|  |  |
| --- | --- |
| DocCoverBackground | CORE FLIGHT SYSTEM  Limit Checker  BUILD 2.1.0.0  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 |
| --- | --- | --- | --- |
| 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 4

2.4 Version Information 9

3 Build Verification Test Preparation 10

3.1 Scenerio Development 10

3.2 Procedure Development and Execution 10

3.3 Test Products 10

4 Build Verification Test Execution 11

4.1 Testbed Overview 11

4.2 Requirements Verification Matrix 12

4.3 Requirements Partially Tested 12

4.4 Requirements/Functionality Deferred 12

4.5 Requirements/Functionality Deferred for Mission Testing 12

5 Build Verficiaton Test Results 13

5.1 Overall Assessment 13

5.2 Procedure Description 13

5.3 Analysis Requirements Verification 14

5.4 Failed Requirements 15

5.5 DCRs and Trac tickets 15

5.5.1 DCRs Verified 15

5.5.2 Outstanding DCRs 16

5.6 Notes 17

Appendix A - RTTM 19

Appendix B - Command, Telemetry, and Events Verification Matrix 20

# Introduction

## Document Purpose

This Test Report describes the test results from the Core Flight System (cFS) Limit Checker (LC) Flight Software (FSW) Test Team build 2.1.0.0 verification testing. It is used to verify that the LC FSW has been tested in a manner that validates that it satisfies the functional and performance requirements defined within the cFS LC Requirements Document. This Test Report summarizes the FSW test history, the build verification process, the build test configuration, and the test execution and results.

This version of LC builds upon the initial Limit Checker eXtended (LCX) version that was implemented for the MMS project. The “eXtended” term for the LC application was added in that release to denote the extended capability for the application to detect stale data. This term has been deprecated in this release.

## Applicable Documents

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

**Parent Documents** (Mission and FSW)

* 582-2012-006 cFS Limit Checker Requirements Document, Version 1.1
* 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://fsb.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

## 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:\Users\sstrege\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\L299AK4Q\Operational_Interface) (OI), [Applications](#Application) (APP), and other cFE-based systems.

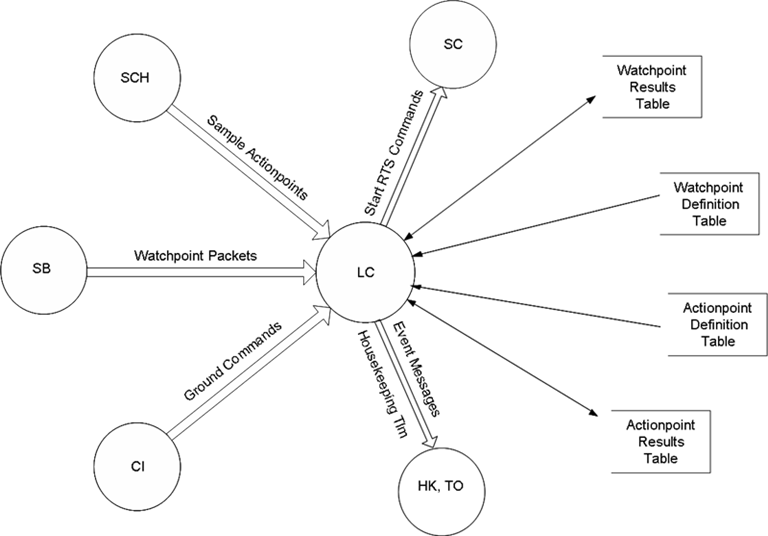


**Figure 2-1 cFS System Context**

The figure below shows major interfaces between the Limit Checker application and other core Flight Executive (cFE) and Core Flight System (cFS) applications. Although it isn’t shown explicitly, all task-to-task communications are accomplished via the cFE Software Bus application.

Inputs to the Limit Checker task include: 1) Wake-up calls from the Scheduler (SCH) application which trigger processing, 2) Housekeeping requests from the Scheduler (SCH) application which trigger housekeeping data collection, 3) configuration commands from the Command Ingest (CI) application, and 4) updates to Limit Checker Tables managed by the Table Services (TBL) application 5) Watchpoint Packets from the Software Bus (SB) application.

Outputs from the Limit Checker application include: 1) Limit Checker housekeeping messages sent to the Housekeeping (HK) application, 2) RTS Commands sent to the Stored Command application for processing, and 3) Event messages.



**Figure 2-1 cFS LC Context**

## Test History

LCX 2.0.0.0 – Build Verification Testing completed 10/2/2012 by Walt Moleski

LC 2.1.0.0 – Build Verification Testing completed 8/16/2017 by Joey Gurganus and 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 not the case, the test procedures will need to be modified to handle this situation.

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

* 1 test application: tst\_lc
* 10 main test procedures: lc\_evtfilter.prc, lc\_gencmds.prc, lc\_monitoring.prc, lc\_noaction.prc, lc\_stale.prc, lc\_resetcds.prc, lc\_resetnocds.prc, lc\_stress.prc, lc\_tabletesting.prc, lc\_withaction.prc
* 17 test procedures that setup the Actionpoint and Watchpoint Definition Tables: lc\_adt1.prc, lc\_adt1a.prc, lc\_adt1b.prc, lc\_adt2.prc, lc\_adt2a.prc, lc\_adt3.prc, lc\_adt4.prc. lc\_adt4a.prc, lc\_adt5.prc, lc\_adt6.prc lc\_wdt1.prc, lc\_wdt2.prc, lc\_wdt2a.prc, lc\_wdt3.prc, lc\_wdt4.prc, lc\_wdt4a.prc, and lc\_wdt5.prc
* 2 test procedures that are called by some of the main procedures: lc\_sendmonpackets.prc and lc\_sendpackets.prc
* 1 test procedure that tests another aspect of the lc\_resetcds.prc: lc\_resetcdsstate.prc
* 1 RTS load file to be used by the lc\_stress.prc to actually execute an RTS: lc\_rts10\_load.scs. Refer to the LC Special Build Instructions document in the lc/docs repository in MKS.
* All tests require the Advanced Spacecraft Integration and System Test (ASIST) Ground Station

The TST\_LC test application is used to send schedule requests for the output of LC’s housekeeping data to the LC application. This was useful when performing build verification testing since it provided great control over the sequence of steps. When deployed for a mission, the Scheduler Application would provide this request. In addition, the test application has 7 ground commands defined to help with the LC testing. These commands are described below:

* TST\_LC\_NOOP
  + This command that issues an event and increments the command processed counter.
* TST\_LC\_ResetCtrs
  + This command resets the command processed and command error counters to zero (0).
* TST\_LC\_SetCounters
  + This command sets several Limit Checker (LC) counters so that the LC\_ResetCtrs command can be tested and verified.
* TST\_LC\_SetWRT
  + This command sets the Watchpoint statistics with non-zero data in order to detect whether the results table was restored properly from the Critical Data Store or if the last dump on the table actually changed the values.
* TST\_LC\_SetART
  + This command sets the Actionpoint statistics with non-zero data in order to detect whether the results table was restored properly from the Critical Data Store or if the last dump on the table actually changed the values.
* TST\_LC\_SendPacket
  + This command sends a packet of data to the LC application so that it can be processed by the LC application.
* TST\_LC\_SendSample
  + This command requests the LC application to sample the specified Actionpoint.

The LC 2.1.0.0 testing was performed using 4 different configurations. Each configuration required a separate compilation with changes to the PLATFORM\_DEFINED configuration parameters. These configurations are described below:

* Normal: LC compiled out of the box with no Critical Data saved.
* Normal CDS: LC compiled with the following parameters set:
  + LC\_SAVE\_TO\_CDS defined
  + LC\_STATE\_WHEN\_CDS\_RESTORED set to LC\_STATE\_FROM\_CDS
* CDS ACTIVE State: LC compiled with the configuration above and the LC\_STATE\_WHEN\_CDS\_RESTORED set to LC\_STATE\_ACTIVE.
* CDS PASSIVE State: LC compiled with the configuration above and the LC\_STATE\_WHEN\_CDS\_RESTORED set to LC\_STATE\_PASSIVE.

The main LC test procedures do the following:

| **Procedure** | **Description** |
| --- | --- |
| lc\_evtfilter | The purpose of this test is to verify that Limit Checker (LC) properly filters the PASS to FAIL and FAIL to PASS transition events when the table-defined maximum number of times has been reached for an ActionPoint (AP). |
| lc\_gencmds | The purpose of this test is to verify that the LC general commands execute and function properly. |
| lc\_monitoring | The purpose of this test is to verify that the LC application functions properly when monitoring WatchPoints (WP). All WPs evaluate to FALSE so no thresholds will be reached to cause ActionPoints (AP) to take action. |
| lc\_noaction | The purpose of this test is to verify that the LC application functions properly when monitoring Watchpoints (WP). WPs evaluate to a mix of TRUE, FALSE, and STALE. All ActionPoints (AP) PASS which results in no thresholds being reached and thus, no actions taken. |
| lc\_stale | The purpose of this test is to verify that the LC application properly handles “stale” data. Stale data is data that has not been received for x Sample commands as specified in the WatchPoint Definition table. |
| lc\_resetcds | The purpose of this test is to verify that the LC application initializes the appropriate data items based upon the type of reset that occurs (Application, Processor, or Power-On). This test should NOT be executed if the configuration parameter indicating Save Critical Data is not set by the Mission. |
| lc\_resetcdsstate | The purpose of this test is to verify that the LC application restores the proper Application State when an Application or cFE Processor Reset occurs. This test should NOT be executed if the configuration parameter indicating Save Critical Data is not set by the Mission. |
| lc\_resetnocds | The purpose of this test is to verify that the LC application does not save any data across a reset (Application, Processor, or Power-On). This test should NOT be executed if the configuration parameter indicating Save Critical Data is set by the Mission. |
| lc\_stress | The purpose of this test is to verify that the LC application functions properly when monitoring the maximum number of WatchPoints (WP). All evaluate to FALSE. The maximum number of ActionPoints (SP) evaluate to FAIL which in turn trigger the execution of an RTS. Also, Anomaly tests are performed to ensure the appropriate action is taken by the LC application. |
| lc\_tabletesting | The purpose of this test is to verify that the LC application functions properly when loading new WatchPoint Definition Tables (WDT) and ActionPoint Definition Tables (ADT). The tables are updated while the LC application is in the ACTIVE, PASSIVE, and DISABLED state. Other tests attempt to load invalid tables to ensure that the validation functions correctly reject the table loads. |
| lc\_withaction | The purpose of this test is to verify that the LC application functions properly when monitoring WatchPoints (WP). WPs evaluate to a mix of TRUE, FALSE, and STALE. ActionPoints (APs) evaluate to a mix of PASS, FAIL, and STALE. |

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

| **Procedure** | **Description** |
| --- | --- |
| lc\_sendmonpackets | This procedure sends packet data that is used by the lc\_monitoring test procedure. |
| lc\_sendpackets | This procedure sends packet data that is used by several test procedures described above. |
| lc\_adt1 | This procedure sets up a version of the ActionPoint Definition Table (ADT) that is used by the main test procedures. NOTE: This ADT is to be used in conjunction with the lc\_wdt1 procedure described below. |
| lc\_adt1a | This procedure sets up a version of the ActionPoint Definition Table (ADT) that is used by the main test procedures. NOTE: This ADT is to be used in conjunction with the lc\_wdt1 procedure described below. |
| lc\_adt1b | This procedure sets up a version of the ActionPoint Definition Table (ADT) that is used by the main test procedures. NOTE: This ADT is to be used in conjunction with the lc\_wdt1 procedure described below. |
| lc\_adt2 | This procedure sets up a version of the ActionPoint Definition Table (ADT) that is used by the main test procedures. NOTE: This ADT is to be used in conjunction with the lc\_wdt2 procedure described below. |
| lc\_adt2a | This procedure sets up a version of the ActionPoint Definition Table (ADT) that is used by the main test procedures. NOTE: This ADT is to be used in conjunction with the lc\_wdt2a procedure described below. |
| lc\_adt3 | This procedure sets up a version of the ActionPoint Definition Table (ADT) that is used by the main test procedures. NOTE: This ADT is to be used in conjunction with the lc\_wdt3 procedure described below. |
| lc\_adt4 | This procedure sets up a version of the ActionPoint Definition Table (ADT) that is used by the main test procedures. NOTE: This ADT is to be used in conjunction with the lc\_wdt4 procedure described below. |
| lc\_adt4a | This procedure sets up a version of the ActionPoint Definition Table (ADT) that is used by the main test procedures. NOTE: This ADT is to be used in conjunction with the lc\_wdt4a procedure described below. |
| lc\_adt5 | This procedure sets up a version of the ActionPoint Definition Table (ADT) that is used by the evtfilter test procedure. NOTE: This ADT is to be used in conjunction with the lc\_wdt1 procedure described below. |
| lc\_adt6 | The purpose of this procedure is to generate an ActionPoint Definition Table (ADT) to be used by the LC application to determine if the correct transitions are achieved when a WatchPoint goes “stale”. |
| lc\_wdt1 | This procedure sets up a version of the WatchPoint Definition Table (WDT) that is used by the main test procedures. NOTE: This WDT is to be used in conjunction with the lc\_adt1, lc\_adt1a, and lc\_adt1b procedures described above. |
| lc\_wdt2 | This procedure sets up a version of the WatchPoint Definition Table (WDT) that is used by the main test procedures. NOTE: This WDT is to be used in conjunction with the lc\_adt2 procedure described above. |
| lc\_wdt2a | This procedure sets up a version of the WatchPoint Definition Table (WDT) that is used by the main test procedures. NOTE: This WDT is to be used in conjunction with the lc\_adt2a procedure described above. |
| lc\_wdt3 | This procedure sets up a version of the WatchPoint Definition Table (WDT) that is used by the main test procedures. NOTE: This WDT is to be used in conjunction with the lc\_adt3 procedure described above. |
| lc\_wdt4 | This procedure sets up a version of the WatchPoint Definition Table (WDT) that is used by the main test procedures. NOTE: This WDT is to be used in conjunction with the lc\_adt4 procedure described above. |
| lc\_wdt4a | This procedure sets up a version of the WatchPoint Definition Table (WDT) that is used by the main test procedures. NOTE: This WDT is to be used in conjunction with the lc\_adt4a procedure described above. |
| lc\_wdt5 | The purpose of this test is to generate the WatchPoint Definition Table (WDT) 5 in order to test the new StaleAge option for the LC application. |

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\_lc (version 2.1.0.0) | Test application required to test the LC 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 |
| LC Requirements | 1.2 |
| LC Application | 2.1.0.0 |
| TST\_LC Application | 2.1.0.0 |
| CFE | 6.5.0.0 |
| OSAL | 4.2.0.0 |
| ASIST | 20.2 |
| VxWorks | 6.9 |

# Build Verification Test Preparation

## Scenerio Development

No scenarios were developed for LC 2.1.0.0 Build Verification Test. All scenarios are stored on the MKS server, in cFS-Repository LC test-and-ground directory within the Scenarios subdirectory. It should be noted that as LC requirements evolve these scenarios are not updated to reflect any changes made.

## Procedure Development and Execution

This build test was completed by running 10 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

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

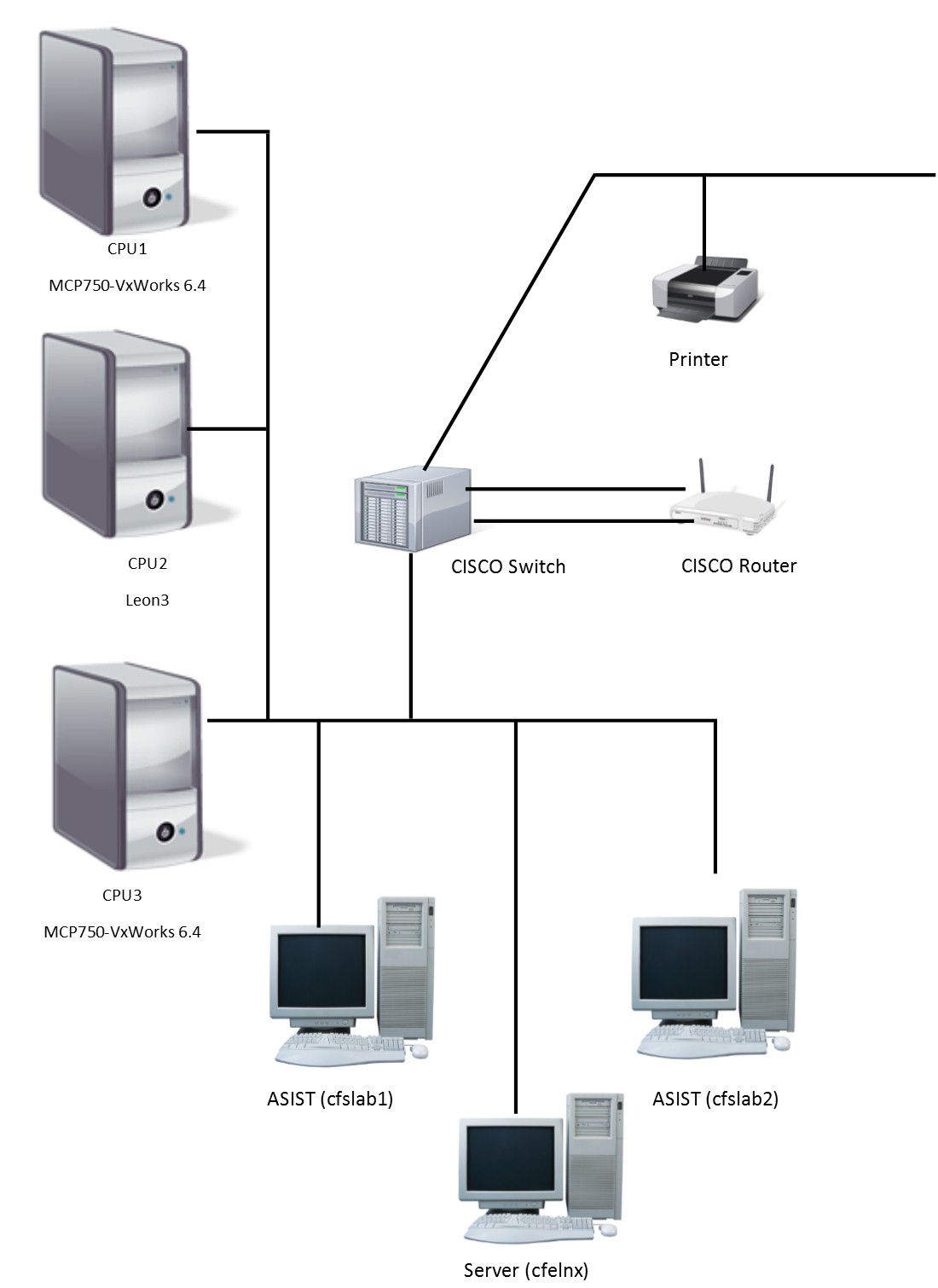
* Logs with the .loge extension list all events sent by the flight software
* Logs with the .logr extension list all requirements that passed validation by demonstration
* Logs with the .logp extension lists all prints that are generated by the test procedure
* Logs with the .logf extension lists everything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

# Build Verification Test Execution

## Testbed Overview

LC 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 N406D. This facility consists of two ASIST workstations running ASIST version 20.2 and three MPC750 CPU boards running VxWorks 6.9 and 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

|  |  |
| --- | --- |
|  | Limit Checker  (LC) |
| Requirements Tested Passed | 62 |
| Requirements Tested Failed | 0 |
| Requirements Tested Partially | 0 |
| Total Tested | 62 |
| Deferred | 0 |
| Total | 62 |

## Requirements Partially Tested

No requirements 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 LC Application the software behaved as expected. Below is a summary of the results:

* 62 requirements passed via demonstration
* 0 requirements failed testing.
* 10 DCRs were verified

## Procedure Description

| **Procedure** | **Description** | **Requirements tested** |
| --- | --- | --- |
| lc\_evtfilter | The purpose of this test is to verify that Limit Checker (LC) properly filters the PASS to FAIL and FAIL to PASS transition events when the table-defined maximum number of times has been reached for an ActionPoint (AP). | LC1003, LC3001, LC3001.2, LC3002, LC3002.1, LC3002.1.1, LC3002.2, LC3002.2.1, LC3002.3, LC3002.3.1, LC3002.4, LC3006, LC4000, LC8000, LC9000, LC9001, LC9002 |
| lc\_gencmds | The purpose of this test is to verify that the LC general commands execute and function properly. | LC1000, LC1001, LC1002, LC1003, LC1004, LC2004, LC3006, LC4000, LC4001, LC4002, LC4003, LC4004, LC4005, LC4006, LC4007, LC4008, LC4009, LC4009.1, LC4010, LC4011, LC4012, LC4013, LC8000, LC9000, LC9001, LC9002 |
| lc\_monitoring | The purpose of this test is to verify that the LC application functions properly when monitoring WatchPoints (WP). All WPs evaluate to FALSE so no thresholds will be reached to cause ActionPoints (AP) to take action. | LC1003, LC2003, LC2003.1, LC2003.2, LC2004, LC3001, LC3001.1, LC3002, LC3003, LC3004, LC3005, LC3006, LC4000, LC4001, LC4002, LC4003, LC4004, LC4005, LC4006, LC4007, LC4008, LC4009, LC8000, LC9000, LC9001, LC9002, LC9005, LC9006; LC9007 |
| lc\_noaction | The purpose of this tese is to verify that the LC application functions properly when monitoring Watchpoints (WP). All WP evaluate to a mix of TRUE, FALSE, and STALE. All ActionPoints (AP) PASS which results in no thresholds being reached and thus, no actions taken. | LC1003, LC2003, LC2003.1, LC2003.3, LC2004, LC3001, LC3001.1, LC3002, LC3003, LC3004, LC3005, LC3006, LC4000, LC4003, LC4004, LC4005, LC4006, LC4007, LC4009, LC8000, LC9000, LC9001, LC9002 |
| lc\_resetcds | The purpose of this test is to verify that the LC application initializes the appropriate data items based upon the type of reset that occurs (Application, Processor, or Power-On). This test should NOT be executed if the configuration parameter indicating Save Critical Data is not set by the Mission. | LC1003, LC2004, LC3006, LC4000, LC4004, LC8000, LC9000, LC9001, LC9002, LC9004, LC9004.1, LC9004.1.1, LC9004.2, LC9005, LC9006, LC9007, LC9007.1, LC9007.2 |
| lc\_resetcdsstate | The purpose of this test is to verify that the LC application restores the proper Application State when an Application or cFE Processor Reset occurs. This test should NOT be executed if the configuration parameter indicating Save Critical Data is not set by the Mission. | LC1003, LC4000, LC8000, LC9000, LC9001, LC9002, LC9004.1, LC9005, LC9006, LC9007, LC9007.2 |
| lc\_resetnocds | The purpose of this test is to verify that the LC application does not save any data across a reset (Application, Processor, or Power-On). This test should NOT be executed if the configuration parameter indicating Save Critical Data is set by the Mission. | LC2004, LC3006, LC8000, LC9000, LC9001, LC9002, LC9003, LC9005, LC9006, LC9007 |
| lc\_stale | The purpose of this test is to verify that the LC application properly handles “stale” data. Stale data is data that has not been received for x Sample commands as specified in the WatchPoint Definition table. | LC1003, LC2001, LC2002, LC2004, LC3001, LC3002, LC3006, LC4000, LC8000, LC9000, LC9001, LC9002 |
| lc\_stress | The purpose of this test is to verify that the LC application functions properly when monitoring the maximum number of WatchPoints (WP). All evaluate to FALSE. The maximum number of ActionPoints (SP) evaluate to FAIL which in turn trigger the execution of an RTS. Also, Anomaly tests are performed to ensure the appropriate action is taken by the LC application. | LC1003, LC2000, LC2003, LC2003.1, LC2003.4, LC2004, LC3000, LC3001, LC3001.1, LC3001.3, LC3002, LC3002.3, LC3002.3.1, LC3002.4, LC3006, LC4000, LC4003, LC4004, LC4006, LC8000, LC9000, LC9001, LC9002 |
| lc\_tabletesting | The purpose of this test is to verify that the LC application functions properly when loading new WatchPoint Definition Tables (WDT) and ActionPoint Definition Tables (ADT). The tables are updated while the LC application is in the ACTIVE, PASSIVE, and DISABLED state. Other tests attempt to load invalid tables to ensure that the validation functions correctly reject the table loads. | LC1003, LC2004, LC2005, LC3006, LC3007, LC4000, LC4001, LC4002, LC4004, LC8000, LC9000, LC9001, LC9002, LC9005, LC9006, LC9007 |
| lc\_withaction | The purpose of this test is to verify that the LC application functions properly when monitoring WathPoints (WP). All WP evaluate to a mix of TRUE, FALSE, and STALE. All ActionPoints (AP) evaluate to a mix of PASS, FAIL, and STALE. | LC1003, LC2003, LC2003.1, LC2003.2, LC2003.3, LC2004, LC2005, LC3001, LC3001.1, LC3001.2, LC3002, LC3002.1, LC3002.1.1; LC3002.2, LC3002.2.1; LC3002.3, LC3002.3.1, LC3002.4, LC3003, LC3004, LC3005, LC3006, LC3007, LC4000, LC4001, LC4002, LC4003, LC4004, LC4005, LC4006, LC4007, LC4009, LC8000, LC9000, LC9001, LC9002; LC9005; LC9006; LC9007 |

## Analysis Requirements Verification

No requirements were verified using analysis.

## Failed Requirements

No requirements failed during LC 2.1.0.0 testing:

## DCRs and Trac tickets

No DCRs were generated during LC 2.1.0.0 testing.

### DCRs Verified

The following DCRs were verified during testing. Trac ticket references are proceeded with a ‘#’ character.

|  |  |  |  |
| --- | --- | --- | --- |
| **DCR/Trac #** | **Description** | **Test Method** | **Test Approach** |
| 3920 | GPM-IVV-1251 – Incorrect implementation of LC3002.4 | Test Procedure | There were 2 counters associated with this requirement. One for the ActionPoint which was checked correctly by the lc\_evtfilter procedure. The other was an LC housekeeping counter that was not incremented or checked prior to this build. The housekeeping counter indicates the total number of RTSs not started or initiated. The lc\_stress and lc\_withaction procedures test that the counter increments in the PASSIVE LC application state.  Confusion occurred here based upon documentation and the wording of the requirement. This DCR is marked passed and a new DCR #146229 addresses the documentation issues. |
| 3972 | GPS-IVV-1309 – LC – Incorrect value representations being stored in WP Results Table | Test Procedure | The Watchpoint Definition Table displays the comparison values. The values displayed during testing did not show any incorrect representations. |
| 4046 | LC – Table Definitions are Unclear | Analysis | The change submitted now prevents the LC application from being configured to cause the “unclear” issue. |
| 4095 | IV & V CFS BVT Findings – LC3006 Item h) Not Implemented | Test Procedure | The counter is now implemented and displayed on the Actionpoint Results Table display page. The lc\_evtfilter test procedure verifies this requirement. |
| 145597 | LC: Misplaced else-case | Test Procedure | The else clause generates an informational event message when LC is started without using the CDS. This message was found in each test procedure that was run in this configuration. |
| 145598 | LC: Unused / Unneeded Variables in lc\_app.c | Inspection | The stated solution was found in the file submitted with this DCR. |
| #5 | LC application is not endian-neutral | Test Procedure | The stated solution was found in the files submitted with this DCR. All test procedures passed on a big endian platform. The LC application was NOT tested on a little endian platform. |
| 145919 | LCX – CFE\_EVS\_SendEvent Format Warnings | Analysis | No compiler warnings were generated during the make process. |
| 146194 | LC – Need to add padding between new variable in WRTTransition\_t type. | Inspection/  Demonstration | The stated padding was found in the file submitted with this DCR. The padding allows the packet to be properly aligned to a 32-bit boundary. |
| 146229 | LC: Requirement 3002.4 and comments for Passive RTSExec Count are wrong | Inspection | The requirement text and the doxygen comments in lc\_msg.h have been updated to reflect the fsw implementation. This clears up any confusion as to the expected behavior of this counter. |

### Outstanding DCRs

The following DCRs exist against LC. Trac ticket references are proceeded with a ‘#’ character.

|  |  |
| --- | --- |
| **DCR/Trac #** | **Description** |
| 4117 | LC - Add Trick Simulation Support (JSC Request) |
| #94 | LC transitions Active APs to Passive When Application is in Passive Mode. During a GPM rehearsal there were several APs that were commanded "active" while the LC application state was in "passive" mode. Before operations could command the application state to "active" mode, some of the APs that were activated and had "tripped" causing the AP to transition back to passive mode. The purpose of changing a "tripped" APs state from active to passive is to prevent an RTS from getting initiated more than once. In "passive" mode, LC performs all limit tests as in "active" mode, but no stored command sequences are invoked as the result of AP failures. Having the AP's state transition while the application is in passive mode will make enabling APs with a low threshold while LC is in passive mode very difficult. The rational for this design feature (LRO heritage) needs to be clearly understood and documented. The LC user's guides (both doxygen and word/pdf) do not make this design feature clear. If no rational exists this design feature should be removed from LC. |
| #80 | LC does not support 64-bit integer or floats (doubles). |

## Notes

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

1. RTTM

The LC Build 2.1.0.0 RTTM can be found on the MKS server, in cFS-Repository LC test-and-ground directory results folder.

1. Command, Telemetry, and Events Verification Matrix

|  |  |  |
| --- | --- | --- |
| **Command** | **Test Procedure(s)** | **Notes/Comments** |
| LC\_NOOP | lc\_gencmds |  |
| LC\_ResetCtrs | lc\_gencmds, lc\_stress, lc\_withaction |  |
| LC\_SetLCState | lc\_evtfilter, lc\_gencmds, lc\_monitoring, lc\_noaction, lc\_resetcds, lc\_resetcdsstate, lc\_stale, lc\_stress, lc\_tabletesting, lc\_withaction |  |
| LC\_SetAPState | lc\_evtfilter, lc\_gencmds, lc\_monitoring, lc\_noaction, lc\_resetcds, lc\_stale, lc\_stress, lxc\_tabletesting, lc\_withaction |  |
| LC\_SetAPPermOff | lc\_gencmds, lc\_monitoring, lc\_noaction |  |
| LC\_ResetAPStats | lc\_gencmds, lc\_stress, lc\_withaction |  |
| LC\_ResetWPStats | lc\_gencmds, lc\_stress, lc\_withaction |  |

|  |  |  |
| --- | --- | --- |
| **Telemetry** | **Test Procedure(s)** | **Notes/Comments** |
| LC\_CMDPC | All |  |
| LC\_CMDEC | All |  |
| LC\_APSAMPLECNT | All |  |
| LC\_MONMSGCNT | All |  |
| LC\_RTSCNT | All |  |
| LC\_PASSRTSCNT | All |  |
| LC\_WPSINUSE | lc\_noaction, lc\_resetcds, lc\_stress, lc\_tabletesting, lc\_withaction |  |
| LC\_ACTIVEAPS | lc\_noaction, lc\_resetcds, lc\_stress, lc\_tabletesting, lc\_withaction |  |
| LC\_CURLCSTATE | All |  |
| WRRESULTS[] | All |  |
| ARRESULTS[] | All |  |
| **Table Telemetry** |  |  |
| LC\_ADT[].DefaultState | lc\_adt1, lc\_adt1a, lc\_adt1b, lc\_adt2, lc\_adt2a, lc\_adt3, lc\_adt4, lc\_adt4a, lc\_adt5, lc\_adt6 |  |
| LC\_ADT[].MaxPassiveEvents | lc\_adt1, lc\_adt1a, lc\_adt1b, lc\_adt2, lc\_adt2a, lc\_adt3, lc\_adt4, lc\_adt4a, lc\_adt5, lc\_adt6 |  |
| LC\_ADT[].MaxPassFailEvents | lc\_adt1, lc\_adt1a, lc\_adt1b, lc\_adt2, lc\_adt2a, lc\_adt3, lc\_adt4, lc\_adt4a, lc\_adt5, lc\_adt6 |  |
| LC\_ADT[].MaxFailPassEvents | lc\_adt1, lc\_adt1a, lc\_adt1b, lc\_adt2, lc\_adt2a, lc\_adt3, lc\_adt4, lc\_adt4a, lc\_adt5, lc\_adt6 |  |
| LC\_ADT[].RTSId | lc\_adt1, lc\_adt1a, lc\_adt1b, lc\_adt2, lc\_adt2a, lc\_adt3, lc\_adt4, lc\_adt4a, lc\_adt5, lc\_adt6 |  |
| LC\_ADT[].MaxFailsBefRTS | lc\_adt1, lc\_adt1a, lc\_adt1b, lc\_adt2, lc\_adt2a, lc\_adt3, lc\_adt4, lc\_adt4a, lc\_adt5, lc\_adt6 |  |
| LC\_ADT[].RPNEquation[] | lc\_adt1, lc\_adt1a, lc\_adt1b, lc\_adt2, lc\_adt2a, lc\_adt3, lc\_adt4, lc\_adt4a, lc\_adt5, lc\_adt6 |  |
| LC\_ADT[].EventType | lc\_adt1, lc\_adt1a, lc\_adt1b, lc\_adt2, lc\_adt2a, lc\_adt3, lc\_adt4, lc\_adt4a, lc\_adt5, lc\_adt6 |  |
| LC\_ADT[].EventId | lc\_adt1, lc\_adt1a, lc\_adt1b, lc\_adt2, lc\_adt2a, lc\_adt3, lc\_adt4, lc\_adt4a, lc\_adt5, lc\_adt6 |  |
| LC\_ADT[].EventText[] | lc\_adt1, lc\_adt1a, lc\_adt1b, lc\_adt2, lc\_adt2a, lc\_adt3, lc\_adt4, lc\_adt4a, lc\_adt5, lc\_adt6 |  |
| LC\_ART[].ActionResult | All |  |
| LC\_ART[].CurrentState | All |  |
| LC\_ART[].PassiveAPCount | lc\_evtfilter, lc\_stale |  |
| LC\_ART[].FailToPassCount | All |  |
| LC\_ART[].PassToFailCount | All |  |
| LC\_ART[].ConsecutiveFailCount | All |  |
| LC\_ART[].CumulativeFailCount | All |  |
| LC\_ART[].CumulativeRTSExecCount | All |  |
| LC\_WDT[].DataType | lc\_wdt1, lc\_wdt2, lc\_wdt2a, lc\_wdt3, lc\_wdt4, lc\_wdt4a, lc\_wdt5 |  |
| LC\_WDT[].OperatorID | lc\_wdt1, lc\_wdt2, lc\_wdt2a, lc\_wdt3, lc\_wdt4, lc\_wdt4a, lc\_wdt5 |  |
| LC\_WDT[].MessageID | lc\_wdt1, lc\_wdt2, lc\_wdt2a, lc\_wdt3, lc\_wdt4, lc\_wdt4a, lc\_wdt5 |  |
| LC\_WDT[].WPOffset | lc\_wdt1, lc\_wdt2, lc\_wdt2a, lc\_wdt3, lc\_wdt4, lc\_wdt4a, lc\_wdt5 |  |
| LC\_WDT[].BitMask | lc\_wdt1, lc\_wdt2, lc\_wdt2a, lc\_wdt3, lc\_wdt4, lc\_wdt4a, lc\_wdt5 |  |
| LC\_WDT[].ComparisonValue.Signed32 | lc\_wdt1, lc\_wdt2, lc\_wdt2a, lc\_wdt3, lc\_wdt4, lc\_wdt4a, lc\_wdt5 |  |
| LC\_WDT[].ComparisonValue.UnSigned32 | lc\_wdt1, lc\_wdt2, lc\_wdt2a, lc\_wdt3, lc\_wdt4, lc\_wdt4a, lc\_wdt5 |  |
| LC\_WDT[].ComparisonValue.Float32 | lc\_wdt1, lc\_wdt2, lc\_wdt2a, lc\_wdt3, lc\_wdt4, lc\_wdt4a, lc\_wdt5 |  |
| LC\_WDT[].StaleAge | lc\_wdt1, lc\_wdt2, lc\_wdt2a, lc\_wdt3, lc\_wdt4, lc\_wdt4a, lc\_wdt5 |  |
| LC\_WDT[].CustFctArgument | lc\_wdt1, lc\_wdt2, lc\_wdt2a, lc\_wdt3, lc\_wdt4, lc\_wdt4a, lc\_wdt5 |  |
| LC\_WRT[].WatchResults | All |  |
| LC\_WRT[].CountdownToStale | lc\_stale |  |
| LC\_WRT[].EvaluationCount | All |  |
| LC\_WRT[].FalseToTrueCount | All |  |
| LC\_WRT[].ConsecutiveTrueCount | All |  |
| LC\_WRT[].CumulativeTrueCount | All |  |
| LC\_WRT[].FtoTValue | All |  |
| LC\_WRT[].SecFtoTTimeStamp | lc\_gencmds |  |
| LC\_WRT[].SubSecFtoTTimeStamp | lc\_gencmds |  |
| LC\_WRT[].TtoFValue | All |  |
| LC\_WRT[].SecTtoFTimeStamp | lc\_gencmds |  |
| LC\_WRT[].SubSecTtoFTimeStamp | lc\_gencmds |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Event Message** | **Test Procedure(s)** | **Notes/Comments** |
| 1 | LC\_TASK\_EXIT\_EID |  |  |
| 2 | LC\_INIT\_INF\_EID | lc\_evtfilter, lc\_gencmds, lc\_monitoring, lc\_noaction, lc\_resetcds, lc\_resetcdsstate, lc\_resetnocds, lc\_stale, lc\_stress, lc\_tabletesting, lc\_withaction |  |
| 3 | LC\_CR\_PIPE\_ERR\_EID |  |  |
| 4 | LC\_SUB\_HK\_REQ\_ERR\_EID |  |  |
| 5 | LC\_SUB\_GND\_CMD\_ERR\_EID |  |  |
| 6 | LC\_SUB\_SAMPLE\_CMD\_ERR\_EID |  |  |
| 7 | LC\_WDT\_REGISTER\_ERR\_EID |  |  |
| 8 | LC\_WDT\_REREGISTER\_ERR\_EID |  |  |
| 9 | LC\_ADT\_REGISTER\_ERR\_EID |  |  |
| 10 | LC\_WRT\_REGISTER\_ERR\_EID |  |  |
| 11 | LC\_ART\_REGISTER\_ERR\_EID |  |  |
| 12 | LC\_WRT\_CDS\_REGISTER\_ERR\_EID |  |  |
| 13 | LC\_ART\_CDS\_REGISTER\_ERR\_EID |  |  |
| 14 | LC\_APP\_CDS\_REGISTER\_ERR\_EID |  |  |
| 15 | LC\_WDT\_LOAD\_ERR\_EID |  |  |
| 16 | LC\_ADT\_LOAD\_ERR\_EID |  |  |
| 17 | LC\_WRT\_GETADDR\_ERR\_EID |  |  |
| 18 | LC\_ART\_GETADDR\_ERR\_EID |  |  |
| 19 | LC\_WDT\_GETADDR\_ERR\_EID |  |  |
| 20 | LC\_ADT\_GETADDR\_ERR\_EID |  |  |
| 21 | LC\_CDS\_RESTORED\_INF\_EID |  |  |
| 22 | LC\_CDS\_UPDATED\_INF\_EID | lc\_resetcds, lc\_resetcdsstate |  |
| 23 | LC\_CDS\_DISABLED\_INF\_EID |  |  |
| 24 | LC\_CC\_ERR\_EID | lc\_gencmds |  |
| 25 | LC\_APSAMPLE\_APNUM\_ERR\_EID | lc\_gencmds, lc\_stress |  |
| 26 | LC\_NOOP\_INF\_EID | lc\_gencmds, lc\_stress |  |
| 27 | LC\_RESET\_DBG\_EID | lc\_gencmds, lc\_stress, lc\_withaction |  |
| 28 | LC\_LCSTATE\_INF\_EID | lc\_evtfilter, lc\_gencmds, lc\_monitoring, lc\_noaction, lc\_resetcds, lc\_resetcdsstate, lc\_stale, lc\_stress, lc\_tabletesting, lc\_withaction |  |
| 29 | LC\_LCSTATE\_ERR\_EID | lc\_gencmds |  |
| 30 | LC\_APSTATE\_NEW\_ERR\_EID | lc\_gencmds |  |
| 31 | LC\_APSTATE\_CURR\_ERR\_EID | lc\_gencmds |  |
| 32 | LC\_APSTATE\_APNUM\_ERR\_EID | lc\_gencmds |  |
| 33 | LC\_APSTATE\_INF\_EID | lc\_evtfilter, lc\_gencmds, lc\_monitoring, lc\_noaction, lc\_resetcds, lc\_stale, lc\_stress, lc\_tabletesting, lc\_withaction |  |
| 34 | LC\_APOFF\_APNUM\_ERR\_EID | lc\_gencmds |  |
| 35 | LC\_APOFF\_CURR\_ERR\_EID | lc\_gencmds |  |
| 36 | LC\_APOFF\_INF\_EID | lc\_gencmds, lc\_monitoring, lc\_noaction |  |
| 37 | LC\_APSTATS\_APNUM\_ERR\_EID | lc\_gencmds |  |
| 38 | LC\_APSTATS\_INF\_EID | lc\_gencmds, lc\_stress, lc\_withaction |  |
| 39 | LC\_WPSTATS\_WPNUM\_ERR\_EID | lc\_gencmds |  |
| 40 | LC\_WPSTATS\_INF\_EID | lc\_gencmds, lc\_stress, lc\_withaction |  |
| 41 | LC\_HKREQ\_LEN\_ERR\_EID | lc\_gencmds |  |
| 42 | LC\_APSAMPLE\_LEN\_ERR\_EID | lc\_gencmds |  |
| 43 | LC\_LEN\_ERR\_EID | lc\_gencmds |  |
| 44 | LC\_UNSUB\_WP\_ERR\_EID |  |  |
| 45 | LC\_SUB\_WP\_ERR\_EID |  |  |
| 46 | LC\_WRT\_NO\_SAVE\_ERR\_EID |  |  |
| 47 | LC\_ART\_NO\_SAVE\_ERR\_EID |  |  |
| 48 | LC\_APP\_NO\_SAVE\_START\_ERR\_EID |  |  |
| 49 | LC\_MID\_INF\_EID |  |  |
| 50 | LC\_WP\_DATATYPE\_ERR\_EID |  |  |
| 51 | LC\_WP\_OPERID\_ERR\_EID |  |  |
| 52 | LC\_WP\_NAN\_ERR\_EID |  |  |
| 53 | LC\_WP\_OFFSET\_ERR\_EID | lc\_stale, lc\_stress |  |
| 54 | LC\_WDTVAL\_FPERR\_EID |  |  |
| 55 | LC\_WDTVAL\_ERR\_EID | lc\_tabletesting |  |
| 56 | LC\_WDTVAL\_INF\_EID | lc\_evtfilter, lc\_gencmds, lc\_monitoring, lc\_noaction, lc\_resetcds, lc\_resetnocds, lc\_resetcdsstate, lc\_stale, lc\_stress, lc\_tabletesting, lc\_withaction |  |
| 57 | LC\_APSAMPLE\_CURR\_ERR\_EID | lc\_gencmds, lc\_monitoring, lc\_noaction |  |
| 58 | LC\_AP\_PASSTOFAIL\_INF\_EID | lc\_evtfilter, lc\_resetcds, lc\_withaction |  |
| 59 | LC\_PASSIVE\_FAIL\_DBG\_EID | lc\_stress, lc\_withaction |  |
| 60 | LC\_AP\_PASSIVE\_FAIL\_DBG\_EID | lc\_evtfilter, lc\_resetcds, lc\_stale, lc\_stress, lc\_withaction |  |
| 61 | LC\_AP\_FAILTOPASS\_INF\_EID | lc\_evtfilter, lc\_resetcds, lc\_withaction |  |
| 62 | LC\_ACTION\_ERROR\_ERR\_EID | lc\_stale, lc\_stress |  |
| 63 | LC\_INVALID\_RPN\_ERR\_EID |  |  |
| 64 | LC\_ADTVAL\_RPNERR\_EID |  |  |
| 65 | LC\_ADTVAL\_ERR\_EID | lc\_tabletesting |  |
| 66 | LC\_ADTVAL\_INF\_EID | lc\_evtfilter, lc\_gencmds, lc\_monitoring, lc\_noaction, lc\_resetcds, lc\_resetnocds, lc\_resetcdsstate, lc\_stale, lc\_stress, lc\_tabletesting, lc\_withaction |  |
| 67 | LC\_CFCALL\_ERR\_EID | lc\_evtfilter, lc\_monitoring, lc\_noaction, lc\_resetcds, lc\_stale, lc\_withaction |  |
| 1000 | LC\_BASE\_AP\_EID |  |  |