|  |  |
| --- | --- |
| DocCoverBackground | 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:



Approved by:



Plan Update History

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

Table of Contents

1 Introduction 1

1.1 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 4

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 11

4.3 Requirements Partially Tested 11

4.4 Requirements/Functionality Deferred 11

4.5 Requirements/Functionality Deferred for Mission Testing 11

5 Build Verficiaton Test Results 12

5.1 Overall Assessment 12

5.2 Procedure Description 12

5.3 Analysis Requirements Verification 13

5.4 DCRs 15

5.4.1 DCRs Verified 15

5.4.2 Outstanding DCRs 15

5.5 Notes 15

Appendix A - RTTM 17

Appendix B - Command, Telemetry, and Events Verification Matrix 18

# Introduction

## 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.

## Applicable Documents

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

**Parent Documents** (Mission and FSW)

1. 582-2008-037 cFS Health and Safety Requirements Document, Version 1.4
2. 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/>

1. 582-2003-001 FSB FSW Test Plan Template
2. 582-2004-001 FSB FSW Test Description Template
3. 582-2004-002 FSB FSW Test Scenario Template
4. 582-2004-003 FSB FSW Test Procedure Template
5. 582-2004-004 FSB FSW Test Execution Summary Template
6. 582-2004-005 FSB Test Product Peer Review Form
7. 582-2000-002 FSB FSW Unit Test Standard

## 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

## Definitions

There were 3 verification 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 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
* 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 has 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.

# OVERVIEW

## Flight Data System Context

Figure 2-1 illustrates the cFS system context. The cFE interfaces to five external systems: an [Operating System](#Operating_System) (OS), a [Hardware Platform](#Hardware_Platform) (HP), an [Operational Interface](file:///C:\582\_Projects\CFS\Management\Testing\Apps\HS\HS230\Artifacts\Review\Operational_Interface) (OI), [Applications](#Application) (APP), and other cFE-based systems.

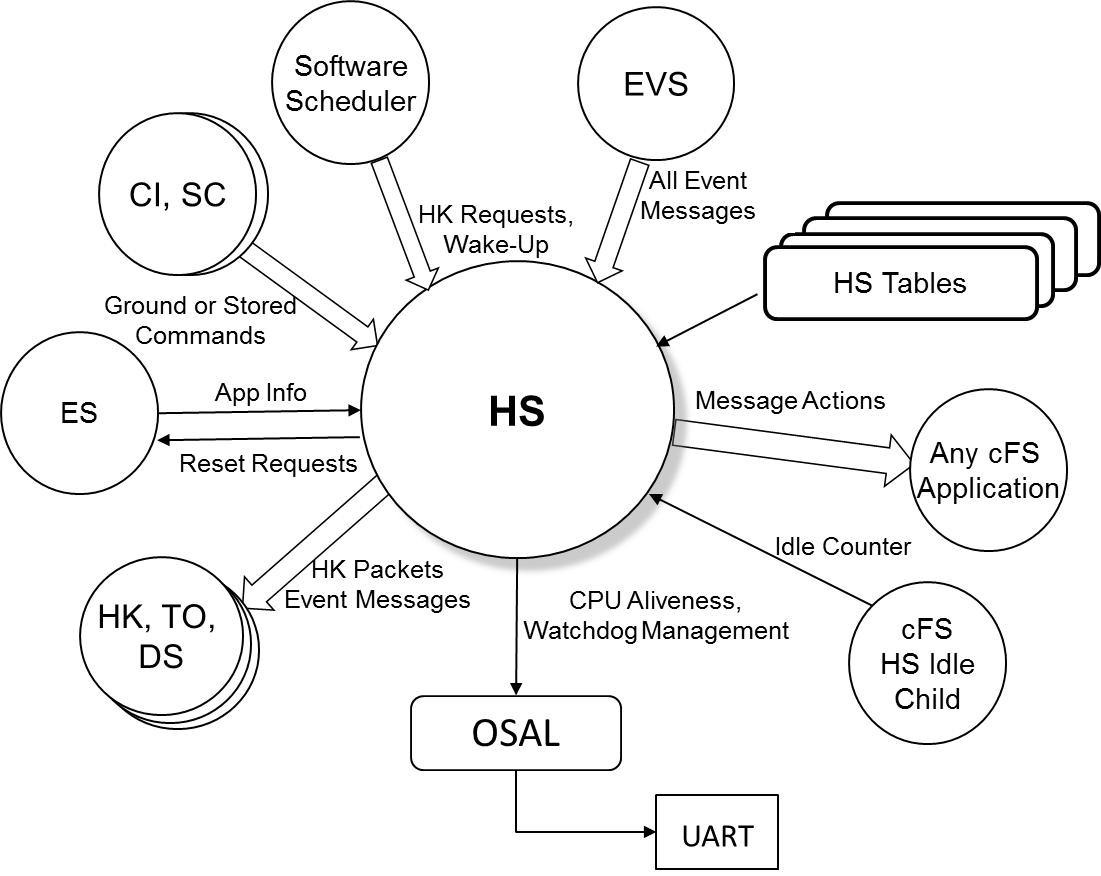


**Figure 2-1 cFS System Context**

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**

## 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

## 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. |
| 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 | Procedure that creates a load file from the specified arguments and cvt |
| load\_start\_app | 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 |

## 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 |

# Build Verification Test Preparation

## 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.

## 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>.

## Test Products

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

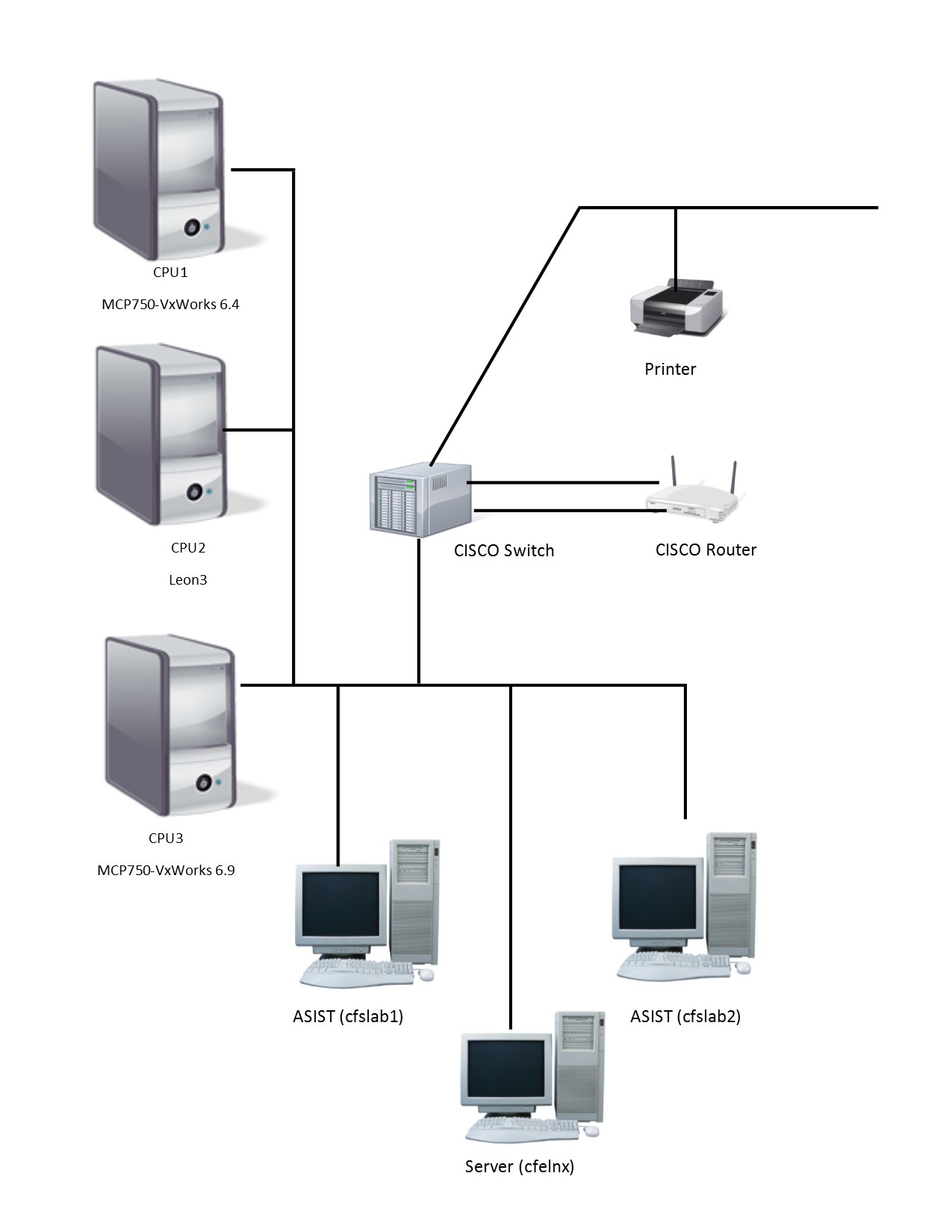
* Logs with the .loge extension list all events sent by the flight software
* Logs with the .logr extension list all requirements that passed validation by demonstration
* Logs with the .logp extension lists all prints that are generated by the test procedure
* Logs with the .logf extension lists everything 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.

# Build Verification Test Execution

## 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**

## 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 |

## Requirements Partially Tested

There were no requirements that were partially tested.

## Requirements/Functionality Deferred

No requirements were deferred to later build testing.

## Requirements/Functionality Deferred for Mission Testing

No requirements/functionality was deferred to mission testing.

# Build Verficiaton Test Results

## 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

## 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 |
| 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 |

## 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   1. cFE Processor Reset 2. Send an Event message 3. 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   1. Restart the Application (that failed to check-in) 2. cFE Processor Reset 3. Send an Event message 4. 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. |
| 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:   1. Restart Application that generated the Event 2. Perform cFE Processor Reset 3. Delete the Application that generated the event 4. 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. |
| 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. |

## DCRs

No new DCRs were generated during HS 2.3.0.0 testing.

### 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. |

### 

### 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) |

## 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.

1. 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.

1. 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 |  |