Utility Module Specification

James E. Francis

MIT 64-30110.2001

Rev. D

October 6, 1993

Contents

Figures

1 Introduction

1.1 Scope

This document describes the components that make up the Utility Module (UTIL). This module consists of a set of utility functions that the Flight Software uses to perform a variety of tasks.

This document does not describe the overall design of the EDS, nor does it describe the operating details of each component in this module. It assumes that the reader is extremely familiar with the Flight Software Global Design, with the "C" programming language and with the architecture of the EDS.

This document should be of primary interest to Flight Software Engineering, Flight Hardware Engineering, Ground Software Engineering and Software Quality Assurance.

1.2 References

ISBN 1-55512-055-5 1987 .... "80286 and 80287 Programmer's Reference Manual"
MIT 64-02009 Rev. D .... "EDS-PCA Science Requirements"
MIT 64-02012 Rev. C .... "EDS-ASM Science Requirements"
MIT 64-30102 Rev. F .... "Flight Software Global Design"
MIT 64-30110.0401 Rev. D .... "Downlink Module Specification" (DOWNLINK)
MIT 64-30110.0101 Rev. C .... "Back Registers Module Specification" (BREG)
MIT 64-30110.1201 Rev. E .... "Interrupt Manager Module Specification" (INTR)
MIT 64-30111.0201 Rev. C .... "Buffer Pool Module Specification" (BUFP)
MIT 64-30110.2301 Rev. D .... "Startup Module Specification" (STARTUP)
MIT 64-30112.0501 Rev. C .... "Supervisor Thread Module Specification" (SUPER)

1.3 Notations

This document assumes the following notations:

INT8 .... Refers to an 8-bit signed value.
UINT8 .... Refers to an 8-bit unsigned value.
INT16 .... Refers to a 16-bit signed value.
UINT16 .... Refers to a 16-bit unsigned value.
INT32 .... Refers to a 32-bit signed value.

UINT32 .... Refers to a 32-bit unsigned value.
BOOL .... Refers to a 16-bit value which represents a TRUE or FALSE condition.

1.4 Module Perspective

The following diagram is a copy of the Level 1 data flow diagram from the **Flight Software Global Design**, Section 4.1.2. The Utility Module does not appear directly in this data flow, but has a scope which indirectly affects all of the Flight Software. These items are indicated by the shaded areas:

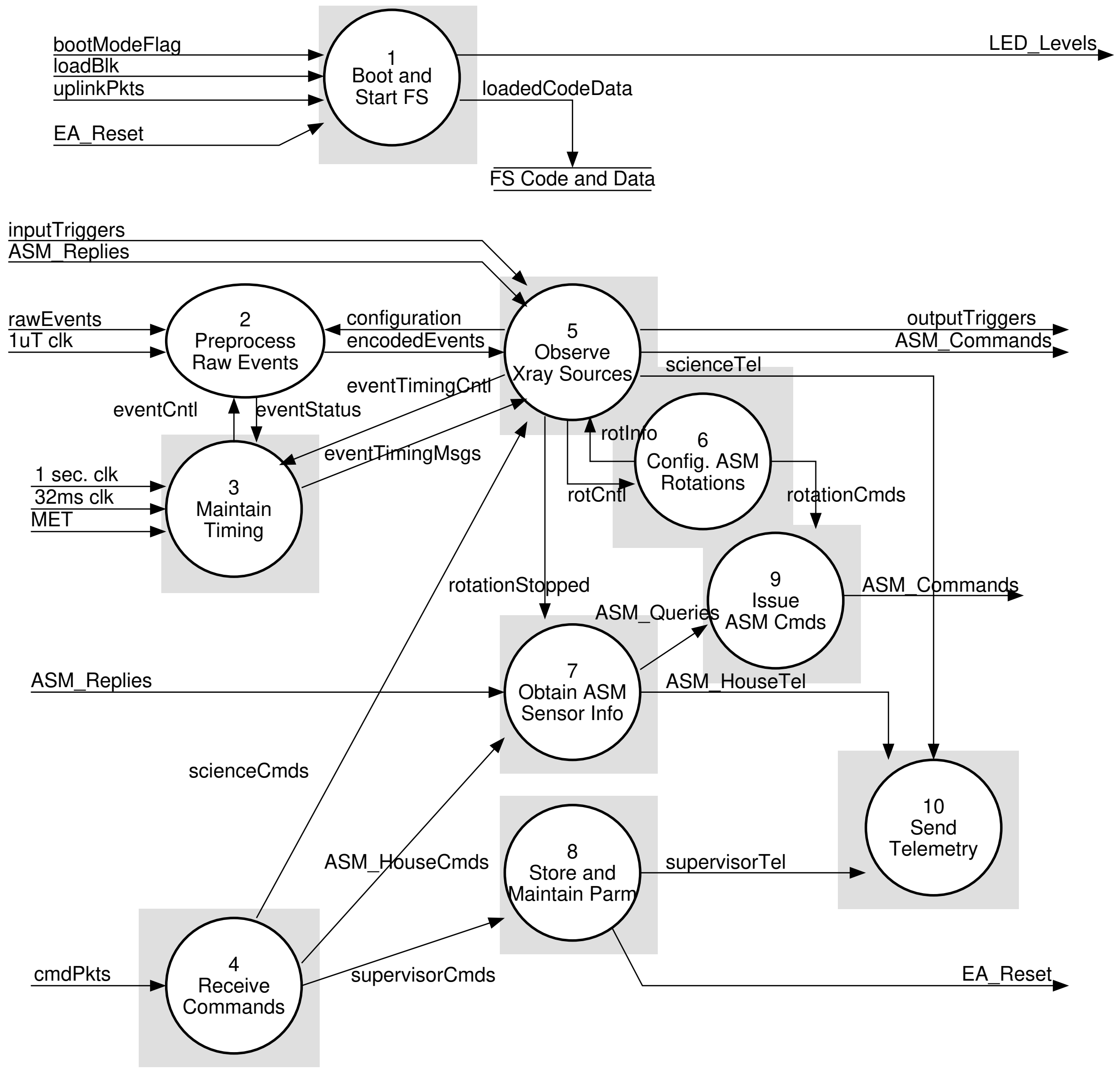

Figure 1 Module Perspective

1.5 Document Overview

Sections 1 and 2 provide an overview of the Utility Module, summarizing the purpose, services, context, and interfaces of the module. Section 3 describes all software structures and constants visible to modules outside of the Utility Module. Section 4 describes the functions provided to other modules by the Utility Module. Section 5 describes some of the implementation details of the module. Section 6 describes which requirements are met by the module. Section 7 describes what types of tests need to be performed to ensure that the implementation of the module is operating properly. Appendix A lists items which are currently undefined, or that need to be reviewed.

1.6 Acronyms

Please refer to the **Flight Software Global Design Specification**, Section 1.5, for a list of acronyms and abbreviations.

2 Overview

2.1 Purpose

The purpose of the Utility Module is to provide functions that the Flight Software uses to perform a variety of tasks. These tasks include issuing startup messages, issuing panic messages and EA resets, forming and issuing command acknowledgments, maintaining the software housekeeping data base, filling memory, buffer copying, and address format conversion.

2.2 Services

The Utility Module provides the following services:

1) Perform system panics .... Issues panic messages and resets the EA, using the Watchdog Timer.
2) Issue Startup Message Pkt .... Forms and issues the Startup Message during system initialization.
3) Issue Command Acks .... Forms and issues Command Acknowledgement packets.
4) Log Housekeeping Data .... Provides a mechanism for maintaining a set of "counts" of software housekeeping statistics.
5) Retrieve and Reset Statistics .... Provides mechanism for retrieving and resetting the set of software housekeeping statistics.
6) Fast buffer copies .... Provides a fast mechanism for copying data from one buffer to another.
7) Fast buffer fills .... Provides a fast mechanism for filling a data buffer with a single 1 byte value.
8) Pointer conversion .... Provides a set of functions for minimizing the offset portion of a native (80286) address, for converting addresses from native format to flat format, and visa-versa, and provides mechanisms for obtaining the segment or offset portion of a native address. It also provides a facility for verifying that a memory pointer (native format) references a memory location within 80286's physical address space.
9) NOP function .... Provides a "nop" function, for callers who need to pause between writing certain hardware registers, or who are executing busy-loops.

2.3 Module Context

The following illustrates the Utility Module's relationships with the rest of the Flight Software residing in an EA:

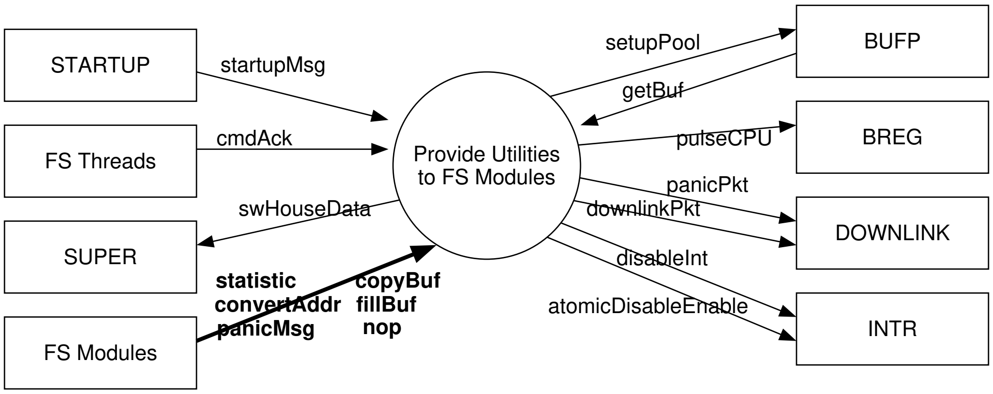

Figure 2 Utility Module Context Diagram

During system initialization, the Utility Module uses the **Buffer Pool Module** to initialize a pool of utility telemetry buffers (setupPool). These buffers are used to contain startup message (startupMsg), panic message (panicMsg), and command acknowledgment (cmdAck) telemetry packets. Once running, the Utility Module obtains these buffers (getBuf) using the **Buffer Pool Module**, and, except for panic messages, posts these packets to the **Downlink Module** (downlinkPkt). When issuing a panic message, the Utility Module synchronously sends the packet using the **Downlink Module** (panicPkt), and resets the EA hardware by disabling interrupts (disableInt), using the **Interrupt Manager Module**, starting the watchdog timer (pulseCPU), using the **Back End Registers Module**, and by busy-looping until the Watchdog Timer expires.

The Utility Module provides the facilities used by the **Startup Module** to send startup messages to the SDS (startupMsg). It provides the facilities used by the Flight Software's threads to build and send command acknowledgment packets (cmdAck). It also provides facilities for the rest of the Flight Software to do the following; increment a software housekeeping statistic (statistic), manipulate native (80286) addresses (convertAddr), issue panic messages (panicMsg), copy data from one buffer to another (copyBuf), fill a buffer with a value (fillBuf), and to burn CPU cycles (nop).

The **Supervisor Thread Module** uses the Utility Module to obtain and reset the software housekeeping statistics (swHouseData). The Utility Module performs this operation as a single atomic step (no interrupts), using the **Interrupt Manager Module** (atomicDisableEnable).

2.4 Theory of operation

This section provides an overview of the operation of the Utility Module. The services provided by the Utility Module can be grouped as follows:

Telemetry Services .... Startup Message, Panic Messages, and Command Acknowledgments.
Software Housekeeping Services ..... Reporting statistics, and obtaining and resetting the database.
Address Manipulation Services .... Address Conversion, adjustment, segment:offset extraction and pointer tests.
Buffer Manipulation Services........... Buffer copies and fills.
Execution Services .... NOP function, EA reset.

The Utility Module uses the **Buffer Pool Module** to construct and maintain a pool of telemetry buffers for use by startup messages, panic messages and command acknowledgments. The only "general purpose" telemetry provided by the Utility Module are command acknowledgments. The startup message only occurs during system initialization, and panic messages only occur during system termination (just before a reset). When issuing a startup message, the Utility Module uses the **Buffer Pool Module** to obtain a telemetry buffer. This facilitates returning the buffer to the pool after the message has been sent. It assumes that a telemetry buffer is available when issuing a startup message, since no other telemetry has been produced by the EA. The absence of a buffer indicates system corruption, and the Utililty Module panics if a buffer is not available.

During panics, the Utility Module disables interrupts and sets an internal flag, preventing recursive panic messages. If the flag had already been set, the Utility Module does not try to send another panic message, but instead immediately resets the EA. If the flag hasn't been set, it grabs the first buffer in the pool. The Utility Module has no need to return the buffer to the pool, since it is about to reset the EA anyway. It fills the buffer with the panic message and argument, and sends the packet syncrhonously using the **Downlink Module**. Once the message has been sent to the SDS, or the transfer has timed out (managed by the **Downlink Module**), the Utility Module starts the Watchdog Timer and busy-waits. Eventually, the Watchdog Timer expires and resets the EA hardware. This is more reliable than just jumping to the reset vector, since it ensures that the EA's hardware is also reset to its default state.

The Utility Module sends command acknowledgments in a kinder, gentler manner than it does with startup and panic messages. The Utility Module attempts to obtain a buffer, using the **Buffer Pool Module**. If a buffer isn't available, the Utility Module increments a count in the Software Housekeeping statistics, which indicates the number of dropped command acknowledgments, and returns. If it obtains a buffer, the Utility Module copies the entire command packet buffer, containing the command being acknowledged, into the obtained telemetry buffer. It then fills in the acknowledgment code and data, and posts the packet for transfer, using the **Downlink Module**. Once the packet has been transferred, the **Downlink Module** will release the buffer back to the pool. This scheme relies on the fact that command packets are less than or equal 64 bytes in size, whereas, telemetry packets can be up to 1024 bytes (see **Flight Software Global Design Specification**, Sections 3.2.1 and 3.2.2).

The Utility Module is responsible for maintaining a set of software housekeeping statistics. Each "statistic" refers to a unique 16-bit counter in an array. Each time a Flight Software Module logs a statistic, the Utility Module increments the corresponding counter in this array. Periodically, the **Supervisor Thread** calls the Utility Module to obtain and reset this array. When it does so, the Utility Module atomically (i.e. with interrupts disabled) copies the array into the passed buffer and zeros the array.

The Utility Module provides a set of routines used to manipulate native (80286) addresses. In real mode, which is what the Flight Software is running in (see **ROM Boot Module Specification**, Section 4.1), the 80286 generates a 20-bit bus (flat) address using a pair of 16-bit values. The first part, known as the "segment", addresses memory in 16 byte chunks. The second part, known as the "offset", is added to the segment to form the final bus address. The common notation for this type of address takes the form of "segment:offset." So, for any given segment, the "offset" portion addresses one of 64K bytes. This addressing scheme produces non-unique addresses. For example, 0x0000:0x00010 is equivalent to 0x0001:0x0000. The Utility Module provides facilities to; minimize the offset portion of an address, convert native address to and from flat address (where each byte address is unique), obtain the segment and offset portions of a native address and test a native address to ensure that the corresponding flat address is within the physical address space of the 80286.

The Utility Module provides two functions to take advantage of the 80286’s block fill and block copy instructions. The first function quickly fills up to one segment’s worth (64K bytes) of data with a single value. The second function quickly copies up to 64K bytes of data from one buffer to another. These buffers must not overlap.

The Utility Module provides a function which, when called, immediately returns. This function is useful when configuring hardware which requires a short delay between register accesses, and when performing busy-loops.

2.5 Data Structures and Constants Summary

This module defines a variety of structures and constants, such as the format of the startup message, the format of panic messages, the format of command acknowledgments, address formats, maximum values for buffer copying, etc.

2.6 Function List Summary

This module defines the following functions:

UTILinit()	Initialize the module.
UTILpanic()	Issue a panic message, and restart the software.
UTILstartup_msg()	Issue a startup message.
UTILack()	Issue a command acknowledgement.
UTILlog_housekeeping()	Increment a housekeeping statistic.
UTILhousekeeping_update()	Retrieve and reset housekeeping statistiscs.
UTILcopy_buf()	Copy a buffer.
UTILfill_buf()	Fill a buffer with a value.
UTILflat2pointer()	Convert flat address to segment:offset format.
UTILpointer2flat()	Convert segment:offset format to flat address.
UTILsegment()	Get the segment portion of a segment:offset addr.
UTILoffset()	Get the offset portion of a segment:offset address.
UTILadjust_address()	Minimize offset portion of a segment:offset address.
UTILvalid_ptr()	Test that a segment:offset address is within the 80286 physical address space.
UTILnop()	A "do-nothing" function for pausing while programming hardware or performing a busy loop.

2.7 System Resource Utilization Summary

Telemetry .... 1 telemetry packet per call to UTILstartup_msg(), UTILack(), or UTILpanic()

3 Data Structures and Constants

3.1 Telemetry Packet Formats

All telemetry packets consist of a CCSDS header, followed by a secondary header, followed by the packet's stream id, followed by a single reserved byte, followed by stream specific data. All stream id's used by the Utility Module are listed in Section 3.3. For a description of the CCSDS and secondary headers, and of the location of the stream id field, refer to the **Flight Software Global Design Specification**, Section 3.2.2.

The Utility Module sends all telemetry packets to the high-priority telemetry level (see **SDS to EDS Interface Control Document**, and the **Downlink Module Specification**, PKT_PRIORITY_HIGH description). The following sections describe the stream specific data for telemetry packets produced by the Utility Module.

3.1.1 Startup Message Packet Format

The following describes the format of the data portion of a Startup Message, whose stream id is STREAM_ID_STARTUP_MSG (see Section 3.3.1):

PRAM Version Id (4 bytes) .... This is a copy of the 4 byte PRAM version id, written into a fixed location in PRAM by the ground. Its use and interpretation is completely up to the ground software.

3.1.2 Panic Message Packet Format

The format of the data portion of a Panic Message Packet is defined by the the **Downlink Module Specification**, Section "Data Structures and Constants: Panic Message Packet Format." It is repeated here for reference:

Message Code (2 bytes) .... A 16-bit value which uniquely identifies a given message. Some messages require an additional 32-bit integer value, which is sent in the next argument.

Message Argument (4 bytes) .... This is a 32-bit integer argument, which may or may not be needed for the message specified in the previous argument. The bytes are always sent. If the message doesn't use this field, the contents are ignored.

3.1.3 Command Acknowledgment Packet Format

The following describes the format of the data portion of a Command Acknowledgment, whose stream id is STREAM_ID_CMD_ACK (see Section 3.3.3):

Copy of Command Pkt (64 bytes) .... This is a complete copy of the command packet buffer which contains the command being acknowledged. The entire packet buffer (64 bytes) is copied into the telemetry packet. All commands are this length or less.

Acknowledgment Code (2 bytes) ..... This is the acknowledgment code, passed by the caller of this module.

Acknowledgment Data (N bytes) ..... The remainder of the packet contains the acknowledgment's data, which was requested by the command. The length and meaning of this data is command specific.

3.2 Address Formats

3.2.1 Flat Address Format

A "flat" address consists of a linear byte position in memory. Addresses in this format refer to a unique memory location (ie. two different addresses never refer to the same memory location).

3.2.2 Segment:Offset Address Format

A "segment:offset" address consists of two parts, which evaluate to a 20-bit linear flat address. The "segment" portion refers to a 16-byte block of memory, and the "offset" portion is a linear offset from that position. For example, 0x0001:0x0003 refers to segment 1, offset 3. This evaluates to a linear address of (1 * 16) + 3, or 19 decimal, or 0x00000013 hex. This addressing format can "alias". Two different addresses can map to the same memory location. For example, 0x0001:0x0003 (0x0000013) and 0x0000:0x0013 (0x0000013) refer to the same flat address, and therefore, the same memory location.

3.3 Constants

3.3.1 STREAM_ID_STARTUP_MSG (DOWNLINK)

This value is written into a telemetry packets "*stream id*" field (see **Downlink Module Specification**, Section 3.1.1.3) to identify it as a startup message packet (see Section 3.1.1).

3.3.2 STREAM_ID_PANIC_MSG (DOWNLINK)

This value is written into a telemetry packets "*stream id*" field (see **Downlink Module Specification**, Section 3.1.1.3) to identify it as a panic message packet (see Section 3.1.2).

3.3.3 STREAM_ID_CMD_ACK (DOWNLINK)

This value is written into a telemetry packets "*stream id*" field (see **Downlink Module Specification**, Section 3.1.1.3) to identify it as a command acknowledgment packet (see Section 3.1.3).

3.3.4 DOWNLINK_PKT_PRIORITY_HIGH (DOWNLINK)

The Utility Module posts Startup Message and Command Acknowledgment Packets to the **Downlink Module** using this telemetry priority.

3.3.5 UTIL_BUFFER_POOL_SIZE (3096)

This is the size, in bytes, of the Utility Module's buffer pool. This pool is used to obtain telemetry packet buffers for startup messages, panic messages, and command acknowledgments.

3.3.6 UTIL_MAX_STATISTICS (64)

This specifies the number of software housekeeping statistics maintained by the Utility Module.

3.3.7 UTIL_MAX_ACKLEN (942)

This specifies the maximum number of bytes of acknowledgment data sent in a command acknowledgment packet (see Section 3.1.3). It is determined by the maximum packet size (FS_DOWNLINK_SIZE=1024), minus the size of the CCSDS Primary Header (6), the Secondary Header (8), the EDS Stream Id (2), the maximum command packet size (FS_UPLINK_SIZE=64) and the command acknowledgment code size (2).

3.3.8 MMAP_MAX_286_ADDR (20-bits, 0x0fffff)

This is the last valid address on a 20-bit address bus.

3.3.9 MSG_UTIL_STARTUP_NO_BUFFER

This message code is passed to UTILpanic() if UTILstartup_msg() fails to obtain a telemetry buffer from the pool for the startup message. The message should read something like:

"PANIC: -- UTILstartup_msg() --No Buffer"

3.3.10 MSG_UTIL_ACK_BAD_DATA_LEN

This message code is passed to UTILpanic() if UTILack() is passed an illegal data length for the acknowledgment's returned data. The message should read something like:

"PANIC: -- UTILack(%d) --Bad Data Length"
where %d represents the passed data length.

3.3.11 MSG_UTIL_LOG_BAD_ARGUMENT

This message code is passed to UTILpanic() if UTILlog_housekeeping() is passed an illegal statistic code. The message should read something like:

"PANIC: -- UTILlog_housekeeping(%d) --Bad Statistic Code"
where %d represents the passed code.

3.3.12 MSG_UTIL_FLATPTR_PTR_RANGE

This message code is passed to UTILpanic() if UTILflat2pointer() is passed an address which is not within the 80286's address space. The message should read something like:

"PANIC: -- UTILflat2pointer(%d) --Bad Address"
where %d represents the passed flat address.

4 Functions

4.1 UTILinit() - Initialize the Module

Usage

void UTILinit(void);

Description

UTILinit() initializes the module's internal data structures and tables, and sets up the module's pool of telemetry buffers. This function must be called before calling all other functions in this module which deal with software housekeeping or telemetry. These functions include UTILpanic(), UTILstartup_msg(), UTILack(), UTILlog_housekeeping(), and UTILhousekeeping_update().

Return Values

NONE - FUNCTION IS VOID.

Caveats

1) This function must be called before certain other functions in this module.

See Also

NONE.

4.2 UTILpanic() - Issue a Panic Message and Restart

Usage

void UTILpanic(UINT16 *msg_code*, UINT32 *arg)*;

Description

UTILpanic() issues a Panic message and then restarts the Flight Software. "*msg_code*" refers to a 16-bit alert message code and "*arg*" refers to a 32-bit numerical argument for the message. The ground will ignore the downlinked "*arg*" value if the message doesn't require a numerical argument.

UTILpanic() catches recursive calls to itself (by either the **Downlink Module** or **DMA Module**). If a panic is in progress when UTILpanic() is called, this function doesn't attempt to send another panic message, but instead just resets the EA, immediately.

Return Values

NONE - THIS FUNCTION SHOULD NEVER RETURN.

Caveats

1) UTILinit() must be called before using this function.

2) Be ready to reboot upon calling this function.

See Also

Downlink Module Specification

4.3 UTILstartup_msg() - Issue a Startup Message Packet

Usage

void UTILstartup_msg (UINT32 *pram_version*);

Description

UTILstartup_msg() constructs and posts a startup message packet. The passed *pram_version* is a copy of the 4 byte PRAM Version id, written into PRAM by the ground.

If UTILstartup_msg() fails to obtain a telemetry buffer, it assumes that the system has been corrupted, and issues a MSG_UTIL_STARTUP_NO_BUFFER panic message and resets the EA.

Return Values

NONE - FUNCTION IS VOID

Caveats

1) UTILinit() must have been called before calling this function.

See Also

Startup Module Specification, **Downlink Module Specification**

4.4 UTILack() - Issue a Command Acknowledgment

Usage

void UTILack (UINT8 **cmd*, UINT16 *ackcode*,
UINT8 **ackdata*, UINT16 *acklen*);

Description

UTILack() forms and posts a command acknowledgment telemetry packet to the ground. "*cmd*" points to the command buffer containing the command packet. This function always copies the entire buffer into the telemetry packet, regardless of the actual length of the packet. "*ackcode*" is the acknowledgment code for the command. "*ackdata*" points to the data to send with the acknowledgment, and "*acklen*" is the length of this data. UTILack() copies "*acklen*" of "*ackdata*" to the acknowledgement telemetry packet.

If UTILack() fails to obtain a telemetry buffer, it records the occurrence in software housekeeping and returns.

If "*acklen*" exceeds the remaining size of the telemetry packet, the UTILack() will issue a MSG_UTIL_ACK_BAD_DATA_LEN panic message and reset the EA.

Return Values

NONE - FUNCTION IS VOID.

Caveats

1) UTILinit() must have been called before calling this function.

See Also

Downlink Module Specification.

4.5 UTILlog_housekeeping() - Increment SW Statistic

Usage

void UTILlog_housekeeping (UINT8 *statistic*);

Description

UTILlog_housekeeping() increments the software housekeeping counter, identified by "*statistic*." If "*statistic*" is invalid, UTILlog_housekeeping() issues a MSG_UTIL_LOG_BAD_ARGUMENT panic message and resets the EA.

Return Values

NONE - FUNCTION IS VOID

Caveats

1) UTILinit() must have been called before calling this function.

2) A given software housekeeping counter will wrap if incremented more than 64K times between successive dumps of housekeeping data (see UTIL-housekeeping_update()).

See Also

Supervisor Thread Module Specification

4.6 UTILhousekeeping_update - Copy and Reset SW Statistics

Usage

void UTILhousekeeping_update (UINT16 **buf*);

Description

UTILhousekeeping_update() copies its internal array of software statistic counters (see UTILlog_housekeeping()) into the buffer, pointed to by "*buf*." It then resets all values in the internal array to zero. "*buf*" must be large enough to hold UTIL_MAX_STATISTICS 16-bit words.

Return Values

NONE - FUNCTION IS VOID.

Caveats

1) UTILinit() must have been called before calling this function.

See Also

Supervisor Thread Module Specification, UTILlog_housekeeping().

4.7 UTILcopy_buf() - Copy a buffer

Usage

void UTILcopy_buf(UINT8 **src*, UINT8 **dst,* UINT16 *count*);

Description

UTILcopy_buf() copies "*count*" bytes from the buffer pointed to by "*src*" to the buffer pointed to by "*dst"*. This function does not handle overlapping buffers.

Return Values

NONE - FUNCTION IS VOID.

Caveats

1) "*src*" and "*dst*" must be non-NULL.

2) The source and destination buffers must not overlap.

See Also

NONE.

4.8 UTILfill_buf() - Fill Buffer with a Value

Usage

void UTILfill_buf (UINT8 **buf*, UINT16 *len*, UINT8 *val*);

Description

UTILfill_buf() sets "*len*" bytes of the buffer, pointed to by "*buf*," with the value "*val*".

Return Values

NONE - FUNCTION IS VOID

Caveats

1) "*buf*" must not be NULL.

See Also

NONE.

4.9 UTILflat2pointer() - Convert Flat Address to CPU Pointer

Usage

UINT8 *UTILflat2pointer(UINT32 *flat_addr*);

Description

UTILflat2pointer() converts a flat-formatted address, "*flat_addr*", to a pointer in machine dependent format. In the case of the 80286, this is a segment:offset format, where this routine attempts to minimize the offset portion of the address (ie. segment is as close to the desired address as possible).

If the passed "*flat_addr*" is outside of the 80286's address space, UTILflat2pointer() issues a MSG_UTIL_FLATPTR_PTR_RANGE panic message, passing "*flat_addr*" as the message argument, and resets the EA.

Return Values

Memory pointer in machine format.

Caveats

1) "*flat_addr"* must be within the address space of the machine.

See Also

NONE.

4.10 UTILpointer2flat() - Convert CPU Address to Flat Address

Usage

UINT32 UTILpointer2flat(UINT8 **ptr*);

Description

UTILpointer2flat() converts the address, "*ptr*", from machine format to a flat-formatted address. In the case of the 80286, "*ptr*" is in segment:offset format.

Return Values

Unsigned, 32-bit, flat-formatted memory address.

Caveats

1) If "*ptr*" refers to memory outside of the 80286 physical address space, the returned 32-bit value will exceed the 80286 20-bit physical address space and cause a system panic if used with UTILflat2pointer().

See Also

NONE.

4.11 UTILsegment() - Get Segment from Segment:Offset Address

Usage

UINT16 UTILsegment(UINT8 **ptr*);

Description

UTILsegment() returns the segment portion of a segment:offset formatted address, specified by "*ptr*".

Return Values

Unsigned, 16-bit segment of the passed address.

Caveats

1) "*ptr*" must be a valid machine address.

See Also

NONE.

4.12 UTILoffset() - Get Offset from Segment:Offset Address

Usage

UINT16 UTILoffset(UINT8 **ptr*);

Description

UTILoffset() returns the offset portion of a segment:offset formatted address, specified by "*ptr*".

Return Values

Unsigned, 16-bit offset of the passed address.

Caveats

1) "*ptr*" must be a valid machine address.

See Also

NONE.

4.13 UTILadjust_address() - Minimize Offset Portion of Address

Usage

UINT8 *UTILadjust_address (UINT8 **ptr*);

Description

UTILadjust_address() minimizes the "offset" portion of the segment:offset formatted address, specified by "*ptr*" and returns the adjusted address. This function divides the "offset" portion by 16, and adds the integer result to the "segment" portion of the address. It then sets the "offset" to the remainder. Any two 80286 addresses which point to the same memory location are normalized to the same value using this function.

Return Values

Adjusted Address

Caveats

1) "*ptr*" must be a valid machine address.

See Also

NONE.

4.14 UTILvalid_ptr() - Verify that Pointer is a Physical Address

Usage

BOOL UTILvalid_ptr (UINT8 *ptr);

Description

UTILvalid_ptr() verifies that the memory location, specified by "*ptr*", is addressable by the 80286 (i.e. resolves to a "flat" address which can be represented in 20-bits or less).

Return Values

TRUE - "*ptr*" resolves to a valid 80286 address
FALSE - "*ptr*" resolves to an address not accessable by the 80286.

Caveats

NONE

See Also

NONE

4.15 UTILnop() - "No Operation" Function

Usage

void UTILnop (void);

Description

UTILnop() returns immediately after being called. This function will take at least 1.5 microseconds to execute (the time it takes to execute a return instruction on an 8MHz 80286), and will never take more than 20 microseconds.

Return Values

NONE - FUNCTION IS VOID.

Caveats

NONE.

See Also

NONE.

5 Detailed Design Considerations

5.1 Data Flow

This section describes the Level 1 data flow within the Utility Module.

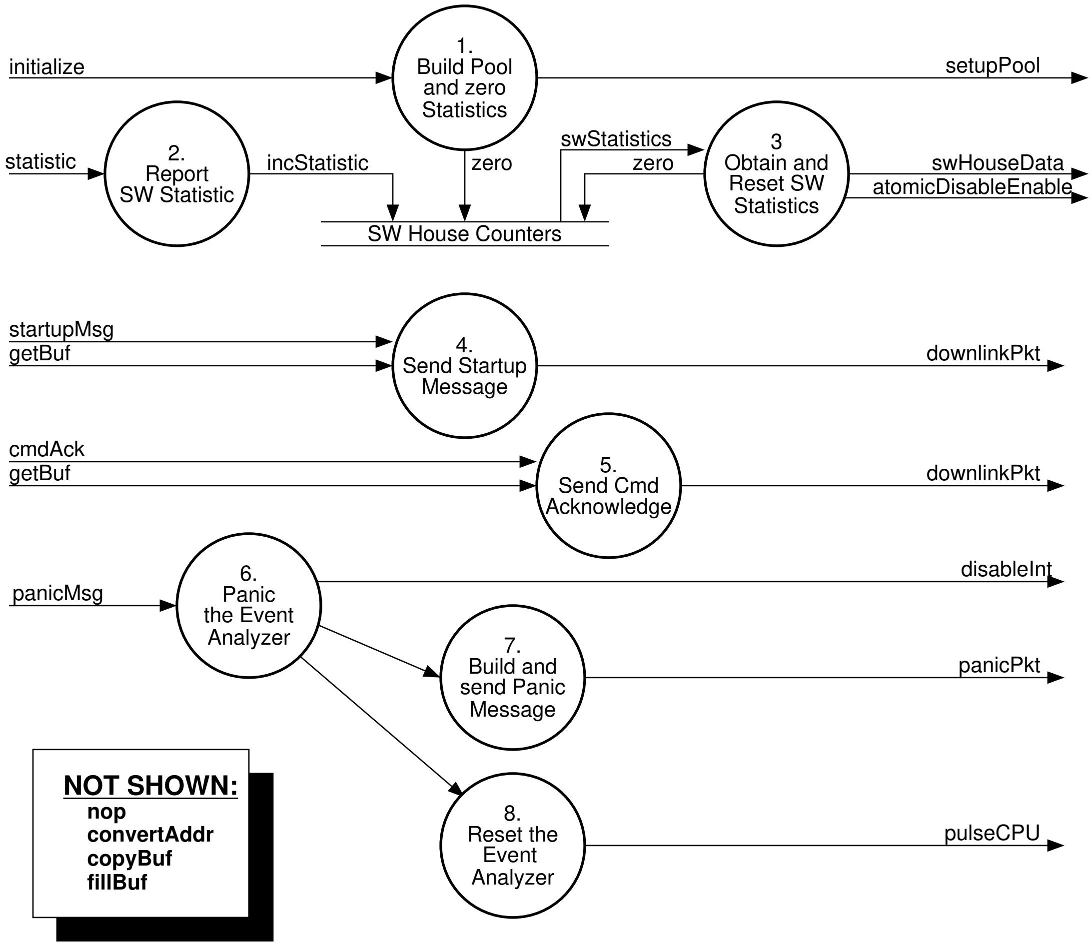

Figure 3 Utility Module Level 1 Data Flow

The following items, described in Section 2.3, are not shown in Figure 3; nop, convertAddr, copyBuf, and fillBuf. The operations on these elements are extremely simple, and are decoupled from the state of the Utility Module.

During initialization (bubble 1), the Utility module initializes and sets up a pool of telemetry buffers (setupPool) and zeros the software housekeeping statistics (zero).

Sometime during system initialization, a request is made to issue a startup message (startupMsg). The Utility Module (bubble 4) obtains a buffer from its pool of telemetry buffers (getBuf), forms the packet and posts the packet for transfer (dowlinkPkt).

Once running, other Flight Software Modules report the occurrence of various events (bubble 2). The Utility Module increments the corresponding counter (incStatistic) in the internal statistics array (SW House Counters). When a request is made to obtain the counters (bubble 3), the Utility Module copies the array (swStatistics) into the callers area (swHouseData) and zeros the internal array (zero).

During the main operation of the Flight Software, the threads receive commands and issue acknowledgments to the commands (cmdAck). The Utility Module (bubble 5) obtains a telemetry buffer from its pool (getBuf), forms the command acknowledgment packet, and posts the packet for transfer (downlinkPkt).

If any part of the Flight Software detects an error, it emits a panic message (panicMsg). The Utility Module (bubble 6) first disables interrupts (disableInt). It then tests to see if it is already attempting to send a panic message (not shown). If it isn't, it grabs a telemetry buffer from the pool (not shown), forms the panic message, and synchronously transfers the packet (panicPkt). In any case, the Utility Module then resets the EA by starting the Watchdog Timer (pulseCPU) and busy waits until the EA resets.

5.2 Control Considerations

5.2.1 Direction Bit for MOVS and STOS

When implementing UTILcopy_buf() and UTILfill_buf(), using the 80286 instructions, MOVS and STOS, the implementation must take care to ensure that the 80286 direction bit is cleared. This bit is contained in the CPU's Flags register, and is cleared using the CLD instruction. If this bit is accidentally set, then the address registers used during these instructions will decrement (instead of increment), and thus cause serious, but hard to detect, data corruptions.

5.2.2 Resetting the Hardware

The Utility Module's panic function resets the EA using the hardware Watchdog Timer. This ensures that all EA hardware is set to its default state, and that, upon restart, the Watchdog Timer is not running.

Another option, not used by the Utility Module, would be to just jump to the reset vector. This technique has two problems. First, the Watchdog Timer would still be running, and if the initialization procedure took too long, the EA would reset again. The second problem is that the hardware would be coming up in a different state each time the EA panicked. This makes testing and trouble-shooting difficult.

5.3 Module Structure

5.3.1 Initialization and Telemetry Function Structures

This section describes the structure of the initialization and telemetry functions provided by the Utility Module. Bolded boxes and text represent functions which are provided by modules other than the Utility Module.

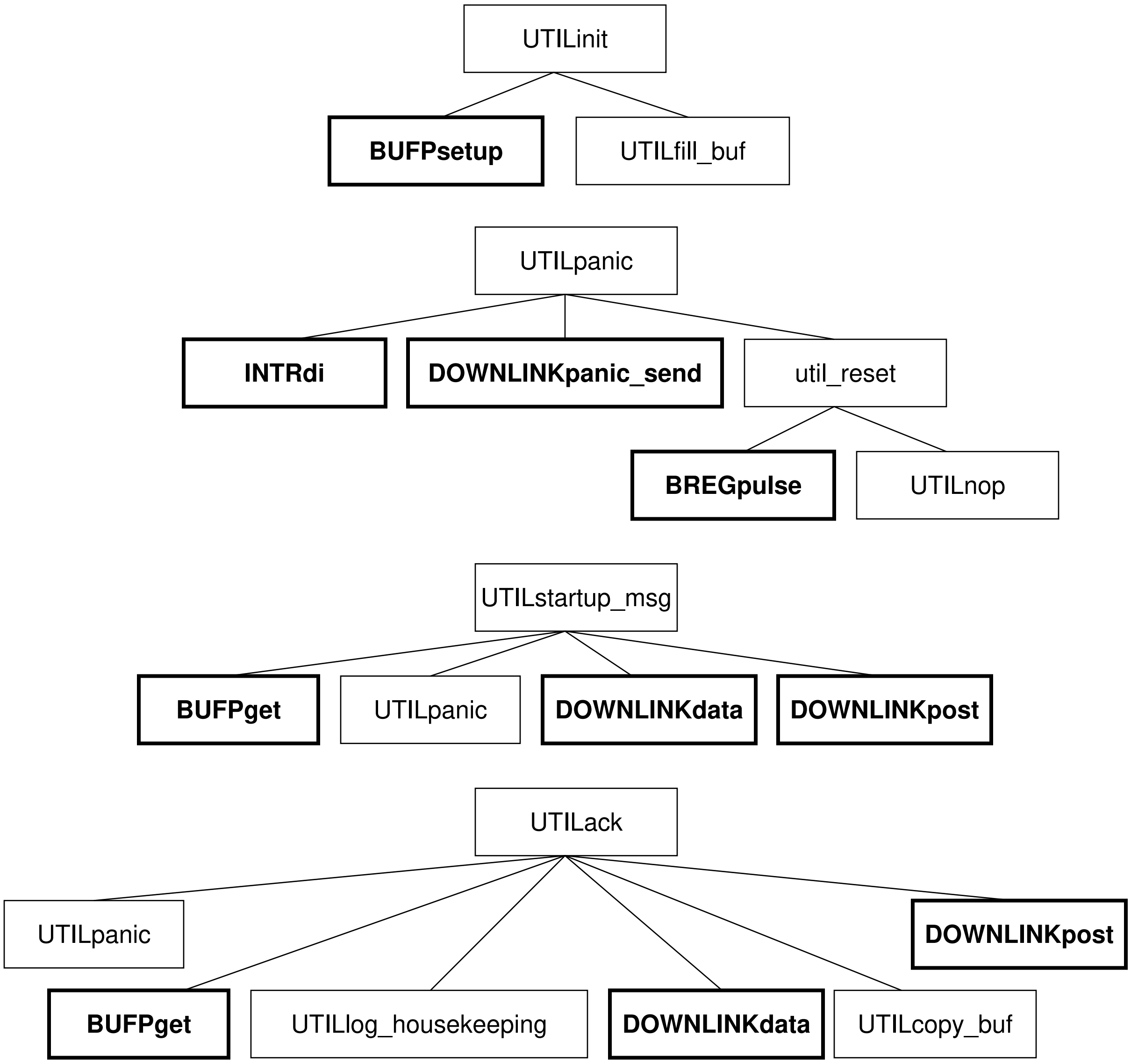

Figure 4 Initialization and Telemetry Function Structures

The following describes each of the functions shown in Figure 4. The details of each function's operation and arguments are described in the Utility Module's detailed design document.

UTILinit .... This function is called by the **Startup Module**, during system initialization. UTILinit deasserts the "panic in progress" flag. It then uses the **Buffer Pool Module** (BUFPsetup) to initialize a pool of telemetry buffers. It then calls UTILfill_buf to zero the contents of the Software Housekeeping Statistics array.

UTILpanic .... This function is called by Flight Software Modules to issue a panic message and reset the EA. UTILpanic starts by disabling interrupts, using the **Interrupt Manager Module** (INTRdi). It then tests the state of the "panic in progress" flag. If the flag not asserted, UTILpanic asserts the flag, directly grabs the first telemetry buffer in the buffer pool, and calls the **Downlink Module** (DOWNLINKpanic_send) to build and transfer the panic message. UTILpanic then resets the EA, by calling util_reset.

util_reset .... This function is called by UTILpanic to reset the EA. It starts or resets the Watchdog Timer, using the **Back End Registers Module** (BREGpulse). It then enters an infinite loop, calling UTILnop for each iteration. When the Watchdog Timer expires, it will reset the EA.

UTILstartup_msg.............................. This function is called by the **Startup Module** to build and transfer a startup message. UTILstartup_msg obtains a buffer from the pool, using the **Buffer Pool Module** (BUFPget). If a buffer is not available, UTILstartup_msg issues a panic message and resets the EA, by calling UTILpanic. If a buffer is available, UTILstartup_msg obtains a pointer to the data portion of the buffer, using the **Downlink Module** (DOWNLINKdata), and copies the passed PRAM Version Id into the buffer. UTILstartup_msg then calls the **Downlink Module** (DOWNLINKpost) to post the packet to be transferred out of the EA.

UTILack.. This function is called by the Flight Software's threads to build and transfer command acknowledgment packets. UTILack first tests the validity of the passed arguments. If a problem is detected, UTILack issues a panic message and resets the EA, by calling UTILpanic. If the arguments are valid, UTILack obtains a buffer, using the **Buffer Pool Module** (BUFPget). If a buffer isn't available, UTILack logs the occurrence to software housekeeping, by calling UTILlog_housekeeping, and then returns. If a buffer is available, UTILack uses the **Downlink Module** (DOWNLINKdata) to obtain a pointer to the data portion of the buffer. It then copies the contents of the command packet buffer into the telemetry buffer, using UTILcopy_buf. It then writes the acknowledgment code. UTILack then copies any acknowledgment data into the telemetry buffer, again using UTILcopy_buf. Once the telemetry packet has been built, UTILack posts the packet for transfer, using the **Downlink Module** (DOWNLINKpost).

5.3.2 Housekeeping, Buffer, and Address Function Structures

This section describes the structure of the housekeeping, buffer utility, and address manipulation functions provided by the Utility Module. Bolded boxes and text represent functions which are provided by modules other than the Utility Module.

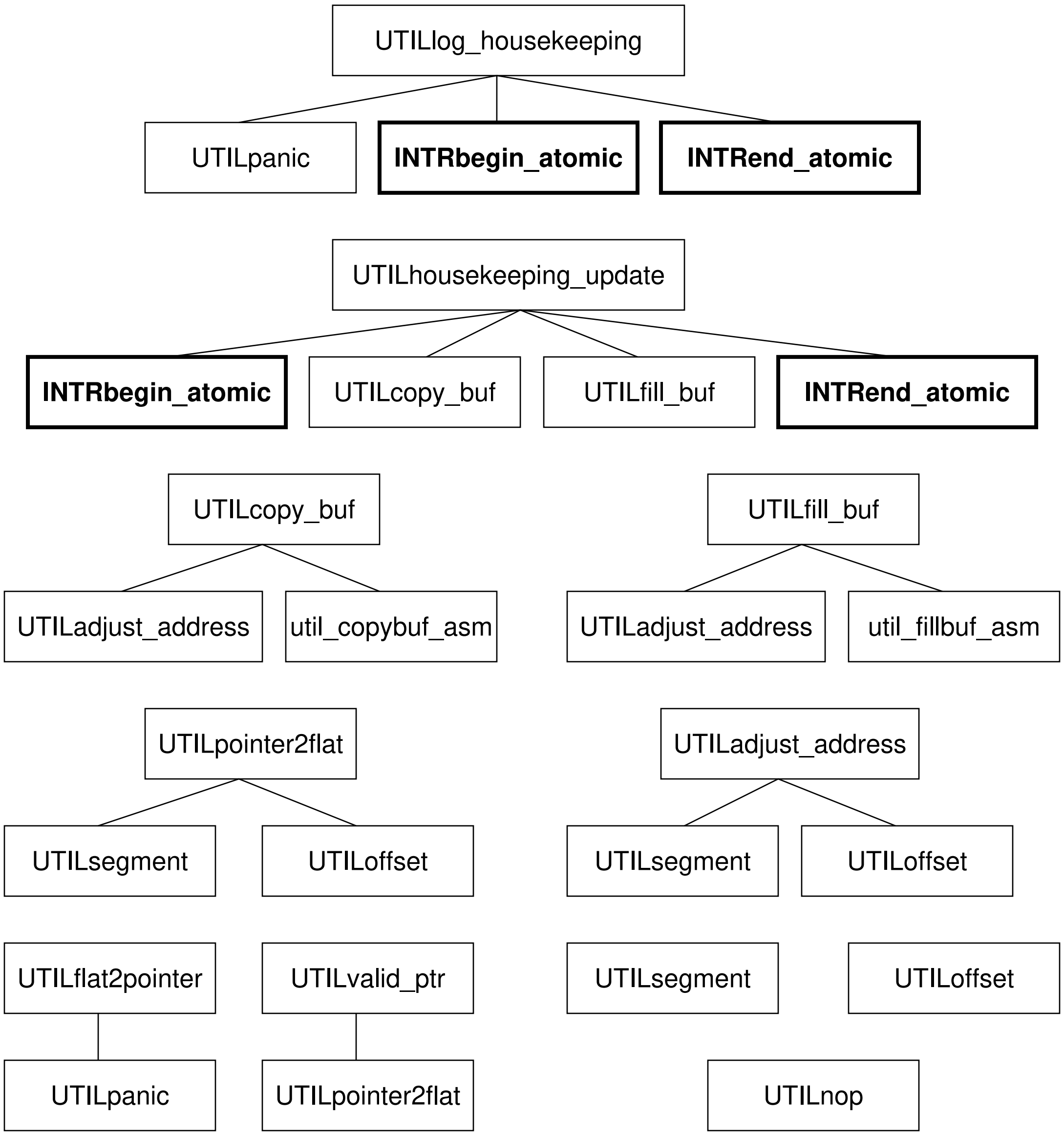

Figure 5 Housekeeping, Buffer and Address Function Structures

The following describes each of the functions shown in Figure 5. The details of each function's operation and arguments are described in the Utility Module's detailed design document.

UTILlog_housekeeping .... This function is called by the Flight Software Modules to log the occurrence of some software "event." UTIL-log_housekeeping tests the passed statistic id, and if it's invalid, issues a panic message and resets the EA, by calling UTILpanic. If the id is valid, UTILlog_housekeeping increments the counter associated with the id. Since the operations involves reading the counter, incrementing the read value and writing the result, UTILog_housekeeping() prevents interrupts while performing the operation, using the **Interrupt Manager Module** (INTRbegin_atomic, INTRend_atomic).

UTILhousekeeping_update .... This function is used by the **Supervisor Thread** to obtain and reset the collection of housekeeping statistics. UTIL-housekeeping_update performs this operation with interrupts disabled, using the **Interrupt Manager Module** (INTRbegin_atomic, INTRend_atomic). It copies the data to the passed buffer using UTILcopy_buf and then zeros the internal set of counts, using UTILfill_buf.

UTILcopy_buf .... This function is called by the Flight Software Modules to perform a fast memory-to-memory copy. This function first calls UTILadjust_address for both the source and destination addresses. It then tests the length of the copy. If the length is odd, it copies the first byte of the set. It then uses util_copybuf_asm to perform a fast word-by-word copy using the 80286 MOVS instruction to quickly copy the remaining data.

UTILfill_buf .... This function is called by the Flight Software Modules to quickly fill a region of memory. This function first calls UTILadjust_address for the destination address. It then tests the length of the fill. If the length is odd, it fills the first byte of the set. It then uses util_fillbuf_asm to perform a fast word-by-word fill using the 80286 STOS instruction to quickly fill the remaining data.

UTILflat2pointer .... This function is called by the Flight Software Modules to convert an address in flat format to a native address (segment:offset). If the address is illegal (i.e. outside of the 80286 address space), UTILflat2pointer issues a panic message and resets the EA, by calling UTILpanic.

UTILpointer2flat .... This function is called by the Flight Software Modules to convert an native address (segment:offset) to flat format. UTILpointer2flat uses UTILsegment and UTILoffset to obtain the segment and offset portions of the passed address respectively.

UTILsegment .... This function is used by the Flight Software Modules to obtain the segment portion of a native address.

UTILoffset .... This function is used by the Flight Software Modules to obtain the offset portion of a native address.

UTILadjust_address .... This function is used by the Flight Software Modules to minimize the offset portion of a native address. UTILadjust_address uses UTILsegment and UTILoffset to obtain the segment and offset portions of the passed address respectively.

UTILvalid_ptr .... This function is used by the Flight Software Modules to determine if a given pointer, in segment:offset format, refers to a location addressable by the 80286. It uses UTILpointer2flat() to convert the passed address to flat format, and then compares the value with the maximum flat address supported by the 80286's 20-bit address bus.

UTILnop .... This function is used by the Flight Software Modules to waste a little time. UTILnop just returns.

6 Requirements Traceability

6.1 Science Requirements

The Utility Module does not directly address any Science Requirements. It does, however, indirectly affect the throughput of the EA. UTILcopy_buf and UTILfill_buf are used extensively when copying and zeroing the DSP's Histogram Buffer, and when building telemetry packets. Since the time it takes to perform these operations governs the frequency at which partitions of data can be produced and posted for transfer, the performance of these functions must be evaluated and, if necessary, further optimized to meet any science throughput requirements.

7 Testing Considerations

7.1 Unit Testing

To perform unit tests on this module, the following equipment is required:

a. An Event Analyzer equipped with the Intermetrics ROM Monitor

b. A workstation running the Intermetrics debugger

c. An RS-232 serial cable between the ROM Monitor's serial port and the workstation

d. A logic analyzer

To unit test this module, the tester must link and locate this module's functions to some EA RAM memory location. Using a resident monitor/debugger, the tester should first verify the operation of all functions internal to this module, and then verify the interactions between all functions internal to the module. These tests should verify all branches, calls, and legal and illegal parameters. The tester should then verify all calls made to functions outside of this module, and ensure that the passed arguments are always within the limits specified by the corresponding module specifications. The tester should also verify that the caller always handles values returned by these functions in an appropriate manner.

In addition to these tests, the tester should ensure that all telemetry packets produced by this module are properly formed, that all address manipulations are correct, that system panics are handled in a reliable fashion, that recursive calls to UTILpanic() are handled correctly, and that modifications to the software housekeeping data is properly protected against unexpected modifications during interrupt handling.

7.2 Integration Testing

For integration tests, the following equipment is required:

a. An Event Analyzer

b. A populated ROM Repository board

c. A logic analyzer

d. Hardware which can:

    1. Receive telemetry packets via the RS422 Interface (S/C Simulator)

The integration tests should verify the operation of the Utility Module in conjunction with the actual hardware and other FS modules. These tests should include:

a. Verify that panic messages are properly formed and transferred out of an EA.

b. Verify that calls to UTILpanic always result in an Event Analyzer reset.

c. Verify that the startup message is properly formed and transferred.

d. Verify that command acknowledgment packets are properly formed and transferred.

e. Measure buffer copy times, as performed by UTILcopy_buf. These measurements assist in determining the overall system throughput.

f. Measure buffer fill times, as performed by UTILfill_buf. These measurements assist in determining the overall system throughput.

g. Measure the time during which interrupts are disabled while copying and zeroing the software housekeeping data (UTILhousekeeping_update). This time must be less than 0.5ms.

h. Measure the execution time of UTILnop. This is to provide general information for use by those modules which call this function between hardware register manipulations.

Appendix A

A.1 Items Under Review

This section lists items and issues which need to be resolved:

Document Reference	Section	Resolution Expected by
NONE		