Partition Map Module Specification

James Francis

MIT 64-30111.0401

Rev. D

November 10, 1994

Contents

Figures

1 Introduction

1.1 Scope

This document describes the Partition Map Software Module (PARTMAP). This Module consists of a set of functions used by the Science Modes to pack partition data into telemetry buffers.

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, Ground Software Engineering and Software Quality Assurance.

1.2 References

MIT 64-30102 Rev. E....................... "Flight Software Global Design".
MIT 64-30112.0601 Rev. D .... "Science Thread Module Specification" (SCIENCE).
MIT 64-30110.2001 Rev. C .... "Utility Module Specification" (UTIL).

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 Partition Map Module does not appear directly in this data flow, but has a scope which directly affects items indicated by the shaded areas:

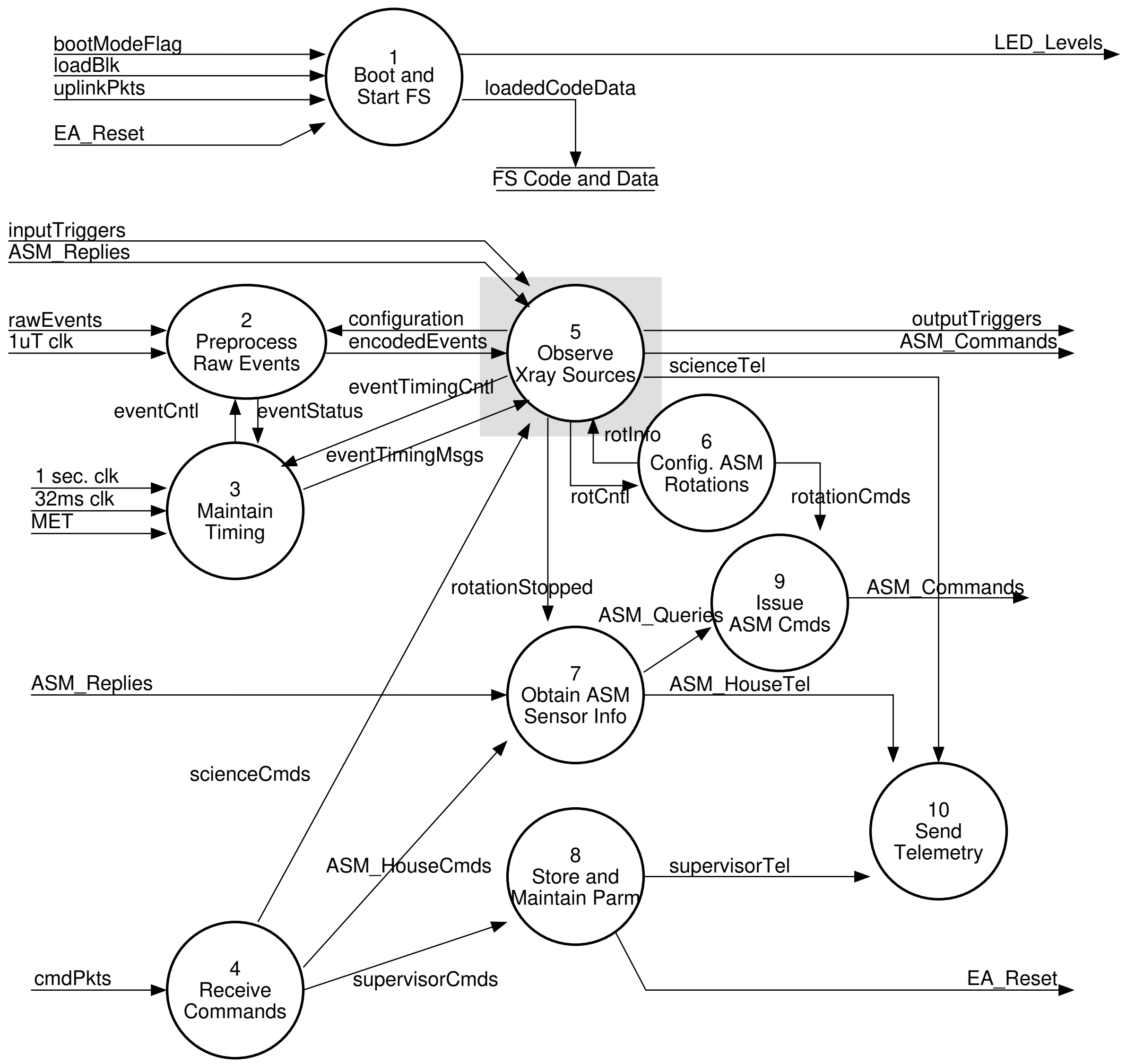

Figure 1 Module Perspective

1.5 Document Overview

Sections 1 and 2 provide an overview of the Partition Map 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 Partition Map Module. Section 4 describes the functions provided to other modules by the Partition Map 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 Partition Map Module is to pack partition data into telemetry buffers according to ground specified parameters.

2.2 Services

The Partition Map Module provides the following services:

1) Provide Length of Partition Map .. The Partition Map Module is responsible for providing the length, in bytes, of a Partition Map to the rest of the Flight Software.
2) Check Partition Map .... The Partition Map Module provides a service to test the consistency of the Partition Map at the beginning and end of an observation. At the start of an observation, the Partition Map Module computes an internal code (i.e. a exclusive-or code) based on the specified Partition Map. At the end of the observation, the Partition Map Module recomputes the code and compares it with the original. The Flight Software uses this feature to detect whether or not the Partition Map was corrupted during the observation.
3) Pack Partition Data .... The Partition Map Module is responsible for packing partition data into telemetry buffers, using the Partition Map specified at the start of the observation.

2.3 Module Context

The following illustrates the Partition Map Module relationships with the rest of the Flight Software residing in an EA:

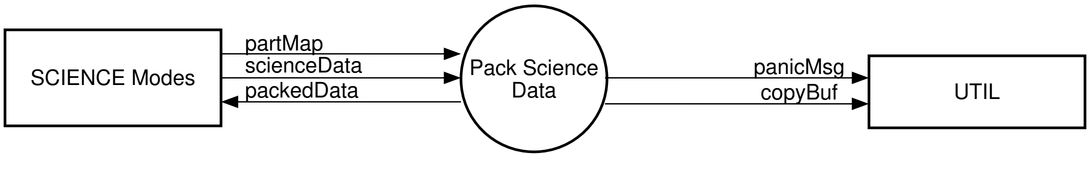

Figure 2 Partition Map Module Context Diagram

The Science Modes (see **Science Thread Module Specification**) use the Partition Map Module to pack science data (scienceData) into telemetry buffers (packedData) according to a ground specified Partition Map (partMap). The Partition Map Module uses the **Utility Module** to perform large word-by-word copies (copyBuf). If the Partition Map Module detects an error, either in the specified Partition Map, or while packing science data, it issues a panic message (panicMsg) which resets the EA using the **Utility Module**.

2.4 Theory of Operation

The Partition Map Module provides the Science Modes with the mechanisms needed to pack science data into telemetry buffers, using a set of ground specified parameters (i.e. a Partition Map). This module packs sections of a partition of science data, representing each 16-bit science data word using 16-bit, 8-bit, or 4-bit elements. This module can also skip over regions of the partition, discarding data words. The Partition Map specifies which element size to use on each region of the partition, the length of each region, and which regions to skip.

At the beginning of an observation, the running science mode registers the Partition Map with the Partition Map Module (see Section 3.1). Once the observation starts, the science mode obtains partitions of science data from the DSP. For each partition to be sent, the science mode first calls the Partition Map Module to re-initialize its internal partition state variables. The science mode then obtains a telemetry buffer and calls the Partition Map Module to pack part of the partition's data into the buffer. The science mode continues to obtain telemetry buffers and pack the data until the Partition Map Module indicates that the partition is complete. At this point, the science mode can start the next partition.

The following illustrates the overall usage of the Partition Map Module from the perspective of the **Science Thread**:

1) Register the Partition Map with the Partition Map Module during the Observation Setup stage.

2) Obtain the length of the Partition Map. If the returned length is not zero, dump the map during Observation Parameter Dump stage.

3) Start running the observation.

4) Obtain a partition of data from the DSP.

5) Tell the Partition Map Module to start a new partition, passing the starting address of the obtained partition data and the number of 16-bit words in the partition.

6) Tell the Partition Map Module to pack partition data into telemetry buffers until the Partition Map Module indicates that the partition is complete.

7) As each buffer is packed, post the packed telemetry buffer for transfer to the SDS.

8) Repeat from Step 4 until the observation completes.

9) Tell the Partition Map Module to re-check the Partition Map, and include the result in the Observation Coda.

2.5 Data Structures and Constants Summary

The Partition Map Module defines the format of the Partition Map, used by the ground to specify how to pack one partition of data. It also assumes that all partitions of science data consist of a series of 16-bit data elements.

2.6 Interface Summary

The Partition Map Module uses the services provided by the following module:

UTIL Module .... Issues panic messages and resets the EA.
Performs fast memory-to-memory copies

2.7 Function List Summary

PARTMAPsetup()	Register Partition Map with the module.
PARTMAPlength()	Obtain length of the Partition Map.
PARTMAPcheck()	Test for corruption of the Partition Map.
PARTMAPnew_partition()	Reset internal partition state variables.
PARTMAPpack_data()	Pack Science Data using decoded map.

3 Data Structures and Constants

3.1 Partition Map Format

The ground specified Partition Map consists of an array of 16-bit Partition Map Entries which tell the Partition Map Module how to pack regions of a partition of science data. Each entry consists of two fields, a "*pack code*" and "*word count*." The "*word count*" field specifies the number of packed 16-bit output words to produce, or how many science data words to ignore, and the "*pack code*" field specifies how many bits to take from each partition word to build the packed output word, or indicates that "*word count*" number of science words should be ignored.

The ground specified Partition Map has the following format:

[Entry Count] .... Number of entries in the map (16-bit word, little-endian)
[Entry ...] .... "Entry Count" Partition Map Entries (each entry is a 16-bit word)

where each entry consists of:
[c,c,w,w,w,w,w,w,w,w,w,w,w,w,w,w] .... Entry (16-bit word)
w .... 14-bits of "*word count*"
c .... Pack Code (2-bits) where:
PARTMAP_PACK_CODE_FULL .... (00). Don't mask partition words.
PARTMAP_PACK_CODE_HALF .... (01). Mask off top 8-bits (use 8-bits).
PARTMAP_PACK_CODE_QUARTER .. (10). Mask off top 12-bits (use 4-bits).
PARTMAP_PACK_CODE_IGNORE...... (11). Skip partition words.

For example, consider a partition consisting of 32,768 words (32K), where the first 2 words of the partition are sent using full 16-bit words, the next 4 words are masked to 4-bit values and packed into 1 16-bit output word, the next 20 words are ignored, the next 32,740 words are masked to 4-bit words and are packed into 8185 16-bit output words, and the last 2 words are ignored. The Partition Map for this scenario is as follows:

Entry Count = 5 .... The number of entries is 5
PARTMAP_PACK_CODE_FULL, 2 .... 2 Output words from 2 science data words.
PARTMAP_PACK_CODE_QUARTER, 1 .... 1 Output word from 4 science data words.
PARTMAP_PACK_CODE_IGNORE, 20.............. No output words, skip 20 science data words.
PARTMAP_PACK_CODE_QUARTER, 8185 ...... 8185 Output words, from 32,740 science

	data words.
PARTMAP_PACK_CODE_IGNORE, 2	No output words, skip 2 science data words.
	TOTAL: 8188 Output words, from 32,768 science data words. Partition Map Size = 12 bytes (6 words)

3.2 Constants

3.2.1 PARTMAP_PACK_COUNT_MASK (constant value 0x03fff)

This is the mask used to extract the "*word count*" field from a Partition Map Entry (see Section 3.1). The "*word count*" is obtained by "and"ing the Partition Entry with this mask. The "*pack code*" (see Sections 3.2.3, 3.2.4, 3.2.5, 3.2.6) is obtained by "and"ing the entry with the inverse of this mask.

3.2.2 PARTMAP_PACK_CODE_MASK (constant value 0x0c000)

This is the inverse of the PARTMAP_PACK_COUNT_MASK (see Section 3.2.1), and is used to extract the "*pack code*" field from a Partition Map Entry (see Section 3.1). The "*pack code*" (see Sections 3.2.3, 3.2.4, 3.2.5, 3.2.6) is obtained by "and"ing the entry with this mask.

3.2.3 PARTMAP_PACK_CODE_FULL (constant value 0)

This is the "*packing code*" value which causes a series of science data words to be copied directly into a telemetry buffer.

3.2.4 PARTMAP_PACK_CODE_HALF (constant value 0x04000)

This is the "*packing code*" value which causes a series of science data words to be masked to 8-bit values and then packed into a telemetry buffer.

3.2.5 PARTMAP_PACK_CODE_QUARTER (constant value 0x08000)

This is the "*packing code*" value which causes a series of science data words to be masked to 4-bit values and then packed into a telemetry buffer.

3.2.6 PARTMAP_PACK_CODE_IGNORE (constant value 0x0c000)

This is the "*packing code*" value which causes a series of science data words to be ignored (i.e. not copied into a telemetry buffer).

3.2.7 MSG_PARTMAP_SETUP_NULL_MAP

This message code is passed to UTILpanic() if PARTMAPsetup() is passed a NULL pointer to a Partition Map. This message should read something like:

"PANIC: --PARTMAPsetup() --NULL pointer to Partition Map"

3.2.8 MSG_PARTMAP_SETUP_INTERNAL_ERROR

This message code is passed to UTILpanic() if PARTMAPsetup() encounters an internal error where a pack code, extracted from one of the entries, was misinterpreted. This message should read something like:

"PANIC: --PARTMAPsetup(%d) --Internal Error - Pack Code Misinterpreted"
where %d represents the address of the partition map passed to PARTMAPsetup().

3.2.9 MSG_PARTMAP_NEW_PARTITION_NO_MAP

This message code is passed to UTILpanic() if PARTMAPnew_partition() is called before a Partition Map has been registered via PARTMAPsetup(). This message should read something like:

"PANIC: --PARTMAPnew_partition() --No Partition Map"

3.2.10 MSG_PARTMAP_NEW_PARTITION_TOO_LONG

This message code is passed to UTILpanic() if PARTMAPnew_partition() is passed a partition length which exceeds the length computed from the Partition Map. The message should read something like:

"PANIC: -- PARTMAPnew_partition(%d) --Partition too long"
where %d represents the passed partition length.

3.2.11 MSG_PARTMAP_PACK_NO_MAP

This message code is passed to UTILpanic() if PARTMAPpack_data() is called before a Partition Map has been registered via PARTMAPsetup(). This message should read something like:

"PANIC: --PARTMAPpack_data() --No Partition Map"

3.2.12 MSG_PARTMAP_PACK_INTERNAL_ERROR

This message code is passed to UTILpanic() if PARTMAPpack_data() encounters an internal error where a pack code, extracted from one of the entries, is misinterpreted. This message should read something like:

"PANIC: --PARTMAPpack_data(%d) --Internal Error - Pack Code Misinterpreted"
where %d represents the address of the installed partition map (via PARTMAPsetup()).

3.2.13 MSG_PARTMAP_PACK_DATA_PARTMAP_TOO_SHORT

This message code is passed to UTILpanic() if PARTMAPpack_data() ran off the end of the Partition Map while ignoring data in the partition. This indicates that the partition was longer than anticipated by the Partition Map. This can be due to a corruption of the map or one of the internal partition state variables. This message should read something like:

"PANIC: -- PARTMAPpack_data() --Partition Map too short"

4 Functions

4.1 PARTMAPsetup() - Register a Partition Map

Usage

void PARTMAPsetup (UINT16 **partmap*);

Description

PARTMAPsetup() registers a Partition Map, pointed to by "*partmap*", with the Partition Map Module. PARTMAPsetup() makes a local copy of the pointer "*partmap*" for use while packing data, computes and stores an internal sanity check code from the map, and computes and stores the maximum length of a partition based on the passed map.

If "*partmap*" is NULL, PARTMAPsetup() will issue a MSG_PARTMAP_SETUP_NULL_MAP panic message and reset the EA, via the **Utility Module** function UTILpanic().

If an internal error occurs, such that a pack code, extracted from the passed partition map, appears to be invalid, PARTMAPsetup() will issue a MSG_PARTMAP_SETUP_INTERNAL_ERROR panic message, passing "*partmap"* as the message argument, and reset the EA via the **Utility Module** function, UTILpanic().

Return Values

NONE - FUNCTION IS VOID.

Caveats

1) This function must be called before calling PARTMAPnew_partition(), and PARTMAPpack_data().

See Also

PARTMAPnew_partition(), PARTMAPpack_data(), **Utility Module Specification**

4.2 PARTMAPlength() - Get Length of Partition Map

Usage

UINT16 PARTMAPlength (UINT16 **partmap*);

Description

PARTMAPlength() returns the length, in bytes, of the Partition Map pointed to by "*partmap*".

If "*partmap*" is NULL, this function will return a length of zero bytes. If "*partmap*" is not NULL, PARTMAPlength() retrieves the length of the map from on the contents of the referenced Partition Map.

Return Values

Number of bytes in the currently registered Partition Map, or zero if "*partmap*" is NULL.

Caveats

NONE.

See Also

Utility Module Specification

4.3 PARTMAPnew_partition() - Reset Partition State Variables

Usage

void PARTMAPnew_partition (UINT16 **buf_addr*, UINT16 *length*);

Description

PARTMAPnew_partition() causes the Partition Map Module to initialize its internal partition specific state variables. "*buf_addr*" points to the first 16-bit science data word in the partition buffer. "*length*" is the number of 16-bit science data words in the partition. "*buf_addr*" and "*length*" are retained by the Partition Map Module for use by PARTMAPpack_data(), which uses these parameters when packing the partition data.

If a Partition Map has not been registered via PARTMAPsetup(), PARTMAPnew_partition() will issue a MSG_PARTMAP_NEW_PARTITION_NO_MAP panic message and reset the EA using the **Utility Module** function UTILpanic().

If the length of the partition, "*length*", exceeds the anticipated length computed from the Partition Map, PARTMAPnew_partition() issues a MSG_PARTMAP_NEW_PARTITION_TOO_LONG panic message, passing "*length*" as the message argument, using the **Utility Module** function UTILpanic().

Return Values

NONE - FUNCTION IS VOID.

Caveats

1) This function must be called before calling PARTMAPpack_data() for each partition to be packed.

2) PARTITIONsetup() must have been called before calling this function.

See Also

PARTMAPpack_data(), PARTMAPsetup(),

Utility Module Specification

4.4 PARTMAPpack_data() - Pack Partition Data into a Buffer

Usage

UINT16 PARTMAPpack_data (UINT16 **buf*, UINT16 *buf_len*);

Description

PARTMAPpack_data() packs a portion of the partition's science data into the buffer, pointed to by "*buf*." "*buf_len*" specifies the number of 16-bit output words that can be stored in "*buf*."

Each invocation of PARTMAPpack_data() packs a portion of the partition into "*buf*" and updates the internal state variables to point to the next part of the Partition Map to use and the next section of the partition to pack. Normally, the output buffer, "*buf*", is completely filled, and PARTMAPpack_data() returns "*buf_len*." If the last section of the partition doesn't completely fill "*buf*", the PARTMAPpack_data() will return the number of words actually placed into the buffer. Subsequent calls will return zero until PARTMAPnew_partition() is called to start a new partition. If the last part of the partition completely fills "*buf*", then PARTMAPpack_data() will return "*buf_len*" and the next and subsequent calls to PARTMAPpack_data() will return zero.

If an error is encountered while packing the data, PARTMAPpack_data(), or one of the internal functions that it calls, will use the **Utility Module** function UTILpanic() to issue one of the following panic messages:

MSG_PARTMAP_PACK_DATA_NO_MAP (see Section 3.2.11)
MSG_PARTMAP_PACK_DATA_INTERNAL_ERROR (see Section 3.2.12)
MSG_PARTMAP_PACK_DATA_PARTMAP_TOO_SHORT (see Section 3.2.13)

Return Values

The number of words stored in "*buf*."
If zero, then a previous call to PARTMAPpack_data() had completed the partition.
If less than "*buf_len*", then this invocation completed the partition.

Caveats

1) PARTITIONnew_partition() must be called before the first call to this function for each new partition.

See Also

PARTMAPnew_partition(), PARTMAPsetup(), **Utility Module Specification**

4.5 PARTMAPcheck() - Check Integrity of Partition Map

Usage

BOOL PARTMAPcheck (void);

Description

PARTMAPcheck() recomputes its internal code, using the Partition Map registered via PARTMAPsetup(), and compares the result against the original code computed by PARTMAPsetup(). If the codes are identical, PARTMAPcheck() assumes that the map hasn't been modified and returns TRUE. If they differ, it returns FALSE, indicating that the map has been corrupted.

If a Partition Map has not been registered via PARTMAPsetup(), PARTMAPcheck() will return FALSE.

Return Values

TRUE - The Partition Map was not corrupted during the observation.

FALSE - The Partition Map changed during the observation, or no map was installed.

Caveats

NONE

See Also

PARTMAPsetup(), **Utility Module Specification**

5 Detailed Design Considerations

5.1 Data Flow

The following diagram illustrates the level 1 data flow within the Partition Map Module:

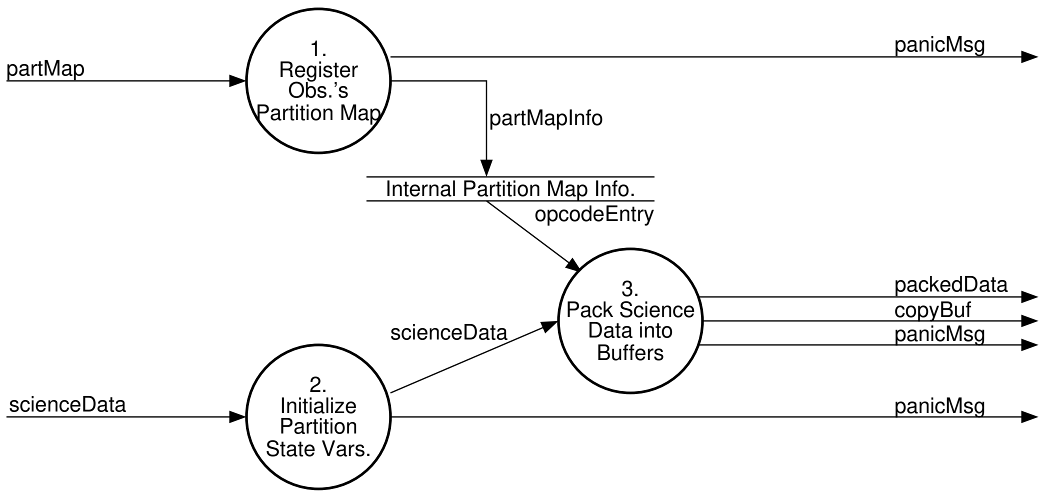

Figure 3 Partition Map Level 1 Data Flow

Before the start of an observation, the Partition Map Module is called to register a Partition Map (bubble 1). At this point the Partition Map Module computes the maximum length of a partition and an integrity check code, based on the passed map. It then stores the computed length, code, and the pointer to the table (partMapInfo) in its internal state variables. During the observation, it packs science data (scienceData) a partition at a time into caller provided telemetry buffers (not shown). At the start of each partition, the Partition Map Module initializes some state variables for use while packing the data (bubble 2). It then proceeds to pack the partition data (bubble 3) into the supplied telemetry buffers (packedData). If needed, the Partition Map Module may use the **Utility Module** to perform fast data copies (copyBuf).

If the Partition Map Module detects an error, it issues a panic message and resets the EA (panicMsg).

5.2 Control Considerations

5.2.1 Partition Map Integrity Check

The Partition Map Module computes an exclusive-or (xor) value for its internal Partition Map integrity code. The integrity code is computed by bit-wise xor'ing the first word in the Partition Map with all one's. The result is then xor'd with the subsequent word in the map. This continues until all of the words in the Partition Map have been included. The result is then used as the integrity code. This method handles any single bit corruptions of the table, or outright corruption of the table.

This form of checking is required because under some scenarios, corruption of the Partition Map may be difficult to detect. For example, consider Event Encoded Mode. Here the actual size of the partition varies, depending on the number of events recorded. If the Partition Map was corrupted in such a way that the map didn't cover the maximum size of the partition, and the partition was filled with event words, a panic would result. However, if the Partition Map was corrupted such that the map could handle MORE than the maximum size of a partition, the ground wouldn't get any indication from the EA, other than that the Event Encoded data might look strange.

Since the expected size of a given Partition Map is small (around 100 bytes), the probability of the Partition Map being directly corrupted by a Single-Event upset is extremely low (about 1 upset/EA in 2 years).

5.2.2 Internal Partition State Variables

The Partition Map Module uses a set of internal state variables to maintain the state of the module across the many calls to PARTMAPnew_partition() and PARTMAPpack_data(). These variables include the following information:

Partition Map Address .... This is the address of the Partition Map, as passed to PARTMAPsetup().
Maximum Partition Length .... This is the maximum size of a partition that can be handled by the registered Partition Map.
Partition Map Integrity Code .... This is the xor integrity code value computed when the map was first registered via PARTMAPsetup().
Partition Address .... This is the address of the current partition, as passed to PARTMAPnew_partition().
Partition Length .... This is the number of 16-bit words in the current partition, as passed to PARTMAPnew_partition().
Partition Index .... This is the index of the next word to pack in the current partition. This field is initialized to zero whenever PARTMAPnew_partition() is called, and is updated during each call to PARTMAPpack_data().
Partition Map Index .... This is an index into the Partition Map which indicates the

current packing operation being performed. This index is reset to zero upon each call to PARTMAPnew_partition() and is updated as PARTMAPpack_data() packs the partition.

Operation Count .... This indicates the number of words that have yet to be produced from the current packing operation. Upon each call to PARTMAPnew_partition, this count is initialized to the count of the first operation and is updated as PARTMAPpack_data() packs the partition data.

5.3 Module Structure

This section describes the structure of the Partition Map Module functions. The following diagram illustrates this structure, where bolded boxes and text indicate functions provided by modules other than the Partition Map Module.

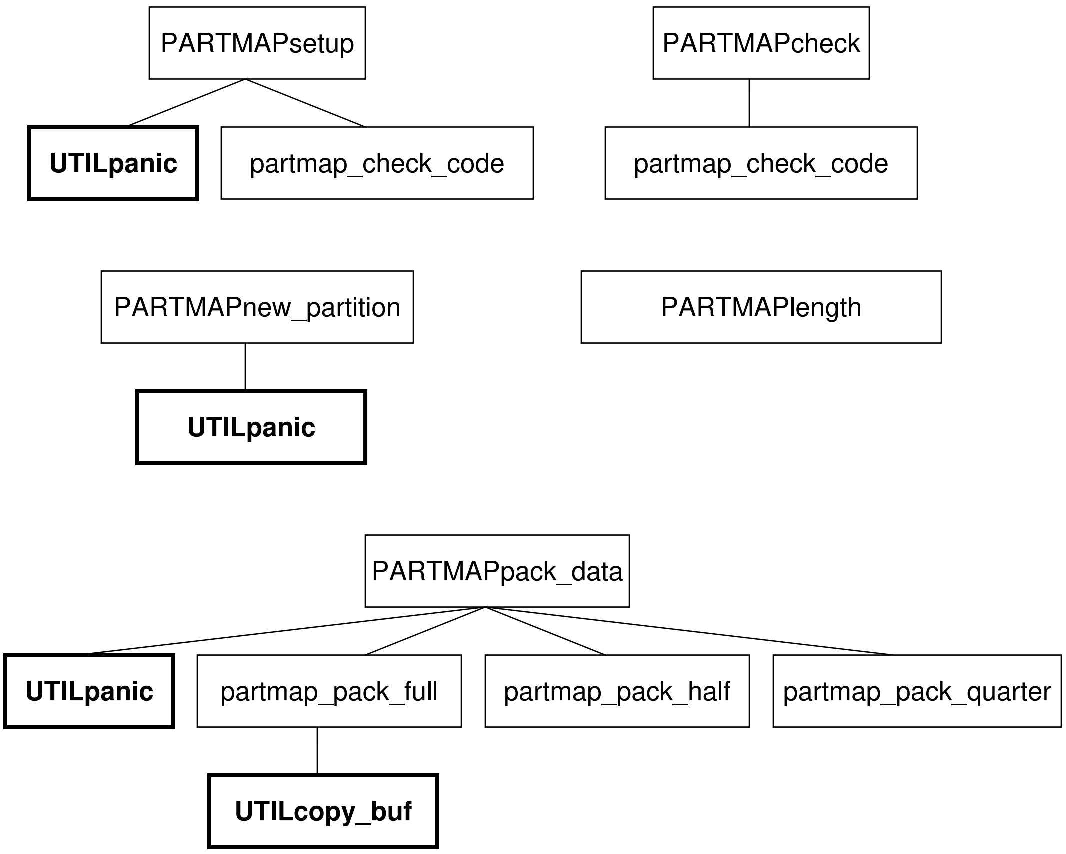

Figure 4 Partition Map Module Function Structures

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

PARTMAPsetup .... Register Partition Map for the Observation
PARTMAPlength .... Obtain Length of the passed Partition Map, in bytes.
PARTMAPnew_partition .... Setup to start processing a new partition of science data.
PARTMAPpack_data .... Pack portion of the science data into one buffer.
PARTMAPcheck .... Check the integrity of the installed Partition Map.
partmap_check_code .... Compute the integrity code for the passed Partition Map.
partmap_pack_full .... Pack science data, 1:1, into the output buffer.
partmap_pack_half .... Pack science data, 2:1, into the output buffer.
partmap_pack_quarter .... Pack science data, 4:1, into the output buffer.

6 Requirements Traceability

6.1 Science Requirements

The Partition Map Module is responsible for providing the mechanism for packing 16-bit science data words into 16, 8 and 4 bit values. This mechanism satisfies all count precision requirements specified in the ASM and PCA Science Requirements documents. The performance of the Partition Map Module directly affects the rate at which science data can be packed into telemetry buffers, and is a factor when considering the science throughput requirements. The **Requirements Traceability Matrix** illustrates which specific requirements are met by this module.

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 test the xor integrity code function against a variety of single-bit corruptions and memory-trashing corruptions and verify that the corruptions are properly detected. The tester should also measure the throughput of each of the internal packing functions; partmap_pack_full, partmap_pack_half, and partmap_pack_quarter, and the main packing function, PARTMAPpack_data. These measurements should then be checked against the science requirements to ensure that the module can meet the system's throughput requirements.

7.2 Integration Testing

For integration tests, the following equipment is required:

a. An Event Analyzer

b. A populated System Manager 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 Partition Map Module in conjunction with the actual hardware and the various Science Modes. These tests should include:

a. Verify the module in conjunction with a science mode that generates fixed length partitions, such as Binned Science Mode. Ensure that the partition's data is packed correctly. Verify that the packed data can then be reliably unpacked using information provided by the dumped Partition Map.

b. Verify the module in conjunction with a science mode that generates variable length partitions, such as Event Encoded Science Mode. Ensure that the stream of partition data words are packed correctly. Verify that the packed data can then be reliably unpacked using information provided by the dumped Partition Map.

c. Measure the overall performance of the module using a variety of Partition Maps. Ensure that the throughput requirements are met.

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		