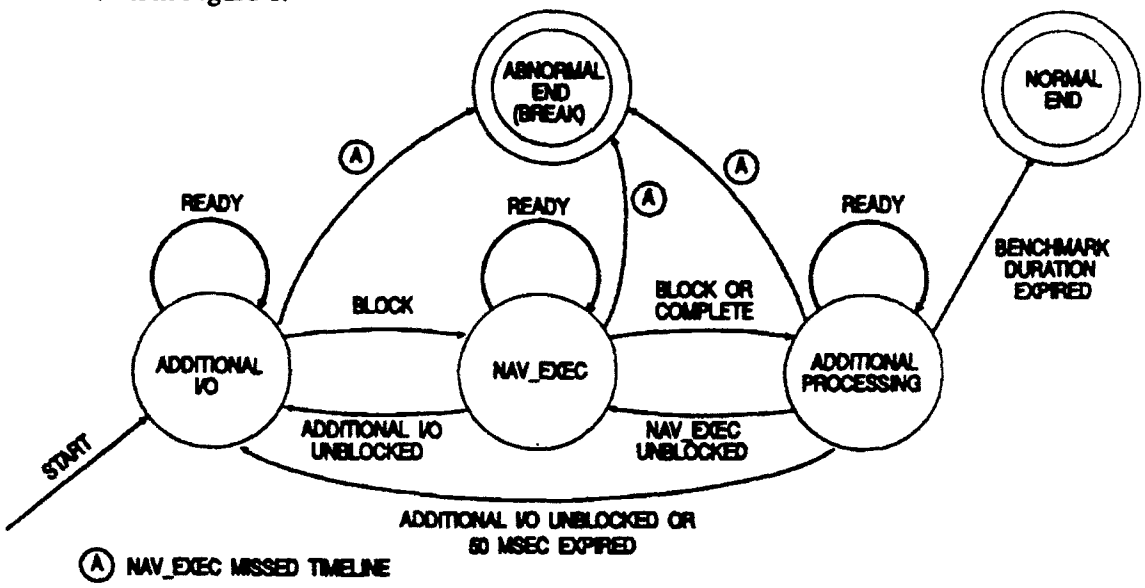


Figure 1. AATD ECPM Execution Profile.

To summarize the flow of the benchmark depicted in Figure 1, the start of a benchmark event is triggered by receipt of a start benchmark message from the Master Computer (MC). During execution of the benchmark, navigational data messages continue to arrive from the Master Computer at the rate of one message every 50 milliseconds or 20 Hz. For a single 20 Hz period, the ECPM will begin by iterating on the additional I/O message mix. The number of executions of the I/O mix per 50 millisecond period is specified to the ECPM program via a command word (reference word 26 in Table III of the ICD). For each iteration of this mix, a counter is incremented by one. When the task controlling the additional I/O mix becomes blocked, control will be transferred to the main portion of the benchmark responsible for the 6-DOF simulation. The simulation processes the navigational data and generates solutions in the form of positional data.

When the additional I/O task becomes unblocked, it again receives control. If the simulation task blocks first, control will be sent to the additional processing task (this is the instruction mix based on the whetstone variant.) Depending on which of the two currently blocked tasks is first to become unblocked, control will then be switched to either the additional I/O task or the simulation task. The interaction between these three tasks continues until either the 20 Hz period expires, the overall benchmark event time has expired, or the simulation task *misses* its timeline. A missed timeline indicates that too much overhead I/O or processing (or both) is taking place concurrently with the simulation task. Consequently, the navigational solutions can no longer be delivered to the Master Computer at the required rate. When this happens, the benchmark essentially "breaks".

By driving the benchmark toward this breaking point, it is possible to quantify the reserve I/O and processing capabilities of the unit under test. In the most general terms, this is the objective of the ECPM. By obtaining hard measurements of this type, the Navy can avoid the risk associated with selecting a computer that does not meet the performance requirements for a given program.

3.1. Interface Diagrams

The remainder of this document discusses procedures used in connection with the sending of messages between the AATD CSCI, its three principal interfaces, the Pi bus backplane, and the MIL-STD-1553B bus. The format and contents of the various messages are described in detail in Reference [1] of Section 2.1.

Figure 2 shows the AATD CSCI context diagram and the external interfaces to this CSCI. The following subparagraphs describe each external interface associated with the AATD CSCI.

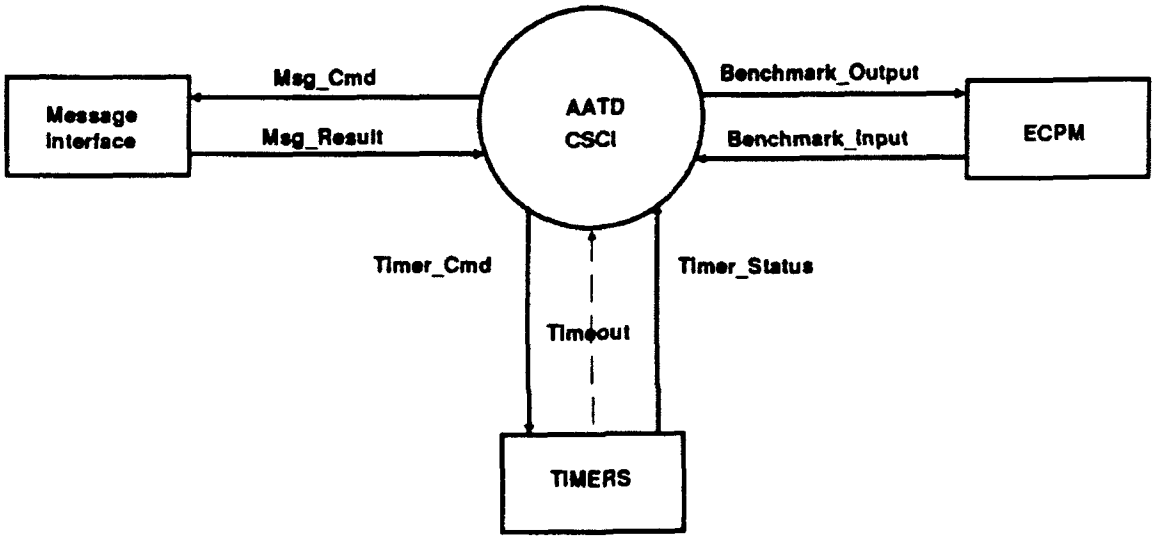


Figure 2. AATD CSCI.

3.2. AATD to ECPM Interface - AATD.IRS.ECPM

The purpose of the interface between the ECPM and the AATD CSCI is to provide operator control and ECPM control of execution of the AATD CSCI and to provide a communications path for returning benchmark results to the ECPM. This interface contains the Benchmark_Input and Benchmark_Output data flows shown in Figure 2. The associated data elements for this interface are documented in the AATD Software Requirements Specification (SRS) (Reference [1], Section 2.2).

3.2.1. Interface Requirements

a. CSCI Synchronization - The AATD CSCI and the ECPM will execute concurrently. The ECPM transmits sensor data to the AATD CSCI and receives navigation solutions and performance information from the AATD CSCI at a rate of 20 Hz.

b. Communication Protocol - The ECPM will send a message to the AATD CSCI directing it to perform one of three possible operations: NAV_Only, Record_Results, or Measure_Max_IO. The format of each message is as described in the *AATD Program ECPM MIL-STD-1553 Interface Definition* referenced in Section 2.1 of this IRS.

The purpose of each of the three functional modes is as follows:

1. *NAV_Only* - Executes only the navigation (6-DOF simulation) portion of the benchmark. This is the code responsible for reading navigation data messages from the Master Computer at the 20 Hz rate and computing navigational solutions.

2. *Record_Results* - Executes the additional processing instruction mix (Digital Avionic Systems Laboratory or *DASL* mix) in a standalone mode to calculate the total time attributable to spare processing. This function is performed after the benchmark has executed for some programmed duration. During execution of the benchmark, a counter is incremented to reflect the number of times the instruction mix was executed in the presence of navigational processing and additional I/O. At completion of the benchmark, the additional processing mix is executed by itself, for the number of iterations just computed, to calculate the total reserve processing time.

3. *Measure_Max_IO* - Executes the additional I/O (IOBIDS) message mix in standalone mode to calculate the maximum possible reserve I/O time.

c. Priority Level - The ECPM will execute independently of the AATD CSCI. No priorities are associated with this interface.

3.2.2. Data Requirements

The data elements for the ECPM Interface are described in Reference [1] of Section 2.1.

3.3. AATD to Message Interface - AATD.IRS.MSG

This interface contains the Msg_Cmd and Msg_Result data flows shown in Figure 2. The purpose of the interface between the AATD CSCI and the Message Interface is to provide 1553B and backplane bus communications capability between the ECPM application program and external hardware or instrumentation (such as a VAX). The implementation of the messaging interface is target dependent. The configuration of the system under test is assumed to contain a MIL-STD-1553B Bus Interface Module (BIM) which connects, along with the desired processor module of interest, to a common backplane. For example, the first prototype of the NAC AATD software was implemented for a MIL-STD-1750A processor module and 1553B bus interface module connected to a Pi bus backplane. These communications capabilities were provided by the Texas Instruments (TI) *Network Operating System (NOS)*. The NOS provides intertask and intermodule communications capabilities based on a message passing paradigm. Only a subset of the TI NOS capabilities were required to implement the ECPM. Equivalent functionality must be implemented by end users of the system to utilize the program with different architectures. The requirements for the basic set of primitives needed by the ECPM are described in the following paragraphs. In general terms, however, the end user must supply a simple message passing scheme, a 1553B bus driver, and a driver that supports the common backplane (e.g., VMEbus, Pi bus, etc.)

In the current version of the ECPM, there are nine message types supported. Each of these messages is described in detail in Reference [1] of Section 2.1.

3.3.1. Interface Requirements

a. CSCI Synchronization - The Message Interface will execute synchronously in response to calls from the AATD CSCI. Ada procedures described in the previous paragraphs implement the required messaging semantics and comprehend the protocol of the particular backplane bus and MIL-STD-1553B.

b. Communication Protocol - The AATD CSCI communicates with the Message Interface using the Ada procedures described in the following paragraphs.

c. Priority Level - There is no priority associated with the Message Interface.

The nine procedures that implement the messaging interface, referred to hereafter as the IO_SERVICES package, are as follows:

- INITIALIZE_1553_COMMUNICATION
- INITIALIZE_NAV_IO
- WAIT_FOR_BENCHMARK_COMMAND
- GET_NAV_DATA
- WRITE_NAV_DATA
- WRITE_BENCHMARK_RESULTS
- BUILD_MESSAGE_GROUP
- SEND_MESSAGE_GROUP
- INITIALIZE_ADDITIONAL_IO

The functional behavior of each of these procedures is described in the following paragraphs. Users of the ECPM must keep in mind that there is an entire message passing paradigm and a set of bus control functions that allow the components of the ECPM to communicate. The functional software that implements the message passing and bus control is *not* delivered with the ECPM. Software to support each functional area must be written uniquely for each processor architecture to be measured. Fortunately, the code has been packaged in a way that facilitates rapid design of these supporting components.

The ECPM consists of five Ada tasks:

1. *Timer Task* - Performs all timing measurements associated with the ECPM.

2. *Additional I/O Task* - Generates additional message traffic to be used to quantify the reserve I/O capacity of the processor under test. The current implementation of this task is based on executing the message mix, defined for use in the IOBIDS, for some number of times given as input.

3. *AATD Control Task* - Handles processing of control messages that govern the operation of the AATD CSCI.

4. *NAV_EXEC Task* - This is the root component of the ECPM that implements a 6-DOF simulation.

5. *Additional Processing Task* - Generates additional processing overhead by iterating on a variation of the synthetic whetstone benchmark. This modified instruction mix is referred to as the *DASL* mix.

The *Timer Task* is the highest priority (most urgent) task and the *Additional Processing Task* is the lowest priority (least urgent) task. Refer to the Software Requirements Specification for a complete description of the various states and control modes associated with each of these tasks.

The following paragraphs describe the parameters and functional behavior needed for each of the procedures required by the Message Interface. The exact format of messages referred to in the following sections are shown in section 3.9

3.3.1.1. INITIALIZE_1553_COMMUNICATION Procedure

The INITIALIZE_1553_COMMUNICATION procedure initializes the interface to the MIL-STD-1553B and establishes an access mechanism to the buffers through which messages will be passed between the AATD CSCI, external devices connected to the 1553 BIM, and any other devices connected via the common backplane. INITIALIZE_1553_COMMUNICATION must be implemented to do the following:

1. Establish a connection to the 1553B BIM that will send and receive messages to/from an external device (for example, a VAX host serving as the Master Computer). The connection process will include any activities required to verify that the BIM is operational, that it is configured for use with the specific backplane and external device communications characteristics, and any other one-time hardware and software setup activities that must precede execution of the ECPM. This routine will be called exactly once following system power-up.

2. Establish a buffer or *pool* of buffers through which messages will be passed. Depending on the actual configuration of the text fixture, the BIM and target processor may be implemented on separate modules. In this case, a buffer must be established on both the BIM and the target processor board. INITIALIZE_1553_COMMUNICATION arranges for the allocation of a buffer at each end of the communications path. The addresses of these buffers will be stored in variables that are local to the IO_SERVICES package and known to the underlying implementation.

Note that one possible approach to implementing the message buffers, which is the one used by the 1750A implementation of the ECPM, is to use the notion of a message *label*. Using this approach, an object called a label is associated with some arbitrary number of buffers. The underlying routine that manages this object arranges for a message to be allocated to the next available buffer. The caller of the routine that reads or writes a message is freed from the responsibility for managing individual buffers and instead simply routes messages to/from a particular label. The code implementing the label object (actually just a queue of message buffers) provides the functionality needed to manage the individual buffers. This approach works well for the ECPM which associates a unique message type with a unique 1553B subaddress.

The INITIALIZE_1553_COMMUNICATION procedure is part of the IO_SERVICES package. This procedure has no parameters.

3.3.1.2. INITIALIZE_NAV_IO Procedure

The INITIALIZE_NAV_IO procedure is used in connection with the NAV_EXEC portion of the ECPM. This procedure is responsible for allocating buffers to receive Type 5 messages on 1553B subaddress 5 (SA5). Type 5 messages contain navigational data such as acceleration, altitude, etc. Recall from section 1.3 that there is a one-to-one correspondence between 1553B subaddresses and message types, i.e., Type 5 messages are always sent via SA5, Type 2 messages via SA2, etc.

The INITIALIZE_NAV_IO procedure is part of the IO_SERVICES package. This procedure has no parameters.

3.3.1.3. WAIT_FOR_BENCHMARK_COMMAND Procedure

The ECPM Interface Definition (Reference [1] Section 2.1) describes a set of messages, sent by the Master Computer, used to control the behavior of the ECPM. The WAIT_FOR_BENCHMARK_COMMAND procedure simply posts a read (with wait) to the appropriate message buffer and returns the next benchmark command message. Note that all benchmark command messages are delivered via 1553B SA7. This procedure basically suspends the encompassing task until a benchmark command is received.

The WAIT_FOR_BENCHMARK_COMMAND is part of the IO_SERVICES package. The Ada specification for this procedure is as follows:

```
procedure WAIT_FOR_BENCHMARK_COMMAND (
   BENCHMARK_COMMAND_ACCESS : out
      MESSAGE_TYPES.BENCHMARK_COMMAND_ACCESS_TYPE );
```

The variable BENCHMARK_COMMAND_ACCESS returns a pointer to a record (buffer) containing a Type 7 message.

3.3.1.4. GET_NAV_DATA Procedure

The GET_NAV_DATA procedure returns an Ada access value to the caller that points to the next logical data message received from the MC. Navigation data messages are delivered via 1553B SA5.

This procedure suspends the encompassing task until a Type 5 message becomes available. Type 5 messages supply navigational information such as acceleration and rate data and are described in Reference [1] of Section 2.1.

The GET_NAV_DATA procedure is part of the IO_SERVICES package. The Ada specification for this procedure is as follows:

```
procedure GET_NAV_DATA (
   RAW_DATA_ACCESS : out
      MESSAGE_TYPES.RAW_DATA_ACCESS_TYPE );
```

The variable RAW_DATA_ACCESS returns a pointer to a record (buffer) containing a Type 5 message.

3.3.1.5. WRITE_NAV_DATA Procedure

As navigation solutions are computed by the ECPM, the results are transmitted back to the MC via the 1553B bus using WRITE_NAV_DATA. These results could consist of any message type other than Type 5, 6, or

7. The caller of this procedure will supply the address of the message to be sent, the length of the message in bytes, and the 1553B subaddress to which the message will be sent. The actual implementation of the write will vary between testbeds. In the 1750A test scenario, for example, the write is implemented as a Direct Memory Access (DMA) transfer to the BIM. The transferred message is then routed from the BIM to external instrumentation via the 1553B.

The WRITE_NAV_DATA procedure is part of the IO_SERVICES package. The Ada specification for this procedure is as follows:

```
procedure WRITE_NAV_DATA (
   RESULTS_ADDRESS : in SYSTEM.address ;
   BYTE_COUNT      : in positive ;
   SUBADDRESS      : in SUBADDRESS_TYPE ) ;
```

The variable RESULTS_ADDRESS is the address of the message to be transferred. The message contained at this address will be something other than Type 5, 6, or 7. The actual memory address of the message is obtained with Ada's ADDRESS representation attribute (as in X'ADDRESS where X is defined as any object, program unit, label, or entry). If X is defined as the data structure containing the message to be transferred, the RESULTS_ADDRESS parameter can be passed to this procedure by coding the parameter as:

```
RESULTS_ADDRESS => X'ADDRESS
```

The ADDRESS representation attribute always returns a value of the type ADDRESS defined in the package SYSTEM (refer to paragraph 13.7.2 in Reference [2] of Section 2.1 for more information on Ada attributes.) This technique of assigning an object's address to a variable can be used, in general, for any parameter declared as type SYSTEM.ADDRESS.

The variable BYTE_COUNT is the number of bytes in the message to be transferred. For example, Type 1 messages are 6 bytes long, Type 2 messages are 28 bytes long, etc. Consult Reference [1] of Section 2.1 for a complete description of all message types. Byte counts for each message type are defined as constants in package MESSAGE_TYPES. For example, the BYTE_COUNT parameter for a type 7 message could be coded as follows:

```
BYTE_COUNT => MESSAGE_TYPES.MESSAGE_7_BYTE_COUNT
```

The variable SUBADDRESS is the 1553B subaddress to which the message will be transferred. Note that there is a one-to-one correspondence between 1553B subaddresses and message types, i.e., Type 1 messages are always sent via SA1, Type 2 messages via SA2, etc.

3.3.1.6. WRITE_BENCHMARK_RESULTS Procedure

The WRITE_BENCHMARK_RESULTS procedure operates similar to WRITE_NAV_DATA, but only writes messages of Type 7. These messages carry the results of the benchmark and include measurements of spare processor and I/O throughput. The procedure uses low-level, NOS specific primitives to transfer benchmark results across the common backplane to the 1553B BIM.

The WRITE_BENCHMARK_RESULTS procedure is part of the IO_SERVICES package. The Ada specification for this procedure is as follows:

```
procedure WRITE_BENCHMARK_RESULTS (
   BENCHMARK_RESULTS_ADDRESS : in SYSTEM.address ;
   BYTE_COUNT                : in positive ) ;
```

The variable BENCHMARK_RESULTS_ADDRESS is the address of the Type 7 message to be transferred.

The variable BYTE_COUNT is the number of bytes in the message to be transferred. This byte count will always be six for Type 7 messages.

3.3.1.7. BUILD_MESSAGE_GROUP Procedure

The BUILD_MESSAGE_GROUP procedure is part of the IO_SERVICES package and builds the Communications Control Block (CCB) chain containing the messages in the IOBIDS message mix. The exact contents of the IOBIDS message mix, defined in Ada package AATD_DATA, is known internally to the BUILD_MESSAGE_GROUP procedure. At runtime, the data structure corresponding to the IOBIDS message mix is loaded by BUILD_MESSAGE_GROUP, via assignments to each individual record field or aggregate, using data supplied by the AATD_DATA package. If the message mix were to be redefined at a later date, the contents of AATD_DATA would have to be updated to reflect this.

The Ada specification for BUILD_MESSAGE_GROUP is as follows:

```
procedure BUILD_MESSAGE_GROUP (
   MESSAGE_GROUP_ID : in MESSAGE_GROUP_ID_TYPE ) ;
```

The variable MESSAGE_GROUP_ID is an integer identifier associated with the message mix. For Phase One of the AATD ECPM program, there is only one message mix and the value assigned to this identifier is superfluous. This parameter facilitates future changes to the message mix.

3.3.1.8. SEND_MESSAGE_GROUP Procedure

The SEND_MESSAGE_GROUP procedure is part of the IO_SERVICES package and transmits the message chains built by BUILD_MESSAGE_GROUP. The Ada specification for this procedure is as follows:

```
procedure SEND_MESSAGE_GROUP (
   MESSAGE_GROUP_ID : in MESSAGE_GROUP_ID_TYPE ) ;
```

The variable MESSAGE_GROUP_ID is an integer identifier associated with the message mix. For Phase One of the AATD ECPM program, there is only one message mix and the value assigned to this identifier is superfluous. This parameter facilitates future changes to the message mix.

3.3.1.9. INITIALIZE_ADDITIONAL_IO Procedure

The INITIALIZE_ADDITIONAL_IO procedure is part of the IO_SERVICES package and establishes communication with the NOS. Part of this process includes identifying the additional I/O task to the underlying NOS communications procedures and reserving buffers through which messages will be passed. The actual mechanics of "connecting" to a NOS will be machine dependent.

The INITIALIZE_ADDITIONAL_IO procedure has no parameters.

3.3.1.10. DISABLE_NAV_IO Procedure

The DISABLE_NAV_IO procedure is part of the IO_SERVICES package and is used to prevent the NAV_EXEC task from receiving navigational data from the MC. The MC continuously sends input data to the UUT, even when the UUT is not processing the data and producing output data. Since a finite number of buffers are reserved for input data, the result is that overflow can occur. The DISABLE_NAV_IO procedure allows the NAV_EXEC task to indicate that it no longer wishes to receive navigational data. The body of this procedure is target-specific. For TI's MIL-STD-1750A implementation, the procedure consists of sending a message to the MIL-STD-1553B BIM indicating that data received via subaddress 5 (SA5) should no longer be routed to the MIL-STD-1750A module.

The DISABLE_NAV_IO procedure has no parameters.

3.3.1.11. ENABLE_NAV_IO Procedure

The ENABLE_NAV_IO procedure is part of the IO_SERVICES package and enables the NAV_EXEC task to receive navigational data from the MC. The body of this procedure is target-specific. For TI's MIL-STD-1750A implementation, the procedure consists of sending a message to the MIL-STD-1553B BIM indicating that data received via subaddress 5 (SA5) should be routed to the MIL-STD-1750A module.

The ENABLE_NAV_IO procedure has no parameters.

3.3.2. Data Requirements

Table 1 defines the data elements for the Message Interface:

IDENTIFIER	DESCRIPTION	SRC	DEST	DATA TYPE	UNITS	RANGE
BENCHMARK_COMMAND_ACCESS	Pointer to Type 7 message buffer.	NOS	ECPM	ACCESS to BENCHMARK_COMMAND_TYPE.	N/A	0..65535
RAW_DATA_ACCESS	Pointer to Type 5 message buffer	ECPM	NOS	ACCESS to RAW_DATA_TYPE.	N/A	0..65535
RESULTS_ADDRESS	Message to be transferred (other than types 5, 6, and 7.)	ECPM	NOS	SYSTEM.ADDRESS	N/A	0..65535
BYTE_COUNT	Number of bytes to transfer.	ECPM	NOS	POSITIVE	Bytes	1..65535
SUBADDRESS	1553 subaddress to which message is transferred.	NOS	ECPM	SUBADDRESS_TYPE	N/A	1..7, 10, 15
BENCHMARK_RESULTS_ADDRESS	Address of Type 7 message to transfer.	ECPM	NOS	SYSTEM.ADDRESS	N/A	0..65535

Table 1. AATD.IRS.MSG Data Elements.

3.4. AATD to TIMERS Interface - AATD.IRS.TIM

3.4.1. Interface Requirements

a. CSCI Synchronization - The Timer Interface is called synchronously from the AATD CSCI.

b. Communication Protocol - Communication with the Timer Interface is achieved with Ada procedure calls documented in the following paragraphs.

c. Priority Level - There is no priority level associated with the Timer Interface.

The AATD.IRS.TIM interface contains the Timer_Cmd, Timer_Status, and Timeout data flows shown in Figure 2. The purpose of the interface between the TIMERS and the AATD CSCI is to provide a mechanism for retrieving the time-of-day and for forcing delays. The Phase One implementation of the ECPM uses Ada package CALENDAR to implement this functionality. However, not all implementations of package CALENDAR provide the same degree of timer granularity. If the particular runtime implementation of package CALENDAR being used to implement a new version of the ECPM does not provide the desired resolution, the user may need to implement bodies in assembly language (or using package MACHINE_CODE) to achieve equivalent timing capabilities.

The following paragraphs describe the parameters and functional behavior required for each of the procedures required by the TIMERS interface.

3.4.1.1. INITIALIZE_TIMERS Procedure

The INITIALIZE_TIMERS procedure will be called once following power-up. The purpose of this routine is to provide any setup or initial configuration required for the particular hardware timer to be used. For example, many timer devices, such as the Intel 8254, have several functional modes. An appropriate mode must be selected that implements the desired timer functionality. When taking this approach to implementing a timer, care must be taken to insure that reprogramming of the timer will not conflict with assumptions made by the Ada runtime to implement its tasking semantics.

The INITIALIZE_TIMERS procedure is part of the TIMER package. This procedure has no parameters.

3.4.1.2. CLOCK Function

The CLOCK function, defined in Ada package CALENDAR, returns a single value of type CALENDAR.DAY_DURATION. The Ada specification for the CLOCK function is as follows:

```
function CLOCK
  return CALENDAR.DAY_DURATION;
```

The function CLOCK is defined in package TIMER.

An alternate approach to implementing this capability without package CALENDAR has been used successfully in other benchmark applications. This method requires the use of assembly language or package MACHINE_CODE to implement START, STOP, and READ functions for an available timer. Use of these auxiliary functions takes advantage of knowing the maximum period of the timer and assumes that an interrupt can be triggered each time the maximum period of the timer is reached. The START procedure essentially zeroes the timer and associates an interrupt with the timer event. The clock begins to increment or decrement with the first call to START. When a timer interrupt occurs signalling that the maximum period of the timer has been reached, control is transferred to an interrupt handler that simply increments a global variable by one and returns control to the interrupted program. At the end of the event to be measured, the STOP function is called and then a READ is issued to retrieve the instantaneous value of the timer. This value is then added to the product of the maximum clock period and the value in the global variable. This calculation will provide the number of clock ticks in the event just measured. Multiplying this value by the time for one clock cycle

gives the total event time. This approach is a bit more cumbersome, but is a reasonable alternative to package CALENDAR functions with poor resolution.

3.4.1.3. WAIT Procedure

The WAIT procedure implements an Ada **delay** statement, suspending the calling process for some specified number of seconds. The Ada specification for procedure WAIT is as follows:

```
procedure WAIT
   ( SECONDS : in CALENDAR.DAY_DURATION ) ;
```

3.4.2. Data Requirements

The following data items are called out for the Timer Interface in the Software Requirements Specification for the AATD CSCI:

IDENTIFIER	DESCRIPTION	SRC	DEST	DATA TYPE	UNITS	RANGE
CALENDAR.DAY_DURATION	Current time.	ECPM	MC	DAY_DURATION	Seconds	0.0 .. 86400.0

Table 2. AATD.IRS.TIM Data Elements.

3.5. Machine Dependent Types Package

The following paragraphs describe additional machine dependent considerations for porting the ECPM to new backplanes and processor architectures.

Ada package MACHINE_DEPENDENT_TYPES contains declarations for data types which are dependent on the specific target processor to be evaluated. Refer to Appendix A for a sample definition of this package for the MIL-STD-1750A target.

3.5.1. WORD_SIZE Constant

The constant WORD_SIZE defines the number of bits in a *word* as defined for the target processor. For example, WORD_SIZE would be 16 for the MIL-STD-1750A and 32 for processors like the Intel 80386, MIPSCo R3000, and the Motorola 68020.

3.5.2. PACKED_INTEGER_TYPE Type

The Ada type PACKED_INTEGER_TYPE is an integer subtype that defines the length of data transferred between the Master Computer and the target processor under test.

3.5.3. NORMAL_PRIORITY Constant

The constant NORMAL_PRIORITY defines the priority of the main AATD CSCI task and the priority of the navigation benchmark itself. Recall from Section 3.3.1 that the Timer Task is the *most* urgent task

(NORMAL_PRIORITY+2) and the Additional Processing Task is the *least* urgent task (NORMAL_PRIORITY-1). The Additional I/O Task is assigned a priority of NORMAL_PRIORITY+1.

3.6. MACHINE_DEPENDENT_PROCEDURES Package

Two procedures are defined in Ada package MACHINE_DEPENDENT_PROCEDURES that allow the ECPM to obtain or release exclusive control of the target processor. Mutual exclusion is required in the ECPM when calculating maximum reserve processing throughput. The precise mechanism for gaining mutual exclusion will vary from one testbed to the next. For the MIL-STD-1750A implementation developed in Phase One, mutual exclusion is achieved by locking the DP module's memory bus and disabling interrupts. For other targets, instructions may be available to disable and enable interrupts explicitly.

3.6.1. RETAIN_EXCLUSIVE_CPU_CONTROL Procedure

A call to this parameterless procedure ensures that the caller will maintain exclusive control of the Central Processing Unit (CPU) until the RELEASE_EXCLUSIVE_CPU_CONTROL procedure is called.

3.6.2. RELEASE_EXCLUSIVE_CPU_CONTROL Procedure

A call to this parameterless procedure allows the calling procedure to be pre-empted by other tasks.

3.7. NUMERIC_CONVERSION_PROCEDURES Package

The Phase One implementation of the ECPM uses packing and unpacking procedures to move data in and out of the various message type fields. The packing is done to facilitate the transmission of 32-bit floating point values to the Master Computer via the 1553B without loss of accuracy. In contrast, unpacking procedures are used to convert packed integers received from the Master Computer to machine dependent floating point values.

3.7.1. PACK Procedure

The PACK procedure is defined in Ada package NUMERIC_CONVERSION_PROCEDURES and has the following specification:

```
procedure PACK (
   LOCAL_REAL      : in float ;
   PACKED_VALUE    : out MACHINE_TYPES.PACKED_INTEGER_TYPE ;
   SCALING_FACTOR  : in float ;
   OFFSET          : in float ) ;
```

The variable LOCAL_REAL is the machine dependent floating-point value to be packed.

The variable PACKED_VALUE is the packed integer equivalent of the LOCAL_REAL input value. The LOCAL_REAL value is packed using the scheme described in Reference [1] of Section 2.1.

The variable SCALING_FACTOR is the machine dependent floating-point value equivalent to the *resolution* variable referred to in section B of Reference [1] in Section 2.1. The scaling factor is used to pack a range of floating point numbers into a 16-bit field.

The variable OFFSET is the machine dependent floating-point value used to shift a floating point input value with a *nonsymmetric* value range into a *symmetric* value range. Refer to Reference [1] in Section 2.1 for descriptions of the packing and unpacking algorithms.

3.7.2. UNPACK Procedure

The UNPACK procedure is defined in Ada package NUMERIC_CONVERSION_PROCEDURES and has the following specification:

```
procedure UNPACK (
   LOCAL_REAL      : out float ;
   PACKED_VALUE    : in MACHINE_TYPES.PACKED_INTEGER_TYPE ;
   SCALING_FACTOR  : in float ;
   OFFSET          : in float ) ;
```

The variable LOCAL_REAL is the machine dependent floating-point number generated from the PACKED_VALUE input parameter.

The variable PACKED_VALUE is the packed integer value to be converted to a machine dependent floating point and returned via the LOCAL_REAL parameter. The LOCAL_REAL value is unpacked using the scheme described in Reference [1] of Section 2.1.

The variable SCALING_FACTOR is the floating-point value equivalent to the *resolution* variable referred to in section B of Reference [1] in Section 2.1. The scaling factor is used to pack and unpack a range of floating point numbers to/from a 16-bit field.

The variable OFFSET is the floating-point value used to shift a floating point input value with a *nonsymmetric* value range into a *symmetric* value range. Refer to Reference [1] in Section 2.1 for descriptions of the packing and unpacking algorithms.

3.7.3. PACK_DOUBLE Procedure

The PACK_DOUBLE procedure is defined in Ada package NUMERIC_CONVERSION_PROCEDURES and has the following specification:

```
procedure PACK_DOUBLE (
   LOCAL_REAL       : in float ;
   PACKED_VALUE_MS  : out MACHINE_TYPES.PACKED_INTEGER_TYPE ;
   PACKED_VALUE_LS  : out MACHINE_TYPES.PACKED_INTEGER_TYPE ;
   SCALING_FACTOR   : in float ;
   OFFSET           : in float ) ;
```

The variable LOCAL_REAL is the floating-point value to be packed.

The variable PACKED_VALUE_MS will be the most significant 16-bits of the packed integer equivalent of the LOCAL_REAL input value. The LOCAL_REAL value is packed using the scheme described in Reference [1] of Section 2.1.

The variable PACKED_VALUE_LS will be the least significant 16-bits of the packed integer equivalent of the LOCAL_REAL input value. The LOCAL_REAL value is packed using the scheme described in Reference [1] of Section 2.1.

The variable SCALING_FACTOR is the floating-point value equivalent to the *resolution* variable referred to in section B of Reference [1] in Section 2.1. The scaling factor is used to pack a range of floating point numbers into a 16-bit field.

The variable OFFSET is the floating-point value used to shift a floating point input value with a *nonsymmetric* value range into a *symmetric* value range. Refer to Reference [1] in Section 2.1 for descriptions of the packing and unpacking algorithms.

3.7.4. UNPACK_DOUBLE Procedure

The UNPACK_DOUBLE procedure is defined in Ada package NUMERIC_CONVERSION_PROCEDURES and has the following specification:

```
procedure UNPACK_DOUBLE (
   LOCAL_REAL       : out float ;
   PACKED_VALUE_MS  : in  MACHINE_TYPES.PACKED_INTEGER_TYPE ;
   PACKED_VALUE_LS  : in  MACHINE_TYPES.PACKED_INTEGER_TYPE ;
   SCALING_FACTOR   : in float ;
   OFFSET           : in float ) ;
```

The variable LOCAL_REAL is the machine dependent floating-point value to be unpacked.

The variable PACKED_VALUE_MS will be the most significant 16 bits of the packed integer equivalent of the LOCAL_REAL output value. The LOCAL_REAL value is unpacked using the scheme described in Reference [1] of Section 2.1.

The variable PACKED_VALUE_LS will be the least significant 16 bits of the packed integer equivalent of the LOCAL_REAL output value. The LOCAL_REAL value is unpacked using the scheme described in Reference [1] of Section 2.1.

The variable SCALING_FACTOR is the machine dependent floating-point value equivalent to the *resolution* variable referred to in section B of Reference [1] in Section 2.1. The scaling factor is used to pack a range of floating point numbers into a 16-bit field.

The variable OFFSET is the machine dependent floating-point value used to shift a floating point input value with a *nonsymmetric* value range into a *symmetric* value range. Refer to Reference [1] in Section 2.1 for descriptions of the packing and unpacking algorithms.

3.7.5. ROUND_TO_NEAREST_INTEGER Function

During the creation of fields in certain message types, it is necessary for the ECPM to convert floating point values to the packed integer format described in Reference [1] of Section 2.1. The Ada language does not specify the precise manner in which this conversion must take place and different compilers will not handle truncation and rounding uniformly. To provide consistent handling irrespective of compiler implementation, the ECPM provides its own internal functions to perform rounding and truncation.

The ROUND_TO_NEAREST_INTEGER function returns a packed integer that is the value of its floating point argument (VALUE) rounded to the nearest whole number. If VALUE is exactly halfway between two whole numbers, the result is the number with the greatest absolute magnitude. The Ada specification for ROUND_TO_NEAREST_INTEGER is as follows:

```
function ROUND_TO_NEAREST_INTEGER
   ( VALUE : in float ) return
     MACHINE_DEPENDENT_TYPES.PACKED_INTEGER_TYPE ;
```

3.7.6. TRUNCATE_TO_0 Function

This function returns a packed integer that is the value of its floating point argument truncated toward zero. The Ada specification for TRUNCATE_TO_0 is as follows:

```
function TRUNCATE_TO_0
   ( VALUE : in float ) return
     MACHINE_DEPENDENT_TYPES.PACKED_INTEGER_TYPE ;
```

3.8. Compiler Pragma Considerations

Use of compiler pragmas for implementations of the ECPM should be avoided as much as possible. The 1750A implementation of the ECPM for TI's MDP uses the PACK and PRIORITY pragmas. The pragma PACK is the only language-defined representation pragma.

Consistent with guidelines published in the Ada Language Reference Manual (LRM), pragma PRIORITY is used in the ECPM only to indicate relative degrees of urgency and not for task synchronization.

3.9. Representation Specifications for Message Types

This paragraph describes the Ada representation specifications for the nine message types used by the ECPM. These specifications must be tailored for each target processor to accommodate differences in addressing modes (byte vs. word) and bit ordering (little endian vs. big endian). For example, the MIL-STD-1750A places bit 0 on the left (big endian) and the VAX places bit 0 on the right (little endian). The record descriptions for the messages are shown here for demonstration purposes and correlate to the definitions in Reference [1] of Section 2.1.

3.9.1. Message Type 1

```
for MESSAGE_1_TYPE use
   record
      PSI   at 0 range 0 .. 15;
      THETA at 1 range 0 .. 15;
      PHI   at 2 range 0 .. 15;
   end record;
```

3.9.2. Message Type 2

```
for MESSAGE_2_TYPE use
   record
      CONSTANT1 at 0 range 0 .. 15;
   end record;
```

3.9.3. Message Type 3

```
for MESSAGE_3_TYPE use
  record
    CONSTANT1                 at  0 range 0 .. 15;
    CONSTANT2                 at  1 range 0 .. 15;
    CONSTANT3                 at  2 range 0 .. 15;
    PSI                       at  3 range 0 .. 15;
    NAV_VEL_X                 at  4 range 0 .. 15;
    NAV_VEL_Y                 at  5 range 0 .. 15;
    NAV_VEL_Z                 at  6 range 0 .. 15;
    PLATFORM_X_ACCELERATION   at  7 range 0 .. 15;
    PLATFORM_Y_ACCELERATION   at  8 range 0 .. 15;
    VERTICAL_ACCELERATION     at  9 range 0 .. 15;
    RATE_X                    at 10 range 0 .. 15;
    RATE_Y                    at 11 range 0 .. 15;
    RATE_Z                    at 12 range 0 .. 15;
    NAV_BAROMETRIC_RATE       at 13 range 0 .. 15;
  end record;
```

3.9.4. Message Type 4

```
for MESSAGE_4_TYPE use
  record
    CONSTANT1                 at  0 range 0 .. 15;
    CONSTANT2                 at  1 range 0 .. 15;
    CONSTANT3                 at  2 range 0 .. 15;
    PSI1                      at  3 range 0 .. 15;
    PSI2                      at  4 range 0 .. 15;
    THETA                     at  5 range 0 .. 15;
    PHI1                      at  6 range 0 .. 15;
    PHI2                      at  7 range 0 .. 15;
    NAV_VEL_Y                 at  8 range 0 .. 15;
    NAV_VEL_X                 at  9 range 0 .. 15;
    NAV_VEL_Z                 at 10 range 0 .. 15;
    NAV_ALTITUDE_1            at 11 range 0 .. 15;
    NAV_ALTITUDE_2            at 12 range 0 .. 15;
    NAV_LATITUDE_DEG          at 13 range 0 .. 15;
    NAV_LONGITUDE_DEG         at 14 range 0 .. 15;
    CONSTANT4                 at 15 range 0 .. 15;
    PLATFORM_Y_ACCELERATION   at 16 range 0 .. 15;
    PLATFORM_X_ACCELERATION   at 17 range 0 .. 15;
    VERTICAL_ACCELERATION     at 18 range 0 .. 15;
    CONSTANT5                 at 19 range 0 .. 15;
    CONSTANT6                 at 20 range 0 .. 15;
    CONSTANT7                 at 21 range 0 .. 15;
    CONSTANT8                 at 22 range 0 .. 15;
    CONSTANT9                 at 23 range 0 .. 15;
    RATE_X                    at 24 range 0 .. 15;
    RATE_Y                    at 25 range 0 .. 15;
    RATE_Z                    at 26 range 0 .. 15;
    CONSTANT10                at 27 range 0 .. 15;
  end record;
```

3.9.5. Message Type 5

```
for RAW_DATA_TYPE use
  record
    PLATFORM_X_ACCELERATION_1 at  0 range 0 .. 15;
    PLATFORM_X_ACCELERATION_2 at  1 range 0 .. 15;
    PLATFORM_Y_ACCELERATION_1 at  2 range 0 .. 15;
    PLATFORM_Y_ACCELERATION_2 at  3 range 0 .. 15;
    VERTICAL_ACCELERATION_1   at  4 range 0 .. 15;
    VERTICAL_ACCELERATION_2   at  5 range 0 .. 15;
    RATE_X_1                  at  6 range 0 .. 15;
    RATE_X_2                  at  7 range 0 .. 15;
    RATE_Y_1                  at  8 range 0 .. 15;
    RATE_Y_2                  at  9 range 0 .. 15;
    RATE_Z_1                  at 10 range 0 .. 15;
    RATE_Z_2                  at 11 range 0 .. 15;
    BAROMETRIC_ALTITUDE_1     at 12 range 0 .. 15;
    BAROMETRIC_ALTITUDE_2     at 13 range 0 .. 15;
  end record;
```

3.9.6. Message Type 6

```
for BENCHMARK_RESULTS_TYPE use
  record
    INPUT_COMMAND                at 0 range 0 .. 63;
    STATUS                       at 4 range 0 .. 15;
    ADDITIONAL_PROCESSING_TIME   at 5 range 0 .. 31;
    MAX_IO_COUNT                 at 7 range 0 .. 15;
  end record;
```

3.9.7. Message Type 7

```
for BENCHMARK_COMMAND_TYPE use
  record
    ECPM_CONTROL_WORD              at 0 range 0 .. 15;
    TYPE_OF_COMMAND                at 1 range 0 .. 15;
    BENCHMARK_DURATION_COUNTER     at 2 range 0 .. 15;
    IO_MIX_ITERATIONS_PER_SECOND   at 3 range 0 .. 15;
  end record;
```

3.9.8. Message Type 10

```
for MESSAGE_10_TYPE use
  record
    CONSTANT1 at 0 range 0 .. 15;
    CONSTANT2 at 1 range 0 .. 15;
    CONSTANT3 at 2 range 0 .. 15;
    CONSTANT4 at 3 range 0 .. 15;
  end record;
```

3.9.9. Message Type 15

```
for MESSAGE_15_TYPE use
  record
    CONSTANT1  at 0 range 0 .. 15;
    CONSTANT2  at 1 range 0 .. 15;
    CONSTANT3  at 2 range 0 .. 15;
    CONSTANT4  at 3 range 0 .. 15;
    CONSTANT5  at 4 range 0 .. 15;
    CONSTANT6  at 5 range 0 .. 15;
    CONSTANT7  at 6 range 0 .. 15;
    CONSTANT8  at 7 range 0 .. 15;
    CONSTANT9  at 8 range 0 .. 15;
    CONSTANT10 at 9 range 0 .. 15;
  end record;
```

4. Quality Assurance Provisions

NONE.

5. Preparations for Delivery

NONE.

6. General Information

6.1. Notes

The following assumptions were made during development of the ECPM for the MIL-STD-1750A and will apply to future implementations of the program unless contrary guidance is received from NAC.

1. When the ECPM is measuring the maximum number of iterations of the I/O message mix, all commands from the Master Computer will be ignored until the measurement is completed. When running the ECPM, users should be aware that the time required to process and ignore commands during a measurement event should invalidate that event (i.e., additional overhead has been introduced).

2. When the ECPM is measuring the maximum spare processor reserve, all commands from the Master Computer will be ignored until the calculation is completed.

3. When a stop command is received from the Master Computer while the ECPM is executing in NAV_ONLY or RECORDING_RESULTS mode, the ECPM will continue executing until the end of the current 50 millisecond period. At that time, a benchmark results message will be returned to the Master Computer. If the ECPM was in RECORDING_RESULTS mode, the status word in the benchmark results message will indicate that the results are invalid due to the receipt of a command while recording.

4. A prioritized, preemptive scheduler will be used.

5. Inputs to the navigation equations (i.e., data received via 1553B subaddresses 1, 2, 3, 4, 10, and 15) are not checked for accuracy in the ECPM.

6. If the ECPM is restarted, all navigation variables defined in the package NAV_DATA will be re-initialized.

7. If a second start command is received while the ECPM is executing, it will be ignored. The program will continue executing until a timeout occurs or a stop command is received.

6.2. MIL-STD-1750A Configuration Information

1. There is no requirement for the ECPM code to run in a specific address state. However, the Communications Services package, which is unique to TI's implementation, must execute from address state 0 on the MIL-STD-1750A.

2. The size of the ECPM object module, excluding the machine dependent code needed to support TI's messaging scheme, is 9006 16-bit words. The default stack size and pre-defined storage for access collections allocated by the Tartan compiler were sufficient to run the benchmark. The Tartan runtime required 7724 words of storage. The total memory required for all components including the ECPM, TI NOS, and Tartan Ada runtime was 21167 words.

6.3. Additional I/O Task Details

Spare I/O is measured in terms of an additional I/O mix. This mix consists of sending six identical Pi bus chains where each chain consists of sending 10 type 16, single slave, block Pi bus messages which use label addressing. For a particular experiment, all the block messages used in the additional I/O mix are either extended headers or short headers. The label table entries (unique to TI's NOS) in the Slave are configured so as not to cause an interrupt on delivery of a message. The Slave ID field (bits 0 - 7 in HWA, where 0 is LSB) is implementation dependent. The value of the data associated with a message is superfluous, but the amount of data associated with a message (HWB) is important. The number of 16-bit words of data associated with each message is shown in Table 3. The value of the label field (HWC0) is implementation specific, and for extended headers, the value of the extended header fields (HWD0-HWD6) is implementation dependent. Pi bus control is vendor specific, but should allow for the Master to be interrupted on completion of a chain being sent.

PI BUS MESSAGE	NUMBER OF WORDS OF DATA
1	120
2	120
3	120
4	15
5	15
6	7
7	48
8	48
9	48
10	48

Table 3. Message Mix Used in Additional I/O Task.

6.4. Comment on 1750 to 1553B I/O

For messages sent to the 1553B module, Pi bus block messages are to be used. Whether these messages use short or extended headers, or whether label or direct addressed messages are used, is implementation dependent. However, the structure of the data and size of the data to be transferred in the data phase of the Pi bus message must be as defined in the ICD. The TI implementation uses type 16, single slave, extended header block Pi bus messages and label addressing.

6.5. List of Acronyms

6-DOF	6 Degrees-of-Freedom
AATD	Advanced Avionics Technology Demonstration
BIM	Bus Interface Module
CCB	Communications Control Block
CSCI	Computer Software Configuration Item
CPU	Central Processing Unit
DASL	Digital Avionic Systems Laboratory
DID	Data Item Description
DMA	Direct Memory Access
DSEG	Defense Systems and Electronics Group
ECPM	Embedded Computer Performance Measurement
ICD	Interface Control Document
IOBIDS	Input/Output Built-In-Test Interface Description Specification
IRS	Interface Requirements Specification
LRM	Language Reference Manual (for Ada)
MC	Master Computer
NAC	Naval Avionics Center
NOS	Network Operating System
SAx	Subaddress x on MIL-STD-1553B (x = 1..31)
STD	Software Technology Department
SRS	Software Requirements Specification
TI	Texas Instruments Incorporated
VMEbus	VersaModule Europe bus

Appendix A. Machine Dependent Types Package for MIL-STD-1750A

```
with SYSTEM;
package MACHINE_DEPENDENT_TYPES is
   WORD_SIZE : constant positive := 16;
   -- Type of data transferred between the target
   -- and master computer.
   type PACKED_INTEGER_TYPE is new integer range -32768 .. 32767;
   for PACKED_INTEGER_TYPE'size use 16;
   -- This is the priority of the MAIN and NAV_EXEC tasks.
   -- It is implementation-dependent.
   NORMAL_PRIORITY : constant SYSTEM.PRIORITY := 11;
end MACHINE_DEPENDENT_TYPES;
```

Appendix B. I/O Services Package for MIL-STD-1750A

```
-----------------------------------------------------------------------------
-- TITLE:                        I/O Services Package                      --
--                                                                         --
-- PURPOSE:                                                                --
--   This package contains procedures which allow the transfer of data     --
--   between the target and the master computer.                           --
--                                                                         --
-- PROCESSING:                                                             --
--   N/A                                                                   --
--                                                                         --
-- INPUTS:                                                                 --
--   None                                                                  --
--                                                                         --
-- OUTPUTS:                                                                --
--   None                                                                  --
--                                                                         --
-- DEPENDENCIES:                                                           --
--   SYSTEM                                                                --
--   MESSAGE_TYPES                                                         --
--                                                                         --
-- GLOBAL VARIABLES DECLARED:                                              --
--   None                                                                  --
--                                                                         --
-- GLOBAL VARIABLES ACCESSED:                                              --
--   None                                                                  --
--                                                                         --
-- EXCEPTIONS RAISED:                                                      --
--   None                                                                  --
--                                                                         --
-- CALLED BY:                                                              --
--   N/A                                                                   --
--                                                                         --
-- CALLS:                                                                  --
--   N/A                                                                   --
--                                                                         --
-- SIDE EFFECTS:                                                           --
--   N/A                                                                   --
--                                                                         --
-- TARGET PROCESSOR:                                                       --
--   implementation-dependent                                              --
--                                                                         --
-- DESIGN MATERIALS:                                                       --
--   Software Requirements Specification for the Advanced Avionics         --
--     Technology Demonstration (AATD) CSCI of the AATD System,            --
--   Advanced Avionics Technology Demonstration (AATD) Program Embedded    --
--     Computer Performance Measurement (ECPM) MIL-STD-1553B Interface     --
--     Definition                                                          --
--                                                                         --
-- HISTORY:                                                                --
--   Original - 8/30/90 Diane Paul                                         --
--                                                                         --
-----------------------------------------------------------------------------
```

```
with SYSTEM;
with MESSAGE_TYPES;

package IO_SERVICES is

    -- Declare the subaddresses used by the benchmark.
    type SUBADDRESS_TYPE is (SA1, SA2, SA3, SA4, INPUT_DATA_SA,
                             PERF_DATA_SA, BENCH_CMD_SA, SA10, SA15);
    for SUBADDRESS_TYPE use (SA1           = 1,
                             SA2           = 2,
                             SA3           = 3,
                             SA4           = 4,
                             INPUT_DATA_SA = 5,
                             PERF_DATA_SA  = 6,
                             BENCH_CMD_SA  = 7,
                             SA10          = 10,
                             SA15          = 15);

    -- Declare the procedures used to communicate with the MC.
    procedure INITIALIZE_1553_COMMUNICATION;

    procedure INITIALIZE_NAV_IO;

    procedure WAIT_FOR_BENCHMARK_COMMAND (
        BENCHMARK_COMMAND_ACCESS : out
           MESSAGE_TYPES.BENCHMARK_COMMAND_ACCESS_TYPE);

    procedure GET_NAV_DATA (
        RAW_DATA_ACCESS : out MESSAGE_TYPES.RAW_DATA_ACCESS_TYPE);

    procedure WRITE_NAV_DATA (RESULTS_ADDRESS : in SYSTEM.address;
                              BYTE_COUNT      : in positive;
                              SUBADDRESS      : in SUBADDRESS_TYPE);

    procedure WRITE_BENCHMARK_RESULTS (
       BENCHMARK_RESULTS_ADDRESS : in SYSTEM.address;
       BYTE_COUNT                : in positive);

end IO_SERVICES;
```

```
-----------------------------------------------------------------------------
-- TITLE:                    I/O Services Package                          --
--                                                                         --
-- PURPOSE:                                                                --
--   This package contains procedures which allow the transfer of data     --
--   between the target and the master computer, as well as data structures --
--   used by the separate procedures.                                      --
--                                                                         --
-- PROCESSING:                                                             --
--   N/A                                                                   --
--                                                                         --
-- INPUTS:                                                                 --
--   None                                                                  --
--                                                                         --
-- OUTPUTS:                                                                --
--   None                                                                  --
--                                                                         --
-- DEPENDENCIES:                                                           --
--   SYSTEM                                                                --
--   VHSIC_1750A_DEPENDENT_TYPES                                           --
--   PIBUS_INTERFACE_TYPES                                                 --
--   COM_TYPES                                                             --
--   MESSAGE_SERVICES                                                      --
--   MDP_COMM_PROCEDURES                                                   --
--   MDP_COMMUNICATIONS                                                    --
--   MACHINE_DEPENDENT_TYPES                                               --
--                                                                         --
-- GLOBAL VARIABLES DECLARED:                                              --
--   None                                                                  --
--                                                                         --
-- GLOBAL VARIABLES ACCESSED:                                              --
--   None                                                                  --
--                                                                         --
-- EXCEPTIONS RAISED:                                                      --
--   None                                                                  --
--                                                                         --
-- CALLED BY:                                                              --
--   N/A                                                                   --
--                                                                         --
-- CALLS:                                                                  --
--   N/A                                                                   --
--                                                                         --
-- SIDE EFFECTS:                                                           --
--   N/A                                                                   --
--                                                                         --
-- TARGET PROCESSOR:                                                       --
--   implementation-dependent                                              --
--                                                                         --
-- DESIGN MATERIALS:                                                       --
--   Software Requirements Specification for the Advanced Avionics         --
--     Technology Demonstration (AATD) CSCI of the AATD System,            --
--   Advanced Avionics Technology Demonstration (AATD) Program Embedded    --
--     Computer Performance Measurement (ECPM) MIL-STD-1553B Interface     --
```

```
--     Definition                                                          --
--                                                                         --
-- HISTORY:                                                                --
--   Original - 8/30/90 Diane Paul                                         --
--                                                                         --
-----------------------------------------------------------------------------

with SYSTEM;
with VHSIC_1750A_DEPENDENT_TYPES;
with MACHINE_DEPENDENT_TYPES;
with PIBUS_INTERFACE_TYPES;
with MDP_COMMUNICATIONS;
with MDP_COMM_PROCEDURES;
with COM_TYPES;
with MESSAGE_SERVICES;
with V1750_UTILITIES;
with UNCHECKED_CONVERSION;

package body IO_SERVICES is

    package MACHINE_TYPES renames MACHINE_DEPENDENT_TYPES;
    package V17_TYPES renames VHSIC_1750A_DEPENDENT_TYPES;
    package PIBUS_TYPES renames PIBUS_INTERFACE_TYPES;

    function CONVERT_INTEGER_TO_SYSTEM_ADDRESS is new
        UNCHECKED_CONVERSION(
            source = integer,
            target = SYSTEM.address);

    function CONVERT_LOGICAL_ADDRESS_TO_SYSTEM_ADDRESS is new
        UNCHECKED_CONVERSION(
            source = V17_TYPES.LOGICAL_ADDRESS_TYPE,
            target = SYSTEM.address);

    function CONVERT_SYSTEM_ADDRESS_TO_LOGICAL_ADDRESS is new
        UNCHECKED_CONVERSION(
            source = SYSTEM.address,
            target = V17_TYPES.LOGICAL_ADDRESS_TYPE);

    function CONVERT_SYSTEM_ADDRESS_TO_TASK_BUFFER_ACCESS is new
        UNCHECKED_CONVERSION (
            source = SYSTEM.address,
            target = COM_TYPES.TASK_BUFFER_ACCESS_TYPE);

    function CONVERT_SUBADDRESS_TO_SA is new UNCHECKED_CONVERSION(
        source = SUBADDRESS_TYPE,
        target = MDP_COMMUNICATIONS.SA_TYPE);

    function CONVERT_INTEGER_TO_LOGICAL_ADDRESS is new
        UNCHECKED_CONVERSION (
            source = integer,
            target = V17_TYPES.LOGICAL_ADDRESS_TYPE);

    function CONVERT_BENCHMARK_COMMAND_ACCESS_TO_LOGICAL_ADDRESS is new
        UNCHECKED_CONVERSION (
```

```
                source = MESSAGE_TYPES.BENCHMARK_COMMAND_ACCESS_TYPE, @CODE A =
target = V17_TYPES.LOGICAL_ADDRESS_TYPE); @CODE A =
    function CONVERT_LOGICAL_ADDRESS_TO_BENCHMARK_COMMAND_ACCESS is new
        UNCHECKED_CONVERSION(
            source = V17_TYPES.LOGICAL_ADDRESS_TYPE,
            target = MESSAGE_TYPES.BENCHMARK_COMMAND_ACCESS_TYPE);

    function CONVERT_SYSTEM_ADDRESS_TO_BENCHMARK_COMMAND_ACCESS is new
        UNCHECKED_CONVERSION(
            source = SYSTEM.address,
            target = MESSAGE_TYPES.BENCHMARK_COMMAND_ACCESS_TYPE);

    function CONVERT_ADDRESS_TO_MSG_BUFFER_ACCESS is new
        UNCHECKED_CONVERSION(
            source = SYSTEM.address,
            target = COM_TYPES.MSG_BUFFER_ACCESS);

    function CONVERT_SYSTEM_ADDRESS_TO_RAW_DATA_ACCESS is new
        UNCHECKED_CONVERSION (
            source = SYSTEM.address,
            target = MESSAGE_TYPES.RAW_DATA_ACCESS_TYPE);

    function CONVERT_MSG_BUFFER_ACCESS_TO_RAW_DATA_ACCESS is new
        UNCHECKED_CONVERSION (
            source = COM_TYPES.MSG_BUFFER_ACCESS,
            target = MESSAGE_TYPES.RAW_DATA_ACCESS_TYPE);

    NUMBER_OF_BENCHMARK_COMMAND_BUFFERS : constant := 2;
   BENCHMARK_COMMAND_BUFFERS : array (1 .. NUMBER_OF_BENCHMARK_COMMAND_BUFFERS)

        of MESSAGE_TYPES.BENCHMARK_COMMAND_TYPE;
    -- Set up the array containing pointers to the buffers for the benchmark
    -- command messages.
   BENCHMARK_COMMAND_BUFFER_PTRS_ACCESS : COM_TYPES.TASK_BUFFER_ACCESS_TYPE :=
        new COM_TYPES.TASK_BUFFER_TYPE'
           (1      = CONVERT_SYSTEM_ADDRESS_TO_LOGICAL_ADDRESS(
                                        BENCHMARK_COMMAND_BUFFERS(1).ECPM_CON-
TROL_WORD'address),
            2      = CONVERT_SYSTEM_ADDRESS_TO_LOGICAL_ADDRESS(
                                        BENCHMARK_COMMAND_BUFFERS(2).ECPM_CON-
TROL_WORD'address),
            others = CONVERT_INTEGER_TO_LOGICAL_ADDRESS(0));

    -- Declare a local variable into which the message can be copied so that
    -- the buffer can be released.
    LOCAL_BENCHMARK_COMMAND : MESSAGE_TYPES.BENCHMARK_COMMAND_TYPE;
    LOCAL_BENCHMARK_CMD_PTR : MESSAGE_TYPES.BENCHMARK_COMMAND_ACCESS_TYPE :=
        CONVERT_SYSTEM_ADDRESS_TO_BENCHMARK_COMMAND_ACCESS(
            LOCAL_BENCHMARK_COMMAND.ECPM_CONTROL_WORD'address);

    -- Declare the input buffer into which the raw (packed) nav data will
    -- be placed.
    INPUT_BUFFER : MESSAGE_TYPES.RAW_DATA_TYPE;
    INPUT_MSG_BUFFER_ACCESS : COM_TYPES.MSG_BUFFER_ACCESS :=
        CONVERT_ADDRESS_TO_MSG_BUFFER_ACCESS(
```

```
            INPUT_BUFFER.PLATFORM_X_ACCELERATION_1'address);

    -- Declare the buffer into which the input data is copied.
    RAW_DATA     : MESSAGE_TYPES.RAW_DATA_TYPE;
    RAW_DATA_PTR : MESSAGE_TYPES.RAW_DATA_ACCESS_TYPE :=
        CONVERT_SYSTEM_ADDRESS_TO_RAW_DATA_ACCESS(
            RAW_DATA.PLATFORM_X_ACCELERATION_1'address);

    -- Declare the table which maps subaddresses to BIM labels to which
    -- messages destined for the MC are to be sent.
    SA_TO_LABEL_MAPPING_TABLE : array (SUBADDRESS_TYPE) of
        MDP_COMMUNICATIONS.BIM_SEND_LABEL_TYPE;

    NULL_SYSTEM_ADDRESS : SYSTEM.address :=
        CONVERT_INTEGER_TO_SYSTEM_ADDRESS(0);

    procedure INITIALIZE_1553_COMMUNICATION is separate;
    procedure INITIALIZE_NAV_IO is separate;

    procedure WAIT_FOR_BENCHMARK_COMMAND (
        BENCHMARK_COMMAND_ACCESS :
            out MESSAGE_TYPES.BENCHMARK_COMMAND_ACCESS_TYPE) is separate;

    procedure GET_NAV_DATA (
        RAW_DATA_ACCESS : out MESSAGE_TYPES.RAW_DATA_ACCESS_TYPE) is separate;

    procedure WRITE_NAV_DATA (RESULTS_ADDRESS : in SYSTEM.address;
                              BYTE_COUNT      : in positive;
                             SUBADDRESS      : in SUBADDRESS_TYPE) is separate;

    procedure WRITE_BENCHMARK_RESULTS (
       BENCHMARK_RESULTS_ADDRESS : in SYSTEM.address;
       BYTE_COUNT                : in positive) is separate;


end IO_SERVICES;
```