Code 582
Flight Software Branch

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:

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: 2017.08.29 07:08:58 -04'00'

Walt Moleski/582
cFS Flight Software Tester

Approved by:

X SUSANNE STREGE

Digitally signed by SUSANNE STREGE
Date: 2017.08.28 18:37:02 -04'00'

Susanne Strege/582
cFS Flight Software Product Development Le...

PLAN UPDATE HISTORY

Version	Date	Description	Affected Pages
1.0		Initial Release	All

TABLE OF CONTENTS

1 INTRODUCTION

1.1 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.

1.2 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

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:

- 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.

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.

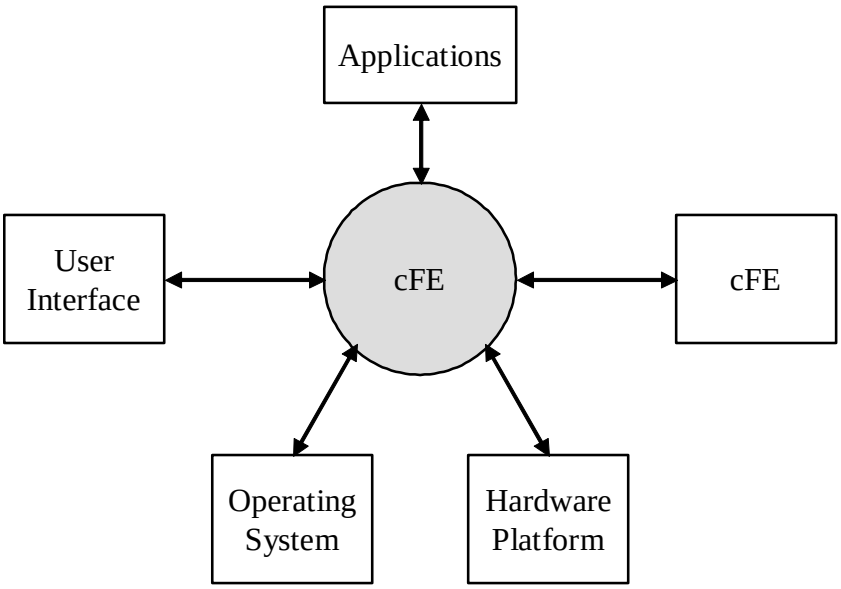

Figure 2-1 cFS System Context

The figure below shows major interfaces between the 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.

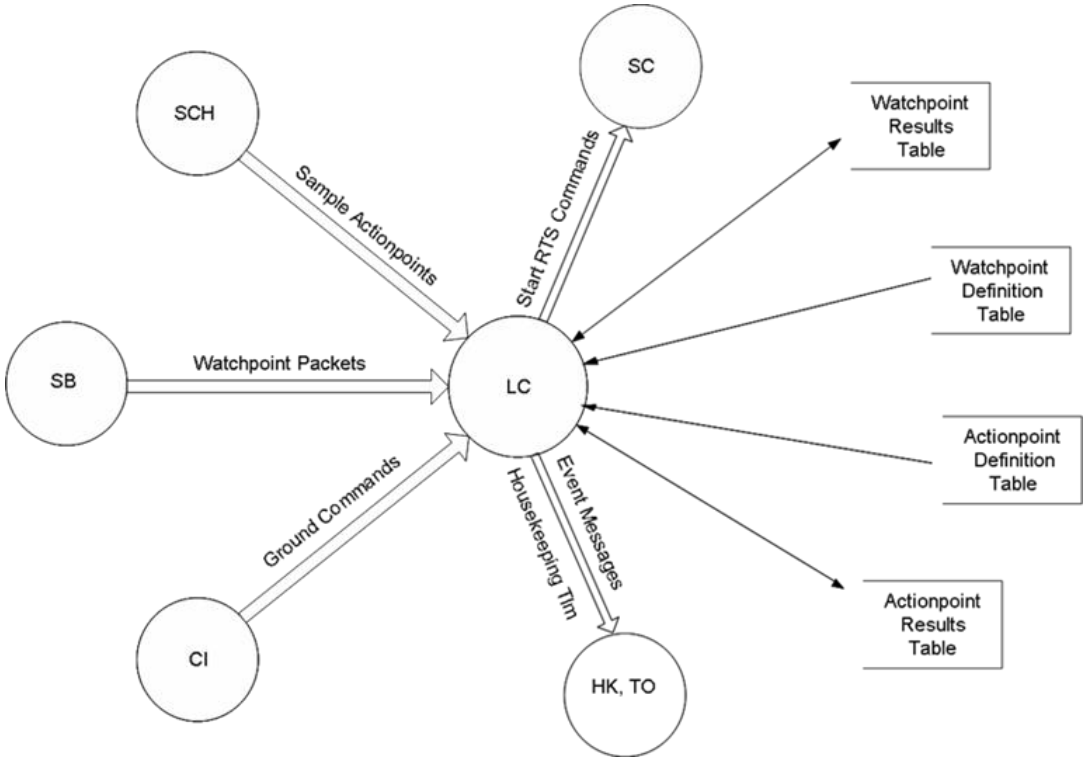

Figure 2-1 cFS LC Context

2.2 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

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 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.

Procedure	Description
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

2.4 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

3 BUILD VERIFICATION TEST PREPARATION

3.1 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.

3.2 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>.

3.3 TEST PRODUCTS

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

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

A test summary 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.

4 BUILD VERIFICATION TEST EXECUTION

4.1 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.

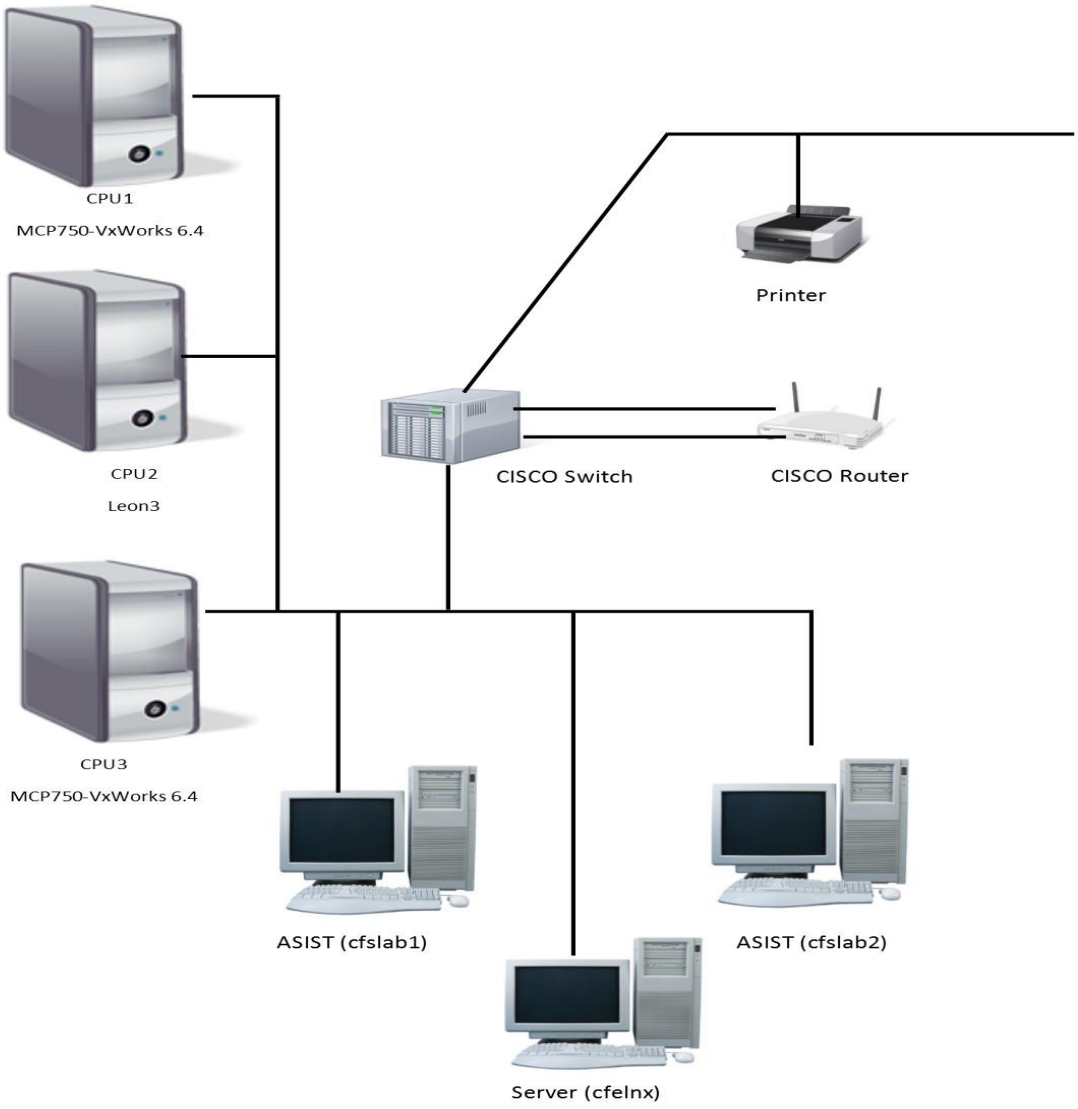

Figure 4-1 cFS FSW Development and Testing Facility

4.2 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

4.3 REQUIREMENTS PARTIALLY TESTED

No requirements 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 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

5.2 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

Procedure	Description	Requirements tested
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

5.3 ANALYSIS REQUIREMENTS VERIFICATION

No requirements were verified using analysis.

5.4 FAILED REQUIREMENTS

No requirements failed during LC 2.1.0.0 testing:

5.5 DCRS AND TRAC TICKETS

No DCRs were generated during LC 2.1.0.0 testing.

5.5.1 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.

5.5.2 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).

5.6 NOTES

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

APPENDIX A - 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.

APPENDIX B - 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		