Code 582
Flight Software Branch

CORE FLIGHT SYSTEM
Health and Safety
BUILD 2.3.0.0

FLIGHT SOFTWARE BUILD VERIFICATON TEST REPORT

Flight Software Branch – Code 582

Version 1.0

SIGNATURES

Submitted by:

X WALTER MOLESKI

Digitally signed by WALTER MOLESKI
DN: c=US, o=U.S. Government, ou=NASA, ou=People, cn=WALTER MOLESKI, 0.9.2342.19200300.100.1.1=wmoleski
Date: 2016.10.12 12:32:31 -04'00'

Walt Moleski/582
cFS Flight Software Tester

Approved by:

X SUSANNE STREGE

Digitally signed by SUSANNE STREGE
DN: c=US, o=U.S. Government, ou=NASA, ou=People, cn=SUSANNE STREGE, 0.9.2342.19200300.100.1.1=sstrege
Date: 2016.10.12 12:18:02 -04'00'

Susanne Strege/582
cFS Flight Software Product Development Lead

PLAN UPDATE HISTORY

Version	Date	Description	Affected Pages
Draft 0.1	09/22/2016	Draft for review	All

TABLE OF CONTENTS

1 INTRODUCTION

1.1 DOCUMENT PURPOSE

This Test Report describes the test results from the Core Flight System (cFS) Health and Safety (HS) Flight Software (FSW) Test Team build 2.3.0.0 verification testing. It is used to verify that the HS FSW has been tested in a manner that validates that it satisfies the functional and performance requirements defined within the cFS HS 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)

a.	582-2008-037	cFS Health and Safety Requirements Document, Version 1.4
b.	582-2008-012	cFS Deployment Guide, Version 3.1

Reference Documents

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

c.	582-2003-001	FSB FSW Test Plan Template
d.	582-2004-001	FSB FSW Test Description Template
e.	582-2004-002	FSB FSW Test Scenario Template
f.	582-2004-003	FSB FSW Test Procedure Template
g.	582-2004-004	FSB FSW Test Execution Summary Template
h.	582-2004-005	FSB Test Product Peer Review Form
i.	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 verification methods used during build verification testing. They were:

- <u>Demonstration:</u> Show compliance with system requirement by exhibiting the required capability (e.g. by demonstrating interactive capability, display capability, print capability, etc.
- <u>Inspection:</u> Show compliance with a system requirement by visual verification of the software (e.g. verifying preparation for delivery, proper interfacing)
- <u>Analysis:</u> 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:

- <u>Requirements Tested Passed</u>: Requirement was fully tested in a build test procedure and passed all tests.
- <u>Requirements Tested Failed</u>: Requirement was fully tested in a build test procedure and failed one or more aspect of the testing.
- <u>Requirements Tested Partially</u>: Requirement was tested partially in a build test procedure. To be fully tested, the partially tested requirement is either tested additionally in one or more other test procedures within the same build **and/or** other aspects of the requirement must be tested in a later build, due to capabilities not present in the current build
- <u>Total Tested</u>: 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.
- <u>Deferred</u>: 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.
- <u>Total</u>: 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:

- <u>Opened:</u> The DCR is currently being addressed
- <u>Assigned:</u> The DCR was accepted and the modification is being addressed
- <u>InTest:</u> The DCR was corrected and is currently in test
- <u>Validated:</u> The DCR was corrected and tested and has been validated, needs to have a CCB to close the DCR
- <u>Closed:</u> The DCR is closed and have been resolved and tested to satisfaction
- <u>Closed with Defect:</u> 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 Hardware Platform (HP), an Operational Interface (OI), Applications (APP), and other cFE-based systems.

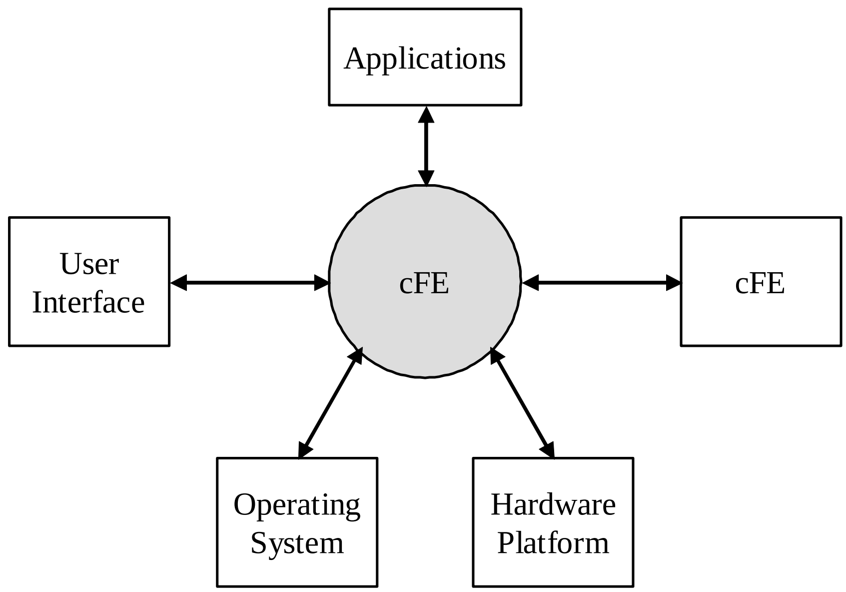

Figure 2-1 cFS System Context

The figure below shows major interfaces between the Health and Safety application and other core Flight Executive (cFE) tasks and cFS applications. Although it isn't shown explicitly, all task-to-task communications are accomplished via the cFE Software Bus (SB) application.

Inputs to the HS application include: 1) Wake-up calls from the Scheduler (SCH) application which trigger processing, 2) Housekeeping requests from the SCH application which trigger housekeeping data collection, 3) configuration commands from the Command Ingest (CI) task, 4) updates to the Health and Safety Tables managed by the Table Services (TBL) application, 5) Event Messages from the Event Services (EVS) application, and 6) Application information from the Executive Services (ES) application.

Outputs from the HS application include: 1) HS housekeeping messages sent to the Housekeeping (HK) application, 2) Processor Reset calls sent to the ES application, 3) Software Bus messages sent to system applications, 4) Event Messages, 5) Watchdog management commands to the Operating System/Board Support Package (OS/BSP), and 6) CPU Aliveness characters to the UART.

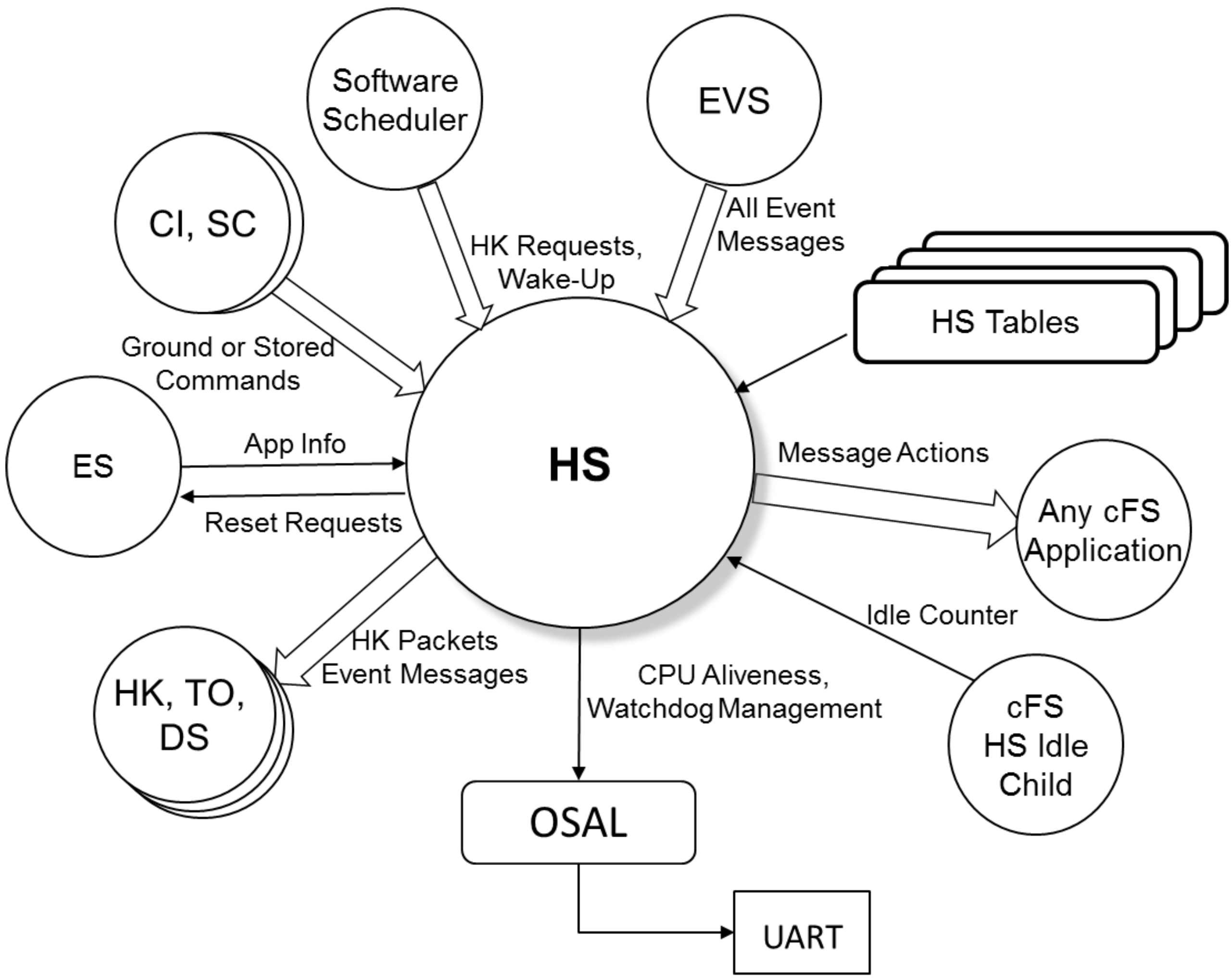

Figure 2-2 cFS HS Context

2.2 TEST HISTORY

HS 1.0.0.0 – Build Verification Testing completed 7/14/2009 by Walt Moleski
HS 2.0.0.0 – Build Verification Testing completed 9/11/2009 by Walt Moleski
HS 2.1.0.0 – Build Verification Testing completed 1/19/2011 by Walt Moleski
HS 2.2.0.0 – Build Verification Testing completed 10/20/2011 by Walt Moleski
HS 2.3.0.0 – Build Verification Testing completed 9/22/2016 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 HS application was tested during Build Verification testing using the following:

- 1 test application: tst_hs

- 8 main test procedures: hs_appmon.prc, hs_cpuhog.prc, hs_eventmon.prc, hs_exectr.prc, hs_gencmds.prc, hs_reset.prc, hs_stress.prc, hs_watchdog.prc
- 13 procedures that setup the Monitored Application Definition Table: hs_amt1.prc to hs_amt13.prc
- 9 procedures that setup the Monitored Event Definition Table: hs_emt1.prc to hs_emt9.prc
- 4 procedures that setup the Execution Counter Definition Table: hs_xct1.prc to hs_xct4.prc
- 2 procedures that setup the Message Action Definition Table: hs_mat1.prc and hs_mat2.prc
- 1 procedures called by the main procedures to startup the HS and TST_HS applications: hs_start_apps.prc
- All tests require the Advanced Spacecraft Integration and System Test (ASIST) Ground Station

The TST_HS test application is used to send schedule requests for the output of HS's housekeeping data to the HS 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 (SCH) Application would provide this request. In addition, the test application has 8 ground commands defined to help with the HS testing. These commands are described below:

- TST_HS_NOOP
  - This command that issues an event and increments the command processed counter.
- TST_HS_ResetCtrs
  - This command resets the command processed and command error counters to zero (0).
- TST_HS_SetCounters
  - This command sets several Health and Safety (HS) counters so that the HS_ResetCtrs command can be tested and verified.
- TST_HS_SetResetsPerformed
  - This command sets the Processor Resets performed by HS counter to the supplied value such that requirements relating to when the Reset Counter is equal to the MAX Resets allowed can be tested.
- TST_HS_EnableWatchdog
  - This command sets the HS Watchdog Update flag to TRUE allowing the HS application to service the WatchDog Timer.
- TST_HS_DisableWatchdog
  - This command sets the HS Watchdog Update flag to FALSE allowing the HS Application to be stopped without causing a Watchdog Reset.
- TST_HS_SetStartupDelay
  - This command sets a flag that will delay the startup of the TST_HS application to more than the HS_STARTUP_SYNC_TIMEOUT configuration parameter in order to test whether HS begins processing after waiting but never receiving the startup-synch.
- TST_HS_RemoveStartupDelay
  - This command clears a flag set by the SetStartupDelay command.
- TST_HS_HogCPU
  - This command attempts to hog the CPU by looping for the command-specified number of iterations.

The main HS test procedures do the following:

Procedure	**Description**
hs_appmon	The purpose of this test is to verify the Health and Safety (HS) Application Monitoring commands function properly. Invalid commands as well as anomalies will be tested to see if the HS application handles these appropriately.
hs_cpuhog	The purpose of this test is to verify the Health and Safety (HS) application performs CPU Utilization as stated by the requirements.

Procedure	**Description**
hs_eventmon	The purpose of this test is to verify the Health and Safety (HS) Event Monitoring commands function properly. Invalid commands as well as anomalies will be tested to see if the HS application handles these appropriately.
hs_exectr	The purpose of this test is to verify the Health and Safety (HS) handles the Execution Counter Management requirements properly. NOTE: If the Mission decides not to use Execution Counters, this test can be skipped.
hs_gencmds	The purpose of this test is to verify the Health and Safety (HS) general commands, CPU aliveness and miscellaneous commands function properly. These commands will be tested as well as invalid commands to see if the HS application handles these appropriately.
hs_reset	The purpose of this test is to verify the Health and Safety (HS) Application initializes the appropriate data items based upon the type of initialization that occurs (Application Reset, Processor Reset, or Power-On Reset). This test also verifies that the proper notifications occur if any anomalies exist with the data items stated in the requirements.
hs_stress	The purpose of this test is to verify the Health and Safety (HS) Application handles the maximum <PLATFORM_DEFINED> entries in the Monitored Application, Monitored Event and Execution Counter Tables.
hs_watchdog	The purpose of this test is to verify the Health and Safety (HS) Application handles the Watchdog Management requirements properly.

The test procedures described in the table below are called by at least one of the test procedures above.

Procedure	**Description**
hs_amt1	This procedure generates the default Monitored Application Table image.
hs_amt2	This procedure generates a Monitored Application Table image that contains a cFE Core Application that triggers a cFE Processor Reset when it fails to check-in.
hs_amt3	This procedure generates a Monitored Application Table image that contains a cFE Core Application that triggers an Event Message to be sent when it fails to check-in.
hs_amt4	This procedure generates a Monitored Application Table image that contains a cFE Core Application that triggers a Software Bus message to be sent when it fails to check-in.
hs_amt5	This procedure generates a Monitored Application Table image that contains a cFE Non-Core Application that triggers an Application Restart when it fails to check-in.
hs_amt6	This procedure generates a Monitored Application Table image that contains a cFE Non-Core Application that triggers a cFE Processor Reset when it fails to check-in.
hs_amt7	This procedure generates a Monitored Application Table image that contains a cFE Non-Core Application that triggers an Event Message to be sent when it fails to check-in.
hs_amt8	This procedure generates a Monitored Application Table image that contains a cFE Non-Core Application that triggers a Software Bus message to be sent when it fails to check-in.
hs_amt9	This procedure generates a Monitored Application Table image that contains the maximum <PLATFORM_DEFINED) entries.
hs_amt10	This procedure generates a Monitored Application Table image that contains two cFE Non-Core Applications that will never execute.

hs_amt11	This procedure generates a Monitored Application Table image that contains invalid data in order to test how HS handles table validation failures.
hs_amt12	This procedure generates a Monitored Application Table image that contains the maximum <PLATFORM_DEFINED> entries with different actions in order to perform some stress testing.
hs_amt13	This procedure generates a Monitored Application Table image that contains multiple entries for the HS application. Each entry specifies a different action.
hs_emt1	This procedure generates the default Monitored Event Table image.
hs_emt2	This procedure generates a Monitored Event Table image that contains an entry for the TST_HS_NOOP event that triggers the TST_HS application to restart.
hs_emt3	This procedure generates a Monitored Event Table image that contains an entry for the HS_NOOP event that triggers a cFE Processor Reset.
hs_emt4	This procedure generates a Monitored Event Table image that contains an entry for the TST_HS_NOOP event that triggers the TST_HS application to be deleted or stopped.
hs_emt5	This procedure generates a Monitored Event Table image that contains an entry for the HS_NOOP event that triggers a Software Bus message to be sent.
hs_emt6	This procedure generates a Monitored Event Table image that contains the maximum <PLATFORM_DEFINED> entries.
hs_emt7	This procedure generates a Monitored Event Table image that contains two entries for applications that never execute.
hs_emt8	This procedure generates a Monitored Event Table image that contains invalid data in order to test how HS handles table validation failures.
hs_emt9	This procedure generated a Monitored Event Table image that contains multiple entries for the TST_HS_NOOP event with different actions for each entry.
hs_mat1	This procedure generates the default Message Actions Table image.
hs_mat2	This procedure generates a Message Actions Table image that contains invalid data in order to test how HS handles table validation failures.
hs_xct1	This procedure generates the default Execution Counter Table image.
hs_xct2	This procedure generates an Execution Counter Table image that contains two applications that will never execute.
hs_xct3	This procedure generates an Execution Counter Table image that contains the maximum <PLATFORM_DEFINED> entries.
hs_xct4	This procedure generates an Execution Counter Table image that contains invalid data in order to test how HS handles table validation failures.

The cFS Deployment Guide contains the instruction for how to set up both the cFS Flight and Ground test environment. The testers use a cFS Test Account for each build test. This account runs ASIST and is setup to contain all the files needed to test the application. These files are extracted from MKS, the source repository tool. Included in these files are test utilities. These utilities can be located in 2 places depending upon whether they are “local” or “global” utilities. The local utilities are extracted into the working prc 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**
CFE_startup	Directive combines the "start_data_center", "open_tlm", and "open cmd <cpu>" ASIST startup commands.
close_data_center	Directive that closes the command and telemetry connection to the CPU being used.

create_tbl_file_from_cvt load_start_app	Procedure that creates a load file from the specified arguments and cvt Procedure to load and start a user application from the /s/opr/accounts/cfstest/apps/cpux directory.
load_table	Procedure that takes the specified file and transfers the file to the specified processor and then issues a TBL_LOAD command using the file.
tst_hs (version 2.2.0.0)	Test application required to test the HS application.
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_tlmupdate	Procedure to wait for a specified telemetry point to update.
ut_tlmwait	Directive that waits for the specified telemetry condition to be met

2.4 VERSION INFORMATION

Item	Version
HS Requirements	1.4
HS Application	2.3.0.0
TST_HS Application	2.2.0.0
CFE	6.5.0.0
ASIST	20.2
VxWorks	6.9

3 BUILD VERIFICATION TEST PREPARATION

3.1 SCENERIO DEVELOPMENT

No new scenarios were developed for HS 2.3.0.0 Build Verification Test. All scenarios are stored on the MKS server, in cFS-Repository HS test-and-ground directory within the Scenarios subdirectory. It should be noted that as HS requirements evolve these scenarios are not updated to reflect any changes made.

3.2 PROCEDURE DEVELOPMENT AND EXECUTION

This build test was completed by running 8 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

Four 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 report 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 HS test-and-ground directory.

4 BUILD VERIFICATION TEST EXECUTION

4.1 TESTBED OVERVIEW

HS 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 N410. This facility consists of two ASIST workstations running ASIST version 9.7k and three MPC750 CPU boards running VxWorks 6.4. CPU1 is primarily used for development testing while CPU2 and CPU3 are used for build verification testing.

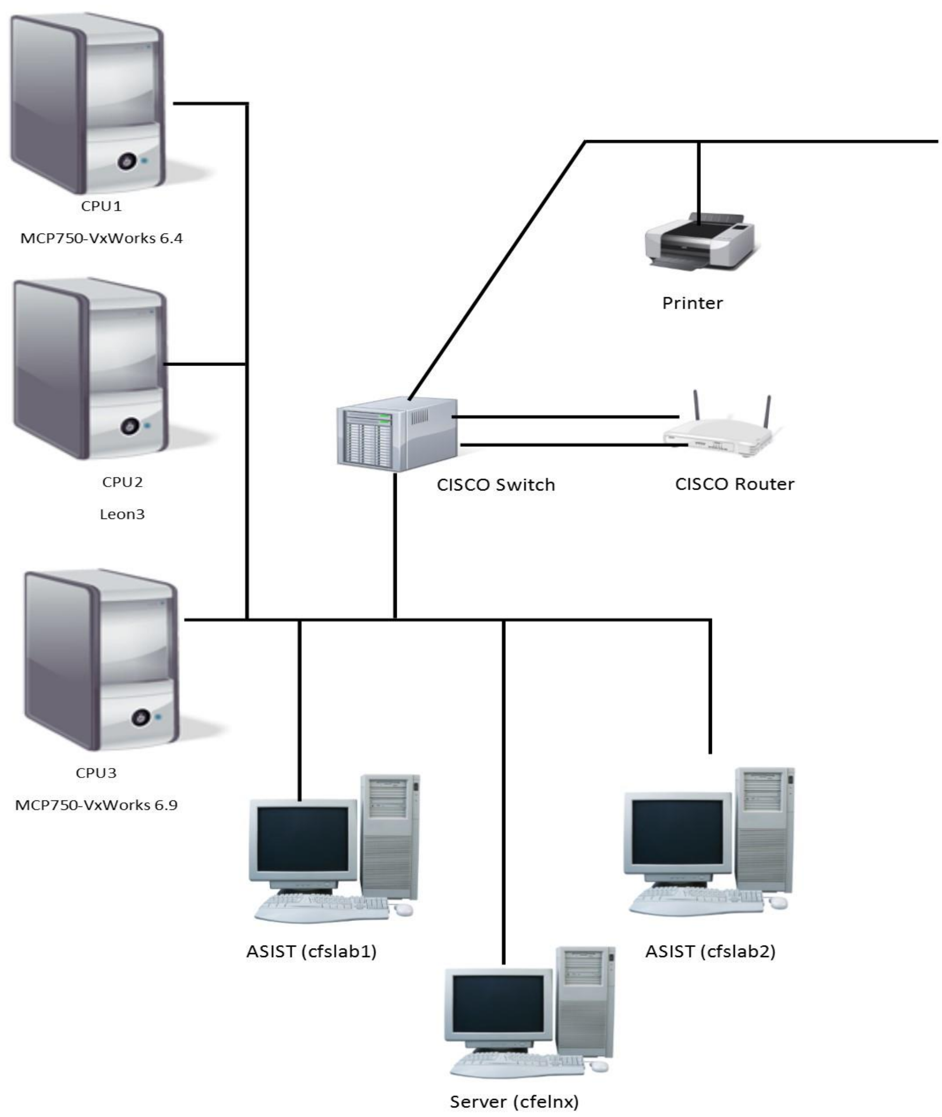

Figure 4-1 cFS FSW Development and Testing Facility

4.2 REQUIREMENTS VERIFICATION MATRIX

	Health and Safety (HS)
Requirements Tested Passed	58
Requirements Tested Failed	0
Requirements Tested Partially	0
Total Tested	58
Deferred	0
Total	58

4.3 REQUIREMENTS PARTIALLY TESTED

There were no requirements that were partially tested.

4.4 REQUIREMENTS/FUNCTIONALITY DEFERRED

No requirements were deferred to later build testing.

4.5 REQUIREMENTS/FUNCTIONALITY DEFERRED FOR MISSION TESTING

No requirements/functionality was deferred to mission testing.

5 BUILD VERFICIATON TEST RESULTS

5.1 OVERALL ASSESSMENT

During this build test of the HS Application, the software behaved as expected.

Below is a summary of the results:

- 49 requirements passed via demonstration
- 9 requirements passed via analysis
- No DCRs were generated
- 10 DCRs were verified

5.2 PROCEDURE DESCRIPTION

Procedure	Description	Requirements tested
hs_appmon	The purpose of this test is to verify the Health and Safety (HS) Application Monitoring commands function properly. Invalid commands as well as anomalies will be tested to see if the HS application handles these appropriately.	HS1002; HS1003; HS1004; HS2000; HS2000.1; HS2000.1.1; HS2000.1.2; HS2000.2; HS2000.2.1; HS2000.2.2; HS2000.2.3; HS2000.3; HS2001; HS2002; HS2003; HS2004; HS2004.1; HS7001; HS7100; HS8000
hs_cpuhog	The purpose of this test is to verify the Health and Safety (HS) application performs CPU Utilization as stated by the requirements.	HS6008; HS6009; HS6010; HS7100; HS8000
hs_eventmon	The purpose of this test is to verify the Health and Safety (HS) Event Monitoring commands function properly. Invalid commands as well as anomalies will be tested to see if the HS application handles these appropriately.	HS1000; HS1002; HS1003; HS1004; HS5000; HS5000.1; HS5000.1.1; HS5000.1.2; HS5000.2; HS5001; HS5002; HS5003; HS5004; HS5004.1; HS7001; HS7100; HS8000
hs_exectr	The purpose of this test is to verify the Health and Safety (HS) handles the Execution Counter Management requirements properly. NOTE: If the Mission decides not to use Execution Counters, this test can be skipped.	HS3000; HS3000.1; HS3001; HS3001.1; HS7100; HS8000
hs_gencmds	The purpose of this test is to verify the Health and Safety (HS) general commands, CPU aliveness and miscellaneous commands function properly. These commands will be tested as well as invalid commands to see if the HS application handles these appropriately.	HS1000; HS1001; HS1002; HS1003; HS1004; HS6005; HS6006; HS6007; HS6011; HS6012; HS7000; HS7001; HS7100; HS8000

Procedure	Description	Requirements tested
hs_reset	The purpose of this test is to verify the Health and Safety (HS) Application initializes the appropriate data items based upon the type of initialization that occurs (Application Reset, Processor Reset, or Power-On Reset). This test also verifies that the proper notifications occur if any anomalies exist with the data items stated in the requirements.	HS1003; HS7001; HS7100; HS8000; HS8001; HS8002; HS8003; HS8003.1; HS8004; HS8004.1; HS8005; HS8005.1; HS8006; HS8006.1
hs_stress	The purpose of this test is to verify the Health and Safety (HS) Application handles the maximum <PLATFORM_DEFINED> entries in the Monitored Application, Monitored Event and Execution Counter Tables.	HS2000.2.3; HS2003; HS3000; HS5003; HS7100; HS8000
hs_watchdog	The purpose of this test is to verify the Health and Safety (HS) Application handles the Watchdog Management requirements properly.	HS4000; HS4000.1; HS4000.2; HS7100; HS8000

5.3 ANALYSIS REQUIREMENTS VERIFICATION

There were 9 requirements verified using analysis and are described below:

Requirement	Requirement Text	Analysis
HS2000.1a	If the entry indicates that the application is a cFE Core Application and it has not executed for the corresponding table-defined number of HS execution cycles, HS shall perform one of the table-defined actions a. cFE Processor Reset b. Send an Event message c. Send a Software Bus message	The cFE Processor Reset action was verified by looking at the UART log file for the appmon test procedure. The UART shows Event ID 42 for CFE_TBL was generated prior to the first reset. The other actions were verified by demonstration.
HS2000.2b	If the entry indicates that the application is a cFE Core Application and it has not executed for the corresponding table-defined number of HS execution cycles, HS shall perform one of the table-defined actions a. Restart the Application (that failed to check-in) b. cFE Processor Reset c. Send an Event message d. Send a Software Bus message	The cFE Processor Reset action was verified by looking at the UART log file for the appmon test procedure. The UART shows Event ID 42 for LC was generated prior to the second reset. The other actions were verified by demonstration.
HS4000	During each HS execution cycle, HS shall check the status of the Update Watchdog Timer flag.	This requirement was verified because its sub-requirements were verified. Since requirements 4000.1 and 4000.2 passed, it can be assumed that the WatchDog Timer Flag's status is checked each cycle.

Requirement	Requirement Text	Analysis
HS4000.1	If it is set to TRUE, HS shall service the Watchdog timer.	This requirement was verified by stopping the HS application after startup and waiting for the WatchDog timer to expire which resulted in a WatchDog Reset of the processor used for this test. The UART log of the HS_WatchDog test verifies this by indicating the WatchDog Reset as the type when the processor restarts.
HS4000.2	If it is set to FALSE, HS shall not service the Watchdog Timer.	This requirement was verified by utilizing the TST_HS application to set the HS WatchDog Timer Flag to false and stopping the HS application after startup. The expected result was that the processor used for this test continues running after the WatchDog timer expires. A TST_HS NOOP command was issued and executed to verify that the processor did not reset. The UART log of the HS_WatchDog test verifies this.
HS5000.1b	If the event is defined in the Monitored Event Table, HS shall execute one of the following table-defined actions: a. Restart Application that generated the Event b. Perform cFE Processor Reset c. Delete the Application that generated the event d. Send a Software Bus message	The cFE Processor Reset action was verified by looking at the UART log file for the eventmon test procedure. The UART shows Event ID 45 for HS EID: (23) was generated prior to the first reset. The other actions were verified by demonstration.
HS6005	During each HS execution cycle, HS shall send a <PLATFORM_DEFINED> character(s) to the UART port every <PLATFORM_DEFINED> second(s).	The UART log of the GenCmds test contained the expected characters which indicate that the system is running.
HS8006	Upon any initialization, HS shall wait until the cFE startup synch has been received indicating all Applications have started.	This requirement was verified by adding the HS application to the cFE startup script and verifying that the CPU Aliveness character printed out after all the applications in the startup script were executing. The UART log of the HS_Reset test verifies this.

Requirement	Requirement Text	Analysis
HS8006.1	If the startup-synch is not received in <PLATFORM_DEFINED> seconds, HS shall begin processing.	This requirement was verified by adding the HS and TST_HS applications to the cFE startup script and having the TST_HS application delay for more than the startup-synch timeout value used by HS. If the HS CPU Aliveness indication character appears in the UART before the TST_HS application Housekeeping request events, then HS started processing before receiving the startup-synch. The UART log of the HS_Reset test verifies this.

5.4 DCRS

No new DCRs were generated during HS 2.3.0.0 testing.

5.4.1 DCRs Verified

The following DCRs were verified during testing:

DCR	Description	Test Method	Test Approach
3284	HS Idle task default priority same as utility task priority	Inspection	Starting the HS application and querying the tasks running provided the priority that was set.
3958	HS – Remove Table Compiler warning from Default table(s)	Demonstration	No compiler warning were generated for the tables.
4015	HS Custom Initialization function variable assignment sets incorrect values.	Inspection	The variable assignments are set properly.
4053	GPM-IVV-1280 – HS – Increments Command Error Counter for Internal Commands	Demonstration	The HS Error Counter did not increment for any internal commands.
4109	HS – Remove Use of the Term “Critical” When Referencing Application and Event Monitoring	Inspection	The word “Critical“ has been removed and changed to :Monitored” where appropriate.
4133	HS - HS_MAT_LD_ERR_EID Doxygen Information is Incorrect	Inspection	The table references are now correct.
4138	HS Default Tables do not get moved to the installation directory	Demonstration	Table files are now moved to the installation directory.
145570	HS - Function HS_CustomGetUtil Does Not Protect Against Divide-By-Zero	Inspection	The divide by zero check is present in the code.
145720	HS - Integrate Babelfish Ticket Fixes	Demonstration and Inspection	The three stated fixes in the DCR were made and verified.
145764	HS: CFE_EVS_SendEvent Format Warnings	Demonstration	The make process for HS 2.3.0.0 did not produce any warnings.

5.4.2 Outstanding DCRs

DCR	Description
4150	Add HS CPU utilization driver for Linux. The HS app has an hs_custom.c file that implements an idle task and calculates the CPU utilization. When running the cFE on Linux this idle task causes the linux system to use 100% CPU, which is not good for a desktop system. Since Linux already maintains its own CPU utilization stats, a version of hs_custom.c that works for linux by eliminating the idle task and instead reporting the CPU utilization stats that linux maintains. This can be added to the project source and selected at compile time.
4116	HS - Add Trick Simulation Support (JSC Request)

5.5 NOTES

Prior to testing, the HS_IDLE_TASK_PRIORITY should be checked to make sure there are no tasks or applications that run at a lower priority that what is specified in this configuration parameter.

Also, other configuration parameters needed to be modified to make the CPU Utilization work in the cFS Lab. The HS_UTIL_PER_INTERVAL_HOGGING was modified from 9900 (99%) to 1000 (10%) in order to generate the event message. Also, in order to see some utilization numbers in the Average and Peak housekeeping items, HS_UTIL_CONV_DIV was changed from 10000 to 1000. These parameters will need to be adjusted for each environment that HS executes.

It should be noted that integration testing is the ultimate verification of the HS application’s performance in a system-like scenario.

APPENDIX A - RTTM

The HS Build 2.3.0.0 RTTM can be found on the MKS server, in cFS-Repository HS test-and-ground directory results folder.

APPENDIX B - COMMAND, TELEMETRY, AND EVENTS VERIFICATION MATRIX

Command	Test Procedure(s)	Notes/Comments
HS_NOOP	EventMon, GenCmds	
HS_ResetCtrs	GenCmds	
HS_EnableAppMon	AppMon	
HS_DisableAppMon	AppMon	
HS_EnableEvtMon	EventMon	
HS_DisableEvtMon	EventMon	
HS_EnableCPUAlive	GenCmds	
HS_DisableCPUAlive	GenCmds	
HS_ResetPRCtr	GenCmds	
HS_SetMaxResetCnt	AppMon, EvnetMon, GenCmds, Reset	
HS_EnableCPUHog	CPUHog, GenCmds	
HS_DisableCPUHog	GenCmds	

Telemetry	Test Procedure(s)	Notes/Comments
HS_CMDPC	AppMon, EventMon, ExeCtr, GenCmds, Reset, Stress, Watchdog	
HS_CMDEC	AppMon, EventMon, ExeCtr, GenCmds, Reset, Stress, Watchdog	
HS_AppMonState	AppMon, EventMon, ExeCtr, GenCmds, Reset, Stress, Watchdog	
HS_EvtMonState	AppMon, EventMon, ExeCtr, GenCmds, Reset, Stress, Watchdog	
HS_CPUAliveState	AppMon, EventMon, ExeCtr, GenCmds, Reset, Stress, Watchdog	
HS_CPUHogState	CPUHog	
HS_StatusFlags		
HS_PRResetCtr	AppMon, EventMon, ExeCtr, GenCmds, Reset, Stress, Watchdog	
HS_MaxResetCnt	AppMon, EventMon, ExeCtr, GenCmds, Reset, Stress, Watchdog	
HS_EVTMonCnt	AppMon, EventMon, ExeCtr, GenCmds, Reset, Stress, Watchdog	
HS_InvalidEVTAppCnt	AppMon, EventMon, ExeCtr, GenCmds, Reset, Stress, Watchdog	
HS_AppStatus[]	AppMon, Stress	
HS_MsgActCnt	AppMon, EventMon	
HS_CPUUtilAve	CPUHog	
HS_CPUUtilPeak	CPUHog	
HS_ExecutionCtr[]	ExeCtr	
Table Telemetry		
HS_AMT[].AppName	AppMon, Reset, Stress, amtx	
HS_AMT[].NullTerm	amtx	
HS_AMT[].CycleCnt	amtx	
HS_AMT[].ActionType	amtx	
HS_EMT[].AppName	EventMon, Reset, Stress, emtx	
HS_EMT[].NullTerm	emtx	
HS_EMT[].EventID	emtx	

HS_EMT[].ActionType	emtx	
HS_XCT[].ResourceName	ExeCtr, Reset, Stress, xctx	
HS_XCT[].NullTerm	xctx	
HS_XCT[].ResourceType	xctx	
HS_MAT[].EnableState	mat1, mat2	
HS_MAT[].Cooldown	mat1, mat2	
HS_MAT[].MessageData[]	mat1, mat2	

Id	Event Message	Test Procedure(s)	Notes/Comments
1	HS_INIT_EID	AppMon, CPUHog, EventMon, ExeCtr, GenCmds, Reset, Stress, Watchdog	
2	HS_APP_EXIT_EID		
3	HS_CDS_RESTORE_ERR_EID		
4	HS_CR_CMD_PIPE_ERR_EID		
5	HS_CR_EVENT_PIPE_ERR_EID		
6	HS_CR_WAKEUP_PIPE_ERR_EID		
7	HS_SUB_EVS_ERR_EID		
8	HS_SUB_REQ_ERR_EID		
9	HS_SUB_CMD_ERR_EID		
10	HS_AMT_REG_ERR_EID		
11	HS_EMT_REG_ERR_EID		
12	HS_XCT_REG_ERR_EID		
13	HS_MAT_REG_ERR_EID		
14	HS_AMT_LD_ERR_EID	AppMon, EventMon, ExeCtr, Reset	
15	HS_EMT_LD_ERR_EID	AppMon, EventMon, ExeCtr, Reset	
16	HS_XCT_LD_ERR_EID	AppMon, EventMon, ExeCtr, Reset	
17	HS_MAT_LD_ERR_EID	AppMon, EventMon, ExeCtr, Reset	
18	HS_CDS_CORRUPT_ERR_EID		
19	HS_CC_ERR_EID	GenCmds	
20	HS_MID_ERR_EID		
21	HS_HKREQ_LEN_ERR_EID		
22	HS_LEN_ERR_EID	AppMon, EventMon, GenCmds	
23	HS_NOOP_INF_EID	AppMon, EventMon, GenCmds	
24	HS_RESET_DBG_EID	GenCmds	
25	HS_ENABLE_APPMON_DBG_EID	AppMon	
26	HS_DISABLE_APPMON_DBG_EID	AppMon	
27	HS_ENABLE_EVENTMON_DBG_EID	EventMon	
28	HS_DISABLE_EVENTMON_DBG_EID	EventMon	
29	HS_ENABLE_ALIVENESS_DBG_EID	GenCmds	
30	HS_DISABLE_ALIVENESS_DBG_EID	GenCmds	
31	HS_RESET_RESETS_DBG_EID	GenCmds	
32	HS_SET_MAX_RESETS_DBG_EID	AppMon, EventMon, GenCmds, Reset	
33	HS_APPMON_GETADDR_ERR_EID		
34	HS_EVENTMON_GETADDR_ERR_EID		
35	HS_EXECOUNT_GETADDR_ERR_EID		
36	HS_MSGACTS_GETADDR_ERR_EID		

37	HS_RESET_LIMIT_ERR_EID	AppMon, EventMon	
38	HS_APPMON_APPNAME_ERR_EID	AppMon, Stress	
39	HS_APPMON_RESTART_ERR_EID	AppMon, Stress	
40	HS_APPMON_NOT_RESTARTED_ERR_EID		
41	HS_APPMON_FAIL_ERR_EID	AppMon, Stress	
42	HS_APPMON_PROC_ERR_EID	AppMon	
43	HS_APPMON_MSGACTS_ERR_EID	AppMon	
44	HS_EVENTMON_MSGACTS_ERR_EID	EventMon	
45	HS_EVENTMON_PROC_ERR_EID	EventMon	
46	HS_EVENTMON_RESTART_ERR_EID	EventMon	
47	HS_EVENTMON_NOT_RESTARTED_ERR_EID		
48	HS_EVENTMON_DELETE_ERR_EID	EventMon	
49	HS_EVENTMON_NOT_DELETED_ERR_EID		
50	HS_AMTVAL_INF_EID	AppMon, CPUHog, EventMon, ExeCtr, GenCmds, Reset, Stress, Watchdog	
51	HS_AMTVAL_ERR_EID	AppMon, Reset	
52	HS_EMTVAL_INF_EID	AppMon, CPUHog, EventMon, ExeCtr, GenCmds, Reset, Stress, Watchdog	
53	HS_EMTVAL_ERR_EID	EventMon, Reset	
54	HS_XCTVAL_INF_EID	AppMon, CPUHog, EventMon, ExeCtr, GenCmds, Reset, Stress, Watchdog	
55	HS_XCTVAL_ERR_EID	ExeCtr, Reset	
56	HS_MATVAL_INF_EID	AppMon, CPUHog, EventMon, ExeCtr, GenCmds, Reset, Stress, Watchdog	
57	HS_MATVAL_ERR_EID	Reset	
58	HS_DISABLE_APPMON_ERR_EID	AppMon, EventMon, ExeCtr, Reset	
59	HS_DISABLE_EVENTMON_ERR_EID	AppMon, EventMon, ExeCtr, Reset	
60	HS_SUB_WAKEUP_ERR_EID		
61	HS_CPUMON_HOGGING_ERR_EID	CPUHog	
64	HS_ENABLE_CPUHOG_DBG_EID	CPUHog, GenCmds	
65	HS_DISABLE_CPUHOG_DBG_EID	GenCmds	
66	HS_EVENTMON_SUB_EID		
67	HS_EVENTMON_UNSUB_EID		
68	HS_BADEMT_UNSUB_EID		
103	Not defined in hs_events.h	GenCmds	
104	Not defined in hs_events.h	GenCmds	
106	Not defined in hs_events.h	CPUHog, GenCmds	