Code 582
Flight Software Branch

CORE FLIGHT SYSTEM MEMORY MANAGER BUILD 2.4.2.0

FLIGHT SOFTWARE BUILD VERIFICATON TEST REPORT

Flight Software Branch – Code 582

Version 1.0

SIGNATURES

Submitted by:

10/7/2020

X Walt Moleski

Walt Moleski/582
cFS Flight Software Tester
Signed by: WALTER MOLESKI

Approved by:

10/7/2020

X Elizabeth Timmons

Elizabeth Timmons/582
GSFC cFS Development Lead
Signed by: cards

PLAN UPDATE HISTORY

Version	Date	Description	Affected Pages
1.0		Initial Release	All

TABLE OF CONTENTS

1 INTRODUCTION 1
1.1 Document Purpose 1
1.2 Applicable Documents 1
1.3 Document Organization 1
1.4 Definitions 2
2 OVERVIEW 3
2.1 Flight Data System Context 3
2.2 Test History 4
2.3 Testing Overview 5
2.4 Version Information 8
3 BUILD VERIFICATION TEST PREPARATION 9
3.1 Scenerio Development 9
3.2 Procedure Development and Execution 9
3.3 Test Products 9
4 BUILD VERIFICATION TEST EXECUTION 10
4.1 Testbed Overview 10
4.2 Requirements Verification Matrix 10
4.3 Requirements Partially Tested 11
4.4 Requirements/Functionality Deferred 11
4.5 Requirements/Functionality Deferred to Mission Testing 11
5 BUILD VERFICIATON TEST RESULTS 12
5.1 Overall Assessment 12
5.2 Procedure Description 12
5.3 Analysis/Inspection Requirements Verification 13
5.4 Failed Requirements 15
5.5 DCRs 16
5.5.1 DCRs Verified 16
5.5.2 Notes 16
APPENDIX A - RTTM 17
APPENDIX B - COMMAND, TELEMETRY, AND EVENTS VERIFICATION MATRIX 18

1 INTRODUCTION

1.1 DOCUMENT PURPOSE

This Test Report describes the test results from the Core Flight System (cFS) Memory Manager (MM) Flight Software (FSW) Test Team build 2.4.2.0 verification testing. It is used to verify that the MM FSW has been tested in a manner that validates that it satisfies the functional and performance requirements defined within the cFS MM Requirements Document. This Test Report summarizes the FSW test history, the build verification process, the build test configuration, and the test execution and results.

1.2 APPLICABLE DOCUMENTS

Unless otherwise stated, these documents refer to the latest version.

Parent Documents (Mission and FSW)

- 582-2007-031 cFS Memory Manager Requirements Document, Version 1.3
- 582-2008-012 cFS Deployment Guide

Reference Documents

All of the references below can be found on the Code 582 internal website at http://fsw.gsfc.nasa.gov/

- 582-2003-001 FSB FSW Test Plan Template
- 582-2004-001 FSB FSW Test Description Template
- 582-2004-002 FSB FSW Test Scenario Template
- 582-2004-003 FSB FSW Test Procedure Template
- 582-2004-004 FSB FSW Test Execution Summary Template
- 582-2004-005 FSB Test Product Peer Review Form
- 582-2000-002 FSB FSW Unit Test Standard

1.3 DOCUMENT ORGANIZATION

Section 1 of this document presents some introductory material.

Section 2 provides a flight software overview and context along with the test history and testing overview.

Section 3 describes the build verification process including procedure development and execution and test products produced.

Section 4 describes the build test configuration which includes an overview of the testbed and the requirements verification matrix.

Section 5 describes the test execution and results by subsystem.

Appendix A - provides the Requirements Traceability Matrix

Appendix B - provides the Command, Telemetry, and Events Verification Matrix

1.4 DEFINITIONS

There were 3 verifications methods used during build verification testing. They were:

- Demonstration: Show compliance with system requirement by exhibiting the required capability (e.g. by demonstrating interactive capability, display capability, print capability, etc.
- Inspection: Show compliance with a system requirement by visual verification of the software (e.g. verifying preparation for delivery, proper interfacing)
- Analysis: Perform detailed analysis of code, generated data (both intermediate data and final output data), etc., to determine compliance with system requirements.

The fields in the Requirements Verification Matrix in Section 4.3 are defined as follows:

- Requirements Tested Passed: Requirement was fully tested in a build test procedure and passed all tests.
- Requirements Tested Failed: Requirement was fully tested in a build test procedure and failed one or more aspect of the testing.
- Requirements Tested Partially: Requirement was tested partially in a build test procedure. To be fully tested, the partially tested requirement must be tested additionally in one or more other test procedures within the same build. The aspects of a partially tested requirement that were not tested in the current build were either tested in an earlier build and no longer need to be retested **and/or** there were capabilities not present required to complete the test.
- Total Tested: Total number of requirements fully tested in a build test procedure. Includes total passed and total failed, but does **not** include requirements tested partially, **unless** (included as a separate entry) testing in multiple procedures within the same build constitutes total testing of a particular requirement. Total Requirements Tested is computed this way in order to avoid multiple counting of individual requirements that are tested partially in more than one procedure.
- Deferred: Number of requirements that were planned to be tested in current build, but were not tested due to some FSW capability or necessary system component not being present.
- Total: Total Requirements Tested + Number of Requirements Deferred

In each software test section in Section 5 there is a table of DCR's. The state definitions are as follows:

- Opened: The DCR is currently being addressed
- Assigned: The DCR was accepted and the modification is being addressed
- InTest: The DCR was corrected and is currently in test
- Validated: The DCR was corrected and tested and have been validated, needs to have a CCB to close the DCR
- Closed: The DCR is closed and have been resolved and tested to satisfaction
- Closed with Defect: The DCR is closed and the defect is most likely assigned a differed DCR number associated with another subsystem.

2 OVERVIEW

2.1 FLIGHT DATA SYSTEM CONTEXT

Figure 2-1 illustrates the cFS system context. The cFE interfaces to five external systems: an Operating System (OS), a Hardward Platform (HP), an Operational Interface (OI), Applications (APP), and other cFE-based systems.

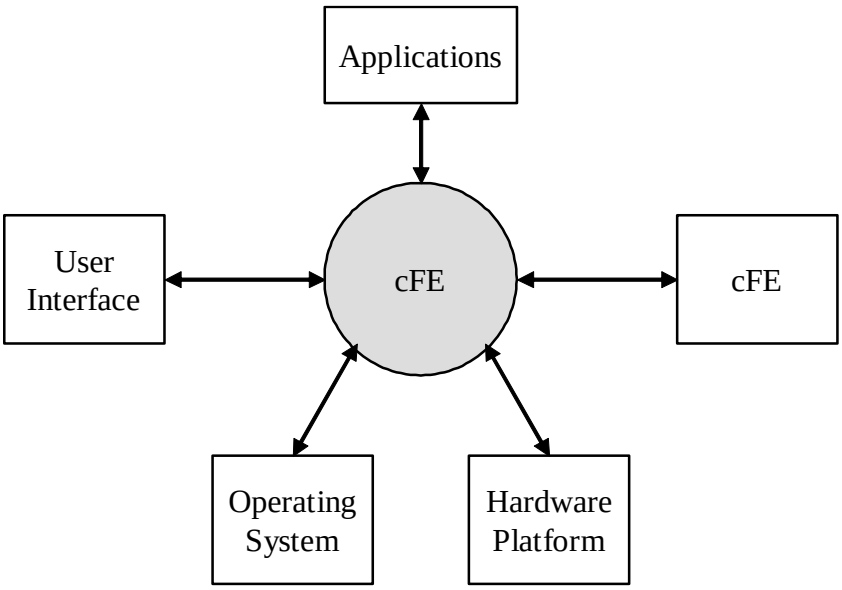

Figure 2-1 cFS System Context

The Memory Manager (MM) application of the Core Flight System (cFS) is responsible for the loading and dumping of flight system memory. MM is basically the operator interface for the Operating System Application Layer (OSAL) memory manipulation. MM provides the ability to load and dump memory via commands as well as from files. If the operating system supports symbolic addressing, MM supports specifying the memory address using a symbolic address.

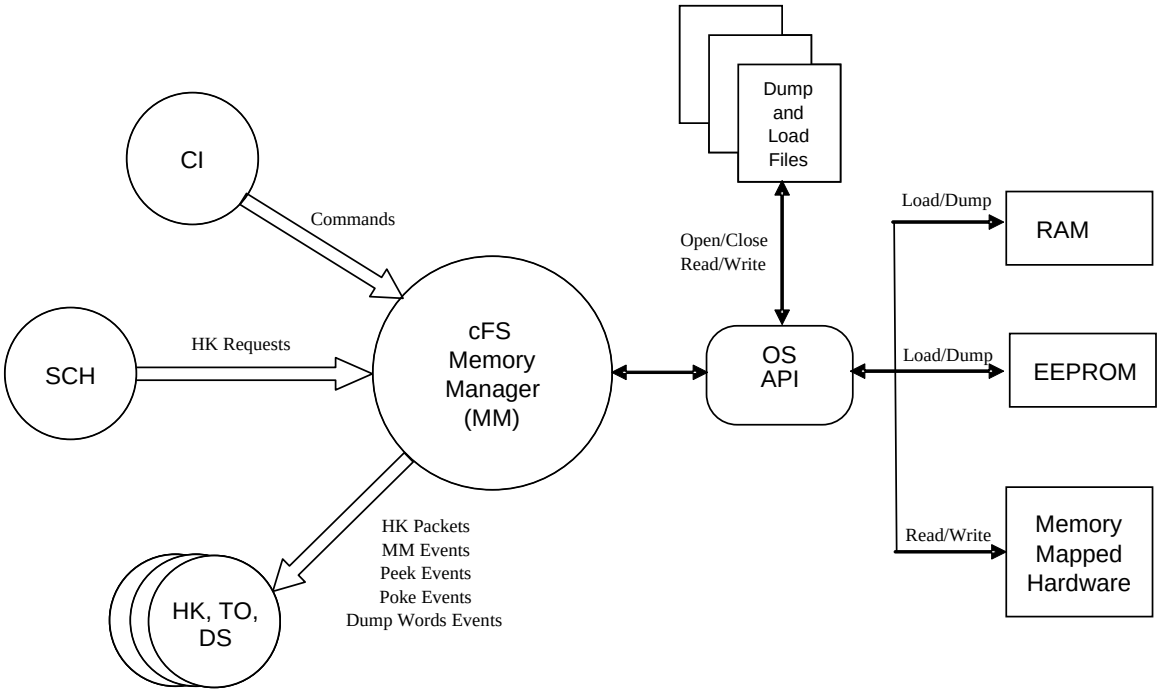

Figure 2.2 – cFS MM Context

Memory Manager makes use of the OSAL when interfacing to memory. Memory Manager assumes that the OSAL will provide routines to access processor memory as well as memory that is not directly accessible (i.e. requires address translation). Address checking is performed using the OSAL. Any addresses specified outside of the valid address range will be considered invalid.

MM performs data transfers between memory and files, but does not handle file dumps or loads. That function must be done with a file transfer application such as the cFS CCSDS File Delivery Protocol (CFDP) application.

Some of the Memory Manager requirements relate to the use of files. MM is responsible for file management operations or directory manipulations. That function is allocated to the cFS File Manager (FM) application. It should be noted that Memory Manager assumes that the files are binary.

There are 3 types of memory that are referred to in Memory Manager:

- RAM – processor memory. Generic term for RAM including DRAM, and SRAM
- EEPROM – Generic term used for non-volatile memory including EEPROM, Flash, PROM, etc.
- Memory Mapped I/O - Addressable Memory that must be read from and written to in 8, 16 or 32 bits at a time

  Note that for the Memory Mapped I/O that is byte addressable and requires no special code to support will be accessed as RAM.

2.2 TEST HISTORY

MM 1.0.0.0 – Build Verification Testing completed 09/22/2008 by Walt Moleski
MM 2.0.0.0 – Build Verification Testing completed 08/26/2009 by Walt Moleski
MM 2.1.1.0 – Build Verification Testing completed 01/10/2010 by Walt Moleski
MM 2.2.0.0 – Build Verification Testing completed 07/26/2011 by Walt Moleski
MM 2.3.0.0 – Build Verification Testing completed 01/09/2012 by Walt Moleski
MM 2.4.0.0 – Build Verification Testing completed 04/16/2015 by Walt Moleski

MM 2.4.1.0 – Build Verification Testing completed 11/15/2016 by Walt Moleski
MM 2.4.2.0 – Build Verification Testing completed 10/01/2020 by Walt Moleski

2.3 TESTING OVERVIEW

The cFS test procedures assume that the cFS application and its corresponding test application are not executing before the start of the test. If this is the case, the test procedures will need to be modified to handle this situation.

The MM application was tested during Build Verification testing using the following:

- 1 test application: tst_mm
- 5 test procedures: mm_cmds.prc, mm_eeprom.prc, mm_memmap.prc, mm_ram.prc, mm_symtab.prc

The tst_mm test application is used to send schedule requests for the output of MM's housekeeping data to the MM application. This was useful when performing build verification testing since it provided great control over the sequence of steps. In addition, having the test application eliminated the need to modify the SCH_LAB application and rebuild. When deployed for a mission, the Scheduler Application would provide this request. In addition, the test application also provides the ability to get the Cyclic Redundancy Check (CRC) value for an array of data and create files. TST_MM has 3 ground commands that are used by the MM test procedures:

- TST_MM_GetCRC
  - This command is used to determine the CRC calculated on the supplied array of data. The arguments to this command are DataSize (uint32) and dataArray[256] (uint8). Thus, the CRC is calculated on the data contained in the dataArray for DataSize bytes.
- TST_MM_CreateFile
  - This command is used to create a Memory Manager load file. The arguments to this command are DataSize (uint32), Address (uint32), Pattern (uint8), MemType (uint8, SymbolName (char) and FileName (char). This command creates an onboard file to use with the MM_Load command. The file will contain DataSize bytes of the Pattern specified. The Address where the data will be loaded depends upon whether the SymbolName argument is specified. If the SymbolName is not null, the MM will attempt to resolve the symbol to an address and then add the Address argument as an offset. Otherwise, the Address argument is used as the absolute address to load the data. The MemType argument specifies the type of memory that this file will be loading.
- TST_MM_CreateErrorFiles
  - This command is used to generate 3 onboard Memory Manager load files that contain errors. The files are created in RAM with the following names:
    - overmaxload.dat – A file that contains data that is larger than the maximum amount of data allowed for the supplied memory type.
    - toomuchdata.dat – A file that contains more data in the file than indicated by the size in the file header.
    - notenoughdata.dat – A file that contains less data in the file than indicated by the size in the file header.

These 5 MM test procedures do the following:

Procedure	Description
MM_Cmds	The purpose of this test is to verify that the Memory Manager (MM) general commands function properly. The MM_NOOP and MM_Reset commands will be tested as well as invalid commands to see if the MM application handles these appropriately. It should be noted that this procedure uses the RAW

Procedure	Description
	command with hard-coded MsgIds to send invalid commands to the MM Application.
MM _EEPROM	The purpose of this test is to verify the Memory Manager (MM) EEPROM commands of the Core Flight System (cFS). This test verifies that the EEPROM commands function properly and that the MM application handles anomalies appropriately.
MM_MemMap	The purpose of this test is to verify the Memory Manager (MM) Memory Mapped I/O commands of the Core Flight System (cFS). This test verifies that the Memory Mapped I/O commands function properly and that the MM application handles anomalies appropriately. Also, these commands are optional in the cFS. If the mission using the MM application does not support Memory Mapped I/O, this test can be eliminated from the test plan.
MM_RAM	The purpose of this test is to verify the Memory Manager (MM) Random Access Memory (RAM) commands of the Core Flight System (cFS). This test verifies that the RAM commands function properly and that the MM application handles anomalies appropriately
MM_SymbolTable	The purpose of this test is to verify the Memory Manager (MM) Symbol Table functionality of the Core Flight System (cFS). Symbol Table support is optional and thus provided in a separate test. If the mission provides Symbol Table support, this test can be used to verify its functionality.

The testers use a cFS Test Account for each build test. This account runs the Advanced Spacecraft Integration and System Test (ASIST) software and is setup to contain all the files needed to test the application. These files are extracted from GIT directory ($WORK/prc). The global utilities are pointed to by ASIST in the global area defined on the test system. Additional tools utilized by the test procedures are located in the $TOOLS directory. It is assumed that test procedures and the ASIST telemetry database used for testing is built using procedure and database templates

The following utilities were used during testing:

Name	Description
close_data_center	Directive that closes the command port from the ASIST machine to the flight cpu.
cfe_startup	Directive combines the "start_data_center", "open_tlm", and "open cmd <cpu>" ASIST startup commands.
load_start_app	Procedure to load and start a user application from the /s/opr/accounts/cfebx/apps/cpux directory.
ut_pfindicate	Directive to print the pass fail status of a particular requirement number.
ut_runproc	Directive to formally run the procedure and capture the log file.
ut_sendcmd	Directive to send EVS commands Verifies command processed and command error counters.
ut_sendrawcmd	Send raw commands to the spacecraft. Verifies command processed and command error counters.
ut_setrequirements	A directive to set the status of the cFE requirements array.
ut_setupevents	Directive to look for multiple events and increment a value for each event to indicate receipt.
ut_tlmwait	Directive that waits for the specified telemetry condition to be met
ftp_file	To ftp a file to/from the FSW/GSW.
get_mm_file_to_cvt	Directive that issues the MM_Dump2File command and downloads the file to the ground and inserts it in the MM_data telemetry item.
create_mm_file_from_cvt	Directive that creates a Memory Manager load file from the MM_data telemetry item.

load_memory	Directive that transfers the supplied file to the specified cpu and issues the MM_LoadFile command.

2.4 VERSION INFORMATION

Item	Version
MM Requirements	1.3
MM Application	2.4.2.0
TST_MM Application	2.4.2.0
cFS Bundle	Bootes
cFE	6.8.0
OSAL	5.0.0.0
ASIST	20.2
VxWorks	6.9

3 BUILD VERIFICATION TEST PREPARATION

3.1 SCENERIO DEVELOPMENT

No new scenarios developed for build verification test 2.4.2.0. All scenarios are stored on the ETD GIT server, https://aetd-git.gsfc.nasa.gov/gsfc-cfs/cfs_mm in the test-and-ground. It should be noted that as MM requirement evolve these scenarios are not updated to reflect any changes made.

3.2 PROCEDURE DEVELOPMENT AND EXECUTION

This build test was completed by running 5 test procedures. All test procedures were written using the STOL scripting language. The naming convention for files created by the test procedures was: scx_cpu<#>_<procedure name>_GMT.<ext>.

3.3 TEST PRODUCTS

Five log files were generated for every procedure that was run. They are defined as follows:

- Logs with the .loge extension list all events sent by the flight software
- Logs with the .logr extension list all requirements that passed validation by demonstration
- Logs with the .logp extension lists all prints that are generated by the test procedure
- Logs with the .logf extension lists everything from the other logs along with the steps in the test procedure
- Logs with the .logs extension lists the SFDU information (if applicable) contained in the full log.

A test summary reported is developed in MKS for each procedure by the tester after build testing is completed. All test products are maintained on MKS in the cFS-Repository MM test-and-ground directory.

4 BUILD VERIFICATION TEST EXECUTION

4.1 TESTBED OVERVIEW

MM FSW testing took place in the cFS FSW Development and Test Facility. A high level view of the cFS FSW Test Bed is shown in Figure 4-1. This facility is located in GSFC Building 23, Room W410N. This facility consists of two ASIST workstations running ASIST version 20.2, two MPC750 CPU boards running VxWorks 6.9, and a Leon3 processor that is currently not being used.

.

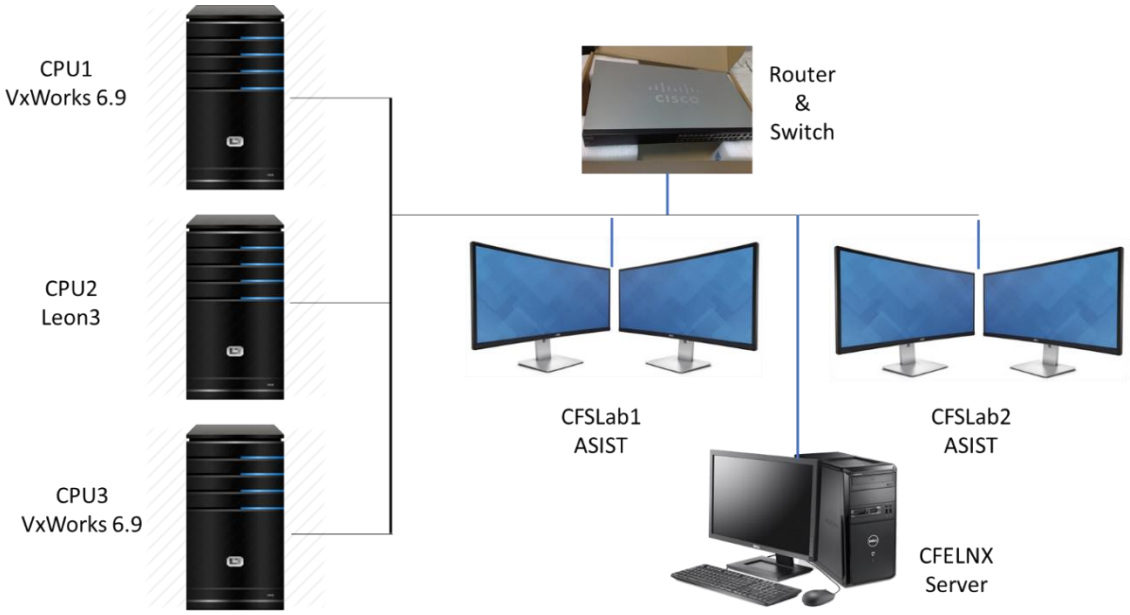

Figure 4-1 cFS FSW Development and Testing Facility

4.2 REQUIREMENTS VERIFICATION MATRIX

	Memory Manager (MM)
Requirements Tested Passed	60
Requirements Tested Failed	0
Requirements Tested Partially	0
Total Tested	60
Deferred	0
Total	60

4.3 REQUIREMENTS PARTIALLY TESTED

No requirements were partially tested.

4.4 REQUIREMENTS/FUNCTIONALITY DEFERRED

No requirements were deferred.

4.5 REQUIREMENTS/FUNCTIONALITY DEFERRED TO MISSION TESTING

The following functionality was deferred to mission testing:

- RAM was the only physical memory type tested. EEPROM, Compact Flash, SSR not tested. EEPROM testing was done by simulating EEPROM in RAM.
- For the Memory Mapped I/O testing there was no forced aligned memory available for testing. Tested was simulated using RAM.

The tests with interrupts disabled wasn't actually able to tell if interrupts were truly disabled

5 BUILD VERFICIATON TEST RESULTS

5.1 OVERALL ASSESSMENT

During this build test of the MM Application the software behaved as expected. Below is a summary of the results:

- 47 requirements passed via demonstration
- 8 requirements were validated by analysis.
- 5 requirements were verified by inspection.
- 5 DCRs were validated

5.2 PROCEDURE DESCRIPTION

Procedure	Description	Requirements tested
MM_Cmds	The purpose of this test is to verify that the Memory Manager (MM) general commands function properly. The MM_NOOP and MM_Reset commands will be tested as well as invalid commands to see if the MM application handles these appropriately	MM1000, MM1001, MM1006, MM1009, MM1010, MM1013, MM7001, MM8000, MM9000
MM _EEPROM	The purpose of this test is to verify the Memory Manager (MM) EEPROM commands of the Core Flight System (cFS). This test verifies that the EEPROM commands function properly and that the MM application handles anomalies appropriately.	MM1006, MM1007, MM1008, MM1009, MM1010, MM3000, MM3000.1, MM3001, MM3002, MM3002.1, MM3100, MM3100.1, MM3100.2, MM3104, MM3104.1, MM3200, MM3200.1, MM3300, MM3301, MM3400, MM3500, MM8000, MM9000
MM_MemMap	The purpose of this test is to verify the Memory Manager (MM) Memory Mapped I/O commands of the Core Flight System (cFS). This test verifies that the Memory Mapped I/O commands function properly and that the MM application handles anomalies appropriately. Also, these commands are optional in the cFS. If the mission using the MM application does not support Memory Mapped I/O, this test can be eliminated from the test plan.	MM1006, MM1007, MM1008, MM1009, MM1010, MM5000, MM5000.1, MM5002, MM5004, MM5004.1, MM5100, MM5100.1, MM5100.2, MM5104, MM5104.1, MM5300, MM5300.1, MM8000, MM9000
MM_RAM	The purpose of this test is to verify the Memory Manager (MM) Random Access Memory (RAM) commands of the Core Flight System (cFS). This test verifies that the RAM commands function properly and that the MM application handles anomalies appropriately	MM1006, MM1007, MM1008, MM1009, MM1010, MM2000, MM2000.1, MM2002, MM2003, MM2003.1, MM2003.2, MM2004, MM2004.1, MM2100, MM2100.1, MM2100.2, MM2104, MM2104.1, MM2300, MM2300.1, MM2500, MM2501, MM8000, MM9000

Procedure	Description	Requirements tested
MM_SymbolTable	The purpose of this test is to verify the Memory Manager (MM) Symbol Table functionality of the Core Flight System (cFS). Symbol Table support is optional and thus provided in a separate test. If the mission provides Symbol Table support, this test can be used to verify its functionality.	MM1009. MM1010, MM1011, MM1012, MM1013, MM2000, MM2000.1, MM2002, MM2003, MM2003.1, MM2004, MM2100, MM2104, MM2300, MM3000, MM3000.1, MM3001, MM3002, MM3100, MM3104, MM3200, MM3400, MM3500, MM7001, MM7002, MM7004, MM8000, MM9000

5.3 ANALYSIS/INSPECTION REQUIREMENTS VERIFICATION

The following requirements were verified using analysis:

Requirement	Description	Status	Justification
MM1013	The MM application shall generate an error event message if symbol table operations are attempted but not supported in the current target environment	Pass	This requirement is set to "U" in mm_cmds and mm_symtab test procedures since vxworks implements Symbol Table Operations. The code was inspected, and the error event was found in mm_app.c.
MM2004	Upon receipt of a Read command, MM shall read the command-specified number of consecutive bytes from the command-specified RAM memory address and generate an event message containing the data.	Pass	This requirement is verified by looking in the logp files for the mm_ram and mm_symtab test procedures. The event text is printed each time the Dump In Event command is issued. The mm_ram steps are 2.1; 2.4; 2.5; 2.8; 2.9; 2.12; 3.7; 3.8; 5.2; and 5.7. The mm_symtab steps are 2.2 and 2.5.
MM2104	Upon receipt of a Dump to File command, MM shall write the data associated with the command-specified RAM address , command-specified number of bytes and calculated <MISSION_DEFINED> CRC to the command-specified file.	Pass	This requirement is verified by examining the dump files generated and downloaded by the mm_ram test procedure in Steps 3.2; 4.2; 4.14; 4.15; 5.5; and 6.4. The mm_symtab test procedure tests this requirement in Step 2.8.
MM2500	When writing data to RAM memory, MM shall write a maximum of <PLATFORM_DEFINED, TBD> bytes per execution cycle	Pass	The code was inspected and found that the MM_SegmentBreak utility function is called after writing MM_MAX_DUMP_DATA_SEG bytes.
MM2501	When writing RAM data to a file, MM shall write a maximum of <PLATFORM_DEFINED, TBD> bytes per execution cycle	Pass	The code was inspected and found that the MM_SegmentBreak utility function is called after writing MM_MAX_DUMP_DATA_SEG bytes.

Requirement	Description	Status	Justification
MM3002	Upon receipt of a Read command, MM shall read the command-specified number of consecutive bytes from the command-specified EEPROM memory address and generate an event message containing the data.	Pass	This requirement is verified by looking in the logp files for the mm_eeprom and mm_symtab test procedures. The event text is printed each time the Dump In Event command is issued. The mm_eeprom steps are 2.1; 2.4; 2.5; 2.8; 2.9; 2.12; 3.4; 3.5; 5.2; and 5.7. The mm_symtab steps are 2.10 and 2.13.
MM3104	Upon receipt of a Dump to File command, MM shall write the data associated with the command-specified EEPROM address , command-specified number of bytes and calculated <MISSION_DEFINED> CRC to the command-specified file.	Pass	This requirement is verified by examining the dump files generated and downloaded by the mm_eeprom test procedure in Steps 3.2; 4.2; 4.14; 4.15; 5.5; and 6.6. The mm_symtab test procedure tests this requirement in Step 2.15.
MM3300	When writing data to EEPROM memory, MM shall write a maximum of <PLATFORM_DEFINED, TBD> bytes per execution cycle	Pass	The code was inspected and found that the MM_SegmentBreak utility function is called after writing MM_MAX_DUMP_DATA_SEG bytes.
MM3301	When writing EEPROM data to a file, MM shall write a maximum of <PLATFORM_DEFINED, TBD> bytes per execution cycle	Pass	The code was inspected and found that the MM_SegmentBreak utility function is called after writing MM_MAX_DUMP_DATA_SEG bytes.
MM5002	Upon receipt of a Peek command, MM shall read <PLATFORM_DEFINED> bytes of data from the command-specified Memory Mapped I/O address and generate an event message containing the following data: a) address read b) length of data read c) value of the data read	Pass	This requirement is verified by looking in the logp files for the mm_memmap test procedure. The event text is printed each time the Peek command is issued. The Steps are 2.1; 2.4; 2.5; 2.8; 2.9; and 2.12.
MM5004	Upon receipt of a Read command, MM shall read the command-specified number of consecutive bytes from the command-specified Memory Mapped I/O memory address and generate an event message containing the data.	Pass	This requirement is verified by looking in the logp files for the mm_memmap test procedure. The event text is printed each time the Dump In Event command is issued. The Steps are 3.4; 3.5; 5.2; 5.7; 5.10; 5.15; 5.18; and 5.23.
MM5104	Upon receipt of a Dump to File command, MM shall write the data associated with the command-specified Memory mapped I/O address, command-specified number of bytes and calculated <MISSION_DEFINED> CRC to the command-specified file.	Pass	This requirement is verified by examining the dump files generated and downloaded by the mm_memmap test procedure in Steps 3.2; 4.2; 4.11; 4.20; 4.32; 4.33.1; 4.33.2; 4.36; 4.37; 4.40; 4.41.1; 4.41.2; 5.5; 5.13 and 5.21.

Requirement	Description	Status	Justification
MM7002	Upon receipt of a Symbol-to-Address command, MM shall report the resolved address in telemetry for the command-specified symbol name.	Pass	This requirement is verified by the mm_symtab test procedure. Steps 2.6 and 2.17 send the LoadWID and LookupSymbol commands respectively. Each command utilized a symbol name and the event generated returned the actual address.

5.4 FAILED REQUIREMENTS

No requirements failed during MM 2.4.2.0 testing.

5.5 DCRS

No new DCRs were generated during MM 2.4.2.0 testing

5.5.1 DCRs Verified

The following DCRs were verified during testing. For each DCR the "Key" column shows the corresponding DCR in the GSFC cFS tracking system.

Key	Summary	Test Method	Test Approach
GSFCCFS-1137	MM Performance Logging doesn't exit	Inspection	The PerfLogExit call exists in the code.
GSFCCFS-1144	MM uses OS_FS_SUCCESS Codes (soon deprecated)	Test	This code is not found in the source.
GSFCCFS-1156	MM does not build against cFE 6.8 with OMIT_DEPRECATED=true and -Werror	Test	MM builds without errors with cFE 6.8
GSFCCFS-1231	MM may have alignment problems on some platforms	Inspection	The command and packet headers were modified to use the proper header structures.
GSFCCFS-1247	MM_Fill command generates Write errors for MEM32 memory type	Test	The mm_memmap test procedure verifies this DCR.

5.5.2 Notes

It should be noted that integration testing is the ultimate verification of the MM applications performance in a system-like scenario.

APPENDIX A - RTTM

The MM Build 2.4.2.0 RTTM can be found on the ETD GIT server, https://aetd-git.gsfc.nasa.gov/gsfc-cfs/cfs_mm in the test-and-ground/results folder.

APPENDIX B - COMMAND, TELEMETRY, AND EVENTS VERIFICATION MATRIX

Command	Test Procedure(s)	Notes/Comments
MM_NOOP	mm_cmds	
MM_RESETCTRS	mm_cmds	
MM_PEEK	mm_eeprom; mm_memmap; mm_ram; mm_symtab	
MM_POKE	mm_eeprom; mm_memmap; mm_ram; mm_symtab	
MM_LOADWID	mm_ram	
MM_LOADFILE	mm_eeprom; mm_memmap; mm_ram; mm_symtab	
MM_DUMP2FILE	mm_eeprom; mm_memmap; mm_ram; mm_symtab	
MM_DUMPINEVENT	mm_eeprom; mm_memmap; mm_ram; mm_symtab	
MM_FILL	mm_eeprom; mm_memmap; mm_ram; mm_symtab	
MM_LookupSymbol	mm_symtab	
MM_SymTbl2File	mm_cmds; mm_symtab	
MM_EnableEEWrite	mm_eeprom; mm_symtab	
MM_DisableEEWrite	mm_eeprom; mm_symtab	

Telemetry	Test Procedure(s)	Notes/Comments
MM_CMDPC	mm_cmds; mm_eeprom; mm_memmap; mm_ram; mm_symtab	
MM_CMDEC	mm_cmds; mm_eeprom; mm_memmap; mm_ram; mm_symtab	
MM_LASTACTN	mm_cmds; mm_eeprom; mm_memmap; mm_ram; mm_symtab	
MM_MEMTYPE	mm_cmds; mm_eeprom; mm_memmap; mm_ram; mm_symtab	
MM_ADDRESS	mm_cmds; mm_eeprom; mm_memmap; mm_ram; mm_symtab	
MM_FILLPATTERN	mm_cmds; mm_eeprom; mm_memmap; mm_ram; mm_symtab	
MM_BYTESPROC	mm_cmds; mm_eeprom; mm_memmap; mm_ram; mm_symtab	
MM_LASTFILE	mm_cmds; mm_eeprom; mm_memmap; mm_ram; mm_symtab	

File Telemetry	Test Procedure(s)	Notes/Comments
MM_data[2048]	mm_eeprom; mm_memmap; mm_ram;	

	Event Message Ids	Test Procedure(s)	Notes/Comments
1	MM_INIT_INF_EID	mm_cmds; mm_eeprom; mm_memmap; mm_ram; mm_symtab	
2	MM_NOOP_INF_EID	mm_cmds	
3	MM_RESET_DBG_EID	mm_cmds	
4	MM_LOAD_WID_INF_EID	mm_ram; mm_symtab	
5	MM_LD_MEM_FILE_INF_EID	mm_eeprom; mm_memmap; mm_ram; mm_symtab	
6	MM_FILL_INF_EID	mm_eeprom; mm_memmap; mm_ram; mm_symtab	
7	MM_PEEK_BYTE_INF_EID	mm_eeprom; mm_memmap; mm_ram; mm_symtab	
8	MM_PEEK_WORD_INF_EID	mm_eeprom; mm_memmap; mm_ram	
9	MM_PEEK_DWORD_INF_EID	mm_eeprom; mm_memmap; mm_ram	
10	MM_POKE_BYTE_INF_EID	mm_eeprom; mm_memmap; mm_ram; mm_symtab	
11	MM_POKE_WORD_INF_EID	mm_eeprom; mm_memmap; mm_ram	
12	MM_POKE_DWORD_INF_EID	mm_eeprom; mm_memmap; mm_ram	
13	MM_DMP_MEM_FILE_INF_EID	mm_eeprom; mm_memmap; mm_ram; mm_symtab	
14	MM_DUMP_INEVENT_INF_EID	mm_eeprom; mm_memmap; mm_ram; mm_symtab	
15	MM_PIPE_ERR_EID		
16	MM_MID_ERR_EID		
17	MM_CC1_ERR_EID	mm_cmds	
18	MM_LEN_ERR_EID	mm_cmds; mm_eeprom; mm_memmap; mm_ram; mm_symtab	
19	MM_MEMTYPE_ERR_EID	mm_eeprom; mm_memmap; mm_ram	
20	MM_SYMNAME_ERR_EID	mm_symtab	
21	MM_DATA_SIZE_BYTES_ERR_EID	mm_eeprom; mm_memmap; mm_ram	
22	MM_DATA_SIZE_BITS_ERR_EID	mm_eeprom; mm_memmap; mm_ram	
23	MM_ALIGN32_ERR_EID	mm_memmap;	
24	MM_ALIGN16_ERR_EID	mm_memmap;	
25	MM_OS_MEMVALIDATE_ERR_EID	mm_eeprom; mm_memmap; mm_ram	
26	MM_LOAD_FILE_CRC_ERR_EID	mm_eeprom; mm_memmap; mm_ram	

27	MM_LOAD_WID_CRC_ERR_EID	mm_ram	
28	MM_OS_EEPROMWRITE8_ERR_EID		
29	MM_OS_EEPROMWRITE16_ERR_EID		
30	MM_OS_EEPROMWRITE32_ERR_EID		
31	MM_OS_CREAT_ERR_EID	mm_eeprom; mm_memmap; mm_ram	
32	MM_OS_OPEN_ERR_EID	mm_eeprom; mm_memmap; mm_ram	
33	MM_OS_CLOSE_ERR_EID		
34	MM_OS_READ_ERR_EID		
35	MM_OS_READ_EXP_ERR_EID		
36	MM_OS_WRITE_EXP_ERR_EID		
37	MM_OS_STAT_ERR_EID		
38	MM_CFS_COMPUTECRCFROMFILE_ERR_EID		
39	MM_CMD_FNAME_ERR_EID	mm_eeprom; mm_memmap; mm_ram	
40	MM_LD_FILE_SIZE_ERR_EID	mm_eeprom; mm_memmap; mm_ram	
41	MM_FILE_LOAD_PARAMS_ERR_EID	mm_eeprom; mm_memmap; mm_ram	
42	MM_CFE_FS_READHDR_ERR_EID		
43	MM_CFE_FS_WRITEHDR_ERR_EID		
44	MM_HKREQ_LEN_ERR_EID		
45	MM_SYM_LOOKUP_INF_EID	mm_symtab	
46	MM_SYMNAME_NUL_ERR_EID	mm_symtab	
47	MM_SYMTBL_TO_FILE_INF_EID	mm_cmds; mm_symtab	
48	MM_SYMFILENAME_NUL_ERR_EID	mm_symtab	
49	MM_SYMTBL_TO_FILE_FAIL_ERR_EID	mm_cmds; mm_symtab	
50	MM_SYMTBL_TO_FILE_INVALID_ERR_EID	mm_symtab	
51	MM_EEPROM_WRITE_ENA_INF_EID	mm_eeprom; mm_symtab	
52	MM_EEPROM_WRITE_ENA_ERR_EID		Cannot generate since the PSP implementation just returns SUCCESS.
53	MM_EEPROM_WRITE_DIS_INF_EID	mm_eeprom; mm_symtab	
54	MM_EEPROM_WRITE_DIS_ERR_EID		Cannot generate since the PSP implementation just returns SUCCESS.
55	MM_OS_ZERO_READ_ERR_EID		